HITACHI
Inspire the Next

Cosminexus V11 アプリケーションサーバ Web サービス開発ガイド

手引書

3021-3-J23-50

前書き

■ 対象製品

マニュアル「アプリケーションサーバ & BPM/ESB 基盤 概説」の前書きの対象製品の説明を参照してください。

■ 輸出時の注意

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。

なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

HITACHI，Cosminexus，JP1，TPBroker，uCosminexus は，株式会社 日立製作所の商標または登録商標です。

Microsoft，Excel，Windows，Windows Server は，マイクロソフト 企業グループの商標です。

Oracle(R)，Java ，MySQL 及び NetSuite は，Oracle，その子会社及び関連会社の米国及びその他の国における登録商標です。

その他記載の会社名，製品名などは，それぞれの会社の商標もしくは登録商標です。

■ マイクロソフト製品のスクリーンショットの使用について

マイクロソフトの許可を得て使用しています。

■ 発行

2024 年 2 月 3021-3-J23-50

■ 著作権

変更内容

変更内容(3021-3-J23-50) uCosminexus Application Server 11-40，uCosminexus Client 11-40，uCosminexus Developer 11-40，uCosminexus Service Architect 11-40，uCosminexus Service Platform 11-40

追加・変更内容	変更個所
マニュアル訂正の内容を反映した。	-

単なる誤字・脱字などはお断りなく訂正しました。

はじめに

このマニュアルをお読みになる際の前提情報については，マニュアル「アプリケーションサーバ & BPM/ESB 基盤 概説」のはじめにの説明を参照してください。

目次

第 2 編　開発と実行

1 Web サービスの開発および実行の概要

Application Server の JAX-WS 機能を使用して，JAX-WS 2.2 仕様に従った SOAP Web サービスを開発できます。また，Application Server の JAX-RS 機能を使用して，JAX-RS 1.1 仕様に従った RESTful Web サービス（Web リソース）を開発できます。

この章では，Web サービス開発の概要，および前提条件について説明します。

1.1 JAX-WS・JAX-RS 仕様の対応バージョン，プレフィクスおよび名前空間 URI

このマニュアルは，Application Server で提供する機能を利用して，JAX-WS 2.2 仕様または JAX-RS 1.1 仕様に対応した Web サービスを開発，実行する方法について説明したものです。

JAX-WS 2.2 仕様，JAX-RS 1.1 仕様，およびプレフィクスと名前空間 URI の対応については，次に示します。

JAX-WS 2.2 仕様の対応バージョン

このマニュアルで「JAX-WS 2.2 仕様」と表記した場合，次のバージョンの仕様を意味します。

```
Specification: JSR-000224 - Java™ API for XML-Based Web Services
Version: 2.2
Status: Maintenance Release 3
Release: 10 December 2009
```

また，このマニュアルで「バインディング宣言のスキーマ」と表記した場合，次のスキーマを意味します。

```
http://java.sun.com/xml/ns/jaxws/wsdl_customizationschema_2_0.xsd(Date Published: May 11,
 2006)
```

JAX-RS 1.1 仕様の対応バージョン

このマニュアルで「JAX-RS 1.1 仕様」と表記した場合，次のバージョンの仕様を意味します。

```
Specification: JSR-000311 - Java™ API for RESTful Web Services
Version: 1.1
Status: Final Release
Release: September 17, 2009
```

プレフィクスと名前空間 URI の対応

このマニュアルで使用するプレフィクスと名前空間 URI の対応を次に示します。特に断りがないかぎり，次のプレフィクスを使用します。

表 1-1　プレフィクスと名前空間 URI の対応

項番	プレフィクス	名前空間 URI
1	cwsrm	http://jaxws.cosminexus.com/cwsrm
2	er	urn:oasis:names:tc:entity:xmlns:xml:catalog
3	javaee	http://java.sun.com/xml/ns/javaee
4	jaxb	http://java.sun.com/xml/ns/jaxb
5	jaxws	http://java.sun.com/xml/ns/jaxws
6	jaxwsdd	http://java.sun.com/xml/ns/jax-ws/ri/runtime
7	net35rmp	http://schemas.microsoft.com/ws-rx/wsrmp/200702

項番	プレフィクス	名前空間 URI
8	S	S11 または S12 を示す。
9	S11	http://schemas.xmlsoap.org/soap/envelope/
10	S12	http://www.w3.org/2003/05/soap-envelope
11	soap	http://schemas.xmlsoap.org/wsdl/soap/
12	soap12	http://schemas.xmlsoap.org/wsdl/soap12/
13	soapenv	http://schemas.xmlsoap.org/soap/envelope/
14	soapenv12	http://www.w3.org/2003/05/soap-envelope
15	wsa	http://www.w3.org/2005/08/addressing
16	wsam	http://www.w3.org/2007/05/addressing/metadata
17	wsaw	http://www.w3.org/2006/05/addressing/wsdl
18	wsdl	http://schemas.xmlsoap.org/wsdl/
19	wsi	http://ws-i.org/profiles/basic/1.1/xsd
20	wsp	http://www.w3.org/ns/ws-policy
21	wsrm	http://docs.oasis-open.org/ws-rx/wsrm/200702
22	wsrmp	http://docs.oasis-open.org/ws-rx/wsrmp/200702
23	wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
24	xmime	http://www.w3.org/2005/05/xmlmime
25	xop	http://www.w3.org/2004/08/xop/include
26	xsd	http://www.w3.org/2001/XMLSchema
27	xsi	http://www.w3.org/2001/XMLSchema-instance

1.2　Web サービスの開発の概要

Application Server では，Web サービスの通信基盤として JAX-WS エンジンおよび JAX-RS エンジンを提供しています。JAX-WS エンジンは，JAX-WS 2.2 仕様に従って，SOAP メッセージのバインディングを実現します。JAX-RS エンジンは，JAX-RS 1.1 仕様に従って，RESTful HTTP メッセージのバインディングを実現します。

このマニュアルでは，JAX-WS エンジンまたは JAX-RS エンジンを介して利用できる Web サービスの開発方法について説明します。

1.2.1　SOAP Web サービスの開発の概要

SOAP Web サービス（JAX-WS エンジンを利用した Web サービス）は，次のどれかの方法で開発します。

- **WSDL を起点とした開発**

  Web サービスの定義を記述した WSDL を起点とした開発方法です。コマンドで WSDL から Java ソースを生成し，必要な処理を実装します。
- **SEI（サービスエンドポイントインタフェース）を起点とした開発**

  SEI を起点とした開発方法です。作成した Web サービス実装クラスから，コマンドで Web サービスの実装に必要な追加の Java ソース（Java Beans クラスなど）を生成します。Web サービス実装クラスは，POJO としても，EJB を基にしても作成できます。
- **プロバイダを起点とした開発**

  プロバイダを起点とした開発方法です。プロバイダ実装クラスを実装します。

それぞれの開発の流れについては，「2.1　SOAP Web サービス開発の流れ」を参照してください。

参考

SOAP アプリケーションの開発について

既存機能である SOAP アプリケーション開発支援機能および SOAP 通信基盤を使用して，Web サービスを開発することもできます（このときに開発するアプリケーションを「SOAP アプリケーション」と呼びます）。ただし，SOAP アプリケーションは SOAP 通信基盤で動作することを前提としているため，JAX-WS エンジン上で利用する場合には条件があります。

SOAP アプリケーションの開発については，マニュアル「アプリケーションサーバ SOAP アプリケーション開発の手引」を参照してください。

JAX-WS エンジン上で利用する場合の条件および移行手順については，「付録 A　旧バージョンからの移行」を参照してください。

1.2.2　RESTful Web サービスの開発の概要

RESTful Web サービス（JAX-RS エンジンを利用した Web サービス）は，Web リソースを実装して開発します。Web リソースは，HTTP リクエストに対しての処理を実装します。なお，RESTful Web サービスそのものを Web リソースと呼ぶこともあります。

開発の流れについては，「2.3　RESTful Web サービスの開発の流れ」を参照してください。

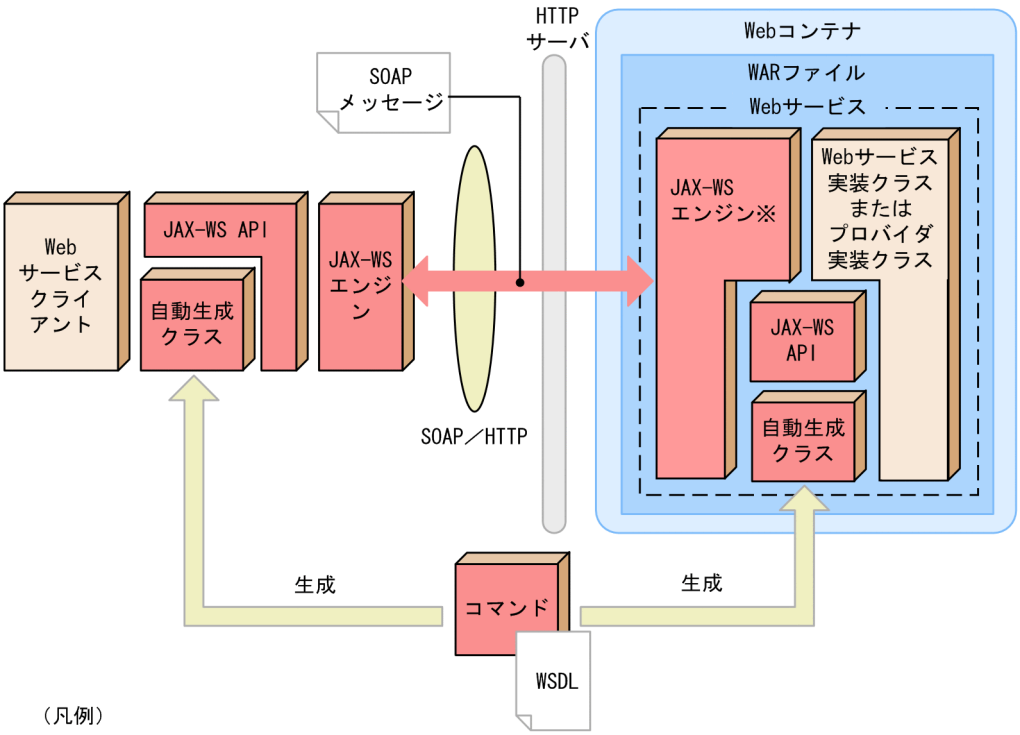
1.3　Web サービスの開発時および実行時に使用する機能

SOAP Web サービスと RESTful Web サービス（Web リソース）の開発時および実行時に使用する機能について説明します。

1.3.1　SOAP Web サービスの機能

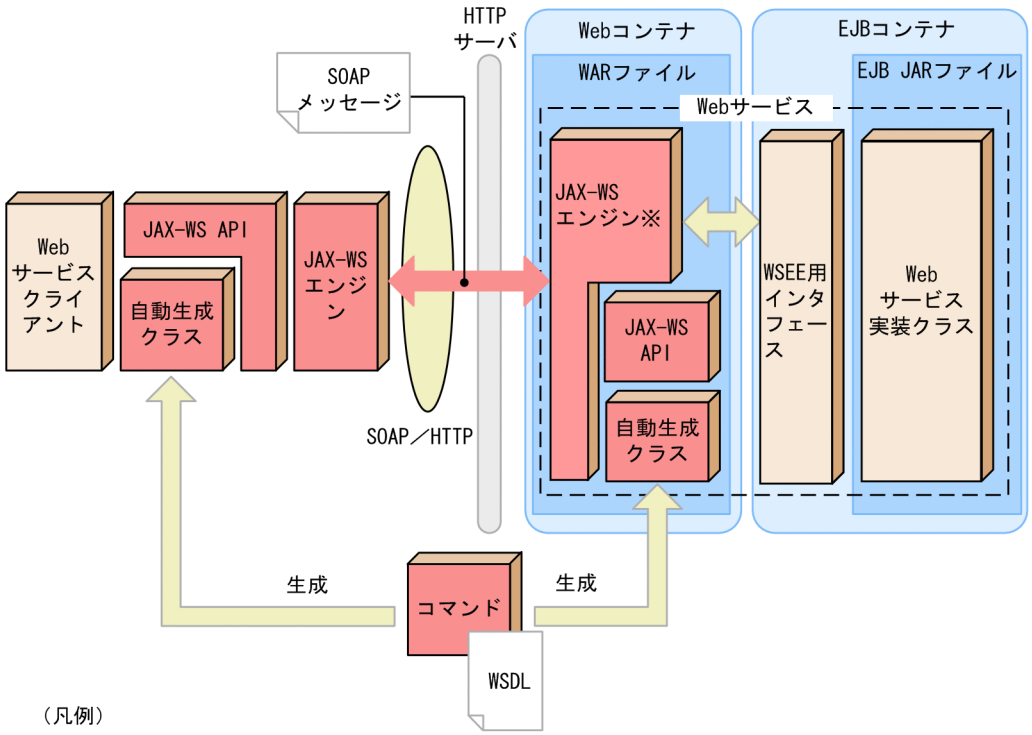
次の図に示す SOAP Web サービスの構成を基に，JAX-WS エンジンの機能について説明します。

図 1-1　SOAP Web サービスの構成（POJO の Web サービス）



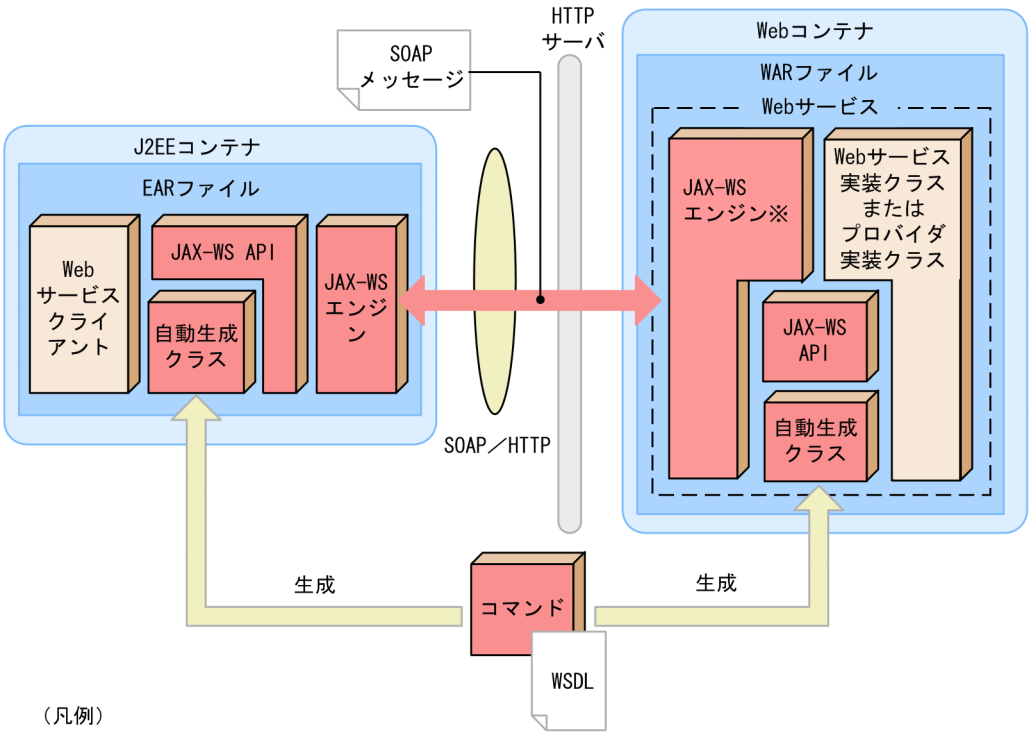
注※
Webサービス側のJAX-WSエンジンおよびJAX-WS APIは，ライブラリとして呼び出されるだけで，WARファイルには含まれません。

図 1-2　SOAP Web サービスの構成（EJB の Web サービス）



注※
Webサービス側のJAX-WSエンジンおよびJAX-WS APIは，ライブラリとして呼び出されるだけで，WARファイルには含まれません。

図 1-3　SOAP Web サービスの構成（J2EE コンテナ上で動作する Web サービスクライアント）



注※
JAX-WSエンジンおよびJAX-WS APIは，ライブラリとして呼び出されるだけで，
WARファイルおよびEARファイルには含まれません。

- JAX-WS エンジン

  SOAP Web サービスの通信基盤となるエンジンです。Web サービス側および Web サービスクライアント側に配置され，送受信された SOAP メッセージのマーシャル／アンマーシャルをする役割を果たします。

  Web サービスクライアント側の JAX-WS エンジン

  Web サービスクライアントから JAX-WS API を介して Java オブジェクトを受け取り，SOAP 要求メッセージを生成（マーシャル）します。生成した SOAP 要求メッセージは呼び出し先の Web サービスに送信します。

  また，Web サービスから SOAP 応答メッセージを受け取り，Java オブジェクトを生成（アンマーシャル）します。生成した Java オブジェクトは Web サービスクライアントに返します。

  Web サービス側の JAX-WS エンジン

  Web サービスクライアント側から SOAP 要求メッセージを受け取り，Java オブジェクトを生成（アンマーシャル）します。このとき，対象となる Web サービス実装クラスまたはプロバイダ実装クラスを見つけ出し（ディスカバリ），オペレーションに対応するメソッドを呼び出します（ディスパッチ）。また，対象となる Web サービス実装クラスまたはプロバイダ実装クラスから Java オブジェクトを受け取り，SOAP 応答メッセージを生成（マーシャル）します。生成した SOAP 応答メッセージは，呼び出し元である Web サービスクライアントに返します。

- コマンド

SOAP Web サービスの開発で使用するコマンドです。Web サービスおよび Web サービスクライアントの実装に必要な Java ソースや WSDL を生成できます。JAX-WS エンジンを利用する Web サービスの開発では次のコマンドを使用します。

- cjwsimport コマンド
- hwsgen コマンド

コマンドの使用方法については，「14.　コマンド」を参照してください。

- **自動生成クラス**

  コマンドを実行して生成される Java ソースです。生成された Java ソースを利用して，Web サービスおよび Web サービスクライアントを実装します。

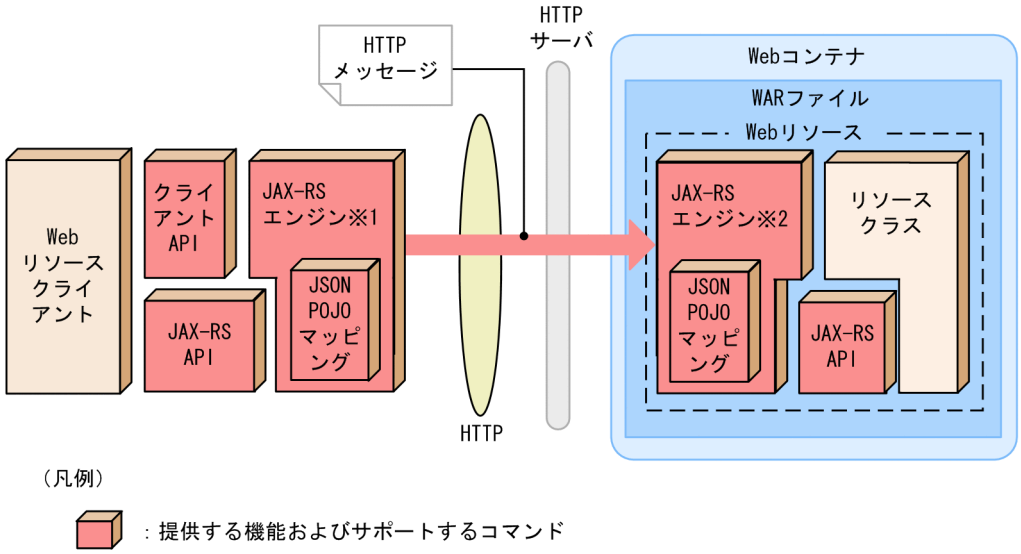
- **JAX-WS API**

  JAX-WS 2.2 仕様の API です。プロバイダを起点とした Web サービスの開発や，ディスパッチベースの Web サービスクライアントを開発する場合に使用します。また，ハンドラフレームワークやアドレッシングなどの機能を追加する場合にも使用できます。

1.3.2　RESTful Web サービスの機能

次の図に示す RESTful Web サービス（Web リソース）の構成を基に，JAX-RS エンジンの機能について説明します。

図 1-4　RESTful Web サービスの構成



注※1
　クライアント側のJAX-RSエンジンはクライアントAPIを経由して呼び出されるライブラリです。
注※2
　サーバ側のJAX-RSエンジンおよびJAX-RS APIは，ライブラリとして呼び出されるだけで，
　WARファイルには含まれません。

- **JAX-RS エンジン**

  RESTful Web サービス（Web リソース）の通信基盤となるエンジンです。

- Web リソースクライアント側の JAX-RS エンジン

  Web リソースクライアントから RESTful Web サービス用クライアント API を介して Java オブジェクトを受け取り，HTTP リクエストを生成します。生成した HTTP リクエストは呼び出し先の Web リソースに送信します。

  また，Web リソースから HTTP レスポンスを受け取り，Java オブジェクトを生成します。生成した Java オブジェクトは Web リソースクライアントに返します。

- Web リソース側の JAX-RS エンジン

  クライアントから HTTP リクエストを受け取り，対象となるリソースクラスを見つけ出し（ディスカバリ），要求に対応するメソッドを呼び出します（ディスパッチ）。また，対象となるリソースクラスから HTTP レスポンスを生成し，呼び出し元に返します。JAX-RS エンジンはディスパッチの際，リソースクラスに含まれるアノテーションに基づき，必要なインジェクションを行います。

- **RESTful Web サービス用クライアント API**

  RESTful Web サービス(Web リソース) を呼び出すクライアントで使用できる API です。

1.4　Web サービスの開発および実行の前提条件

Application Server 上で Web サービスを開発する場合の前提条件，およびサポート範囲について説明します。

1.4.1　前提となる構成ソフトウェア

Web サービスは Developer を利用して開発し，Application Server を利用して実行します。

Developer の前提ソフトウェアについては，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「1.3　開発環境のマシン構成」を参照してください。

Application Server の前提ソフトウェアについては，マニュアル「アプリケーションサーバ & BPM/ESB 基盤 概説」の「2.2　構成ソフトウェア」を参照してください。

1.4.2　機能および仕様に関する前提条件

(1)　SOAP Web サービスの機能および仕様に関する前提条件

Application Server 上で SOAP Web サービスを開発するときに利用できる機能および仕様について説明します。標準仕様のサポート範囲については，次に示す個所を参照してください。

- 「19.　JAX-WS 仕様のサポート範囲」
- 「20.　WSDL 仕様のサポート範囲」
- 「22.　SAAJ 仕様のサポート範囲」
- 「23.　WS-RM 仕様のサポート範囲」

(a)　デフォルトマッピング

Application Server の JAX-WS 機能で提供しているコマンドは，JAX-WS 2.2 仕様の 2 章で規定された WSDL から Java へのデフォルトのマッピング，JAX-WS 2.2 仕様の 3 章で規定された Java から WSDL へのデフォルトのマッピングに従って動作します。

また，Web サービス側の JAX-WS エンジンは，Web サービスのメタデータである WSDL を要求された場合，WAR ファイルまたは EJB JAR ファイルに WSDL がなければ，JAX-WS 2.2 仕様の 3 章で規定された Java から WSDL へのデフォルトのマッピングに従って，WSDL を生成します。

WSDL から Java へのデフォルトのマッピングについては，「15.1　WSDL から Java へのデフォルトマッピング」を参照してください。Java から WSDL へのデフォルトのマッピングについては，「16.1　Java から WSDL へのデフォルトマッピング」を参照してください。

(b) マッピングのカスタマイズ

Application Server の JAX-WS 機能で提供しているコマンドは，JAX-WS 2.2 仕様の 7 章で規定された WSDL から Java へのマッピングのカスタマイズ（バインディング宣言），および JAX-WS 2.2 仕様の 8 章で規定された Java から WSDL へのマッピングのカスタマイズ（アノテーション）に従って動作します。

また，Web サービス側の JAX-WS エンジンは，Web サービスのメタデータである WSDL を要求された場合，WAR ファイルまたは EJB JAR ファイルに WSDL がなければ，JAX-WS 2.2 仕様の 7 章で規定された Java から WSDL へのマッピングのカスタマイズに従って，WSDL を生成します。

WAR ファイルについては「3.5.1　WAR ファイルの構成」を参照してください。EJB JAR ファイルについては「3.5.2　EJB JAR ファイルの構成」を参照してください。WSDL から Java へのマッピングのカスタマイズについては，「15.2　WSDL から Java へのマッピングのカスタマイズ」を参照してください。Java から WSDL へのデフォルトのマッピングについては，「16.2　Java から WSDL へのマッピングのカスタマイズ」を参照してください。

(c) Java と WSDL 間のバインディング

Application Server の JAX-WS エンジンは，Web サービスおよび Web サービスクライアントともに，JAX-WS 2.2 仕様の 2 章および 3 章に従って，Java と WSDL 間をバインディングします。

Application Server の JAX-WS エンジンのサポート範囲については，「10.2　JAX-WS エンジンの動作」を参照してください。

(d) WSDL 仕様

Application Server の JAX-WS 機能では，WSDL 1.1 仕様の WSDL をサポートしています。WSDL 定義のスタイルとしては，document/literal スタイルだけサポートしています。document/literal スタイルであれば，wrapper スタイルおよび non-wrapper スタイルのどちらも利用できます。

WSDL 1.1 仕様のサポート範囲については，「20.1　WSDL 1.1 仕様のサポート範囲」を参照してください。

(e) SOAP 仕様

Application Server の JAX-WS 機能では，SOAP 1.1 仕様および SOAP 1.2 仕様の SOAP メッセージをサポートしています。

(f) Message Exchange Pattern（MEP）

Application Server の JAX-WS 機能では，MEP として request-response オペレーション，および one-way オペレーションをサポートしています。request-response オペレーション，および one-way オペレーションの定義方法については，次に示す個所を参照してください。

- 「15.1.4　メッセージのパートからパラメタおよび戻り値へのマッピング（wrapper スタイルの場合）」
- 「15.1.5　メッセージのパートからパラメタおよび戻り値へのマッピング（non-wrapper スタイルの場合）」

- 「16.2.4　javax.jws.Oneway アノテーション」

また，one-way オペレーションおよび注意事項については，「10.22　one-way オペレーション」を参照してください。

(g)　非同期関連の機能

Application Server の JAX-WS 機能では，JAX-WS 2.2 仕様の 2.3.4 項や，4 章などに記載されている Web サービスクライアントでの非同期呼び出しを実現する機能はサポートしていません。

(h)　Dispatch／Provider インタフェース関連の機能

Application Server の JAX-WS 機能では，JAX-WS 2.2 仕様の 4 章に記載されている Dispatch インタフェース，JAX-WS 2.2 仕様の 5 章に記載されている Provider インタフェース，および Dispatch インタフェースと Provider インタフェースに関連する機能をサポートしています。ただし，JAX-WS 2.2 仕様の 4 章に記載されているオブジェクトのうち，次のオブジェクトはサポートしていません。

- javax.activation.DataSource
- javax.xml.transform.stax.StAXSource

(i)　Endpoint クラスおよび発行関連の機能

Application Server の JAX-WS 機能では，JAX-WS 2.2 仕様の 5 章に記載されている Web サービスのエンドポイントを動的に生成し，発行するための機能はサポートしていません。

(j)　ハンドラ関連の機能

Application Server の JAX-WS 機能では，Web サービスクライアントの実装で，API による動的なハンドラ設定をサポートしています。また，Web サービスの実装で，アノテーションによる動的なハンドラ設定をサポートしています。JSR-109 仕様を前提とする静的な設定はサポートしていません。

(k)　添付ファイルの使用

Application Server の JAX-WS 機能では，SAAJ 1.3 仕様，および WSDL に wsi:swaRef 型を記述する形式の添付ファイルおよび MTOM/XOP 仕様形式の添付ファイルをサポートしています。WSDL 1.1 仕様の MIME 拡張要素を使用する記述（MIME バインディング）はサポートしていません。wsi:swaRef 形式の添付ファイルの使用方法については，「28.　添付ファイル機能（wsi:swaRef 形式）」を参照してください。MTOM/XOP 仕様形式の添付ファイルの使用方法については，「30.　添付ファイル機能（MTOM/XOP）」を参照してください。

(l)　メッセージコンテキスト

Application Server の JAX-WS 機能では，JAX-WS 2.2 仕様の 9 章に記載されている標準のメッセージコンテキストプロパティは，読み取りだけサポートしています。Web サービスクライアントの実装でタイムアウトを設定するためのプロパティをサポートしています。メッセージコンテキストの使用方法および注意事項については，「19.2.5　メッセージコンテキストの使用」を参照してください。

(m)　API

Application Server の JAX-WS 機能では，JAX-WS 2.2 仕様の API をサポートしています。JAX-WS API のサポート範囲については，「19.2　API のサポート範囲」を参照してください。

(n)　XML／HTTP バインディング

Application Server の JAX-WS 機能では，JAX-WS 2.2 仕様の 11 章に記載されている XML／HTTP バインディングはサポートしていません。

(o)　ラッパ bean の動的生成

Application Server の JAX-WS 機能では，SEI を起点として開発した Web サービスで，JAX-WS2.2 仕様の 3.6.2.1 項および 3.7 節に記載されているラッパ bean（リクエスト bean とレスポンス bean），およびフォルト bean の JavaBeans クラスを JAX-WS エンジンが動的に生成する機能（以降，ラッパ bean の動的生成機能と呼ぶ場合があります）をサポートしています。Web サービスクライアントや，WSDL を起点とする Web サービスでは，ラッパ bean の動的生成機能はサポートしていません。

ラッパ bean の動的生成機能については，「10.23　ラッパ bean の動的生成機能」を参照してください。

(p)　関連する標準仕様

関連する標準仕様について説明します。

- MTOM 仕様に関連する機能

  Application Server の JAX-WS 機能では，JAX-WS 2.2 仕様の 2.4 節や 6.5 節などに記載された MTOM 仕様に関連する機能をサポートしています。

- WS-Addressing 仕様

  Application Server の JAX-WS 機能では，JAX-WS 2.2 仕様の 5.2.8 項などに記載された WS-Addressing 仕様に関連する機能をサポートしています。詳細については，「37.　アドレッシング機能」を参照してください。

- XML Catalogs 1.1 仕様

  Application Server の JAX-WS 機能では，JAX-WS 2.2 仕様の 4.4 節などに記載された XML Catalogs 1.1 仕様に関連する機能をサポートしています。

  XML Catalogs 1.1 仕様のサポート範囲については，「21.　XML Catalogs 1.1 のサポート範囲」を参照してください。

- WSEE（JSR-109）仕様

  Application Server の JAX-WS 機能では，WSEE（JSR-109）仕様をサポートしています。WSEE（JSR-109）仕様の項目とサポートの対応を次の表に示します。

表 1-2　WSEE（JSR-109）仕様の項目とサポートの対応

項番	項目		サポート
1	ステートレスセッションビーンおよびシングルトンセッションビーンの次の方法。 • WSDL へのマッピングまたはバインディング方法。 • SOAP へのマッピングまたはバインディング方法。	ステートレス	○
		シングルトン	－
2	クライアントモデル。特に，JNDI を使用してサービスインタフェースを発見する方法と，JAX-WS 仕様の WebServiceRef アノテーションを使用する方法。	4.2.2 項の内容（サービスクラスおよびポートのインジェクション）	○
		4.2.2 項以外の内容	－
3	デプロイメントモデル。EAR ファイルへのパッケージング方法とライフサイクル。	5.4 節記載の内容	○
		5.4 節以外の内容	－
4	デプロイメントディスクリプタ。webservices.xml の構文と記述する必要がある内容，および JAX-WS 仕様（Web Services Metadata （JSR-181）仕様）で定義されるアノテーションとのマッピング。		－
5	ロールなど既存の Java EE コンテナの機能とのマッピング。		－

（凡例）

○：サポートしていることを示します。

－：サポートしていないことを示します。

なお，web.xml を省略した場合の動作はサポートしています。web.xml を省略した場合の動作については，「3.4.3　web.xml を WAR ファイルに含めない場合の動作」を参照してください。

- SAAJ 1.3 仕様

  Application Server の JAX-WS 機能では，SAAJ 1.3 仕様をサポートしています。SAAJ 1.3 仕様の API については，「22.1　SAAJ 1.3 仕様のサポート範囲」を参照してください。

(2)　RESTful Web サービスの機能および仕様に関する前提条件

Application Server 上で RESTful Web サービス（Web リソース）を開発するときに利用できる機能および仕様について説明します。標準仕様のサポート範囲については，「24.　JAX-RS 仕様のサポート範囲」を参照してください。

(a)　リソースクラス

Web リソースを実装するクラスで，JAX-RS 1.1 仕様の 3 章で定義されています。ルートリソースクラスとサブリソースクラスの 2 種類があります。Application Server の JAX-RS 機能はどちらのリソースクラスもサポートしています。

ルートリソースクラスについては，「17.1.1　ルートリソースクラス」を参照してください。サブリソースクラスについては，「17.1.7　サブリソースクラス」を参照してください。

(b) プロバイダ

JAX-RS エンジンを拡張する機能で，JAX-RS 1.1 仕様の 4 章で定義されています。エンティティプロバイダ，コンテキストプロバイダ，例外マッピングプロバイダの 3 種類があります。

エンティティプロバイダは，HTTP エンティティボディと Java 型をマッピングするプロバイダです。JAX-RS 機能は，ビルトインのエンティティプロバイダをサポートしています。ビルトインのエンティティプロバイダがサポートする MIME タイプと Java 型については，「17.1.1(4)(c)　エンティティパラメタ」を参照してください。

コンテキストプロバイダは，リソースやほかのプロバイダにコンテキストを提供するプロバイダです。JAX-RS 機能は，ビルトインのコンテキストプロバイダをサポートしています。ビルトインのコンテキストプロバイダがサポートするコンテキストについては，「24.4　コンテキスト」を参照してください。

例外マッピングプロバイダは，Web リソースの例外の HTTP レスポンスへのマッピングをカスタマイズするプロバイダです。例外マッピングプロバイダについては，「17.2.2　例外マッピングプロバイダ」を参照してください。

(c) *Application*

Application はリソースクラスとプロバイダのファクトリで，JAX-RS 1.1 仕様の 2 章で定義されています。JAX-RS 1.1 仕様はビルトインの *Application* をサポートしています。ビルトインの *Application* については，「11.3.1　WAR ファイルの構成」を参照してください。

1.5　Web サービスとクライアントの形態

ここでは，Application Server がサポートする Web サービスとクライアントの形態について説明します。

1.5.1　Web サービスの形態

Application Server では，次に示す形態の Web サービスをサポートしています。

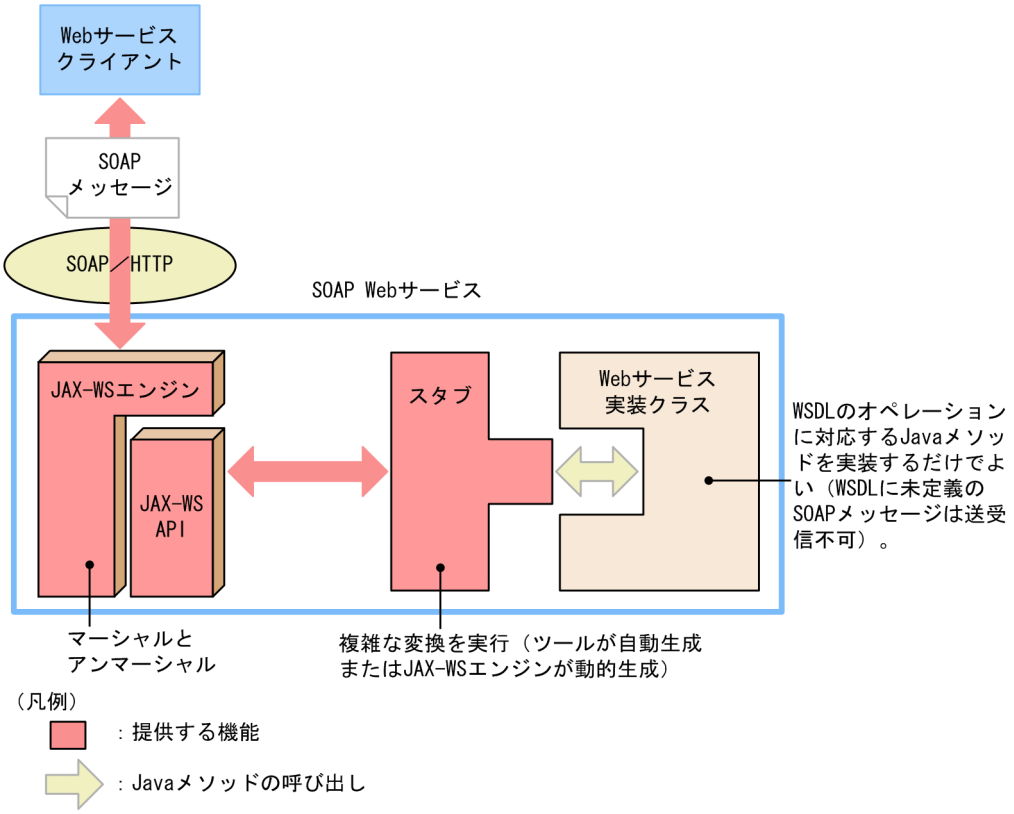
- Web サービス実装クラスを使用した SOAP Web サービス
- プロバイダ実装クラスを使用した SOAP Web サービス
- RESTful Web サービス（Web リソース）

それぞれの形態について説明します。

(1)　Web サービス実装クラスを使用した SOAP Web サービス

Web サービス実装クラスを使用する場合の形態を次に示します。

図 1-5　Web サービス実装クラスを使用した SOAP Web サービスの形態



Web サービス実装クラスを使用すると，WSDL のオペレーションに対応する Java メソッドを実装するだけで Web サービスを実現できます。複雑な変換はスタブが実施するため，Web サービス開発時に，

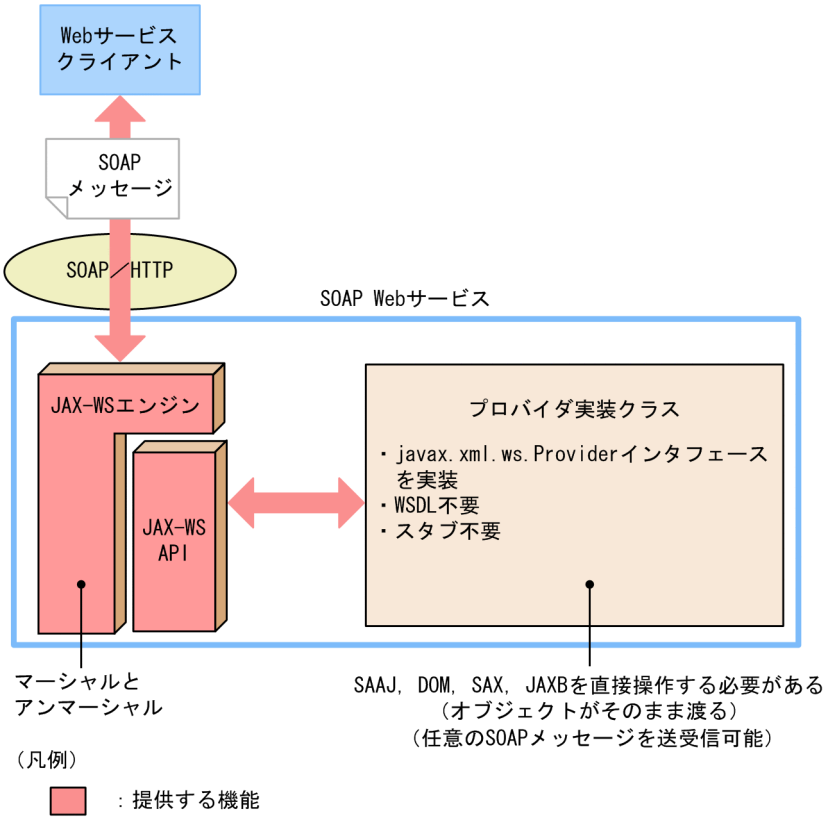
API を利用して SOAP メッセージを構成する XML を組み立てるような，複雑なプログラミングをする必要はありません。

スタブは，ツールが自動生成するか，または JAX-WS エンジンが動的に生成します。WSDL は自動生成でき，必須となります。また，WSDL に含まれない SOAP メッセージを送受信することはできないので注意してください。なお，Web サービス実装クラスは，POJO としても，EJB を基にしても作成できます。

(2) プロバイダ実装クラスを使用した SOAP Web サービス

プロバイダ実装クラスを使用する場合の形態を次に示します。

図 1-6 プロバイダ実装クラスを使用した SOAP Web サービスの形態

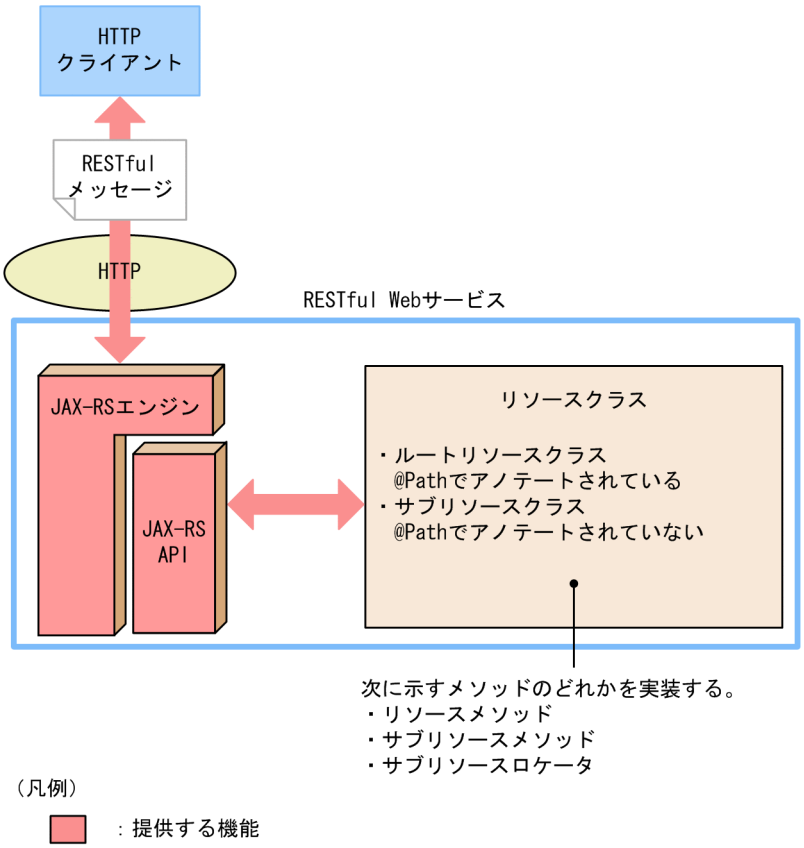


プロバイダ実装クラスを使用する場合はスタブも WSDL も必要ないため，任意の SOAP メッセージを動的に送受信したいときに適しています。プロバイダ実装クラスは JAX-WS エンジンがマーシャルしたオブジェクトをそのまま受け取るため，Web サービス開発時に，API を利用して SOAP メッセージを構成する XML から値を直接取得したり，API を利用して SOAP メッセージを構成する XML を組み立てたりする必要があります。

(3) RESTful Web サービス（Web リソース）

RESTful Web サービス（Web リソース）はリソースクラスを実装することで実現します。

図 1-7　RESTful Web サービス（Web リソース）



リソースクラスには，ルートリソースクラスとサブリソースクラスがあります。

ルートリソースクラスは，クラスレベルで Path アノテーションでアノテートされた Java クラスです。サブリソースクラスは，クラスレベルで Path アノテーションでアノテートされていない Java クラスです。

リソースクラスには次に示すメソッドのどれかを実装します。

- リソースメソッド
- サブリソースメソッド
- サブリソースロケータ

1.5.2　クライアントの形態

Application Server では，SOAP Web サービスを呼び出すクライアントとして，次に示す形態をサポートしています。

- スタブベースの Web サービスクライアント
- ディスパッチベースの Web サービスクライアント

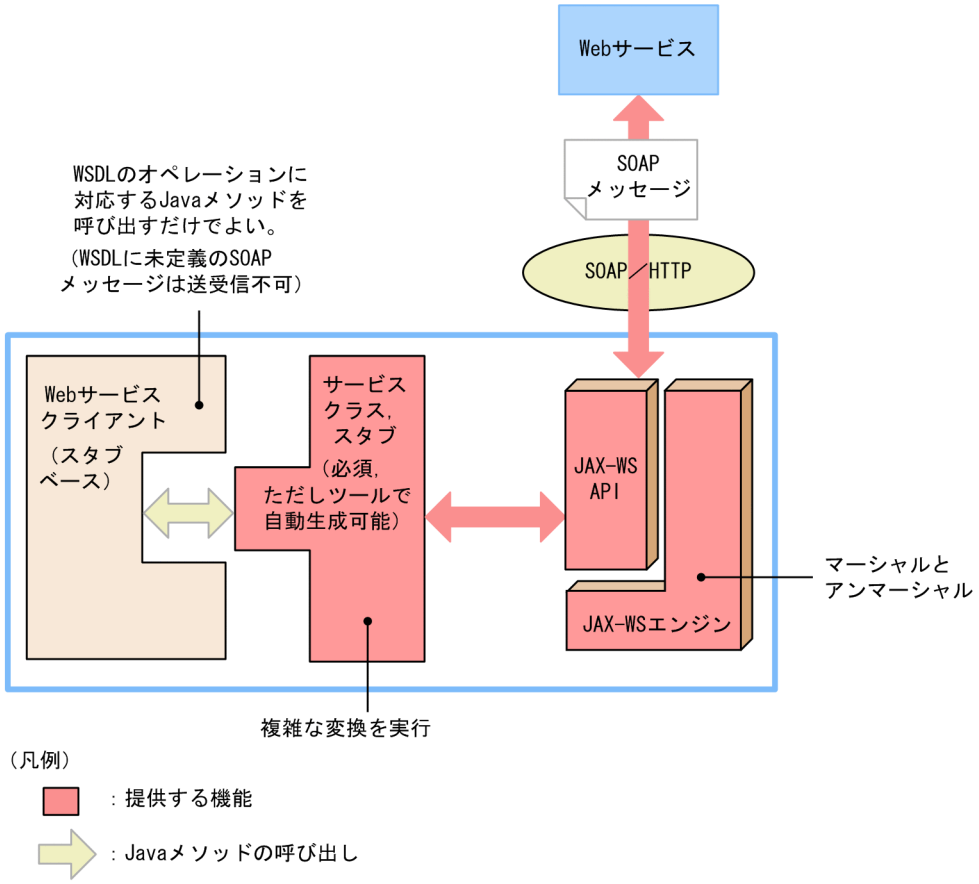
それぞれの形態について説明します。

なお，Web リソースクライアントは，RESTful Web サービス用クライアント API か，または java.net.URL や java.net.HttpURLConnection などの標準的な Java API を利用して実装してください。

(1)　スタブベースの Web サービスクライアント

スタブベースの Web サービスクライアントを使用する場合の形態を次に示します。

図 1-8　Web サービスクライアント（スタブベース）

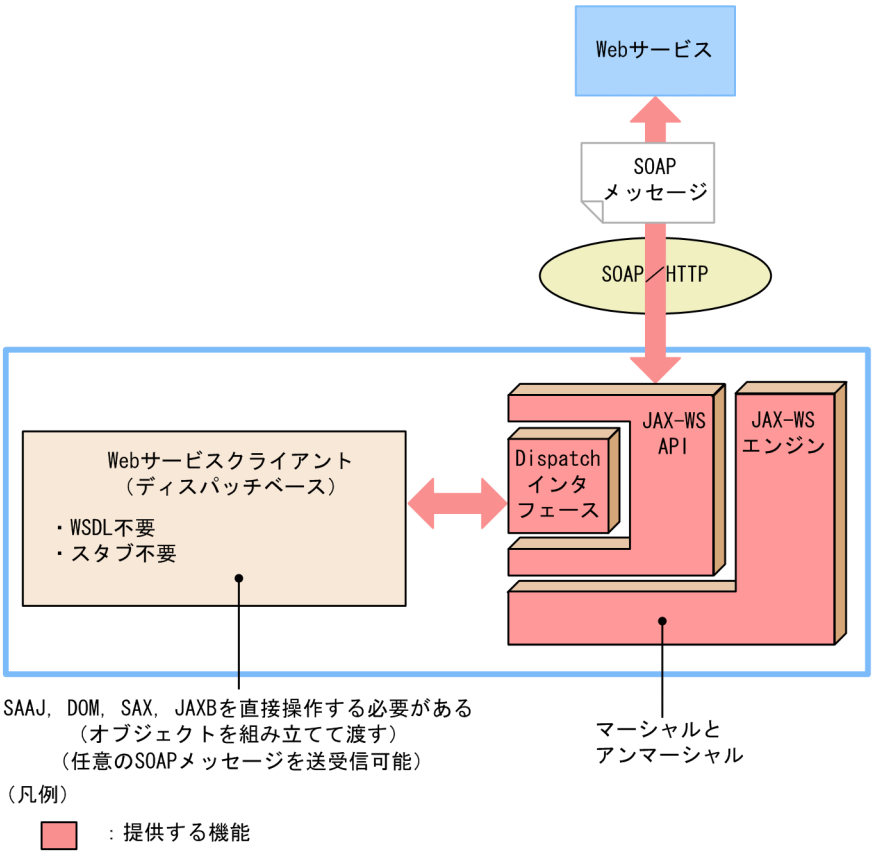


スタブベースの Web サービスクライアントを使用すると，WSDL のオペレーションに対応する Java メソッドを呼び出すだけで，Web サービスを呼び出せます。複雑な変換は自動生成されるスタブが実施するため，Web サービスクライアント開発時に，API を利用して SOAP メッセージを構成する XML を組み立てるような，複雑なプログラミングをする必要はありません。ただし，WSDL に定義されていない SOAP メッセージを送受信することはできないので注意してください。また，スタブおよび WSDL は必須です。WSDL は，対象の Web サービスから入手する必要があります。

(2)　ディスパッチベースの Web サービスクライアント

ディスパッチベースの Web サービスクライアントを使用する場合の形態を次に示します。

図 1-9　Web サービスクライアント（ディスパッチベース）



ディスパッチベースの Web サービスクライアントを使用する場合は，スタブも WSDL も必要ないため，任意の SOAP メッセージを動的に送受信したいときに適しています。ディスパッチベースの Web サービスクライアントは，生成したオブジェクトを javax.xml.ws.Dispatch インタフェース経由で JAX-WS エンジンに受け渡すため，Web サービスクライアント開発時に，API を利用して SOAP メッセージを構成する XML から値を直接取得したり，API を利用して SOAP メッセージを構成する XML を組み立てたりする必要があります。

注意事項

ディスパッチベースの Web サービスクライアントでは，javax.xml.ws.Dispatch インタフェースの代わりに javax.xml.soap.SOAPConnection クラスを使用して SOAP メッセージを送受信することもできます。ただし，javax.xml.soap.SOAPConnection クラスを使用した場合，動作定義ファイルおよびメッセージコンテキストの設定は無効になり，ログも出力されないため，注意してください。

また，このマニュアルでは，javax.xml.soap.SOAPConnection クラスを使用する場合の動作については説明しません。ほかの SAAJ 1.3 仕様の API と同様に，JDK のドキュメントを参照してください。なお，特に問題のないかぎり，javax.xml.soap.SOAPConnection クラスではなく javax.xml.ws.Dispatch インタフェースを使用してください。

1.6　JAX-WS エンジンと JAX-RS エンジンの設定

JAX-WS エンジンと JAX-RS エンジンを利用するには，J2EE サーバ用オプション定義ファイルに JAX-WS エンジンと JAX-RS エンジンを有効にするための定義をする必要があります。

JAX-WS エンジンと JAX-RS エンジンを有効にするための定義については，「付録 A.1(3)　動作環境の切り替え」の JAX-WS エンジンと JAX-RS エンジンを利用する場合の説明を参照してください。

2 開発の流れ

JAX-WS 2.2 仕様に従った SOAP Web サービスは，WSDL を起点とした開発，SEI を起点とした開発，またはプロバイダを起点とした開発のどれかで開発します。SOAP Web サービスを呼び出すクライアントは，スタブベース，またはディスパッチベースのどちらかで開発します。

RESTful Web サービス（Web リソース）は，リソースクラスを実装して開発します。RESTful Web サービスを呼び出すクライアントは，RESTful Web リソース用クライアント API，または標準的な Java API を使用して開発します。

この章では，Web サービスおよびクライアントの開発の流れについて説明します。

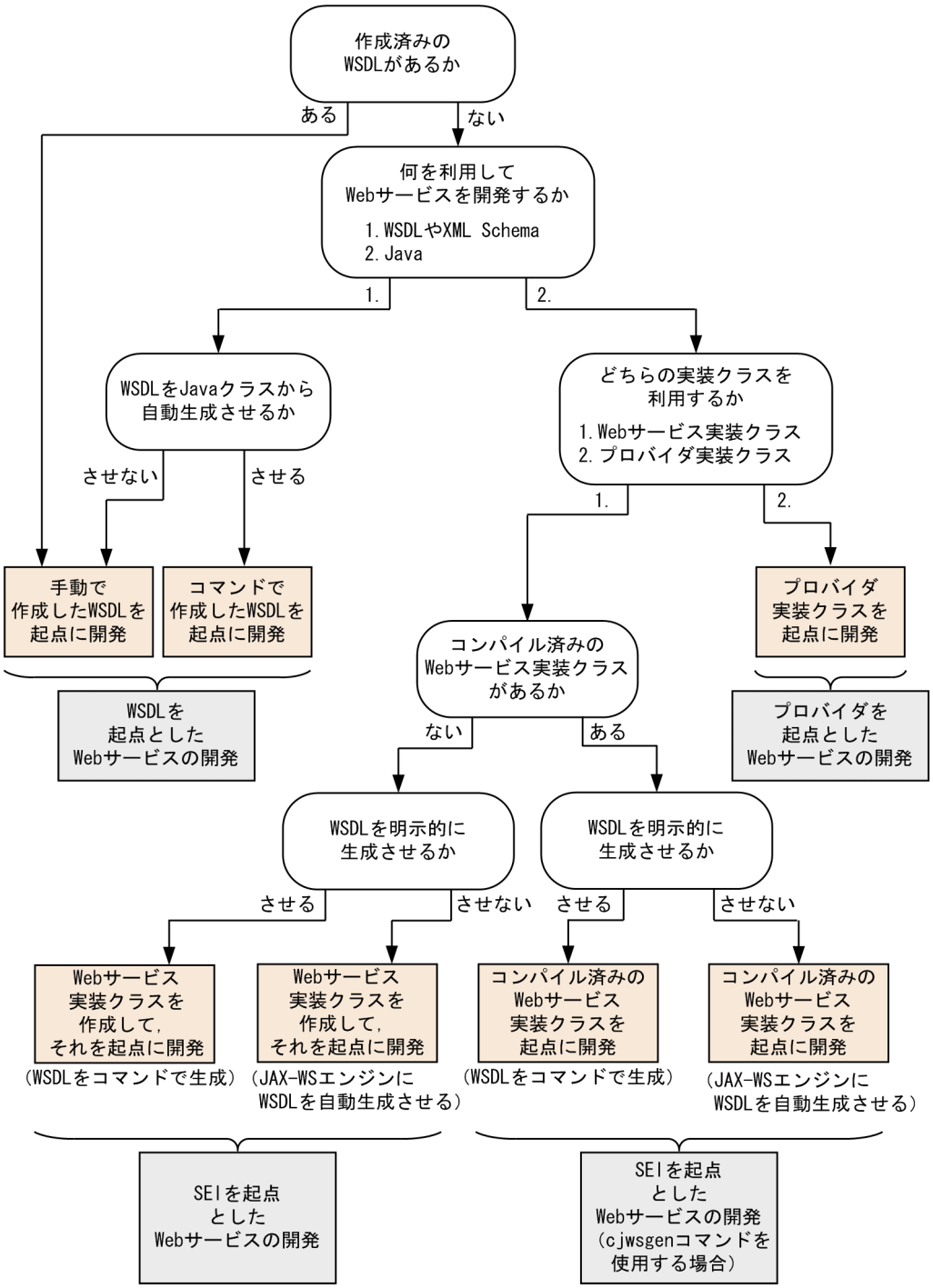
2.1 SOAP Web サービス開発の流れ

SOAP Web サービスには，次に示す開発方法があります。

- WSDL を起点とした開発（2.1.1）
- SEI を起点とした開発（2.1.2）
- SEI を起点とした開発（hwsgen コマンドを使用する場合）（2.1.3）
- プロバイダを起点とした開発（2.1.4）

開発方法を選択する流れを次に示します。

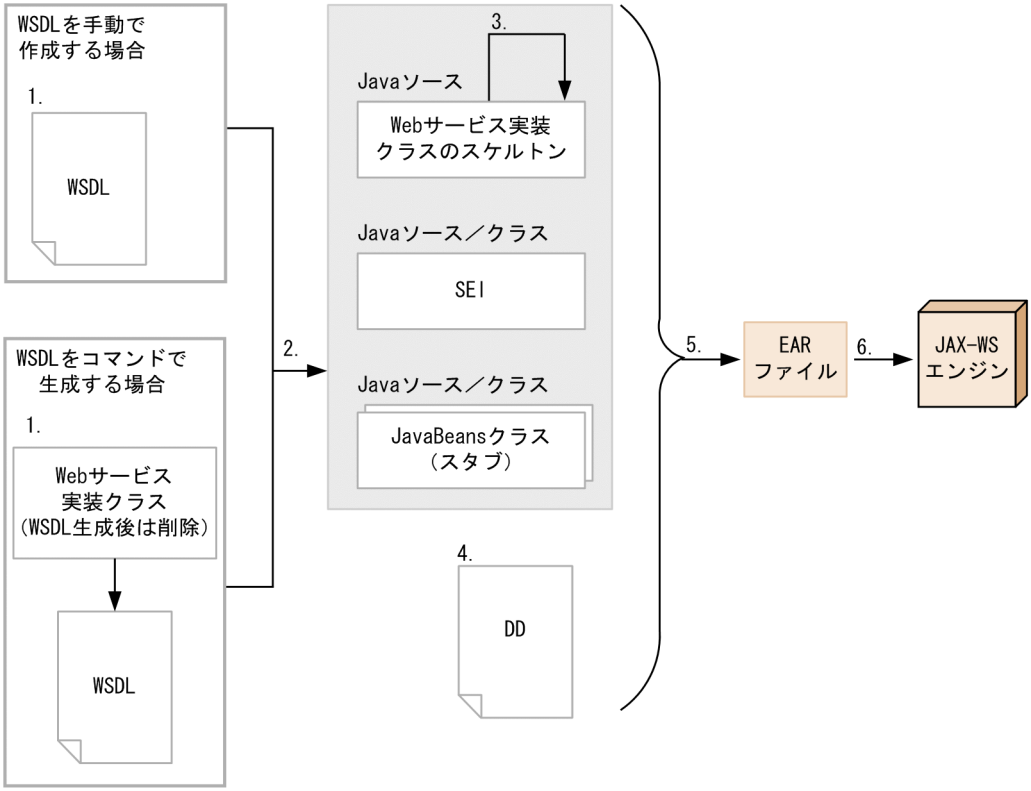
図 2-1　SOAP Web サービス開発方法の選択の流れ



2.1.1　WSDL を起点とした開発の流れ

WSDL を起点とした Web サービスの開発の流れを次の図に示します。

図 2-2　WSDL を起点とした Web サービスの開発の流れ



1. WSDL ファイルを作成する

WSDL ファイルは，手動またはコマンドで作成できます。

- 手動で作成する場合

  Web サービスのメタ情報として，WSDL 1.1 仕様，XML Schema 仕様，および WS-I Basic Profile 1.1 に従って WSDL ファイルを作成します。なお，SOAP 1.1 のメッセージを受信する場合は SOAP 1.1 仕様の拡張要素を記述し，SOAP 1.2 のメッセージを受信する場合は SOAP 1.2 仕様の拡張要素を記述してください。

  WSDL 1.1 仕様のサポート範囲については，「20.1　WSDL 1.1 仕様のサポート範囲」を参照してください。

- コマンドで生成する場合

  コマンドでの WSDL の生成は，ユーザが WSDL や XML Schema の構文ではなく，Java 言語での開発に慣れている場合にお勧めします。

  一時的に Web サービス実装クラスを作成・コンパイルしたあと，-wsdl オプションを指定して hwsgen コマンドの WSDL 生成機能を実行し，WSDL ファイルを生成します。生成した WSDL は，必要に応じて変更してください。ここで作成する Web サービス実装クラスは，hwsgen コマンドの入力だけに利用します。そのため，メソッドを実装する必要はありません。

  SOAP 1.2 仕様のメッセージを受信する場合は，Web サービス実装クラスの作成時，javax.xml.ws.BindingType アノテーションに"http://www.w3.org/2003/05/soap/bindings/HTTP/"を指定してください。

  WSDL 生成後，不要になった Web サービス実装クラスは削除してください。

2. cjwsimport コマンドを実行する（Java ソースの生成）

cjwsimport コマンドを実行して，作成した WSDL ファイルから，SEI，Web サービス実装クラスのスケルトン，JavaBeans クラス（スタブ）など，Web サービスの開発・実行に必要な Java ソースを生成します。cjwsimport コマンドは，-generateService オプションを指定して実行します。
cjwsimport コマンドについては，「14.1.1　cjwsimport コマンド」を参照してください。

3. Web サービスを実装する

手順 2 で生成されたスタブを利用して，Web サービス実装クラスのスケルトンに必要な処理を記述し，Web サービスを実装します。また，実装した Web サービス実装クラスをコンパイルします。なお，javax.xml.ws.BindingType アノテーションは WSDL の内容に応じて自動的に付加されます。

4. DD を作成する

web.xml および application.xml を作成します。web.xml には，Web サービス固有の情報を記述します。web.xml の作成については，「3.4　web.xml の作成」を参照してください。

5. EAR ファイルを作成する

作成したファイルを含む EAR ファイルを作成します。EAR ファイルの作成については，「3.5.3　EAR ファイルの作成」を参照してください。

6. EAR ファイルをデプロイし，開始する

作成した EAR ファイルをデプロイし，J2EE アプリケーション（Web サービス）として開始します。J2EE アプリケーションのインポートおよび開始コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート）」および「cjstartapp（J2EE アプリケーションの開始）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションをデプロイ（インポート）する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.3　J2EE アプリケーションのインポート」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションを開始する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.1　J2EE アプリケーションの開始」を参照してください。

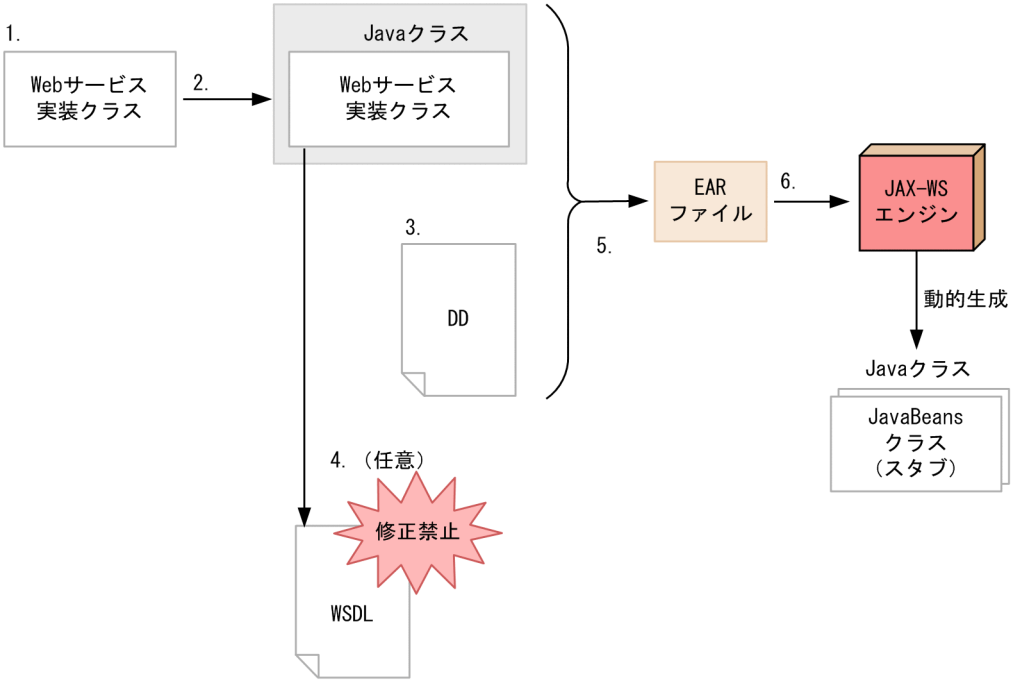
WSDL を起点とした Web サービスの開発例については，次に示す個所を参照してください。

- 「4.3　Web サービスの開発例（WSDL 起点）」
- 「35.3　Web サービスの開発例（WSDL 起点・WS-RM 1.2）」

2.1.2　SEI を起点とした開発の流れ

SEI を起点とした Web サービスの開発の流れを次の図に示します。

図 2-3　SEI を起点とした Web サービスの開発の流れ



WSDL は JAX-WS エンジンによって自動生成されますが，手順 4 に従いコマンドを実行することで WSDL を生成することもできます。WSDL をコマンドで生成しない場合，手順 4 は不要です。

1. Web サービス実装クラスを作成する

Web サービス実装クラスは，POJO として作成する場合と EJB を基に作成する場合があります。どちらの場合も，JAX-WS 2.2 仕様および JAXB 2.2 仕様に従って，Web サービス実装クラスを作成します。SOAP 1.2 仕様のメッセージを受信する場合は，javax.xml.ws.BindingType アノテーションに"http://www.w3.org/2003/05/soap/bindings/HTTP/"を指定してください。

JAX-WS 2.2 仕様のサポート範囲については，「19.1　JAX-WS 2.2 仕様のサポート範囲」を参照してください。JAXB 2.2 仕様のサポート範囲については，マニュアル「XML Processor ユーザーズガイド」の「付録 B　JAXB 仕様のサポート範囲」を参照してください。

2. Web サービス実装クラスをコンパイルする

javac コマンドを実行して，作成した Web サービス実装クラスをコンパイルします。javac コマンドについては，JDK のドキュメントを参照してください。

3. DD を作成する

web.xml および application.xml を作成します。web.xml には，Web サービス固有の情報を記述します。web.xml の作成については，「3.4　web.xml の作成」を参照してください。

4. hwsgen コマンドの WSDL 生成機能を実行する（任意）

-wsdl オプションを指定して hwsgen コマンドを実行し，Web サービス実装クラスから WSDL を生成，およびアノテーションなどのエラーチェックをします。手順 6.の JavaBeans クラス（スタブ）の動的生成時にエラーが発生しないようにするには，コンパイルした Web サービス実装クラスに対して

hwsgen コマンドを実行してください。この手順は任意です。なお，ここで生成した WSDL は，変更しないでください。

hwsgen コマンドの WSDL 生成機能で生成した WSDL は，メールで送付するなど，メタデータの発行機能以外の任意の方法で展開できます。

hwsgen コマンドについては，「14.1.2　hwsgen コマンド」を参照してください。

5. EAR ファイルを作成する

作成したファイルを含む EAR ファイルを作成します。EAR ファイルの作成については，「3.5.3　EAR ファイルの作成」を参照してください。

6. EAR ファイルをデプロイし，開始する

作成した EAR ファイルをデプロイし，J2EE アプリケーション（Web サービス）として開始します。J2EE アプリケーションのインポートおよび開始コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート）」および「cjstartapp（J2EE アプリケーションの開始）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションをデプロイ（インポート）する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.3　J2EE アプリケーションのインポート」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションを開始する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.1　J2EE アプリケーションの開始」を参照してください。

JavaBeans クラス（スタブ）は，J2EE アプリケーション（Web サービス）を開始するときに JAX-WS エンジンが動的に生成します。また，JavaBeans クラス（スタブ）の動的生成でエラーが発生しないように，コンパイルした Web サービス実装クラスに対して手順 4.に従いコマンドを実行すると，事前にアノテーションなどのエラーチェックができます。エラーチェックについては，「10.23.1 hwsgen コマンドによるエラーチェックについて」を参照してください。

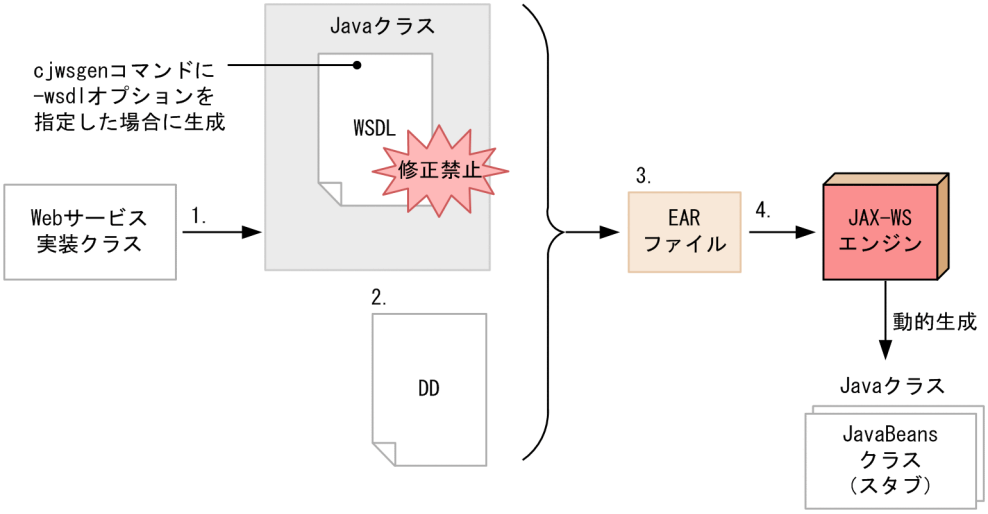
SEI を起点とした Web サービスの開発例については，次に示す個所を参照してください。

- 「5.3　Web サービスの開発例（SEI 起点）」
- 「6.3　Web サービスの開発例（SEI 起点・hwsgen コマンド）」
- 「7.3　Web サービスの開発例（SEI 起点・カスタマイズ）」
- 「8.3　Web サービスの開発例（SEI 起点・EJB の Web サービス）」
- 「29.3　Web サービスの開発例（SEI 起点・wsi:swaRef 形式の添付ファイル）」
- 「33.3　Web サービスの開発例（SEI 起点・ストリーミング）」
- 「38.3　Web サービスの開発例（SEI 起点・アドレッシング）」

2.1.3　SEI を起点とした開発の流れ（hwsgen コマンドを使用する場合）

コンパイル済みの Web サービス実装クラスから，hwsgen コマンドで Java ソースを生成する場合の，SEI を起点とした Web サービスの開発の流れを次の図に示します。

図 2-4　SEI を起点とした Web サービスの開発の流れ（hwsgen コマンドを使用する場合）



1. hwsgen コマンドを実行する

   Web サービス実装クラスは，POJO として作成する場合と EJB を基に作成する場合があります。どちらの場合も，hwsgen コマンドを実行するとき，-wsdl オプションを指定すると，WSDL の生成，およびアノテーションなどのエラーチェックができます。この場合，生成した WSDL は変更しないでください。

   hwsgen コマンドについては，「14.1.2　hwsgen コマンド」を参照してください。

2. DD を作成する

   web.xml および application.xml を作成します。web.xml には，Web サービス固有の情報を記述します。web.xml の作成については，「3.4　web.xml の作成」を参照してください。

3. EAR ファイルを作成する

   作成したファイルを含む EAR ファイルを作成します。EAR ファイルの作成については，「3.5.3　EAR ファイルの作成」を参照してください。

4. EAR ファイルをデプロイし，開始する

   作成した EAR ファイルをデプロイし，J2EE アプリケーション（Web サービス）として開始します。J2EE アプリケーションのインポートおよび開始コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート）」および「cjstartapp（J2EE アプリケーションの開始）」を参照してください。

   運用管理ポータルを使用して，J2EE アプリケーションをデプロイ（インポート）する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.3　J2EE アプリケーションのインポート」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションを開始する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.1　J2EE アプリケーションの開始」を参照してください。

JavaBeans クラス（スタブ）は，J2EE アプリケーション（Web サービス）を開始するときに JAX-WS エンジンが動的に生成します。

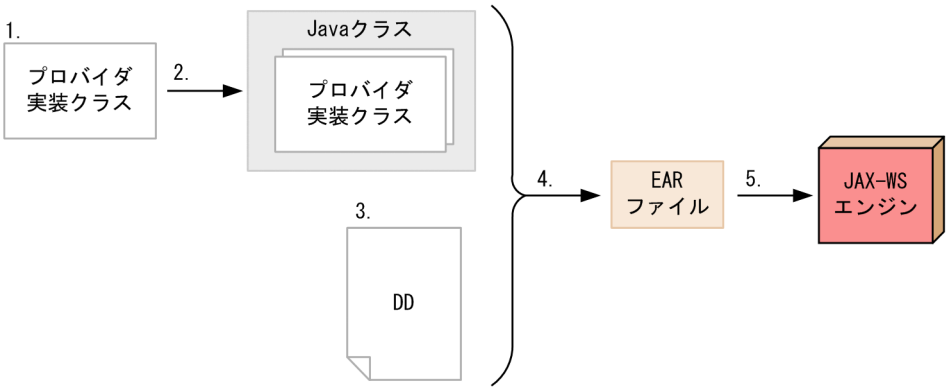
SEI を起点とした Web サービスの開発例については，次に示す個所を参照してください。

- 「5.3　Web サービスの開発例（SEI 起点）」
- 「6.3　Web サービスの開発例（SEI 起点・hwsgen コマンド）」
- 「7.3　Web サービスの開発例（SEI 起点・カスタマイズ）」
- 「8.3　Web サービスの開発例（SEI 起点・EJB の Web サービス）」
- 「29.3　Web サービスの開発例（SEI 起点・wsi:swaRef 形式の添付ファイル）」
- 「33.3　Web サービスの開発例（SEI 起点・ストリーミング）」
- 「38.3　Web サービスの開発例（SEI 起点・アドレッシング）」

2.1.4　プロバイダを起点とした開発の流れ

プロバイダを起点とした Web サービスの開発の流れを次の図に示します。

図 2-5　プロバイダを起点とした Web サービスの開発の流れ



1. プロバイダ実装クラスを作成する

Web サービス実装クラスは，POJO として作成します。JAX-WS 2.2 仕様および JAXB 2.2 仕様に従って，プロバイダ実装クラスを作成します。SOAP 1.2 仕様のメッセージを受信する場合は，javax.xml.ws.BindingType アノテーションに"http://www.w3.org/2003/05/soap/bindings/HTTP/"を指定してください。

JAX-WS 2.2 仕様のサポート範囲については，「19.1　JAX-WS 2.2 仕様のサポート範囲」を参照してください。JAXB 2.2 仕様のサポート範囲については，マニュアル「XML Processor ユーザーズガイド」の「付録 B　JAXB 仕様のサポート範囲」を参照してください。

2. javac コマンドを実行する（コンパイルとエラーチェック）

javac コマンドを実行し，作成したプロバイダ実装クラスのコンパイル，およびエラーチェックをします。

3. DD を作成する

web.xml および application.xml を作成します。web.xml には，Web サービス固有の情報を記述します。web.xml の作成については，「3.4　web.xml の作成」を参照してください。

4. EAR ファイルを作成する

作成したファイルを含む EAR ファイルを作成します。EAR ファイルの作成については，「3.5.3　EAR ファイルの作成」を参照してください。

5. EAR ファイルをデプロイし，開始する

作成した EAR ファイルをデプロイし，J2EE アプリケーション（Web サービス）として開始します。J2EE アプリケーションのインポートおよび開始コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート）」および「cjstartapp（J2EE アプリケーションの開始）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションをデプロイ（インポート）する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.3　J2EE アプリケーションのインポート」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションを開始する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.1　J2EE アプリケーションの開始」を参照してください。

プロバイダを起点とした Web サービスの開発例については，「9.3　Web サービスの開発例（プロバイダ起点・SAAJ）」を参照してください。

2.2　Web サービスクライアント開発の流れ

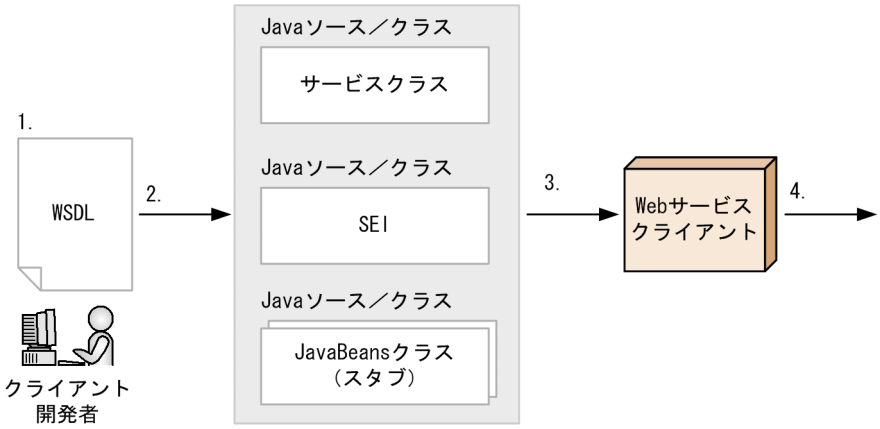
SOAP Web サービスを呼び出すクライアントには，次に示す開発方法があります。

- スタブベースの Web サービスクライアントの開発（2.2.1）
- ディスパッチベースの Web サービスクライアントの開発（2.2.2）

2.2.1　スタブベースの Web サービスクライアントの開発の流れ

スタブベースの Web サービスクライアントの開発の流れを次の図に示します。

図 2-6　Web サービスクライアントの開発の流れ（スタブベース）



1. WSDL ファイルを入手する（または，公開されている WSDL の URL を取得する）

   呼び出そうとしている Web サービスのメタ情報が記述された WSDL ファイルを入手します。または，呼び出そうとしている Web サービスが WSDL ファイルの URL を公開している場合，URL を取得します。

2. cjwsimport コマンドを実行する（Java ソースの生成）

   cjwsimport コマンドを実行して，入手した WSDL ファイル，または取得した WSDL ファイルの URL から，サービスクラスや SEI など，Web サービスクライアントの開発・実行に必要な Java ソースを生成します。cjwsimport コマンドは，-generateService オプションを指定しないで実行します。cjwsimport コマンドについては，「14.1.1　cjwsimport コマンド」を参照してください。

3. Web サービスクライアントを実装する

   生成された Java ソースを利用して，Web サービスクライアントを実装します。実装した Web サービスクライアントは，javac コマンドでコンパイルします。Web サービスクライアントの実装については，「3.6　Web サービスクライアントの実装」を参照してください。

4. Web サービスを呼び出す

   作成した Web サービスクライアントを実行して，Web サービスを呼び出します。

WSDL を起点に開発する場合のスタブベースの Web サービスクライアントの開発例については，「4.5 Web サービスクライアントの開発例（WSDL 起点）」を参照してください。

SEI を起点に開発する場合のスタブベースの Web サービスクライアントの開発例については，次に示す個所を参照してください。

- 「5.5　Web サービスクライアントの開発例（SEI 起点）」
- 「6.5　Web サービスクライアントの開発例（SEI 起点・hwsgen コマンド）」
- 「7.5　Web サービスクライアントの開発例（SEI 起点・カスタマイズ）」
- 「8.5　Web サービスクライアントの開発例（SEI 起点・EJB の Web サービス）」
- 「29.5　Web サービスクライアントの開発例（SEI 起点・wsi:swaRef 形式の添付ファイル）」
- 「33.3　Web サービスの開発例（SEI 起点・ストリーミング）」
- 「38.5　Web サービスクライアントの開発例（SEI 起点・アドレッシング）」

2.2.2　ディスパッチベースの Web サービスクライアントの開発の流れ

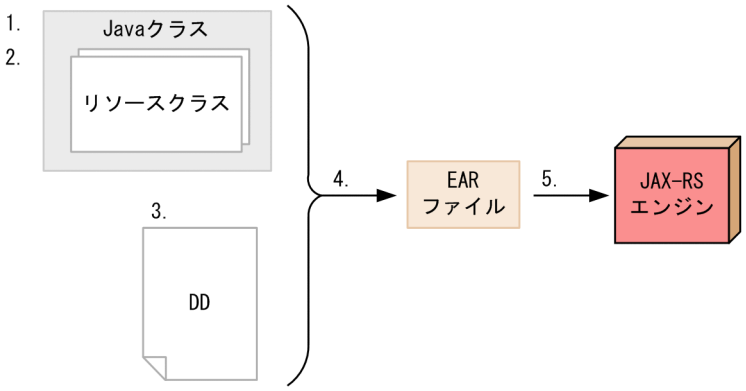
ディスパッチベースの Web サービスクライアントを開発する場合は，JAX-WS 2.2 仕様および JAXB 2.2 仕様に従って，Web サービスクライアントを実装してください。

ディスパッチベースの Web サービスクライアントの開発例については，「9.5　Web サービスクライアントの開発例（プロバイダ起点・SAAJ）」を参照してください。

2.3　RESTful Web サービスの開発の流れ

RESTful Web サービス（Web サービス）の開発の流れを次の図に示します。

図 2-7　RESTful Web サービスの開発の流れ



1. ルートリソースクラスを作成する

ルートリソースクラスを作成し，リソースメソッド，サブリソースメソッド，またはサブリソースロケータのどれか一つ以上を実装します。必要があればサブリソースクラスや例外マッピングプロバイダも作成してください。

2. javac コマンドを実行する

javac コマンドを実行して，作成した Java ソースをコンパイルします。

3. DD を作成する

web.xml および application.xml を作成します。web.xml には，Web サービス固有の情報を記述します。web.xml の作成については，「11.2　web.xml の作成」を参照してください。

4. EAR ファイルを作成する

作成したファイルを含む EAR ファイルを作成します。EAR ファイルの作成については，「11.3.2　EAR ファイルの作成」を参照してください。

5. EAR ファイルをデプロイし，開始する

作成した EAR ファイルをデプロイし，J2EE アプリケーション（Web サービス）として開始します。J2EE アプリケーションのインポートおよび開始コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート）」および「cjstartapp（J2EE アプリケーションの開始）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションをデプロイ（インポート）する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.3　J2EE アプリケーションのインポート」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションを開始する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.1　J2EE アプリケーションの開始」を参照してください。

RESTful Web サービス（Web サービス）の開発例については，「12.3　Web リソースの開発例」を参照してください。

3 SOAP Web サービス開発のポイント

この章では，Web サービス開発の作業ごとに，あらかじめ理解しておく必要がある点と注意事項について説明します。

3.1　WSDL の作成

WSDL を起点とした開発では，WSDL 1.1 仕様および WS-I Basic Profile 1.1 に従って WSDL を作成します。WSDL 1.1 仕様のサポート範囲については，「20.1　WSDL 1.1 仕様のサポート範囲」を参照してください。

POJO の Web サービスの場合，作成した WSDL は，WAR ファイルを構成する wsdl ディレクトリに格納します。WSDL の格納先については，「3.5.1　WAR ファイルの構成」を参照してください。

EJB の Web サービスの場合，作成した WSDL は，EJB JAR ファイルを構成する wsdl ディレクトリに格納します。WSDL の格納先については，「3.5.2　EJB JAR ファイルの構成」を参照してください。

wsdl ディレクトリに WSDL が含まれていない場合，Web サービスをデプロイするときに，Web サービス側の JAX-WS エンジンによって JAX-WS 2.2 仕様に従った WSDL が自動生成されます。

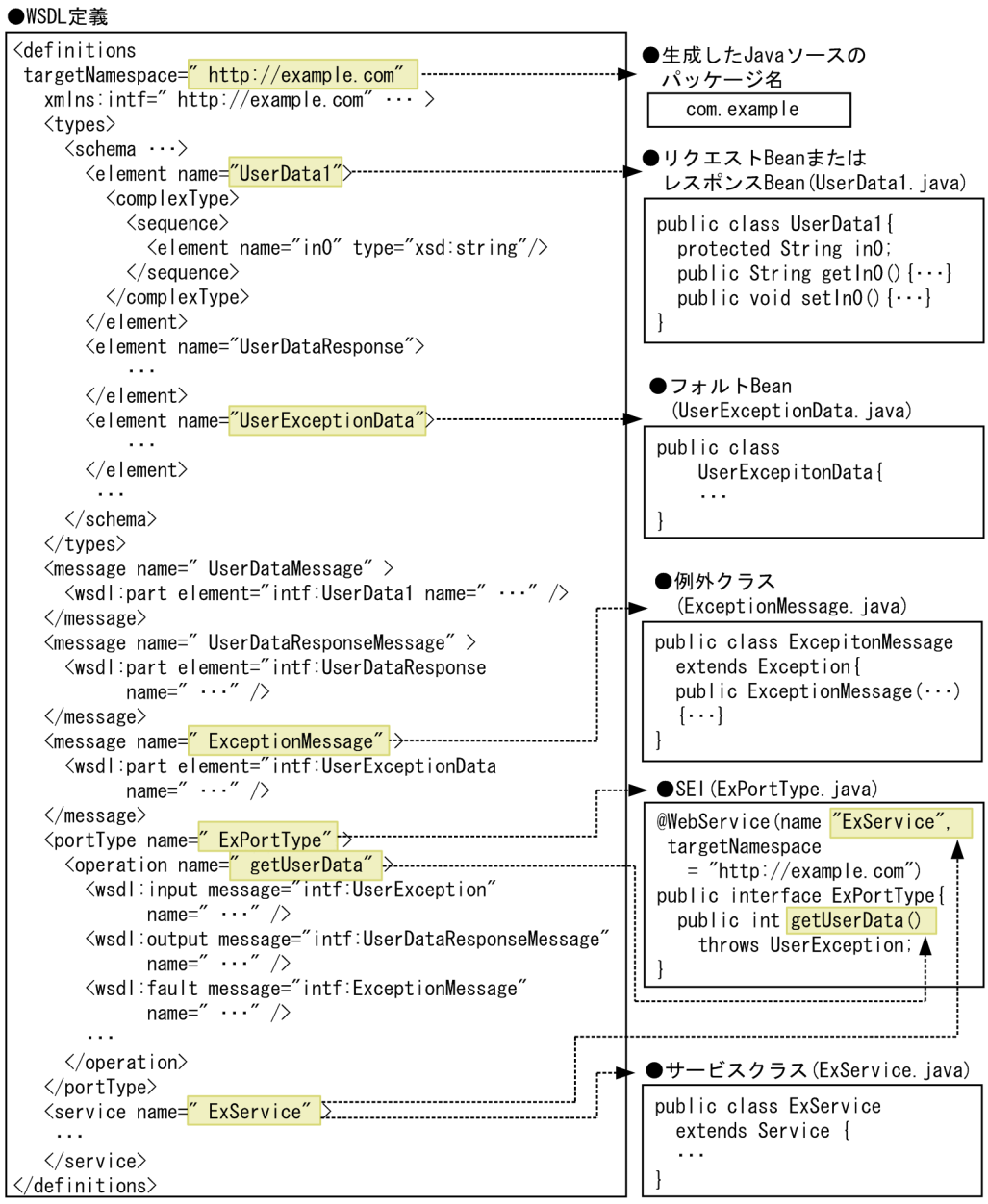
3.2　WSDL と Java ソースのマッピング

cjwsimport コマンド実行時に，WSDL と Java ソース間でマッピングされます。ここでは，WSDL と Java ソース間のマッピング例（デフォルトマッピング）を示します。

3.2.1　WSDL から Java ソースへのマッピング例

cjwsimport コマンドを実行すると，WSDL から Java ソースへマッピングされます。WSDL から Java ソースへのマッピング例を次の図に示します。

図 3-1　WSDL から Java ソースへのマッピング例

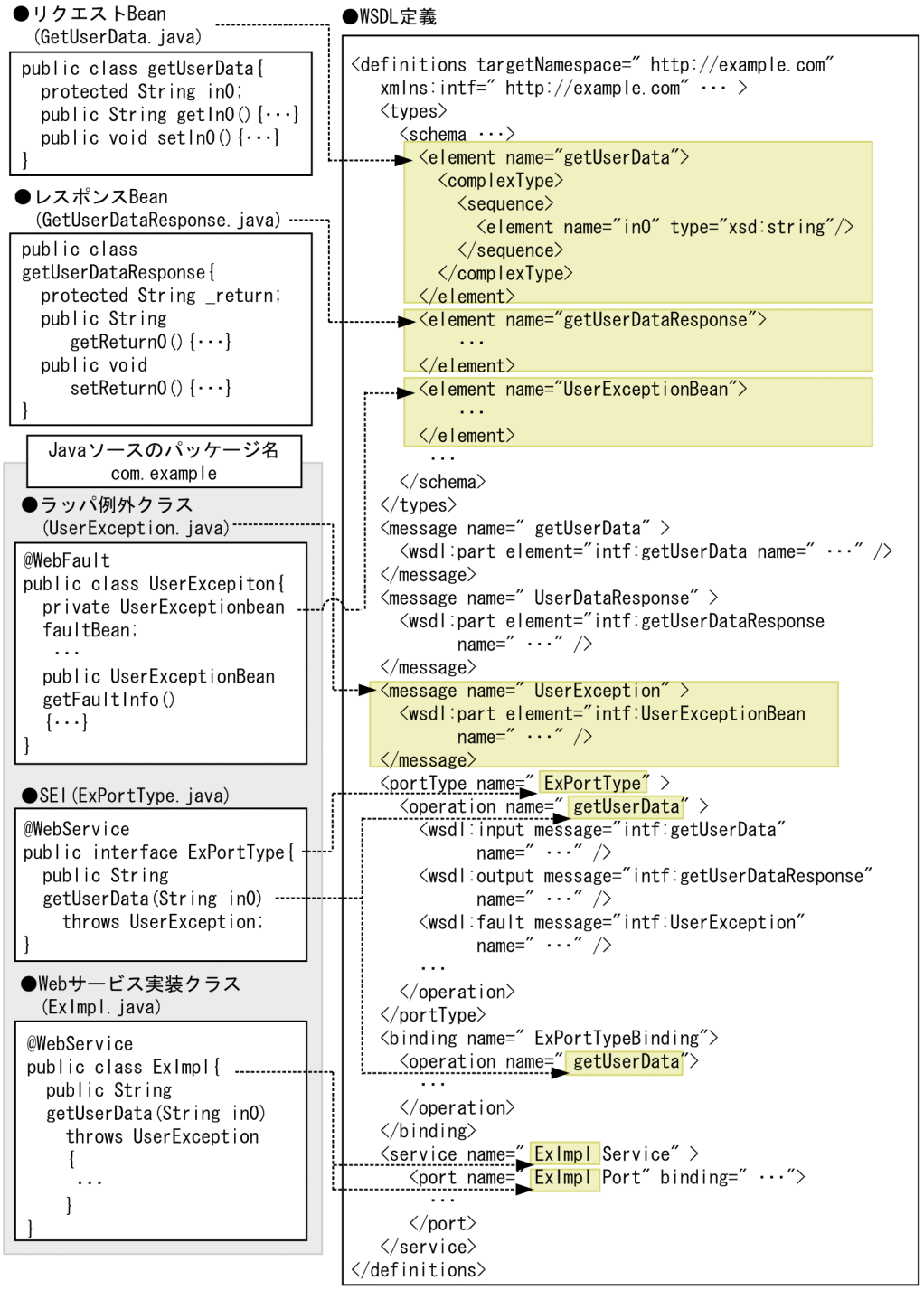


マッピングについては，「15.　WSDL から Java へのマッピング」を参照してください。

3.2.2　Java ソースから WSDL へのマッピング例

Java ソースから WSDL へのマッピング例を次の図に示します。

図 3-2　Java ソースから WSDL へのマッピング例



マッピングについては，「16.　Java から WSDL へのマッピング」を参照してください。

3.3　Web サービス実装クラスおよびプロバイダ実装クラスの作成

Web サービス実装クラスまたはプロバイダ実装クラスは，コンパイルした Java クラスファイル（*.class）として作成します。

POJO の Web サービスの場合

Web サービス実装クラスまたはプロバイダ実装クラスは，WAR ファイルを構成するディレクトリに含めます。次に示すどちらか，または両方に一つ以上含めてください。

- classes ディレクトリ以下
- lib ディレクトリ以下に含まれる JAR ファイル内

Web サービス実装クラスまたはプロバイダ実装クラスの格納先については，「3.5.1　WAR ファイルの構成」を参照してください。

EJB の Web サービスの場合

Web サービス実装クラスは，EJB JAR ファイルを構成するディレクトリに含めます。次に示すディレクトリに一つ以上含めてください。

- EJB-JAR ファイルのルートディレクトリ以下

Web サービス実装クラスの格納先については，「3.5.2　EJB JAR ファイルの構成」を参照してください。

作成時の注意事項

通常，Web サービスはステートレスです。Web サービス実装クラスまたはプロバイダ実装クラスのフィールドを，複数回の Web サービスの呼び出しで共有しないでください。

3.4 web.xml の作成

ここでは，POJO の Web サービスで使用する WAR ファイルに含まれる web.xml について説明します。

web.xml を作成する場合，ファイル名称は"web.xml"とし，WAR ファイルを構成する WEB-INF ディレクトリの直下に格納します。web.xml の格納が必須かどうかは，J2EE サーバ用ユーザプロパティファイル（usrconf.properties）の webserver.container.jaxws.webservice.no_webxml.enabled プロパティの設定値によって異なります。

- "strict"または"true"（推奨は"strict"）を設定した場合

  WEB-INF ディレクトリの直下に"web.xml"という名称で web.xml を格納するかどうかは任意です。格納する場合，Web サービスの実行に必要な定義が記述されていなければなりません。
- "lax"を設定した場合

  WEB-INF ディレクトリの直下に"web.xml"という名称で web.xml を格納するかどうかは任意です。格納する場合，Web サービスの実行に必要な定義が記述されていなくてもかまいません。
- "none"または"false"（推奨は"none"）を設定した場合

  必ず WEB-INF ディレクトリの直下に"web.xml"という名称で web.xml を格納してください。

次に，Web サービスの実行に必要な定義，web.xml の例，および web.xml を WAR ファイルに含めない場合の動作について説明します。

3.4.1 Web サービスの実行に必要な定義

webserver.container.jaxws.webservice.no_webxml.enabled プロパティに"strict"を設定して web.xml を WAR ファイルに含める場合，または"none"を設定した場合，web.xml は，次に示す条件を満たすように作成してください。

- バージョン

  web.xml のバージョンは，2.5 以上にしてください。
- リスナ

  web-app 要素に，次に示す listener 要素を含めてください。

```
<listener>
  <listener-class>
    com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
  </listener-class>
</listener>
```

- サーブレット

  web-app 要素に，次に示す servlet 要素を含めてください。

```
<servlet>
  <description>Endpoint servlet for Cosminexus JAX-WS</description>
  <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
```

```
  <servlet-name>CosminexusJaxwsServlet</servlet-name>
  <servlet-class>
    com.cosminexus.xml.ws.transport.http.servlet.WSServlet
  </servlet-class>
</servlet>
```

- **サーブレットマッピング**

  web-app 要素以下に servlet-mapping 要素を記述し，Web サービス実装クラスまたはプロバイダ実装クラスと同じ数の url-pattern 要素を含めてください。

  servlet-mapping 要素の記述例を次に示します。

```
<servlet-mapping>
  <servlet-name>CosminexusJaxwsServlet</servlet-name>
  <url-pattern>”/ ”＋Webサービス1のサービス名</url-pattern>
  <url-pattern>”/ ”＋Webサービス2のサービス名</url-pattern>
    :
  <url-pattern>”/ ”＋Webサービスnのサービス名</url-pattern>
</servlet-mapping>
```

  url-pattern 要素の「" / "＋ Web サービス 1 のサービス名」は，次の値にプレフィクスとして「/」（スラッシュ）を付けた文字列を記述します。

  - Web サービス実装クラスの javax.jws.WebService アノテーションの serviceName 属性の値
  - プロバイダ実装クラスの javax.xml.ws.WebServiceProvider アノテーションの serviceName 属性の値

  参考

  WAR ファイルに cosminexus-jaxws.xml を含める場合

  cosminexus-jaxws.xml の Web サービス実装クラス，またはプロバイダ実装クラスに対応する endpoint 要素の url-pattern 属性の値を記述してください。cosminexus-jaxws.xml については，「10.3　cosminexus-jaxws.xml によるカスタマイズ」を参照してください。

- **そのほかの要素**

  任意に記述できます。WAR ファイルに作成したサーブレット，リスナ，JSP などを含める場合，適宜，web.xml に定義してください。

3.4.2　web.xml の例

web.xml の例を次に示します。

```
<?xml version=”1.0” encoding=”UTF-8”?>
<web-app version=”3.0” xmlns=”http://java.sun.com/xml/ns/javaee”
  xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
  xsi:schemaLocation=”http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd”>
  <description>Sample web service &amp;quot;fromwsdl&amp;quot;</description>
```

```
  <display-name>Sample_web_service_fromwsdl</display-name>
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/TestJaxWsService</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

バージョン 2.5 の web.xml を作成する場合は，web-app 要素の version 属性を"2.5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"にしてください。

この例では，次の属性の値が「TestJaxWsService」であることを想定しています。

- Web サービス実装クラスの javax.jws.WebService アノテーションの serviceName 属性
- プロバイダ実装クラスの javax.xml.ws.WebServiceProvider アノテーションの serviceName 属性

このとき，url-pattern 要素の値は「/」（スラッシュ）を付けて「/TestJaxWsService」と記述します。

参考

WAR ファイルに cosminexus-jaxws.xml を含める場合

Web サービス実装クラスまたはプロバイダ実装クラスに対応する endpoint 要素の url-pattern 属性の値が「/TestJaxWsService」であれば，例と同じように記述します。

3.4.3 web.xml を WAR ファイルに含めない場合の動作

Application Server の JAX-WS 機能では，webserver.container.jaxws.webservice.no_webxml.enabled プロパティに"strict"，または"lax"を設定して web.xml を WAR ファイルに含めない場合，Web サービス呼び出し時に次に示す内容の web.xml があるものとして処理されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- web-app要素を記述する -->
  <description>Cosminexus JAX-WS Default web.xml</description>
  <display-name>Cosminexus_JAX_WS_Default_web_xml</display-name>
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>"/"＋Webサービス1のサービス名</url-pattern>
    <url-pattern>"/"＋Webサービス2のサービス名</url-pattern>
      :
    <url-pattern>"/"＋Webサービスnのサービス名</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

url-pattern 要素の「" / "+ Web サービス 1 のサービス名」は，次の属性の値に，プレフィクスとして「/」（スラッシュ）を付けた文字列が定義されます。

- Web サービス実装クラスの javax.jws.WebService アノテーションの serviceName 属性の値
- プロバイダ実装クラスの javax.xml.ws.WebServiceProvider アノテーションの serviceName 属性の値

WAR ファイル内に格納したすべての Web サービス実装クラスまたはプロバイダ実装クラスについて，url-pattern 要素があるかのように動作します。

webserver.container.jaxws.webservice.no_webxml.enabled プロパティに"lax"を設定して web.xml を WAR ファイルに含める場合でも，「3.4.1　Web サービスの実行に必要な定義」の内容が正しく web.xml に記述されていないときは，web.xml が含まれない場合と同様に，上記に示す内容の web.xml があるものと仮定して Web サービスを呼び出します。なお，このとき仮定される web.xml は，上記の内容から session-config 要素以下を除いた内容です。

注意事項

- JAX-WS エンジンが仮定した web.xml は，実際に WAR ファイル内に生成されるわけではありません。Web サービスが呼び出される場合を除いて，すべてこの仮定はないものとして扱われます。

（例）

cjgetappprop コマンドで取得できる属性ファイルには，web.xml に関する情報は含まれません。また，webserver.container.jaxws.webservice.no_webxml.enabled プロパティに"lax"を設定して不正な内容の web.xml を WAR ファイルに含めた場合は，実際に含まれている web.xml に関する情報だけが取得できます。

- webserver.container.jaxws.webservice.no_webxml.enabled プロパティに"lax"を設定して不正な内容の web.xml を WAR ファイルに含めた場合，JAX-WS エンジンが Web サービス呼び出し時に仮定する web.xml の内容が，実際の web.xml に記載されるわけではありません。
- webserver.container.jaxws.webservice.no_webxml.enabled プロパティに"lax"を設定する場合，「3.4.1　Web サービスの実行に必要な定義」の内容をすべて定義した web.xml を WAR ファイルに含めてください。一部だけを定義した web.xml を WAR ファイルに含めた場合の動作は保証されません。

3.5　アーカイブの作成

WAR ファイルの構成，EJB JAR ファイルの構成および EAR ファイルの作成について説明します。

3.5.1　WAR ファイルの構成

POJO の Web サービスで使用する WAR ファイルは，次の表に示す構成にする必要があります。

表 3-1　WAR ファイルの構成

ディレクトリ	備考
/	－
┣ META-INF/	－
┃ ┗ MANIFEST.MF	－
┗ WEB-INF/	－
┣ web.xml	作成した web.xml です。
┣ classes/	コンパイルした Java クラスを格納します。
┣ lib/	コンパイルした Java クラスを含む JAR ファイルを格納します。
┗ wsdl/	作成した WSDL を格納します。WSDL を Web サービスのメタデータとして公開するには，このディレクトリに含める必要があります。なお，別の WAR ファイルおよび EJB JAR ファイルに含まれる WSDL ファイルをこのディレクトリに含めないでください。

（凡例）

－：説明や補足事項が特にないことを示します。

注

- cosminexus-jaxws.xml を WAR ファイルに含める場合は，WEB-INF ディレクトリ直下に含めます。cosminexus-jaxws.xml については，「10.3　cosminexus-jaxws.xml によるカスタマイズ」を参照してください。
- Application Server の JAX-WS エンジンでは，同一の WAR ファイルに，複数の Web サービスを含められます。複数の Web サービスを含める場合，異なる機能を持つクラスのクラス名は重複できません。異なる機能を持つクラスのクラス名が重複していると，Web サービスが正常に動作しないおそれがあります。ただし，複数の Web サービスで同じクラスを利用する場合は，クラス名が重複していてもかまいません。
- cjwsimport コマンドが package-info.java を生成した場合は，パッケージ名から名前空間へのマッピングに必要なアノテーションなどの Web サービスに必要な情報が出力されているため，必ず package-info.java を WAR ファイルに含めてください。

3.5.2　EJB JAR ファイルの構成

EJB の Web サービスで使用する EJB JAR ファイルは，次の表に示す構成にする必要があります。

表 3–2　EJB JAR ファイルの構成

ディレクトリ			備考
/			コンパイルした Java クラスを格納します。EJB の Web サービス実装クラスは，このディレクトリに含める必要があります。
└	META-INF/		－
	├	wsdl/	作成した WSDL を格納します。WSDL を Web サービスのメタデータとして公開するには，このディレクトリに含める必要があります。なお，別の WAR ファイルおよび EJB JAR ファイルに含まれる WSDL ファイルをこのディレクトリに含めないでください。
	└	MANIFEST.MF	－

（凡例）

－：説明や補足事項が特にないことを示します。

注

- Application Server の JAX-WS エンジンでは，同一の EJB JAR ファイルに，複数の Web サービスを含められます。複数の Web サービスを含める場合，異なる機能を持つクラスのクラス名は重複できません。異なる機能を持つクラスのクラス名が重複していると，Web サービスが正常に動作しないおそれがあります。ただし，複数の Web サービスで同じクラスを利用する場合は，クラス名が重複していてもかまいません。
- cjwsimport コマンドが package-info.java を生成した場合は，パッケージ名から名前空間へのマッピングに必要なアノテーションなどの Web サービスに必要な情報が出力されているため，必ず package-info.java を EJB JAR ファイルに含めてください。

3.5.3　EAR ファイルの作成

Web サービスを J2EE サーバにデプロイするには，作成した WAR ファイルまたは EJB JAR ファイルを含めた EAR ファイルを作成します。EAR ファイルを作成するには，application.xml が必要になります。

EAR ファイルの構成については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「1.4.2　アーカイブ形式の J2EE アプリケーション」を参照してください。

3.5.4　EJB の Web サービスの設定用 WAR ファイルの作成

EJB の Web サービス呼び出し機能は，EAR ファイルに設定用 WAR ファイルを含めない構成で利用できます。ただし，WAR ファイルの指定が必要なアプリケーションサーバの機能を利用するために，EAR ファイルに設定用 WAR ファイルを含めることができます。ここでは設定用 WAR ファイルについて説明します。

(1)　EAR ファイルに設定用 WAR ファイルを含めない場合

EAR ファイルに EJB JAR ファイルが含まれていて，さらに EJB JAR ファイルに EJB の Web サービス実装クラスが含まれるとき，JAX-WS エンジンは設定用 WAR ファイルを含むと仮定して動作します。仮定される設定用 WAR ファイルの構成を次の表に示します。

表 3-3　仮定される設定用 WAR ファイルの構成

ディレクトリ	備考
/	－
┣ WEB-INF/	－
┃ ┗ web.xml	3.5.4(4)を参照してください。
┗ META-INF/	－

（凡例）

－：説明や補足事項が特にないことを示します。

(2)　EAR ファイルに設定用 WAR ファイルを含める場合

EJB の Web サービス実装クラスを呼び出す際に同時にサーブレットフィルタ機能を適用するなど，web.xml に設定を加えたい場合，作成した web.xml を設定用 WAR ファイルに格納し，EAR ファイルに含めます。設定用 WAR ファイルの構成を次の表に示します。なお，設定用 WAR ファイルには命名規則があります。設定用 WAR ファイルのファイル名については，「3.5.4(3)　設定用 WAR ファイル名」を参照してください。

表 3-4　設定用 WAR ファイルの構成

ディレクトリ	備考
/	－
┣ WEB-INF/	－
┃ ┣ classes/	コンパイルした Java クラスを格納します。フィルタを使用する場合は，フィルタの Java クラスをこのディレクトリに格納します。※
┃ ┣ lib/	コンパイルした Java クラスを含む JAR ファイルを格納します。フィルタを使用する場合は，フィルタの Java クラスを含む JAR ファイルをこのディレクトリに格納します。※

ディレクトリ			備考
│	│		
│	└	web.xml	3.5.4(4)を参照してください。
└	META-INF/		－

（凡例）

－：説明や補足事項が特にないことを示します。

注※

設定用 WAR ファイルの classes に POJO の Web サービスを含めないでください。POJO の Web サービスを含めた場合の動作は保証されません。

(3) 設定用 WAR ファイル名

EAR ファイルに設定用 WAR ファイルを含める場合，EJB の Web サービスの設定用 WAR ファイル名は，J2EE サーバ用ユーザプロパティファイル（usrconf.properties）の webserver.container.jaxws.webservice.wsee.warname プロパティで指定したファイル名にする必要があります。

EAR ファイルに設定用 WAR ファイルを含めない場合，webserver.container.jaxws.webservice.wsee.warname プロパティで指定したファイル名の設定用 WAR ファイルを含むと仮定して動作します。EAR ファイルに設定用 WAR ファイルを含めない場合，メッセージが J2EE サーバのログに出力されるので，JAX-WS エンジンが設定用 WAR ファイルを含むと仮定したことを確認できます（KDJE42391-I）。このメッセージが出力されなかった場合は，プロパティで指定した設定用 WAR ファイルが EAR ファイルに含まれることを確認できます。

なお，プロパティのデフォルト値は「CosminexusWSEE.war」です。

EJB の Web サービス呼び出しに対して，次に示す設定を使用できます。これらの設定を使用する場合は，設定用 WAR ファイルを EAR ファイルに含めて，設定用 WAR ファイル名を指定してください。

- コンテキストルートの設定

  「application.xml の<web-uri>要素」に設定用 WAR ファイル名を指定します。

  EJB の Web サービスの設定用 WAR ファイルに対してコンテキストルートを設定しなかった場合，コンテキストルートは「/」と仮定されます。

- cosminexus.xml の<war>要素で指定する設定

  「cosminexus.xml の<module-name>要素」に設定用 WAR ファイル名を指定します。

注意事項

webserver.container.jaxws.webservice.wsee.warname プロパティの値を変更するには，EJB の Web サービスを含む Web アプリケーションを停止してください。Web アプリケー

ションを開始した状態で，プロパティの値を変更した場合は動作を保証されません。ほかのアプリケーションが不正となって，予期しない例外が発生する場合があります。

(4) 設定用 WAR ファイルの web.xml

ここでは EJB の Web サービスの設定用 WAR ファイルに含まれる web.xml について説明します。

web.xml を作成する場合，ファイル名称は"web.xml"とし，WAR ファイルを構成する WEB-INF ディレクトリの直下に格納します。web.xml の格納が必須かどうかは，J2EE サーバ用ユーザプロパティファイル（usrconf.properties）の webserver.container.jaxws.webservice.wsee.no_webxml.enabled プロパティの設定値によって異なります。

- "strict"を設定した場合

  WEB-INF ディレクトリの直下に"web.xml"という名称で web.xml を格納するかどうかは任意です。格納する場合，Web サービスの実行に必要な定義が記述されていなければなりません。

- "lax"を設定した場合

  WEB-INF ディレクトリの直下に"web.xml"という名称で web.xml を格納するかどうかは任意です。格納する場合，Web サービスの実行に必要な定義が記述されていなくてもかまいません。

- "none"を設定した場合

  必ず WEB-INF ディレクトリの直下に"web.xml"という名称で web.xml を格納してください。

次に，設定用 WAR ファイルに web.xml を含めない場合の動作および設定用 WAR ファイルに web.xml を含める場合の動作について説明します。

(a) 設定用 WAR ファイルに web.xml を含めない場合

webserver.container.jaxws.webservice.wsee.no_webxml.enabled プロパティの設定値に"strict"，または"lax"を設定して，EJB の Web サービスの設定用 WAR ファイルに web.xml を含めない場合，Web サービス呼び出し時に次に示す内容の web.xml があるものとして処理されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- web-app要素を記述する -->
  <description>Cosminexus JAX-WS Default web.xml</description>
  <display-name>Cosminexus_JAX_WS_Default_web_xml</display-name>
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.EJBWSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>EJB Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>EJB_Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsEjbServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.EJBWSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
```

```
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxwsEjbServlet</servlet-name>
    <url-pattern>”/” + Webサービス1のサービス名+ ”/” + Webサービス1のクラス名</url-pattern>
    <url-pattern>”/” + Webサービス2のサービス名+ ”/” + Webサービス2のクラス名</url-pattern>
      :
    <url-pattern>”/” + Webサービスnのサービス名+ ”/” + Webサービスnのクラス名</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

url-pattern 要素の「"/" + Web サービス 1 のサービス名+ "/" + Web サービス 1 のクラス名」は，Web サービス実装クラスの javax.jws.WebService アノテーションの serviceName 属性と name 属性の値に，プレフィクスとして「/」（スラッシュ）を付けた文字列が定義されます。

設定用 WAR ファイル内に格納したすべての Web サービス実装クラスについて，url-pattern 要素があるかのように動作します。

注意事項

- JAX-WS エンジンが仮定した web.xml は，実際に設定用 WAR ファイル内に生成されるわけではありません。Web サービスが呼び出される場合を除いて，すべてこの仮定はないものとして扱われます。

（例）

cjgetappprop コマンドで取得できる属性ファイルには，web.xml に関する情報は含まれません。また，webserver.container.jaxws.webservice.wsee.no_webxml.enabled プロパティに"lax"を設定して不正な内容の web.xml を設定用 WAR ファイルに含めた場合は，実際に含まれている web.xml に関する情報だけが取得できます。

- webserver.container.jaxws.webservice.wsee.no_webxml.enabled プロパティに"lax"を設定して不正な内容の web.xml を設定用 WAR ファイルに含めた場合，JAX-WS エンジンが Web サービス呼び出し時に仮定する web.xml の内容が，実際の web.xml に記載されるわけではありません。

(b)　設定用 WAR ファイルに web.xml を含める場合

webserver.container.jaxws.webservice.wsee.no_webxml.enabled プロパティに"strict"を設定して web.xml を WAR ファイルに含める場合，または"none"を設定した場合，web.xml は，次に示す条件を満たすように作成してください。

- バージョン

  web.xml のバージョンは，2.5 以上にしてください。

- リスナ

web-app 要素に，次に示す listener 要素を含めてください。

```
<listener>
  <listener-class>
    com.cosminexus.xml.ws.transport.http.servlet.EJBWSServletContextListener
  </listener-class>
</listener>
```

- サーブレット

  web-app 要素に，次に示す servlet 要素を含めてください。

```
<servlet>
  <description>EJB Endpoint servlet for Cosminexus JAX-WS</description>
  <display-name>EJB_Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
  <servlet-name>CosminexusJaxwsEjbServlet</servlet-name>
  <servlet-class>
    com.cosminexus.xml.ws.transport.http.servlet.EJBWSServlet
  </servlet-class>
</servlet>
```

- サーブレットマッピング

  web-app 要素以下に servlet-mapping 要素を記述し，Web サービス実装クラスと同じ数の url-pattern 要素を含めてください。

  servlet-mapping 要素の記述例を次に示します。

```
<servlet-mapping>
  <servlet-name>CosminexusJaxwsEjbServlet</servlet-name>
  <url-pattern>"/" + Webサービス1のサービス名+ "/" + Webサービス1のクラス名</url-pattern>
  <url-pattern>"/" + Webサービス2のサービス名+ "/" + Webサービス2のクラス名</url-pattern>
    :
  <url-pattern>"/" + Webサービスnのサービス名+ "/" + Webサービスnのクラス名</url-pattern>
</servlet-mapping>
```

  url-pattern 要素の「Web サービス n のサービス名」および「Web サービス n のクラス名」には，次の文字列を記述します。

  - 「Web サービス n のサービス名」

    Web サービス実装クラスの javax.jws.WebService アノテーションの serviceName 属性の値を記述します。serviceName 属性が省略されている場合，Web サービス実装クラスのクラス名（単純名）にサフィックスとして"Service"を付けた文字列を記述します。

  - 「Web サービス n のクラス名」

    Web サービス実装クラスの javax.jws.WebService アノテーションの name 属性の値を記述します。name 属性が省略されている場合，Web サービス実装クラスのクラス名（単純名）を記述します。

- そのほかの要素

  任意に記述できます。WAR ファイルに作成したサーブレットなどを含める場合，適宜，web.xml に定義してください。

EJB の Web サービスの設定用 WAR ファイルに含める web.xml の例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/we
b-app_3_0.xsd">
  <description>Cosminexus JAX-WS Default web.xml</description>
  <display-name>Cosminexus_JAX_WS_Default_web_xml</display-name>
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.EJBWSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>EJB Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>EJB_Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.EJBWSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/AddNumbersImplService/AddNumbersImpl</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

バージョン 2.5 の web.xml を作成する場合は，web-app 要素の version 属性を"2.5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"にしてください。

この例では，Web サービス実装クラスの javax.jws.WebService アノテーションの serviceName 属性の値が「AddNumbersImplService」で，さらに name 属性が「AddNumbersImpl」であることを想定しています。

webserver.container.jaxws.webservice.wsee.no_webxml.enabled プロパティの設定値に"lax"を設定し，設定用 WAR ファイルに web.xml を含める場合，リスナ，サーブレット，サーブレットマッピング，およびそのほかの要素が不完全に含まれるとき，動作は保証されません。

3.6 Web サービスクライアントの実装

Web サービスクライアントの形態に制限はありません。例えば，次に示すような Web サービスクライアントを開発できます。なお，J2EE コンテナ上で動作する Web サービスクライアントを開発する場合の条件は，EJB のときは EJB 3.0 以降，JSP やサーブレットなどの Web アプリケーションのときは Servlet 2.5 以降となります。例えば，Web アプリケーションで web.xml を WAR ファイルに含める場合は，web.xml のバージョンを 2.5 以上にしてください。

- Java アプリケーション（Java アプリケーションから Web サービスを呼び出す）
- JSP（JSP から Web サービスを呼び出す）
- サーブレット（サーブレットから Web サービスを呼び出す）
- EJB（EJB から Web サービスを呼び出す）
- Web サービス（Web サービス実装クラスから別の Web サービスをさらに呼び出す）

Application Server の JAX-WS 機能では，次のどれかの方法で Web サービスクライアントを実装できます。

- サービスクラスとスタブを使用する（スタブベースの Web サービスクライアントの開発）
- javax.xml.ws.Dispatch インタフェースを利用する（ディスパッチベースの Web サービスクライアントの開発）
- その他の JAX-WS API を利用する（API ベースの Web サービスクライアントの開発）

ここでは，Web サービスクライアントの実装例について説明します。また，サービスクラスやポートを使用する上での注意事項についても説明します。

3.6.1 スタブベースの実装例

cjwsimport コマンドで自動生成したサービスクラスやスタブを使用して，Web サービスクライアントを開発できます。スタブベースの Web サービスクライアントの実装で，サービスクラスの生成やポートを取得するには，次の方法があります。

- サービスクラスのコンストラクタの利用

  サービスクラスのコンストラクタを利用して，サービスクラスのインスタンスを生成します。生成したサービスクラスのインスタンスから，ポートを取得します。
- javax.xml.ws.WebServiceRef アノテーションの利用

  javax.xml.ws.WebServiceRef アノテーションを利用して，サービスクラスやポートをインジェクトします。

サービスクラスやポートのインジェクションについては，「10.21.1　サービスクラスおよびポートのインジェクション」を，javax.xml.ws.WebServiceRef アノテーションについては，「19.3　アノテーションのサポート範囲」を参照してください。

(1)　参照される WSDL

スタブベースの Web サービスクライアントを実行する場合，WSDL のパスまたは URL が必要になります。このとき，次の条件の組み合わせによって，参照される WSDL が異なります。

- javax.xml.ws.WebServiceRef アノテーションを利用してインジェクションする場合
  インジェクションする場合に参照される WSDL については，「19.3.1(2)　wsdlLocation 要素(javax.xml.ws.WebServiceRef)」を参照してください。
- サービスクラスのコンストラクタを利用してインスタンスを生成する場合
  参照される WSDL は次に示す条件の組み合わせによって異なります。
  - 利用するコンストラクタの種類
  - cjwsimport コマンドに-wsdllocation オプションを指定して実行したかどうか

条件ごとの参照される WSDL の対応を次の表に示します。なお，それぞれの条件での例については，「3.6.1(5)(a)　サービスクラスをインスタンス化する」を参照してください。

表 3-5　条件の組み合わせと参照される WSDL の対応

項番	利用するコンストラクタ	-wsdllocation オプション	参照される WSDL
1	デフォルトコンストラクタ	×	cjwsimport コマンドの引数に指定した WSDL※1
2	デフォルトコンストラクタ	○	-wsdllocation オプションに指定した WSDL※2
3	java.net.URL オブジェクトおよび javax.xml.namespace.QName オブジェクトをパラメタに持つコンストラクタ	－	パラメタの URL に指定した WSDL※2

(凡例)

○：指定した場合。

×：指定しなかった場合。

－：指定の有無は参照される WSDL に影響しません。

注※1

Web サービスクライアントを実行する場合，WSDL は cjwsimport コマンド実行時と同じ場所に存在している必要があります。cjwsimport コマンド実行時に相対パスを指定した場合も，WSDL は cjwsimport コマンド実行時と同じ場所に存在している必要があります。

注※2

絶対 URL を指定した場合，Web サービスクライアントの実行時に，WSDL がその URL が示す場所に存在している必要があります。

相対 URL を指定した場合，Web サービスクライアントの実行時に，WSDL がカレントディレクトリを基準として相対 URL を解決した場所に存在している必要があります。

(2)　参照されるエンドポイントアドレス

接続する Web サービスの URL（エンドポイントアドレス）は，デフォルトでは WSDL のポート（wsdl:port 要素）が持つアドレス情報（soap:address 子要素の location 属性）が利用されます。ただし，メッセージコンテキストの javax.xml.ws.service.endpoint.address プロパティを利用すると，エンドポイントアドレスを動的に変更できます。エンドポイントアドレスを動的に変更する例については，「3.6.1(5)(c)　ポートのメソッドを呼び出す」を参照してください。

(3)　サービスクラスの生成およびポートの取得

サービスクラスの生成やポートの取得には処理コストが掛かるので，次のように，ポートをインジェクションまたは再利用することをお推めします。

- Web サービスクライアントをサーブレットや EJB として実装する場合
  ポートをインジェクションします。詳細については，「10.21　インジェクション」を参照してください。
- ほかのアプリケーション（コマンドラインアプリケーションなど）で Web サービスクライアントを実装する場合
  初期化処理でサービスクラスの生成やポートの取得を実施し再利用します。ポートを取得するたびにサービスクラスを生成したり，ポートのメソッドを呼び出すたびにポートを取得したりする必要はありません。

(4)　サービスの選択

呼び出す Web サービスの WSDL に複数のサービス（wsdl:service 要素）が含まれる場合，サービスクラスを生成するときに，java.net.URL オブジェクトと javax.xml.namespace.QName オブジェクトをパラメタに持つコンストラクタを使用し，javax.xml.namespace.QName オブジェクトでそのサービス（wsdl:service 要素）を呼び出すのか明示的に指定してください。

(5)　基本的な実装例

Web サービスクライアントの実装には，コンストラクタを利用してサービスクラスを生成する方法と，インジェクションを利用する方法があります。ここでは，コンストラクタを利用してサービスクラスを生成する実装の手順を説明します。インジェクションについては，「10.21.1　サービスクラスおよびポートのインジェクション」を参照してください。

スタブベースの Web サービスクライアントを開発する場合，cjwsimport コマンドを使用して Web サービスを呼び出すために必要な Java ソースを生成します。cjwsimport コマンドは，-generateService オプションを指定しないで実行してください。

スタブベースの Web サービスクライアントは，次の Java ソースを使用して Web サービスを呼び出します。

- **サービスクラス**

  サービスクラスは，JAX-WS 2.2 仕様で規定された Web サービスを呼び出すための WSDL のサービス（wsdl:service 要素）に対応するクラスです。wsdl:service 要素が wsdl:port 要素をまとめるように，複数のポートをまとめています。

  Web サービスクライアントを実装する場合は，はじめにサービスクラスのインスタンスを生成します。

- **ポート**

  WSDL のポート（wsdl:port 要素）に対応するインスタンスで，インタフェースはサービスエンドポイントインタフェース（SEI）です。リモートにある接続先 Web サービスのプロキシのように動作するため，Web サービスクライアントがこのポートのメソッドを呼び出すことで，Web サービスのオペレーションを透過的に呼び出せます。

Web サービスクライアントの実装で Web サービスのオペレーションを呼び出す手順は，次のとおりです。

1. サービスクラスをインスタンス化する
2. ポートを取得する
3. ポートのメソッドを呼び出す

ここでは，「5.1　開発例の構成（SEI 起点）」に構成を示す Web サービス（足し算をする Web サービス）を呼び出す Web サービスクライアントの実装例を示します。

Web サービスクライアントの実装に使用するクラスおよびメソッドを次の表に示します。必要に応じて，「5.5.1　サービスクラスを生成する」に記載されたサービスクラスの生成物の内容を参照してください。

表 3-6　Web サービスクライアントの実装例で使用するクラスおよびメソッド

項番	種別	クラスおよびメソッド
1	サービスクラス	AddNumbersImplService
2	SEI	AddNumbersImpl
3	SEI のメソッド	int add(int, int)
4	クライアントのメインクラス	TestClient

(a)　サービスクラスをインスタンス化する

デフォルトのコンストラクタを使用してサービスクラスのオブジェクトを生成する例を次に示します。

```
// サービスクラスをインスタンス化する
AddNumbersImplService service = new AddNumbersImplService();
```

この場合，cjwsimport コマンドの引数または-wsdllocation オプションに指定した WSDL が参照されます。

cjwsimport コマンドの実行時に-wsdllocation オプションを指定しなかった場合の三つの例を次に示します。

実行例 1

```
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" D:¥dev¥development.wsdl
```

D:¥dev¥development.wsdl は，Web サービスクライアント実行時にも存在し，参照できる状態である必要があります。

実行例 2

```
> D:
> cd D:¥dev¥
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" relative¥development.wsdl
```

D:¥dev¥relative¥development.wsdl は，Web サービスクライアント実行時にも存在し，参照できる状態である必要があります。

実行例 3

```
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" http://sample.com/fromjava/AddNumbersImplS
ervice?wsdl
```

http://sample.com/fromjava/AddNumbersImplService?wsdl は，Web サービスクライアント実行時にも存在し，参照できる状態である必要があります。

cjwsimport コマンドの実行時に-wsdllocation オプションを指定した場合の二つの例を次に示します。

実行例 1

```
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" -wsdllocation file:/home/wsdl4runtime/mast
er.wsdl D:¥dev¥development.wsdl
```

file:/home/wsdl4runtime/master.wsdl は，Web サービスクライアント実行時にも存在し，参照できる状態である必要があります。D:¥dev¥development.wsdl は，Web サービスクライアント開発時に実装に必要な Java コードの生成に使用されるだけです。そのため，D:¥dev¥development.wsdl は実行時には存在しない，または参照できない状態でも問題ありません。

実行例 2

```
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" -wsdllocation ./wsdl4runtime/master.wsdl D
:¥dev¥development.wsdl
```

Web サービスクライアント実行時のカレントディレクトリを<実行時カレントディレクトリ>とした場合，<実行時カレントディレクトリ>/wsdl4runtime/master.wsdl が存在し，参照できる状態である必要があります。

なお，デフォルトのコンストラクタではなく，java.net.URL オブジェクトおよび javax.xml.namespace.QName オブジェクトをパラメタに持つコンストラクタを使用する場合は，次の例のようになります。

```
// サービスクラスをインスタンス化する
java.net.URL wsdlLocation = new java.io.File( "./wsdl4runtime/master.wsdl" ).toURL();
javax.xml.namespace.QName serviceName =
  new javax.xml.namespace.QName( "http:/sample.com/", "AddNumbersImplService");
```

```
AddNumbersImplService service =
  new AddNumbersImplService( wsdlLocation, serviceName );
```

java.net.URL オブジェクトで指定した URL の WSDL が参照されるため，cjwsimport コマンドの引数や-wsdllocation オプションで指定した WSDL は，Web サービスクライアント実行時には存在しない，または参照できない状態でも問題ありません。ただし，Web サービスクライアント実行時のカレントディレクトリを<実行時カレントディレクトリ>とした場合は，<実行時カレントディレクトリ>/wsdl4runtime/master.wsdl が存在し，参照できる状態である必要があります。

(b) ポートを取得する

インスタンス化したサービスクラスからポートを取得する例を次に示します。

```
// ポートを取得する
AddNumbersImpl port = service.getAddNumbersImplPort();
```

(c) ポートのメソッドを呼び出す

インスタンス化したサービスクラスから取得したポートのメソッドを呼び出す例を次に示します。

```
// ポートのメソッドを呼び出す
int returnValue = port.add(205, 103)
```

この例では，ポートのメソッドの引数に二つの値を渡すと，Web サービスで足し算の処理が行われます。戻り値として足し算の演算結果が返されます。

接続する Web サービスの URL（エンドポイントアドレス）は，デフォルトでは，WSDL のポート（wsdl:port 要素）が持つアドレス情報（soap:address 子要素の location 属性）が利用されます。ただし，メッセージコンテキストの javax.xml.ws.service.endpoint.address プロパティを利用すると，エンドポイントアドレスを動的に変更できます。

エンドポイントアドレスを動的に変更しない場合，参照される WSDL の soap:address 子要素の location 属性に，Web サービスクライアントからアクセスできる URL が記述されているかどうか確認してください。

エンドポイントアドレスを動的に変更する場合，ポートのメソッドを呼び出す前に要求コンテキストを取得し，javax.xml.ws.service.endpoint.address プロパティの値を変更します。例を次に示します。

```
// 要求コンテキストを取得する
java.util.Map<String, Object> context =
  ( ( javax.xml.ws.BindingProvider )port ).getRequestContext();

// エンドポイントアドレスを変更する
// (javax.xml.ws.BindingProvider.ENDPOINT_ADDRESS_PROPERTYは
//  "javax.xml.ws.service.endpoint.address"を定義した定数)
context.put( javax.xml.ws.BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
   "http://other.remote.org/fromjava/AddNumbersImplService" );
```

```
// ポートのメソッドを呼び出す
int returnValue = port.add(205, 103)
```

注意事項

サービスクラスの生成やポートの取得には処理コストが掛かるので，一度生成，取得したサービスクラスとポートは再利用することをお勧めします。ポートの Web メソッドを複数回呼び出す場合に，サービスクラスを複数回生成したり，ポートを複数回取得したりする必要はありません。ただし，複数スレッドでポートを共有する場合，共有するポートの要求コンテキストのプロパティに対する設定は，複数スレッドが動作する前に実行してください。複数スレッドの動作中にプロパティに対する設定を行うと，通信が失敗したり，不正な SOAP メッセージが送信されたりすることがあります。なお，ポートは要求コンテキスト以外，スレッドセーフです。

ポートのメソッドを複数回呼び出す例を次に示します。

```
// サービスクラスをインスタンス化する
AddNumbersImplService service = new AddNumbersImplService();
// ポートを取得する
AddNumbersImpl port = service.getAddNumbersImplPort();

// ポートのメソッドを複数回呼び出す
for (int i = 0; i < 10; i++) {
      int returnValue = port.add(i, i);
}
```

(6) Java アプリケーションの Web サービスクライアントの実装例

Java アプリケーションの Web サービスクライアントから，サービスクラスを使用して Web サービスを呼び出す処理を実装します。

ここでは，「5.1 開発例の構成（SEI 起点）」に構成を示す Web サービス（足し算をする Web サービス）を呼び出す Web サービスクライアントの実装例を示します。Web サービスクライアントの実装に使用するクラスおよびメソッドを次の表に示します。必要に応じて，「5.5.1 サービスクラスを生成する」に記載されたサービスクラスの生成物の内容を参照してください。

表 3-7 Web サービスクライアントの実装例で使用するクラスおよびメソッド（Java アプリケーションの Web サービスクライアントの場合）

項番	種別	クラスおよびメソッド
1	サービスクラス	AddNumbersImplService
2	ポート	AddNumbersImpl
3	ポートのメソッド	int add(int, int)
4	Web サービスクライアントのクライアントのメインクラス	TestClient

Java アプリケーションの実装例を次に示します。

```
package com.example.sample.client;
```

```
import com.example.sample.AddNumbersImplTestJaxWs;
import com.example.sample.AddNumbersImplTestJaxWsService;
import com.sample.AddNumbersFault_Exception;

// Sample implementation of web service's client
public class TestClient {
    public static void main( String[] args ) {
        try {
            // サービスクラスをインスタンス化する
            AddNumbersImplTestJaxWsService service = new AddNumbersImplTestJaxWsService();
            // ポートを取得する
            AddNumbersImplTestJaxWs port = service.getAddNumbersImplPortTestJaxWs();

            // ポートのメソッドを呼び出す
            port.jaxWsTest1( ... );
            int number1 = 205;
            int number2 = 103;
            int returnValue = port.add(number1, number2);

            // 結果を表示する
            System.out.println( "[RESULT] " + number1 + " + " + number2 + " = " + returnValu
e );
        }
        catch( Exception e ){
            // 例外処理(ここでは単にスタックトレースを出力)
            e.printStackTrace();
        }
    }
}
```

プログラムの実行結果を次に示します。

```
[RESULT] 205 + 103 = 308
```

(7)　サーブレットの Web サービスクライアントの実装例

サーブレット形態の Web サービスクライアントから，サービスクラスを使用して Web サービスを呼び出す処理を実装します。

ここでは，「5.1　開発例の構成（SEI 起点）」に構成を示す Web サービス（足し算をする Web サービス）を呼び出す Web サービスクライアントの実装例を示します。Web サービスクライアントの実装に使用するクラスおよびメソッドを次の表に示します。必要に応じて，「5.5.1　サービスクラスを生成する」に記載されたサービスクラスの生成物の内容を参照してください。

表 3-8　Web サービスクライアントの実装例で使用するクラスおよびメソッド（サーブレットから呼び出す場合）

項番	種別	クラスおよびメソッド
1	サービスクラス	AddNumbersImplService
2	ポート	AddNumbersImpl

項番	種別	クラスおよびメソッド
3	ポートのメソッド	int add(int, int)
4	Web サービスクライアントとなるサーブレット実装クラス	TestClient

(a) ポートのインジェクション

サーブレットから Web サービスを呼び出す実装では，ポート型のフィールドに javax.xml.ws.WebServiceRef アノテーションを指定して，サーブレットインスタンスの生成時にポートのインジェクションを実行します。

インジェクションの例を次に示します。

```
...
public class TestClient extends HttpServlet {

    // ポート(AddNumbersImpl)型のフィールドに，ポートをインジェクトする
    @WebServiceRef(AddNumbersImplService.class)
    AddNumbersImpl port;

    @Override
    public void init() {
    }
    ...
}
```

(b) ポートのメソッド呼び出しによる Web サービスのオペレーション実行

サーブレットのフィールドとして生成したポートを使って，メソッドを呼び出します。

メソッド呼び出しの例を次に示します。

```
...
public class TestClient extends HttpServlet {

    // ポート(AddNumbersImpl)型のフィールドに，ポートをインジェクトする
    @WebServiceRef(AddNumbersImplService.class)
    AddNumbersImpl port;
    ...
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        ...
            int number1 = ...; // requestオブジェクトから取得した値を代入
            int number2 = ...; // requestオブジェクトから取得した値を代入
            // ポートのメソッドを呼び出す
            int returnValue = port.add(number1, number2);
            ...
    }
}
```

サーブレットから Web サービスを呼び出す場合の実装例の全体を次に示します。

```
package com.sample.client;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.sample.AddNumbersFault_Exception;
import com.sample.AddNumbersImpl;
import com.sample.AddNumbersImplService;

public class TestClient extends HttpServlet {

    // ポート(AddNumbersImpl)型のフィールドに, ポートをインジェクトする
    @WebServiceRef(AddNumbersImplService.class)
    AddNumbersImpl port;

    @Override
    public void init() {
    }

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        PrintWriter out = response.getWriter();

        try {
            // Invoke a method of the target web service.
            int number1 = ...; // requestオブジェクトから取得した値を代入
            int number2 = ...; // requestオブジェクトから取得した値を代入
            // ポートのメソッドを呼び出す
            int returnValue = port.add(number1, number2);
            // 結果を表示する
            out.println("<html><body>");
            out.println("<h1>RESULT</h1>");
            out.println(number1 + " + " + number2 + " = " + returnValue);
            out.println("</body></html>");
        } catch(AddNumbersFault_Exception e) {
            // 例外処理(ここでは単に例外の詳細メッセージを出力)
            out.println("<html><body>");
            out.println("<h1>" + e.getMessage() + "</h1>");
            out.println("</body></html>");
        }
    }
}
```

プログラムの実行結果を次に示します。ブラウザなどからサーブレットに接続すると実行結果が表示されます。

```
RESULT
205 + 103 = 308
```

(c) 注意事項

Web サービスクライアント（サーブレット）と接続先の Web サービスが，同じ J2EE サーバ上にデプロイされている環境では，次の場合，J2EE サーバの起動時に例外が発生します。

- init メソッドまたは javax.annotation.PostConstruct アノテーションを指定したメソッドの中で，Web サービスから WSDL を取得してサービスクラスを生成するときに，Web サービスクライアントが含まれる WAR ファイルの web.xml に load-on-startup 要素を指定した場合
- javax.xml.ws.WebServiceRef アノテーションを利用して，Web サービスから WSDL を取得してサービスクラスやポートをインジェクトする場合

例外が発生する場合の対策方法を次に示します。

- J2EE サーバを停止する前に，Web サービスクライアントが含まれる Web アプリケーションを停止してください。J2EE サーバを再起動したあとに，Web サービスクライアントが含まれる Web アプリケーションを開始してください。
- Web サービスと Web サービスクライアントを別の J2EE サーバにデプロイして，Web サービスをデプロイした J2EE サーバを起動したあとに Web サービスクライアントをデプロイした J2EE サーバを起動してください。
- カタログ機能を使用して，サービスクラスの生成時にローカルに格納した WSDL 文書を参照するように設定してください。
- init メソッドまたは javax.annotation.PostConstruct アノテーションを指定したメソッドで，サービスクラスを生成する場合は，URL をパラメタに取るコンストラクタにローカルの WSDL 文書を指定してサービスクラスを生成してください。または，web.xml に load-on-startup 要素を指定しないでください。
- javax.xml.ws.WebServiceRef アノテーションを利用する場合は，wsdlLocation 要素に相対パスまたは絶対パスで，ローカルに格納した WSDL 文書を指定してください。

3.6.2 ディスパッチベースの実装例

Application Server でサポートしている javax.xml.ws.Dispatch インタフェース，JAX-WS 2.2 仕様，JAXB 2.2 仕様，および SAAJ 1.3 仕様などの API を使用して開発してください。

(1) ディスパッチベースの Web サービスクライアントの実装例

ディスパッチベースの Web サービスクライアントの実装例を次に示します。

```
package com.example.sample.client;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPBodyElement;
```

```
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class TestClient {
    public static void main( String[] args ) {
        QName port = new QName( "http://sample.com", "AddNumbersImplPort" );
        SOAPBody soapBody = null;

        // サービスを生成する
        Service service = Service.create(
            new QName("http://sample.com", "UserInfoPort") );

        // サービスにポートを追加する
        service.addPort( port, SOAPBinding.SOAP11HTTP_BINDING,
            "http://localhost:80/dispatch_provider/UserInfoService" );

        // ディスパッチを生成する
        Dispatch<SOAPMessage> dispatch = service.createDispatch(
            port, SOAPMessage.class, Service.Mode.MESSAGE );

        SOAPMessage request = null;
        try{
            // 要求メッセージを SAAJ 1.3仕様のAPIを使用して生成していく
            request = MessageFactory.newInstance().createMessage();
            SOAPBody reqSoapBody = request.getSOAPBody();
            SOAPElement soapElement = null;
            // SOAPボディに要素を追加
            SOAPBodyElement requestRoot= reqSoapBody.addBodyElement(
                new QName( ... ) );
            soapElement = requestRoot.addChildElement(
                new QName( ... ) );
            soapElement.addTextNode( ... );
            // 添付ファイルを追加する
            File attachment = new File( ... );
            FileDataSource fds = new FileDataSource( attachment );
            AttachmentPart apPart = request.createAttachmentPart( new DataHandler( fds ) );
            request.addAttachmentPart(apPart);
        }
        catch( SOAPException e ){
            // 例外処理
        }
        // 作成した要求メッセージを指定し，ディスパッチを利用してWebサービスを呼び出す
        SOAPMessage response = dispatch.invoke( request );

        try{
            // 応答メッセージについて必要な処理を行う
            SOAPBody resSoapBody = response.getSOAPBody();
            ...
        }
        catch( SOAPException e ){
            // 例外処理
        }
```

```
    }
}
```

プロバイダ実装クラスから送信された SOAP フォルトを Web サービスクライアントの実装で利用する場合は，try-catch ブロック内で invoke()メソッドを呼び出し，javax.xml.ws.soap.SOAPFaultException 例外を取得します。SOAP フォルトも，javax.xml.ws.soap.SOAPFaultException 例外から取得できます。実装例を次に示します。

```
package com.example.sample.client;

import javax.xml.namespace.QName;
...
import javax.xml.ws.soap.SOAPBinding;
import javax.xml.ws.soap.SOAPFaultException;

public class TestClient {
    public static void main( String[] args ) {
...
        try{
            // 要求メッセージを SAAJ 1.3仕様のAPIを使用して生成していく
            ...
        }
        catch( SOAPException e ){
            // 例外処理
        }

        SOAPMessage response = null;
        try{
            // 作成した要求メッセージを指定し，ディスパッチを利用してWebサービスを呼び出す
            response = dispatch.invoke( request );
        }
        catch( SOAPFaultException e ){
            // SOAPフォルトを取得する処理
            SOAPFault fault = e.getFault();
            // 取得した SOAPフォルトに対して必要な処理を行う
            String faultCode = fault.getFaultCode();
            ...
        }

        try{
            // 応答メッセージについて必要な処理を行う
            ...
        }
        catch( SOAPException e ){
            e.printStackTrace();
        }
    }
}
```

(2)　参照されるエンドポイントアドレス

接続する Web サービスの URL（エンドポイントアドレス）は，メッセージコンテキストの javax.xml.ws.service.endpoint.address プロパティで指定および変更できます。エンドポイントアドレスを指定・変更する例については，「3.6.1(5)(c)　ポートのメソッドを呼び出す」を参照してください。

(3)　サービスクラスおよびディスパッチの再利用

サービスクラスおよびディスパッチの生成には処理コストが掛かるので，一度生成したサービスクラスおよびディスパッチは再利用することをお勧めします。ポートを追加したりディスパッチを生成したりするためにサービスクラスを複数回生成する必要はありません。同様に，ディスパッチの invoke メソッドを複数回呼び出すためにディスパッチを複数回取得する必要はありません。ただし，複数スレッドでディスパッチを共有する場合，共有するディスパッチの要求コンテキストのプロパティに対する設定は，複数スレッドが動作する前に実行してください。複数スレッドの動作中にプロパティに対する設定を行うと，通信が失敗したり，不正な SOAP メッセージが送信されたりすることがあります。なお，ディスパッチは要求コンテキスト以外，スレッドセーフです。

Web サービスクライアントをサーブレットや EJB などで実装する場合は，それぞれの初期化メソッドでサービスクラスおよびディスパッチを取得し，再利用することをお勧めします。ディスパッチの要求コンテキストのプロパティに対する変更も，それぞれの初期化メソッドで実行してください。

3.6.3　JAX-WS API を使用する場合の実装例

Application Server の JAX-WS 機能がサポートしている JAX-WS API を利用して，Web サービスクライアントを実装できます。JAX-WS API のサポート範囲については，「19.2　API のサポート範囲」を参照してください。

(1)　JAX-WS API を使用する場合の Web サービスクライアントの実装例

JAX-WS API を使用した Web サービスクライアントの実装例を示します。

```
package com.example.sample.client;

import com.example.sample.TestJaxWs;
import com.example.sample.TestJaxWsService;

import java.net.URL;
import java.net.MalformedURLException;
import java.util.Iterator;
import java.util.Map;
import javax.xml.namespace.QName;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.Service;
import javax.xml.ws.WebServiceException;

public class TestClient {
    public static void main( String[] args ) {
```

```
        // WSDLのURLとサービス名を設定
        URL url = null;
        try {
            url = new URL("http://localhost:8085/fromwsdl/test?wsdl");
        } catch (MalformedURLException e) {
            // 例外処理
        }
        QName serviceName =
            new QName("http://example.com/sample", "TestJaxWsService");

        // Serviceインスタンスの生成
        Service service = Service.create(url, serviceName);
        System.out.println(service.getWSDLDocumentLocation());

        QName portName = null;

        // ポート名のリストを表示
        Iterator it = service.getPorts();
        while(it.hasNext()) {
            portName = (QName)it.next();
            System.out.println(portName);
        }


        // ポートの取得
        TestJaxWs port = (TestJaxWs)service.getPort(TestJaxWs.class);

        // 送信コンテキストの取得
        Map<java.lang.String, java.lang.Object>context =
((BindingProvider)port).getRequestContext();
        System.out.println(context.entrySet());

        // サービスのメソッド呼び出し
        try {
            port.jaxWsTest1( "TEST", 123);
        } catch (WebServiceException e) {
            // 例外処理
        }
    }
}
```

(2)　javax.xml.ws.Service オブジェクトおよびポートの再利用

javax.xml.ws.Service オブジェクトの生成には処理コストが掛かるので，一度生成した javax.xml.ws.Service オブジェクトは再利用することをお勧めします。ポートを取得するために javax.xml.ws.Service オブジェクトを複数回生成する必要はありません。

同様に，ポートの取得にも処理コストが掛かるので，一度取得したポートは再利用することをお勧めします。ポートの Web メソッドを複数回呼び出すためにポートを複数回取得する必要はありません。ただし，複数スレッドでポートを共有する場合，共有するポートの要求コンテキストのプロパティに対する設定は，複数スレッドが動作する前に実行してください。複数スレッドの動作中にプロパティに対する設定を行うと，通信が失敗したり，不正な SOAP メッセージが送信されたりすることがあります。なお，ポートは要求コンテキスト以外，スレッドセーフです。

Web サービスクライアントをサーブレットや EJB などで実装する場合は，それぞれの初期化メソッドで javax.xml.ws.Service オブジェクトの生成，またはポートの取得を実施し，再利用してください。ポートの要求コンテキストのプロパティに対する変更も，それぞれの初期化メソッドで実行してください。

3.6.4　注意事項

Web サービスクライアント実装時の注意事項について説明します。

(1)　オブジェクトの再利用

サービスクラス，ポート，およびディスパッチの生成には処理コストが掛かるので，インジェクションを利用するか（スタブベースの場合だけ），再利用することをお勧めします。オブジェクトの再利用については，それぞれ次の個所を参照してください。

- スタブベースの Web サービスクライアントの場合
  「3.6.1(3)　サービスクラスの生成およびポートの取得」
- ディスパッチベースの Web サービスクライアントの場合
  「3.6.2(3)　サービスクラスおよびディスパッチの再利用」
- API ベースの Web サービスクライアントの場合
  「3.6.3(2)　javax.xml.ws.Service オブジェクトおよびポートの再利用」

(2)　プロキシ・SSL 接続・ベーシック認証の設定

必要に応じて，Web サービスクライアントの実行環境に，プロキシ，SSL 接続，およびベーシック認証の設定を行ってください。詳細については，それぞれ次の個所を参照してください。

- プロキシの設定
  「10.10　プロキシサーバ経由の接続」
- SSL 接続の設定
  「10.11　SSL プロトコルによる接続」
- ベーシック認証の設定
  「10.12　ベーシック認証による接続」

(3)　Windows 環境での注意事項

Web サービスクライアントから大量のリクエストを送信するような環境では，次の例外が記録されることがあります。

java.net.BindException: Address already in use: connect [errno=10048, syscall=select]

例えば，サーブレットとして実装した Web サービスクライアントに対して大量のリクエストが到着すると，例外が発生します。

このような場合は，次に示すどちらか，または両方の値を見直してください。

- OS で使用できるポート番号の範囲を広げる
- TIME_WAIT の継続時間を短くする

例えば，レジストリの MaxUserPort や TcpTimedWaitDelay の設定を見直します。ただし，OS のバージョンやエディション，セキュリティ更新プログラムの適用状況によって，仕様が異なるため，詳細については各 OS のドキュメントを参照してください。また，これらの設定は OS 全体に影響が及ぶため，注意が必要です。

(4) 通信失敗時の Java 例外

Web サービスとの通信が失敗した場合，次の例外が記録されることがあります。

java.net.ConnectException: ["Connection refused: connect"または"接続を拒否されました"]

このメッセージは，OS や環境によって出力内容が異なる場合があります。

3.6.5 アドレッシング機能を使用した Web サービスにアクセスする場合の注意事項

アドレッシング機能が有効な Web サービスにアクセスする場合は，スタブベースの Web サービスクライアントを使用します。ディスパッチベースの Web サービスクライアントは使用できないので，注意してください。

4 WSDL を起点とした開発の例

この章では，WSDL を起点とした Web サービスを開発する場合の例を説明します。

4.1　開発例の構成（WSDL 起点）

この章で説明する開発例では，WSDL を起点とした Web サービスを開発します。

開発する Web サービスの構成を次の表に示します。

表 4-1　Web サービスの構成（WSDL 起点）

項番	項目		値
1	デプロイする J2EE サーバの名称		jaxwsserver
2	Web サーバのホスト名とポート番号		webhost:8085
3	ネーミングサーバの URL		corbaname::testserver:900
4	コンテキストルート		fromwsdl
5	スタイル		document/literal/wrapped
6	名前空間 URI		http://example.com/sample
7	ポートタイプ	個数	1
8		ローカル名	TestJaxWs
9	オペレーション	個数	1
10		ローカル名	jaxWsTest1
11	サービス	個数	1
12		ローカル名	TestJaxWsService
13	ポート	個数	1
14		ローカル名	testJaxWs
15	WSDL のファイル名		input.wsdl

Web サービス開発時のカレントディレクトリの構成を次の表に示します。

表 4-2　カレントディレクトリの構成（WSDL 起点）

ディレクトリ	説明
c:¥temp¥jaxws¥works¥fromwsdl	カレントディレクトリです。
┣server¥	Web サービスの開発で使用します。
┃┣META-INF¥	EAR ファイルの META-INF ディレクトリに対応します。
┃┗application.xml	「4.3.6　application.xml を作成する」で作成します。
┃┣src¥	Web サービスのソースファイル（*.java）を格納します。「4.3.2　SEI を生成する」および「4.3.4　Web サービス実装クラスをコンパイルする」で使用します。

ディレクトリ	説明
│	
├ WEB-INF¥	WAR ファイルの WEB-INF ディレクトリに対応します。
│ ├ web.xml	「4.3.5　web.xml を作成する」で作成します。
│ ├ classes¥	コンパイルしたクラスファイル（*.class）を格納します。
│ └ wsdl¥	作成した wsdl を格納します。
├ temporary¥	Java で記述，変換したものを基に WSDL を作成する場合に一時ファイルを格納します。このディレクトリの作成は任意です。
│ ├ src¥	
│ └ classes¥	
├ fromwsdl.ear	「4.3.7　EAR ファイルを作成する」で作成します。
└ fromwsdl.war	
└ client¥	Web サービスクライアントの開発で使用します。
├ src¥	Web サービスクライアントのソースファイル（*.java）を格納します。「4.5.1　サービスクラスを生成する」および「4.5.2　Web サービスクライアントの実装クラスを作成する」で使用します。
├ classes¥	コンパイルしたクラスファイル（*.class）を格納します。「4.5.3　Web サービスクライアントの実装クラスをコンパイルする」で使用します。
├ usrconf.cfg	「4.6.1　Java アプリケーション用オプション定義ファイルを作成する」で作成します。
└ usrconf.properties	「4.6.2　Java アプリケーション用ユーザプロパティファイルを作成する」で作成します。

カレントディレクトリのパスは，開発する環境に合わせて変更してください。

なお，以降の説明では，この表に示すディレクトリおよびファイル名を使用します。コマンド実行例や Java ソースなどで背景色付きの太字になっている部分は，この例で使用する指定値や生成される値を示します。構築する環境に合わせて読み替えてください。

また，この章で説明する開発例では，Web サービスと Web サービスクライアントを同じ環境で開発しますが，別の環境で開発することもできます。別の環境で開発する場合は，それぞれの環境に合わせて，カレントディレクトリのパスを読み替えてください。

4.2　開発例の流れ（WSDL 起点）

この章で説明する開発例では，次の流れで開発および実行します。

Web サービスの開発

1. WSDL ファイルを作成する（4.3.1）
2. cjwsimport コマンドを実行し，SEI を生成する（4.3.2）
3. Web サービス実装クラスを作成する（4.3.3）
4. Web サービス実装クラスをコンパイルする（4.3.4）
5. web.xml を作成する（4.3.5）
6. application.xml を作成する（4.3.6）
7. EAR ファイルを作成する（4.3.7）

デプロイと開始

1. EAR ファイルをデプロイする（4.4.1）
2. Web サービスを開始する（4.4.2）

Web サービスクライアントの開発

1. cjwsimport コマンドを実行し，サービスクラスを生成する（4.5.1）
2. Web サービスクライアントの実装クラスを作成する（4.5.2）
3. Web サービスクライアントの実装クラスをコンパイルする（4.5.3）

Web サービスの実行

1. Java アプリケーション用オプション定義ファイルを作成する（4.6.1）
2. Java アプリケーション用ユーザプロパティファイルを作成する（4.6.2）
3. Web サービスクライアントを実行する（4.6.3）

4.3　Web サービスの開発例（WSDL 起点）

ここでは，WSDL を起点とした場合の Web サービスの開発例を説明します。

4.3.1　WSDL ファイルを作成する

WSDL ファイルを作成し，Web サービスのメタ情報を定義します。WSDL 定義は，次の仕様のサポート範囲内で定義してください。

- WSDL 1.1 仕様
  サポート範囲については，「20.1　WSDL 1.1 仕様のサポート範囲」を参照してください。
- XML Schema 仕様
  サポート範囲については，マニュアル「XML Processor ユーザーズガイド」を参照してください。
- WS-I Basic Profile 1.1

WSDL ファイルの作成方法には，新規に作成する方法と，Java ソースを変換したものを利用して作成する方法の 2 種類があります。

(1)　新規に WSDL ファイルを作成する

ここでは，WSDL ファイル（input.wsdl）を作成します。作成した WSDL ファイルは，UTF-8 形式で c:¥temp¥jaxws¥works¥fromwsdl¥server¥WEB-INF¥wsdl¥ディレクトリに保存してください。

SOAP 1.1 の場合の作成例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="TestJaxWsService"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://example.com/sample"
  targetNamespace="http://example.com/sample">

  <wsdl:types>
    <xsd:schema targetNamespace="http://example.com/sample">
      <!-- 要求メッセージの wrapper要素 -->
      <xsd:element name="jaxWsTest1" type="tns:jaxWsTest1"/>

      <!-- 応答メッセージの wrapper要素 -->
      <xsd:element name="jaxWsTest1Response" type="tns:jaxWsTest1Response"/>

      <!-- フォルトメッセージの wrapper要素 -->
      <xsd:element name="UserDefinedFault" type="tns:UserDefinedFault"/>

      <!-- 要求メッセージの wrapper要素が参照する型 -->
      <xsd:complexType name="jaxWsTest1">
        <xsd:sequence>
```

```
        <xsd:element name="information" type="xsd:string"/>
        <xsd:element name="count" type="xsd:int"/>
      </xsd:sequence>
    </xsd:complexType>

    <!-- 応答メッセージの wrapper要素が参照する型 -->
    <xsd:complexType name="jaxWsTest1Response">
      <xsd:sequence>
        <xsd:element name="return" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>

    <!-- フォルトメッセージの wrapper要素が参照する型 -->
    <xsd:complexType name="UserDefinedFault">
      <xsd:sequence>
        <xsd:element name="additionalInfo" type="xsd:int"/>
        <xsd:element name="detail" type="xsd:string"/>
        <xsd:element name="message" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
</wsdl:types>

<!-- 要求メッセージ -->
<wsdl:message name="jaxWsTest1Request">
  <wsdl:part name="inputParameters" element="tns:jaxWsTest1"/>
</wsdl:message>

<!-- 応答メッセージ -->
<wsdl:message name="jaxWsTest1Response">
  <wsdl:part name="outputParameters" element="tns:jaxWsTest1Response"/>
</wsdl:message>

<!-- フォルトメッセージ -->
<wsdl:message name="UserDefinedException">
  <wsdl:part name="fault" element="tns:UserDefinedFault"/>
</wsdl:message>

<!-- ポートタイプ -->
<wsdl:portType name="TestJaxWs">
  <!-- オペレーション -->
  <wsdl:operation name="jaxWsTest1">
    <wsdl:input message="tns:jaxWsTest1Request"/>
    <wsdl:output message="tns:jaxWsTest1Response"/>
    <wsdl:fault name="UserDefinedFault"
      message="tns:UserDefinedException"/>
  </wsdl:operation>
</wsdl:portType>

<!-- バインディング(SOAP 1.1/HTTPバインディング) -->
<wsdl:binding name="testJaxWsBinding" type="tns:TestJaxWs">
  <!-- document/literal/wrapped -->
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <!-- オペレーション -->
  <wsdl:operation name="jaxWsTest1">
    <soap:operation/>
    <wsdl:input>
      <soap:body use="literal"/>
```

```
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="UserDefinedFault">
        <soap:fault name="UserDefinedFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>

  <!-- サービス -->
  <wsdl:service name="TestJaxWsService">
    <!-- ポート -->
    <wsdl:port name="testJaxWs" binding="tns:testJaxWsBinding">
      <soap:address location="http://webhost:8085/fromwsdl/TestJaxWsService"/>
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

SOAP 1.2 の場合の作成例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="TestJaxWsService"
  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://example.com/sample"
  targetNamespace="http://example.com/sample">

  <wsdl:types>
    <xsd:schema targetNamespace="http://example.com/sample">
      <!-- 要求メッセージの wrapper要素 -->
      <xsd:element name="jaxWsTest1" type="tns:jaxWsTest1"/>

      <!-- 応答メッセージの wrapper要素 -->
      <xsd:element name="jaxWsTest1Response" type="tns:jaxWsTest1Response"/>

      <!-- フォルトメッセージの wrapper要素 -->
      <xsd:element name="UserDefinedFault" type="tns:UserDefinedFault"/>

      <!-- 要求メッセージの wrapper要素が参照する型 -->
      <xsd:complexType name="jaxWsTest1">
        <xsd:sequence>
          <xsd:element name="information" type="xsd:string"/>
          <xsd:element name="count" type="xsd:int"/>
        </xsd:sequence>
      </xsd:complexType>

      <!-- 応答メッセージの wrapper要素が参照する型 -->
      <xsd:complexType name="jaxWsTest1Response">
        <xsd:sequence>
          <xsd:element name="return" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>

      <!-- フォルトメッセージの wrapper要素が参照する型 -->
```

```
      <xsd:complexType name="UserDefinedFault">
        <xsd:sequence>
          <xsd:element name="additionalInfo" type="xsd:int"/>
          <xsd:element name="detail" type="xsd:string"/>
          <xsd:element name="message" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </wsdl:types>

  <!-- 要求メッセージ -->
  <wsdl:message name="jaxWsTest1Request">
    <wsdl:part name="inputParameters" element="tns:jaxWsTest1"/>
  </wsdl:message>

  <!-- 応答メッセージ -->
  <wsdl:message name="jaxWsTest1Response">
    <wsdl:part name="outputParameters" element="tns:jaxWsTest1Response"/>
  </wsdl:message>

  <!-- フォルトメッセージ -->
  <wsdl:message name="UserDefinedException">
    <wsdl:part name="fault" element="tns:UserDefinedFault"/>
  </wsdl:message>

  <!-- ポートタイプ -->
  <wsdl:portType name="TestJaxWs">
    <!-- オペレーション -->
    <wsdl:operation name="jaxWsTest1">
      <wsdl:input message="tns:jaxWsTest1Request"/>
      <wsdl:output message="tns:jaxWsTest1Response"/>
      <wsdl:fault name="UserDefinedFault"
        message="tns:UserDefinedException"/>
    </wsdl:operation>
  </wsdl:portType>

  <!-- バインディング(SOAP 1.2/HTTPバインディング) -->
  <wsdl:binding name="testJaxWsBinding" type="tns:TestJaxWs">
    <!-- document/literal/wrapped -->
    <soap12:binding style="document" transport="http://www.w3.org/2003/05/soap/bindings/HTTP
/"/>
    <!-- オペレーション -->
    <wsdl:operation name="jaxWsTest1">
      <soap12:operation/>
      <wsdl:input>
        <soap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="UserDefinedFault">
        <soap12:fault name="UserDefinedFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>

  <!-- サービス -->
  <wsdl:service name="TestJaxWsService">
```

```
    <!-- ポート -->
    <wsdl:port name="testJaxWs" binding="tns:testJaxWsBinding">
      <soap12:address location="http://webhost:8085/fromwsdl/TestJaxWsService"/>
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

(2) Java ソースを変換したものを基に WSDL ファイルを作成する

ここでは，WSDL 変換用に仮実装の Web サービス実装クラスと例外クラスを作成し，hwsgen コマンドの WSDL 生成機能を実行して，コンパイル済みの Java ソースから WSDL ファイルを作成します。作成したクラスは，javax.jws.WebService アノテーションでアノテートしてください。メソッドを実装する必要はありません。

仮実装の Web サービス実装クラスの例を次に示します。

```
package com.example.sample;

@javax.jws.WebService
public class TestJaxWsImpl {

    public String jaxWsTest1(String information, int count)
        throws UserDefinedException
    {
        // 実装不要
        return null;
    }

}
```

仮実装の例外クラスの例を次に示します。

```
package com.example.sample;

public class UserDefinedFault extends Exception{
    // 実装不要
    public int additionalInfo;
    public String detail;
    public String message;
}
```

作成した TestJaxWsImpl.java と UserDefinedFault.java を UTF-8 形式で c:¥temp¥jaxws¥works¥fromwsdl¥server¥temporary¥src¥com¥example¥sample¥ディレクトリに保存し，コンパイルします。コンパイルの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥fromwsdl¥server¥
> mkdir .¥temporary
> mkdir .¥temporary¥classes
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥ja
```

```
vaee¥1100¥lib¥javaee-api.jar" -d .¥temporary¥classes .¥temporary¥src¥com¥example¥sample¥Test
JaxWsImpl.java .¥temporary¥src¥com¥example¥sample¥UserDefinedFault.java
```

コンパイルが正常に終了すると，
c:¥temp¥jaxws¥works¥fromwsdl¥server¥temporary¥classes¥com¥example¥sample¥ディレクトリに TestJaxWsImpl.class と UserDefinedFault.class が生成されます。これらのクラスファイルを利用して，hwsgen コマンドの WSDL 生成機能で WSDL ファイルを作成します。

hwsgen コマンドの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥fromwsdl¥server¥
> mkdir .¥WEB-INF¥wsdl¥
> "%COSMINEXUS_HOME%¥jaxws¥bin¥hwsgen.bat" -r .¥WEB-INF¥wsdl -d .¥temporary¥classes -cp .¥te
mporary¥classes com.example.sample.TestJaxWsImpl
```

hwsgen コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥fromwsdl¥WEB-INF¥wsdl¥ディレクトリに TestJaxWsService.wsdl と TestJaxWsService_schema1.xsd が生成されます。
c:¥temp¥jaxws¥works¥fromwsdl¥temporary¥classes¥ディレクトリにあるクラスは削除してください。

生成された TestJaxWsService.wsdl と TestJaxWsService_schema1.xsd は，一部修正する必要があります。

TestJaxWsService.wsdl の修正例を次に示します。イタリック体になっている個所が，修正した個所です。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions targetNamespace="http://example.com/sample" name="TestJaxWsImplService" xmlns:t
ns="http://example.com/sample" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http
://schemas.xmlsoap.org/wsdl/soap/" xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <xsd:schema targetNamespace="http://example.com/sample">
      <xsd:include schemaLocation="TestJaxWsImplService_schema1.xsd"/>
    </xsd:schema>
  </types>
  <message name="jaxWsTest1">
    <part name="parameters" element="tns:jaxWsTest1"/>
  </message>
  <message name="jaxWsTest1Response">
    <part name="parameters" element="tns:jaxWsTest1Response"/>
  </message>
  <message name="UserDefinedFault">
    <part name="fault" element="tns:UserDefinedFault"/>
  </message>
  <portType name="TestJaxWs">
    <operation name="jaxWsTest1">
      <input message="tns:jaxWsTest1"/>
      <output message="tns:jaxWsTest1Response"/>
      <fault message="tns:UserDefinedFault" name="UserDefinedFault"/>
    </operation>
  </portType>
  <binding name="testJaxWsBinding" type="tns:TestJaxWs">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="jaxWsTest1">
      <soap:operation soapAction=""/>
```

```
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
      <fault name="UserDefinedFault">
        <soap:fault name="UserDefinedFault" use="literal"/>
      </fault>
    </operation>
  </binding>
  <service name="TestJaxWsService">
    <port name="testJaxWs" binding="tns:testJaxWsBinding">
      <soap:address location="http://webhost:8085/fromwsdl/TestJaxWsService"/>
    </port>
  </service>
</definitions>
```

TestJaxWsService_schema1.xsd の修正例を次に示します。イタリック体になっている個所が，修正した個所です。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" targetNamespace="http://example.com/sample" xmlns:tns="http://examp
le.com/sample" xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="UserDefinedFault" type="tns:UserDefinedFault"/>

  <xs:element name="jaxWsTest1" type="tns:jaxWsTest1"/>

  <xs:element name="jaxWsTest1Response" type="tns:jaxWsTest1Response"/>

  <xs:complexType name="jaxWsTest1">
    <xs:sequence>
      <xs:element name="arg0" type="xs:string" minOccurs="0"/>
      <xs:element name="arg1" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="jaxWsTest1Response">
    <xs:sequence>
      <xs:element name="return" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="UserDefinedFault">
    <xs:sequence>
      <xs:element name="message" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

修正した TestJaxWsService.wsdl は，名前を input.wsdl に変更して，
c:¥temp¥jaxws¥works¥fromwsdl¥server¥WEB-INF¥wsdl¥ディレクトリに保存してください。

4.3.2　SEI を生成する

cjwsimport コマンドを実行すると，SEI など，Web サービスの開発に必要な Java ソースが生成されます。cjwsimport コマンドについては，「14.1.1　cjwsimport コマンド」を参照してください。

cjwsimport コマンドの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥fromwsdl¥server¥
> mkdir src¥
> mkdir WEB-INF¥classes¥
> ”%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat” -generateService -s src -d WEB-INF¥classes WE
B-INF¥wsdl¥input.wsdl
```

cjwsimport コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥fromwsdl¥server¥src¥com¥example¥sample¥ディレクトリに，Java ソースが生成されます。com¥example¥sample¥（パッケージに対応するディレクトリパス）は，名前空間 URI の記述によって変わります。名前空間 URI とパッケージとのマッピングについては，「15.1.1　名前空間からパッケージ名へのマッピング」を参照してください。

生成物の一覧を次の表に示します。

表 4-3　SEI 生成時の生成物（WSDL 起点）

ファイル名	説明
JaxWsTest1.java	WSDL 定義の「要求メッセージの wrapper 要素が参照する型」に対応する JavaBean クラスです。
JaxWsTest1Response.java	WSDL 定義の「応答メッセージの wrapper 要素が参照する型」に対応する JavaBean クラスです。
ObjectFactory.java	JAXB 2.2 仕様の ObjectFactory クラスです。
package-info.java	package-info.java ファイルです。
TestJaxWs.java	TestJaxWs ポートタイプに対応する SEI です。
TestJaxWsImpl.java	TestJaxWs ポートタイプに対応するスケルトンクラスです。
UserDefinedFault.java	WSDL 定義の「フォルトメッセージの wrapper 要素が参照する型」に対応する JavaBean クラス（フォルト bean）です。
UserDefinedException.java	フォルト bean のラッパ例外クラスです。

ファイル名の JaxWsTest1，TestJaxWs，および TestJaxWsImpl は，オペレーションのローカル名，ポートタイプのローカル名，およびサービスのローカル名の記述によって変わります。オペレーションのローカル名，ポートタイプのローカル名，およびサービスのローカル名と Java ソースとのマッピングについては，「15.　WSDL から Java へのマッピング」を参照してください。

4.3.3　Web サービス実装クラスを作成する

スケルトンクラスに Web サービスの処理を追加し，Web サービス実装クラスを作成します。ここでは，受け取った要求メッセージの内容を，日付情報とともに応答メッセージとして返す処理を追加します。

Web サービス実装クラスの作成例を次に示します。

```
package com.example.sample;

import java.util.Calendar;
import javax.jws.WebService;


@WebService(endpointInterface = "com.example.sample.TestJaxWs", targetNamespace = "http://ex
ample.com/sample", serviceName = "TestJaxWsService", portName = "testJaxWs")
public class TestJaxWsImpl {

    public String jaxWsTest1(String information, int count)
        throws UserDefinedException
    {
        Calendar today = Calendar.getInstance();
        StringBuffer result = new StringBuffer( 256 );
        result.append( "We've got your #" );
        result.append( new Integer( count ) );
        result.append( " message ¥"" );
        result.append( information );
        result.append( "¥"! It's " );
        result.append( String.format( "%04d.%02d.%02d %02d:%02d:%02d", new Object[]{
            new Integer( today.get( Calendar.YEAR ) ),
            new Integer( today.get( Calendar.MONTH ) + 1 ),
            new Integer( today.get( Calendar.DAY_OF_MONTH ) ),
            new Integer( today.get( Calendar.HOUR_OF_DAY ) ),
            new Integer( today.get( Calendar.MINUTE ) ),
            new Integer( today.get( Calendar.SECOND ) ) } ) );
        result.append( " now. See ya!" );

        return result.toString();
    }

}
```

イタリック体になっている個所が，スケルトンに対して追加した実装です。

4.3.4　Web サービス実装クラスをコンパイルする

javac コマンドを使用して，作成した Web サービス実装クラスをコンパイルします。

コンパイルの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥fromwsdl¥server¥
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥ja
```

```
vaee¥1100¥lib¥javaee-api.jar;.¥WEB-INF¥classes" -d .¥WEB-INF¥classes src¥com¥example¥sample¥TestJaxWsImpl.java
```

javac コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥fromwsdl¥server¥WEB-INF¥classes¥com¥example¥sample¥ディレクトリの TestJaxWsImpl.class が上書きされます。

javac コマンドについては，JDK のドキュメントを参照してください。

4.3.5　web.xml を作成する

WAR ファイルの構成要素として必要な web.xml を作成します。

web.xml の作成例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <description>Sample web service &quot;fromwsdl&quot;</description>
  <display-name>Sample_web_service_fromwsdl</display-name>
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/TestJaxWsService</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

バージョン 2.5 の web.xml を作成する場合は，web-app 要素の version 属性を"2.5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"にしてください。

作成した web.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥fromwsdl¥server¥WEB-INF¥ディレクトリに保存します。web.xml の設定項目については，「3.4　web.xml の作成」を参照してください。

4.3.6　application.xml を作成する

EAR ファイルの構成要素として必要な application.xml を作成します。

application.xml の作成例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/ap
plication_6.xsd">

  <description>Sample application &quot;fromwsdl&quot;</description>
  <display-name>Sample_application_fromwsdl</display-name>
  <module>
    <web>
      <web-uri>fromwsdl.war</web-uri>
      <context-root>fromwsdl</context-root>
    </web>
  </module>
</application>
```

バージョン 5 の application.xml を作成する場合は，application 要素の version 属性を"5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/application_5.xsd"にしてください。

作成した application.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥fromwsdl¥server¥META-INF¥ディレクトリに保存します。application.xml を作成するときの注意事項については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「5.2.2　application.xml 編集時の注意事項」を参照してください。

4.3.7　EAR ファイルを作成する

jar コマンドを使用して，EAR ファイルを作成します。

EAR ファイルの作成例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥fromwsdl¥server¥
> jar cvf fromwsdl.war .¥WEB-INF
> jar cvf fromwsdl.ear .¥fromwsdl.war .¥META-INF¥application.xml
```

jar コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥fromwsdl¥server¥ディレクトリに fromwsdl.ear が作成されます。

jar コマンドについては，JDK のドキュメントを参照してください。

4.4　デプロイと開始の例（WSDL 起点）

ここでは，WSDL を起点とした場合のデプロイと開始の例を説明します。

4.4.1　EAR ファイルをデプロイする

cjimportapp コマンドを使用して，作成した EAR ファイルを J2EE サーバにデプロイします。

デプロイの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥fromwsdl¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjimportapp" jaxwsserver -nameserver corbaname::testserver
:900 -f fromwsdl.ear
```

cjimportapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションをデプロイ（インポート）する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.3　J2EE アプリケーションのインポート」を参照してください。

4.4.2　Web サービスを開始する

cjstartapp コマンドを使用して，Web サービスを開始します。

Web サービスを開始する例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥fromwsdl¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjstartapp" jaxwsserver -nameserver corbaname::testserver:
900 -name Sample_application_fromwsdl
```

cjstartapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartapp（J2EE アプリケーションの開始）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションを開始する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.1　J2EE アプリケーションの開始」を参照してください。

4.5　Web サービスクライアントの開発例（WSDL 起点）

ここでは，WSDL を起点とした場合の Web サービスクライアントの開発例を説明します。

4.5.1　サービスクラスを生成する

cjwsimport コマンドを実行すると，サービスクラスなど Web サービスクライアントの開発に必要な Java ソースが生成されます。cjwsimport コマンドについては，「14.1.1　cjwsimport コマンド」を参照してください。

Web サービスの開発と同じ環境で，Web サービスクライアントを開発する場合の実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥fromwsdl¥client¥
> mkdir src/
> mkdir classes/
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" -s src -d classes ..¥server¥WEB-INF¥wsdl¥input.wsdl
```

Web サービスの開発と異なる環境で，Web サービスクライアントを開発する場合の実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥fromwsdl¥client¥
> mkdir src/
> mkdir classes/
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" -s src -d classes http://webhost:8085/fromwsdl/TestJaxWsService?wsdl
```

正常に終了すると，c:¥temp¥jaxws¥works¥fromwsdl¥client¥src¥com¥example¥sample¥ディレクトリに Java ソースが生成されます。com¥example¥sample¥（パッケージに対応するディレクトリパス）は，名前空間 URI の記述によって変わります。名前空間 URI とパッケージとのマッピングについては，「15.1.1　名前空間からパッケージ名へのマッピング」を参照してください。

生成物の一覧を次の表に示します。

表 4-4　サービスクラス生成時の生成物（WSDL 起点）

ファイル名	説明
JaxWsTest1.java	WSDL 定義の「要求メッセージの wrapper 要素が参照する型」に対応する JavaBean クラスです。
JaxWsTest1Response.java	WSDL 定義の「応答メッセージの wrapper 要素が参照する型」に対応する JavaBean クラスです。
ObjectFactory.java	JAXB 2.2 仕様の ObjectFactory クラスです。
package-info.java	package-info.java ファイルです。
TestJaxWs.java	TestJaxWs ポートタイプに対応する SEI です。

ファイル名	説明
TestJaxWsService.java	サービスクラスです。
UserDefinedFault.java	WSDL 定義の「フォルトメッセージの wrapper 要素が参照する型」に対応する JavaBean クラスです。
UserDefinedException.java	フォルト bean のラッパ例外クラスです。

ファイル名の JaxWsTest1，TestJaxWs，および TestJaxWsService は，オペレーションのローカル名，ポートタイプのローカル名，およびサービスのローカル名の記述によって変わります。オペレーションのローカル名，ポートタイプのローカル名，およびサービスのローカル名と Java ソースとのマッピングについては，次の個所を参照してください。

- 「15.1.2　ポートタイプから SEI 名へのマッピング」
- 「15.1.3　オペレーションからメソッド名へのマッピング」
- 「15.1.4　メッセージのパートからパラメタおよび戻り値へのマッピング（wrapper スタイルの場合）」
- 「15.1.5　メッセージのパートからパラメタおよび戻り値へのマッピング（non-wrapper スタイルの場合）」

4.5.2　Web サービスクライアントの実装クラスを作成する

Web サービスを利用する Web サービスクライアントの実装クラスを作成します。

Web サービスに対して 1 回の呼び出しをする Web サービスクライアントの実装クラスの作成例を次に示します。

```
package com.example.sample.client;

import com.example.sample.TestJaxWs;
import com.example.sample.TestJaxWsService;
import com.example.sample.UserDefinedException;

public class TestClient {
    public static void main( String[] args ) {
        try {
            TestJaxWsService service = new TestJaxWsService();
            TestJaxWs port = service.getTestJaxWs();

            String returnValue = port.jaxWsTest1( "Invocation test.", 1003 );

            System.out.println( "[RESULT] " + returnValue );
        }
        catch( UserDefinedException e ){
            e.printStackTrace();
        }
    }

}
```

作成した TestClient.java は，UTF-8 形式で c:¥temp¥jaxws¥works¥fromwsdl¥client¥src¥com¥example¥sample¥client¥ディレクトリに保存します。com.example.sample，TestJaxWs，TestJaxWsService，TestJaxWs，および jaxWsTest1 は，生成された Java ソースのパッケージ名，クラス名，およびクラス内のメソッド名によって変わります。異なる構成の Web サービスを開発する場合には，パッケージ名，クラス名，およびクラス内のメソッド名の記述を見直す必要があります。

4.5.3 Web サービスクライアントの実装クラスをコンパイルする

javac コマンドを使用して，作成した Web サービスクライアントをコンパイルします。

コンパイルの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥fromwsdl¥client¥
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥ja
vaee¥1100¥lib¥javaee-api.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;.¥classes" -d .¥classes
src¥com¥example¥sample¥client¥TestClient.java
```

javac コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥fromwsdl¥client¥classes¥com¥example¥sample¥client¥ディレクトリに，TestClient.class が生成されます。

javac コマンドについては，JDK のドキュメントを参照してください。

4.6　Web サービスの実行例（WSDL 起点）

ここでは，WSDL を起点とした場合の Web サービスクライアントの実行例を説明します。

4.6.1　Java アプリケーション用オプション定義ファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用オプション定義ファイル（usrconf.cfg）を作成します。

Java アプリケーション用オプション定義ファイルの作成例を次に示します。

```
add.class.path=<Application Serverのインストールディレクトリ>¥jaxws¥lib¥cjjaxws.jar
add.class.path=.¥classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home=<Application Serverのインストールディレクトリ>
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

<Application Server のインストールディレクトリ>の部分は，Application Server をインストールしているパスを絶対パスで指定します。

作成した Java アプリケーション用オプション定義ファイルは，UTF-8 形式で c:¥temp¥jaxws¥works¥fromwsdl¥client¥ディレクトリに保存します。Java アプリケーション用オプション定義ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.1　usrconf.cfg（Java アプリケーション用オプション定義ファイル）」を参照してください。

4.6.2　Java アプリケーション用ユーザプロパティファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用ユーザプロパティファイルを作成します。

ここでは特に設定を変更しないため，c:¥temp¥jaxws¥works¥fromwsdl¥client¥ディレクトリに usrconf.properties という名称の空のファイルを作成します。Java アプリケーション用ユーザプロパティファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.2　usrconf.properties（Java アプリケーション用ユーザプロパティファイル）」を参照してください。

4.6.3　Web サービスクライアントを実行する

cjclstartap コマンドを使用して，Web サービスクライアントを実行します。

Web サービスクライアントの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥fromwsdl¥client¥
> "%COSMINEXUS_HOME%¥CC¥client¥bin¥cjclstartap"  com.example.sample.client.TestClient
```

cjclstartap コマンドが正常に終了すると，Web サービスクライアントの実行結果が表示されます。実行結果の表示例を次に示します。

```
KDJE40053-I The cjclstartap command will now start. (directory for the user definition file
= c:¥temp¥jaxws¥works¥fromwsdl¥client, PID = 2636)
--------------------
[RESULT] We've got your #1003 message "Invocation test."! It's 2007.11.28 14:50:50 now. See
ya!
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

イタリック体になっている個所は，実行したタイミングや環境によって変わります。

cjclstartap コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjclstartap（Java アプリケーションの開始）」を参照してください。

5 SEI を起点とした開発の例

この章では，SEI を起点とした Web サービスを開発する場合の例を説明します。

5.1　開発例の構成（SEI 起点）

この章で説明する開発例では，SEI を起点とした Web サービスを開発します。

開発する Web サービスの構成を次の表に示します。

表 5-1　Web サービスの構成（SEI 起点）

項番	項目		値
1	デプロイする J2EE サーバの名称		jaxwsserver
2	Web サーバのホスト名とポート番号		webhost:8085
3	ネーミングサーバの URL		corbaname::testserver:900
4	コンテキストルート		fromjava_dynamic_generate
5	スタイル		document/literal/wrapped
6	名前空間 URI		http://sample.com
7	ポートタイプ	個数	1
8		ローカル名	AddNumbersImpl
9	オペレーション	個数	1
10		ローカル名	add
11	サービス	個数	1
12		ローカル名	AddNumbersImplService
13	ポート	個数	1
14		ローカル名	AddNumbersImplPort
15	Web サービス実装クラス		com.sample.AddNumbersImpl
16	Web サービス実装クラスで公開するメソッド	個数	1
17		メソッド名	add
18	Web サービス実装内のメソッドでスローする例外	個数	1
19		クラス名	com.sample.AddNumbersFault

Web サービス開発時のカレントディレクトリの構成を次の表に示します。

表 5-2　カレントディレクトリの構成（SEI 起点）

ディレクトリ	説明
c:¥temp¥jaxws¥works¥fromjava	カレントディレクトリです。
├ server¥	Web サービスの開発で使用します。
│　└ META-INF¥	EAR ファイルの META-INF ディレクトリに対応します。

ディレクトリ	説明
│ │ └ application.xml	「5.3.4　application.xml を作成する」で作成します。
├ src¥	Web サービスのソースファイル（*.java）を格納します。「5.3.1　Web サービス実装クラスを作成する」および「5.3.2　Web サービス実装クラスをコンパイルする」で使用します。
├ WEB-INF¥	WAR ファイルの WEB-INF ディレクトリに対応します。
│ ├ web.xml	「5.3.3　web.xml を作成する」で作成します。
│ ├ classes¥	コンパイルしたクラスファイル（*.class）を格納します。「5.3.2　Web サービス実装クラスをコンパイルする」で使用します。
│ └ wsdl¥	「5.3.5　WSDL ファイルを作成する（任意）」で作成します。
├ fromjava_dynamic_generate.ear	「5.3.6　EAR ファイルを作成する」で作成します。
└ fromjava_dynamic_generate.war	
└ client¥	Web サービスクライアントの開発で使用します。
├ src¥	Web サービスクライアントのソースファイル（*.java）を格納します。「5.5.1　サービスクラスを生成する」および「5.5.2　Web サービスクライアントの実装クラスを作成する」で使用します。
├ classes¥	コンパイルしたクラスファイル（*.class）を格納します。「5.5.3　Web サービスクライアントの実装クラスをコンパイルする」で使用します。
├ usrconf.cfg	「5.6.1　Java アプリケーション用オプション定義ファイルを作成する」で作成します。
└ usrconf.properties	「5.6.2　Java アプリケーション用ユーザプロパティファイルを作成する」で作成します。

なお，以降の説明では，この表に示すディレクトリおよびファイル名を使用します。コマンド実行例や Java ソースなどで背景色付きの太字になっている部分は，この例で使用する指定値や生成される値を示します。構築する環境に合わせて読み替えてください。

また，この章で説明する開発例では，Web サービスと Web サービスクライアントを同じ環境で開発しますが，別の環境で開発することもできます。別の環境で開発する場合は，それぞれの環境に合わせて，カレントディレクトリのパスを読み替えてください。

5.2　開発例の流れ（SEI 起点）

この章で説明する開発例では，次の流れで開発および実行します。

Web サービスの開発

1. Web サービス実装クラスを作成する（5.3.1）
2. Web サービス実装クラスをコンパイルする（5.3.2）
3. web.xml を作成する（5.3.3）
4. application.xml を作成する（5.3.4）
5. WSDL ファイルを作成する（任意）（5.3.5）
6. EAR ファイルを作成する（5.3.6）

デプロイと開始

1. EAR ファイルをデプロイする（5.4.1）
2. Web サービスを開始する（5.4.2）

Web サービスクライアントの開発

1. cjwsimport コマンドを実行し，サービスクラスを生成する（5.5.1）
2. Web サービスクライアントの実装クラスを作成する（5.5.2）
3. Web サービスクライアントの実装クラスをコンパイルする（5.5.3）

Web サービスの実行

1. Java アプリケーション用オプション定義ファイルを作成する（5.6.1）
2. Java アプリケーション用ユーザプロパティファイルを作成する（5.6.2）
3. Web サービスクライアントを実行する（5.6.3）

5.3　Web サービスの開発例（SEI 起点）

ここでは，SEI を起点とした場合の Web サービスの開発例を説明します。

5.3.1　Web サービス実装クラスを作成する

Web サービスの処理を記述した Web サービス実装クラスを新規に作成します。

ここでは，受け取った要求メッセージの内容を計算して，応答メッセージを返す Web サービス実装クラスを作成します。

SOAP 1.1 の場合の Web サービス実装クラスの作成例を次に示します。作成した AddNumbersImpl.java は，UTF-8 形式で c:¥temp¥jaxws¥works¥fromjava¥server¥src¥com¥sample¥ディレクトリに保存してください。

```
package com.sample;

@javax.jws.WebService
public class AddNumbersImpl{

    public int add( int number1, int number2 ) throws AddNumbersFault{

        if( ( number1 < 0 ) || ( number2 < 0 ) ){
            throw new AddNumbersFault( "Negative number cannot be added!",
                "Numbers: " + number1 + ", " + number2 );
        }
        return number1 + number2;
    }

}
```

また，com.sample.AddNumbersImpl でスローしている例外クラス com.sample.AddNumbersFault を作成します。通常，例外クラスの作成は任意ですが，ここでは例外クラスを作成します。

例外クラスの作成例を次に示します。作成した AddNumbersFault.java は，UTF-8 形式で，c:¥temp¥jaxws¥works¥fromjava¥server¥src¥com¥sample¥ディレクトリに保存してください。

```
package com.sample;

public class AddNumbersFault extends Exception {

    String detail;

    public AddNumbersFault( String message, String detail ){
        super( message );
        this.detail = detail;
    }

    public String getDetail(){
```

```
        return detail;
    }
}
```

SOAP 1.2 の場合の Web サービス実装クラスの作成例を次に示します。

```
package com.sample;

@javax.jws.WebService
@javax.xml.ws.BindingType(javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_BINDING)
public class AddNumbersImpl{

    public int add( int number1, int number2 ) throws AddNumbersFault{

        if( ( number1 < 0 ) || ( number2 < 0 ) ){
            throw new AddNumbersFault( "Negative number cannot be added!",
                "Numbers: " + number1 + ", " + number2 );
        }
        return number1 + number2;
    }

}
```

5.3.2　Web サービス実装クラスをコンパイルする

javac コマンドを実行して，Web サービス実装クラスをコンパイルします。javac コマンドについては，JDK のドキュメントを参照してください。

javac コマンドの実行例を次に示します。添付ファイル機能を使用した Java プログラムを javac コマンドでコンパイルする場合，javac コマンドの引数に"--add-modules=java.activation"を指定してください。

```
> cd c:¥temp¥jaxws¥works¥fromjava¥server¥
> mkdir .¥WEB-INF¥classes¥
> javac -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥javaee¥1100¥lib¥ja
vaee-api.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxp.jar;%
COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar" -d WEB-INF¥classes¥ -s src src¥com¥sample¥AddNumbersI
mpl.java src¥com¥sample¥AddNumbersFault.java
```

javac コマンドが正常に終了すると，コンパイルしたクラスが，c:¥temp¥jaxws¥works¥fromjava¥server¥WEB-INF¥classes¥com¥sample¥ディレクトリに出力されます。なお，コンパイルした Web サービス実装クラスに対して hwsgen コマンドを実行すると，事前にエラーチェックができます。hwsgen コマンドについては，「14.1.2　hwsgen コマンド」を，エラーチェックについては，「10.23.1　hwsgen コマンドによるエラーチェックについて」を参照してください。

5.3.3　web.xml を作成する

WAR ファイルの構成要素として必要な web.xml を作成します。

web.xml の作成例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/we
b-app_3_0.xsd">
  <description>Sample web service &amp;quot;fromjava_dynamic_generate&amp;quot;</description>
  <display-name>Sample_web_service_fromjava_dynamic_generate</display-name>
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/AddNumbersImplService</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

バージョン 2.5 の web.xml を作成する場合は，web-app 要素の version 属性を"2.5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"にしてください。

作成した web.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥fromjava¥server¥WEB-INF¥ディレクトリに保存します。web.xml の設定項目については，「3.4　web.xml の作成」を参照してください。

5.3.4　application.xml を作成する

EAR ファイルの構成要素として必要な application.xml を作成します。

application.xml の作成例を次に示します。なお，Web サービスとして application.xml に設定する項目はありません。

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/ap
plication_6.xsd">
```

```
  <description>Sample application &quot;fromjava_dynamic_generate&quot;</description>
  <display-name>Sample_application_fromjava_dynamic_generate</display-name>
  <module>
    <web>
      <web-uri>fromjava_dynamic_generate.war</web-uri>
      <context-root>fromjava_dynamic_generate</context-root>
    </web>
  </module>
</application>
```

バージョン 5 の application.xml を作成する場合は，application 要素の version 属性を"5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/application_5.xsd"にしてください。

作成した application.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥fromjava¥server¥META-INF¥ディレクトリに保存します。application.xml を作成するときの注意事項については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「5.2.2　application.xml 編集時の注意事項」を参照してください。

5.3.5　WSDL ファイルを作成する（任意）

SEI 起点の開発では WSDL ファイルの作成は任意ですが，作成した場合は EAR ファイルに含めます。ここでは，hwsgen コマンドの WSDL 生成機能を実行することで，コンパイル済みの Java ソースから WSDL ファイルを作成する例を説明します。hwsgen コマンドについては，「14.1.2　hwsgen コマンド」を参照してください。

hwsgen コマンドの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥fromwsdl¥server¥
> mkdir .¥WEB-INF¥wsdl¥
> mkdir .¥temporary
> "%COSMINEXUS_HOME%¥jaxws¥bin¥hwsgen.bat" -r .¥WEB-INF¥wsdl -d .¥temporary -cp .¥WEB-INF¥cl
asses com.sample.AddNumbersImpl
> rmdir /S /Q .¥temporary
```

hwsgen コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥fromjava¥WEB-INF¥wsdl ディレクトリに，リソースファイルが生成されます。生成物の一覧を次の表に示します。

表 5-3　hwsgen コマンド実行時の生成物

ファイル名	説明
AddNumbersImplService.wsdl	指定した Java ソースに対応する WSDL ファイルです。
AddNumbersImplService_schema1.xsd	WSDL ファイルから参照される XML Schema 定義です。

なお，c:¥temp¥jaxws¥works¥fromjava¥temporary ディレクトリに生成されるファイルは必要ないため削除してください。

5.3.6　EAR ファイルを作成する

jar コマンドを使用して，これまでに作成したファイルを含めた EAR ファイルを作成します。

EAR ファイルの作成例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥fromjava¥server¥
> jar cvf fromjava_dynamic_generate.war .¥WEB-INF
> jar cvf fromjava_dynamic_generate.ear .¥fromjava_dynamic_generate.war .¥META-INF¥application.xml
```

jar コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥fromjava¥server¥ディレクトリに fromjava_dynamic_generate.ear が作成されます。

jar コマンドについては，JDK のドキュメントを参照してください。

5.4　デプロイと開始の例（SEI 起点）

ここでは，SEI を起点とした場合のデプロイと開始の例を説明します。

5.4.1　EAR ファイルをデプロイする

cjimportapp コマンドを使用して，作成した EAR ファイルを J2EE サーバにデプロイします。

デプロイの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥fromjava¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjimportapp" jaxwsserver -nameserver corbaname::testserver
:900 -f fromjava_dynamic_generate.ear
```

cjimportapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションをデプロイ（インポート）する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.3　J2EE アプリケーションのインポート」を参照してください。

5.4.2　Web サービスを開始する

cjstartapp コマンドを使用して，Web サービスを開始します。

Web サービスを開始する例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥fromjava¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjstartapp" jaxwsserver -nameserver corbaname::testserver:
900 -name Sample_application_fromjava_dynamic_generate
```

cjstartapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartapp（J2EE アプリケーションの開始）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションを開始する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.1　J2EE アプリケーションの開始」を参照してください。

5.5　Web サービスクライアントの開発例（SEI 起点）

ここでは，SEI を起点とした場合の Web サービスクライアントの開発例を説明します。

5.5.1　サービスクラスを生成する

cjwsimport コマンドを実行すると，サービスクラスなど，Web サービスクライアントの開発に必要な Java ソースが生成されます。cjwsimport コマンドについては，「14.1.1　cjwsimport コマンド」を参照してください。

cjwsimport コマンドの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥fromjava¥client¥
> mkdir src¥
> mkdir classes¥
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" -s src -d classes http://webhost:8085/fromjava_dynamic_generate/AddNumbersImplService?wsdl
```

cjwsimport コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥fromjava¥client¥src¥com¥sample¥ディレクトリに，Java ソースが生成されます。なお，com¥sample¥（パッケージに対応するディレクトリパス）は，名前空間 URI の記述によって変わります。名前空間 URI とパッケージとのマッピングについては，「15.1.1　名前空間からパッケージ名へのマッピング」を参照してください。

生成物の一覧を次の表に示します。

表 5-4　サービスクラス生成時の生成物（SEI 起点）

ファイル名	説明
Add.java	WSDL 定義の「オペレーション」の要求メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
AddResponse.java	WSDL 定義の「オペレーション」の応答メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
ObjectFactory.java	JAXB 2.2 仕様の ObjectFactory クラスです。
package-info.java	package-info.java ファイルです。
AddNumbersImpl.java	WSDL 定義の「サービス」に対応する SEI です。
AddNumbersImplService.java	サービスクラスです。
AddNumbersFault.java	AddNumbersFault に対応する JavaBean クラスです。
AddNumbersFault_Exception.java	フォルト bean のラッパ例外クラスです。

ファイル名の Add，AddNumbersImpl，および AddNumbersImplService は，オペレーションのローカル名，ポートタイプのローカル名，およびサービスのローカル名の記述によって変わります。オペレーショ

ンのローカル名，ポートタイプのローカル名，およびサービスのローカル名のマッピングについては，次の個所を参照してください。

- 「15.1.2　ポートタイプから SEI 名へのマッピング」
- 「15.1.3　オペレーションからメソッド名へのマッピング」
- 「15.1.4　メッセージのパートからパラメタおよび戻り値へのマッピング（wrapper スタイルの場合）」
- 「15.1.5　メッセージのパートからパラメタおよび戻り値へのマッピング（non-wrapper スタイルの場合）」

5.5.2　Web サービスクライアントの実装クラスを作成する

Web サービスを利用する Web サービスクライアントの実装クラスを作成します。

Web サービスに対して 1 回の呼び出しをする Web サービスクライアントの作成例を次に示します。

```
package com.sample.client;

import com.sample.AddNumbersImpl;
import com.sample.AddNumbersImplService;
import com.sample.AddNumbersFault_Exception;

public class TestClient {
    public static void main( String[] args ) {
        try {
            AddNumbersImplService service = new AddNumbersImplService();
            AddNumbersImpl port = service.getAddNumbersImplPort();

            int returnValue = port.add( 205, 103 );

            System.out.println( "[RESULT] " + returnValue );
        }
        catch( AddNumbersFault_Exception e ){
            e.printStackTrace();
        }
    }

}
```

作成した TestClient.java は，UTF-8 形式で
c:¥temp¥jaxws¥works¥fromjava¥client¥src¥com¥sample¥client¥ディレクトリに保存します。

なお，com.sample，AddNumbersImpl，AddNumbersImplService，AddNumbersImplPort，および add は，生成された Java ソースのパッケージ名，クラス名，およびクラス内のメソッド名によって変わります。異なる構成の Web サービスを開発する場合には，パッケージ名，クラス名，およびクラス内のメソッド名の記述を見直す必要があります。

5.5.3　Web サービスクライアントの実装クラスをコンパイルする

javac コマンドを使用して，作成した Web サービスクライアントをコンパイルします。

コンパイルの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥fromjava¥client¥
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥ja
vaee¥1100¥lib¥javaee-api.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;.¥classes" -d .¥classes
src¥com¥sample¥client¥TestClient.java
```

javac コマンドが正常に終了すると，
c:¥temp¥jaxws¥works¥fromjava¥client¥classes¥com¥sample¥client¥ディレクトリに，
TestClient.class が生成されます。

javac コマンドについては，JDK のドキュメントを参照してください。

5.6　Web サービスの実行例（SEI 起点）

ここでは，SEI を起点とした場合の Web サービスクライアントの実行例を説明します。

5.6.1　Java アプリケーション用オプション定義ファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用オプション定義ファイル（usrconf.cfg）を作成します。

Java アプリケーション用オプション定義ファイルの作成例を次に示します。

```
add.class.path=<Application Serverのインストールディレクトリ>¥jaxws¥lib¥cjjaxws.jar
add.class.path=.¥classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home=<Application Serverのインストールディレクトリ>
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

<Application Server のインストールディレクトリ>の部分は，Application Server をインストールしているパスを絶対パスで指定します。

作成した Java アプリケーション用オプション定義ファイルは，c:¥temp¥jaxws¥works¥fromjava¥client¥ディレクトリに保存します。Java アプリケーション用オプション定義ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.1　usrconf.cfg（Java アプリケーション用オプション定義ファイル）」を参照してください。

5.6.2　Java アプリケーション用ユーザプロパティファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用ユーザプロパティファイルを作成します。

ここでは特に設定を変更しないため，c:¥temp¥jaxws¥works¥fromjava¥client¥ディレクトリに usrconf.properties という名称の空のファイルを作成します。Java アプリケーション用ユーザプロパティファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.2　usrconf.properties（Java アプリケーション用ユーザプロパティファイル）」を参照してください。

5.6.3　Web サービスクライアントを実行する

cjclstartap コマンドを使用して，Web サービスクライアントを実行します。

Web サービスクライアントの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥fromjava¥client¥
> "%COSMINEXUS_HOME%¥CC¥client¥bin¥cjclstartap"  com.sample.client.TestClient
```

cjclstartap コマンドが正常に終了すると，Web サービスクライアントの実行結果が表示されます。実行結果の表示例を次に示します。

```
KDJE40053-I The cjclstartap command will now start. (directory for the user definition file
= c:¥temp¥jaxws¥works¥fromjava¥client, PID = 2636)
[RESULT] 308
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

イタリック体になっている個所は，実行したタイミングや環境によって変わります。

cjclstartap コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjclstartap（Java アプリケーションの開始)」を参照してください。

6 SEI を起点とした開発の例（hwsgen コマンドを使用する場合）

この章では，SEI を起点として hwsgen コマンドを使用して Web サービスを開発する場合の例を説明します。

6.1　開発例の構成（SEI 起点・hwsgen コマンド）

この章で説明する開発例では，Application Server が提供する hwsgen コマンドを使用して，SEI を起点とした Web サービスを開発します。hwsgen コマンドの使用方法については，「14.1.2　hwsgen コマンド」を参照してください。

開発する Web サービスの構成を次の表に示します。

表 6-1　Web サービスの構成（SEI 起点・hwsgen コマンド）

項番	項目		値
1	デプロイする J2EE サーバの名称		jaxwsserver
2	Web サーバのホスト名とポート番号		webhost:8085
3	ネーミングサーバの URL		corbaname::testserver:900
4	コンテキストルート		wsgen
5	スタイル		document/literal/wrapped
6	名前空間 URI		http://sample.com
7	ポートタイプ	個数	1
8		ローカル名	AddNumbersImpl
9	オペレーション	個数	1
10		ローカル名	add
11	サービス	個数	1
12		ローカル名	AddNumbersImplService
13	ポート	個数	1
14		ローカル名	AddNumbersImplPort
15	Web サービス実装クラス		com.sample.AddNumbersImpl
16	Web サービス実装クラスで公開するメソッド	個数	1
17		メソッド名	add
18	Web サービス実装内のメソッドでスローする例外	個数	1
19		クラス名	com.sample.AddNumbersFault

Web サービス開発時のカレントディレクトリの構成を次の表に示します。

表 6-2　カレントディレクトリの構成（SEI 起点・hwsgen コマンド）

ディレクトリ	説明
c:¥temp¥jaxws¥works¥wsgen	カレントディレクトリです。

ディレクトリ	説明
├ server¥	Web サービスの開発で使用します。
├ META-INF¥	EAR ファイルの META-INF ディレクトリに対応します。
└ application.xml	「6.3.4　application.xml を作成する」で作成します。
├ WEB-INF¥	WAR ファイルの WEB-INF ディレクトリに対応します。
├ web.xml	「6.3.3　web.xml を作成する」で作成します。
├ classes¥	コンパイルしたクラスファイル（*.class）を格納します。「6.3.2　Java ソースを生成する」で使用します。
└ wsdl¥	「6.3.2　Java ソースを生成する」で作成します。
├ wsgen.ear	「6.3.5　EAR ファイルを作成する」で作成します。
└ wsgen.war	
└ client¥	Web サービスクライアントの開発で使用します。
├ src¥	Web サービスクライアントのソースファイル（*.java）を格納します。「6.5.1　サービスクラスを生成する」および「6.5.2　Web サービスクライアントの実装クラスを作成する」で使用します。
├ classes¥	コンパイルしたクラスファイル（*.class）を格納します。「6.5.3　Web サービスクライアントの実装クラスをコンパイルする」で使用します。
├ usrconf.cfg	「6.6.1　Java アプリケーション用オプション定義ファイルを作成する」で作成します。
└ usrconf.properties	「6.6.2　Java アプリケーション用ユーザプロパティファイルを作成する」で作成します。

カレントディレクトリのパスは，開発する環境に合わせて変更してください。

なお，以降の説明では，この表に示すディレクトリおよびファイル名を使用します。コマンド実行例や Java ソースなどで背景色付きの太字になっている部分は，この例で使用する指定値や生成される値を示します。構築する環境に合わせて読み替えてください。

また，この章で説明する開発例では，Web サービスと Web サービスクライアントを同じ環境で開発しますが，別の環境で開発することもできます。別の環境で開発する場合は，それぞれの環境に合わせて，カレントディレクトリのパスを読み替えてください。

6.2　開発例の流れ（SEI 起点・hwsgen コマンド）

この章で説明する開発例では，次の流れで開発および実行します。

Web サービスの開発

1. コンパイル済みのクラスファイルを保存する（6.3.1）
2. hwsgen コマンドを実行して追加の Java コードを生成し，任意で WSDL も生成する（6.3.2）
3. web.xml を作成する（6.3.3）
4. application.xml を作成する（6.3.4）
5. EAR ファイルを作成する（6.3.5）

デプロイと開始

1. EAR ファイルをデプロイする（6.4.1）
2. Web サービスを開始する（6.4.2）

Web サービスクライアントの開発

1. cjwsimport コマンドを実行し，サービスクラスを生成する（6.5.1）
2. Web サービスクライアントの実装クラスを作成する（6.5.2）
3. Web サービスクライアントの実装クラスをコンパイルする（6.5.3）

Web サービスの実行

1. Java アプリケーション用オプション定義ファイルを作成する（6.6.1）
2. Java アプリケーション用ユーザプロパティファイルを作成する（6.6.2）
3. Web サービスクライアントを実行する（6.6.3）

6.3　Web サービスの開発例（SEI 起点・hwsgen コマンド）

ここでは，SEI を起点とし，hwsgen コマンドを使用する場合の Web サービスの開発例を説明します。

6.3.1　コンパイル済みのクラスファイルを保存する

コンパイル済みの Web サービス実装クラス AddNumbersImpl.class を c:¥temp¥jaxws¥works¥wsgen¥server¥WEB-INF¥classes¥com¥sample¥ディレクトリに保存します。また，コンパイル済みの例外クラス AddNumbersFault.class は，c:¥temp¥jaxws¥works¥wsgen¥server¥WEB-INF¥classes¥com¥sample¥ディレクトリに保存してください。

6.3.2　Java ソースを生成する

hwsgen コマンドを実行して，Web サービスの開発に必要な Java コードを追加し，任意で Web サービスのメタ情報を示すリソースファイル（WSDL および XML Schema 定義）を生成します。

リソースファイルも生成する場合の hwsgen コマンドの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥wsgen¥server¥
> "%COSMINEXUS_HOME%¥jaxws¥bin¥hwsgen.bat" -r WEB-INF¥wsdl -d WEB-INF¥classes -cp WEB-INF¥cl
asses com.sample.AddNumbersImpl
```

hwsgen コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥wsgen¥server¥WEB-INF¥classes¥com¥sample¥ディレクトリに，Java ソースが生成されます。なお，com¥sample¥（パッケージに対応するディレクトリパス）は，名前空間 URI の記述によって変わります。名前空間 URI とパッケージとのマッピングについては，「15.1.1　名前空間からパッケージ名へのマッピング」を参照してください。

生成物の一覧を次の表に示します。

表 6-3　Java ソース生成時の生成物（SEI 起点・hwsgen コマンド）

ファイル名	説明
Add.java	WSDL 定義の「オペレーション」の要求メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
AddResponse.java	WSDL 定義の「オペレーション」の応答メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
AddNumbersFaultBean.java	例外クラスの AddNumbersFault に対応する JavaBean クラスです。

ファイル名の Add および AddNumbersFault は，Web サービス実装クラスで公開するメソッド名，ポートタイプのローカル名，および Web サービス実装クラスでスローする例外のクラス名の記述によって変

わります。オペレーションのローカル名のマッピングについては，「15.1.3　オペレーションからメソッド名へのマッピング」を参照してください。

なお，リソースファイルは c:¥temp¥jaxws¥works¥wsgen¥WEB-INF¥wsdl¥ディレクトリに生成されます。

6.3.3　web.xml を作成する

WAR ファイルの構成要素として必要な web.xml を作成します。

web.xml の作成例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/we
b-app_3_0.xsd">
  <description>Sample web service &quot;wsgen &quot;</description>
  <display-name>Sample_web_service_wsgen</display-name>
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/AddNumbersImplService</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

バージョン 2.5 の web.xml を作成する場合は，web-app 要素の version 属性を"2.5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"にしてください。

作成した web.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥wsgen¥server¥WEB-INF¥ディレクトリに保存します。web.xml の設定項目については，「3.4　web.xml の作成」を参照してください。

6.3.4　application.xml を作成する

EAR ファイルの構成要素として必要な application.xml を作成します。

application.xml の作成例を次に示します。なお，Web サービスとして application.xml に設定する項目はありません。

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/application_6.xsd">

  <description>Sample application &amp;quot;wsgen&amp;quot;</description>
  <display-name>Sample_application_wsgen</display-name>
  <module>
    <web>
      <web-uri>wsgen.war</web-uri>
      <context-root>wsgen</context-root>
    </web>
  </module>
</application>
```

バージョン 5 の application.xml を作成する場合は，application 要素の version 属性を"5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/application_5.xsd"にしてください。

作成した application.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥ wsgen¥server¥META-INF¥ディレクトリに保存します。application.xml を作成するときの注意事項については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「5.2.2　application.xml 編集時の注意事項」を参照してください。

6.3.5　EAR ファイルを作成する

jar コマンドを使用して，これまでに作成したファイルを含めた EAR ファイルを作成します。

EAR ファイルの作成例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥fromjava¥server¥
> jar cvf wsgen.war .¥WEB-INF
> jar cvf wsgen.ear .¥wsgen.war .¥META-INF¥application.xml
```

jar コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥wsgen¥server¥ディレクトリに wsgen.ear が作成されます。

jar コマンドについては，JDK のドキュメントを参照してください。

6.4　デプロイと開始の例（SEI 起点・hwsgen コマンド）

ここでは，SEI を起点として hwsgen コマンドを使用した場合のデプロイと開始の例を説明します。

6.4.1　EAR ファイルをデプロイする

cjimportapp コマンドを使用して，作成した EAR ファイルを J2EE サーバにデプロイします。

デプロイの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥wsgen¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjimportapp" jaxwsserver -nameserver corbaname::testserver
:900 -f fromjava.ear
```

cjimportapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションをデプロイ（インポート）する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.3　J2EE アプリケーションのインポート」を参照してください。

6.4.2　Web サービスを開始する

cjstartapp コマンドを使用して，Web サービスを開始します。

Web サービスを開始する例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥wsgen¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjstartapp" jaxwsserver -nameserver corbaname::testserver:
900 -name Sample_application_fromjava
```

cjstartapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartapp（J2EE アプリケーションの開始）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションを開始する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.1　J2EE アプリケーションの開始」を参照してください。

6.5　Web サービスクライアントの開発例（SEI 起点・hwsgen コマンド）

ここでは，SEI を起点とし，hwsgen コマンドを使用した場合の Web サービスクライアントの開発例を説明します。

6.5.1　サービスクラスを生成する

cjwsimport コマンドを実行すると，サービスクラスなど，Web サービスクライアントの開発に必要な Java ソースが生成されます。cjwsimport コマンドについては，「14.1.1　cjwsimport コマンド」を参照してください。

cjwsimport コマンドの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥wsgen¥client¥
> mkdir src¥
> mkdir classes¥
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" -s src -d classes http://webhost:8085/wsgen/AddNumbersImplService?wsdl
```

cjwsimport コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥wsgen¥client¥src¥com¥sample¥ディレクトリに，Java ソースが生成されます。なお，com¥sample¥（パッケージに対応するディレクトリパス）は，名前空間 URI の記述によって変わります。名前空間 URI とパッケージとのマッピングについては，「15.1.1　名前空間からパッケージ名へのマッピング」を参照してください。

生成物の一覧を次の表に示します。

表 6-4　サービスクラス生成時の生成物（SEI 起点・hwsgen コマンド）

ファイル名	説明
Add.java	WSDL 定義の「オペレーション」の要求メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
AddResponse.java	WSDL 定義の「オペレーション」の応答メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
ObjectFactory.java	JAXB 2.2 仕様の ObjectFactory クラスです。
package-info.java	package-info.java ファイルです。
AddNumbersImpl.java	WSDL 定義の「サービス」に対応する SEI です。
AddNumbersImplService.java	サービスクラスです。
AddNumbersFault.java	AddNumbersFault に対応する JavaBean クラスです。
AddNumbersFault_Exception.java	フォルト bean のラッパ例外クラスです。

ファイル名の Add，AddNumbersImpl，および AddNumbersImplService は，オペレーションのローカル名，ポートタイプのローカル名，およびサービスのローカル名の記述によって変わります。オペレーショ

ンのローカル名，ポートタイプのローカル名，およびサービスのローカル名のマッピングについては，次の個所を参照してください。

- 「15.1.2　ポートタイプから SEI 名へのマッピング」
- 「15.1.3　オペレーションからメソッド名へのマッピング」
- 「15.1.4　メッセージのパートからパラメタおよび戻り値へのマッピング（wrapper スタイルの場合）」
- 「15.1.5　メッセージのパートからパラメタおよび戻り値へのマッピング（non-wrapper スタイルの場合）」

6.5.2　Web サービスクライアントの実装クラスを作成する

Web サービスを利用する Web サービスクライアントの実装クラスを作成します。

Web サービスに対して 1 回の呼び出しをする Web サービスクライアントの作成例を次に示します。

```
package com.sample.client;

import com.sample.AddNumbersImpl;
import com.sample.AddNumbersImplService;
import com.sample.AddNumbersFault_Exception;

public class TestClient {
    public static void main( String[] args ) {
        try {
            AddNumbersImplService service = new AddNumbersImplService();
            AddNumbersImpl port = service.getAddNumbersImplPort();

            int returnValue = port.add( 205, 103 );

            System.out.println( "[RESULT] " + returnValue );
        }
        catch( AddNumbersFault_Exception e ){
            e.printStackTrace();
        }
    }

}
```

作成した TestClient.java は，UTF-8 形式で
c:¥temp¥jaxws¥works¥wsgen¥client¥src¥com¥sample¥client¥ディレクトリに保存します。

なお，com.sample，AddNumbersImpl，AddNumbersImplService，AddNumbersImplPort，および add は，生成された Java ソースのパッケージ名，クラス名，およびクラス内のメソッド名によって変わります。異なる構成の Web サービスを開発する場合には，パッケージ名，クラス名，およびクラス内のメソッド名の記述を見直す必要があります。

6.5.3　Web サービスクライアントの実装クラスをコンパイルする

javac コマンドを使用して，作成した Web サービスクライアントをコンパイルします。

コンパイルの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥wsgen¥client¥
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥client¥lib¥j2ee-javax.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;.¥classes" -d .¥classes src¥com¥sample¥client¥TestClient.java
```

javac コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥wsgen¥client¥classes¥com¥sample¥client¥ディレクトリに，TestClient.class が生成されます。

javac コマンドについては，JDK のドキュメントを参照してください。

6.6　Web サービスの実行例（SEI 起点・hwsgen コマンド）

ここでは，SEI を起点とし，hwsgen コマンドを使用した場合の Web サービスクライアントの実行例を説明します。

6.6.1　Java アプリケーション用オプション定義ファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用オプション定義ファイル（usrconf.cfg）を作成します。

Java アプリケーション用オプション定義ファイルの作成例を次に示します。

```
add.class.path=<Application Serverのインストールディレクトリ>¥jaxws¥lib¥cjjaxws.jar
add.class.path=.¥classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home=<Application Serverのインストールディレクトリ>
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

<Application Server のインストールディレクトリ>の部分は，Application Server をインストールしているパスを絶対パスで指定します。

作成した Java アプリケーション用オプション定義ファイルは，c:¥temp¥jaxws¥works¥fromjava¥client¥ディレクトリに保存します。Java アプリケーション用オプション定義ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.1　usrconf.cfg（Java アプリケーション用オプション定義ファイル）」を参照してください。

6.6.2　Java アプリケーション用ユーザプロパティファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用ユーザプロパティファイルを作成します。

ここでは特に設定を変更しないため，c:¥temp¥jaxws¥works¥wsgen¥client¥ディレクトリに usrconf.properties という名称の空のファイルを作成します。Java アプリケーション用ユーザプロパティファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.2　usrconf.properties（Java アプリケーション用ユーザプロパティファイル）」を参照してください。

6.6.3　Web サービスクライアントを実行する

cjclstartap コマンドを使用して，Web サービスクライアントを実行します。

Web サービスクライアントの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥wsgen¥client¥
> "%COSMINEXUS_HOME%¥CC¥client¥bin¥cjclstartap"  com.sample.client.TestClient
```

cjclstartap コマンドが正常に終了すると，Web サービスクライアントの実行結果が表示されます。実行結果の表示例を次に示します。

```
KDJE40053-I The cjclstartap command will now start. (directory for the user definition file
= c:¥temp¥jaxws¥works¥wsgen¥client, PID = 2636)
[RESULT] 308
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

イタリック体になっている個所は，実行したタイミングや環境によって変わります。

cjclstartap コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjclstartap（Java アプリケーションの開始)」を参照してください。

7 SEI を起点とした開発の例（カスタマイズする場合）

この章では，SEI を起点とした Web サービスをカスタマイズする場合の例を説明します。

7.1　開発例の構成（SEI 起点・カスタマイズ）

この章で説明する開発例では，SEI を起点とした Web サービスを開発します。開発する Web サービスは，アノテーションを使用してカスタマイズします。

開発する Web サービスの構成を次の表に示します。

表 7-1　Web サービスの構成（SEI 起点・カスタマイズ）

項番	項目		値
1	デプロイする J2EE サーバの名称		jaxwsserver
2	Web サーバのホスト名とポート番号		webhost:8085
3	ネーミングサーバの URL		corbaname::testserver:900
4	コンテキストルート		annotations_dynamic_generate
5	スタイル		document/literal/wrapped
6	名前空間 URI		http://example.org/sample
7	ポートタイプ	個数	1
8		ローカル名	TestJaxWs
9	オペレーション	個数	1
10		ローカル名	jaxWsTest1
11	サービス	個数	1
12		ローカル名	TestJaxWsService
13	ポート	個数	1
14		ローカル名	testJaxWs
15	Web サービス実装クラス		com.sample.AddNumbersImpl
16	Web サービス実装クラスで公開するメソッド	個数	1
17		ローカル名	add
18	Web サービス実装内のメソッドでスローする例外	個数	1
19		ローカル名	com.sample.AddNumbersFault

Web サービス開発時のカレントディレクトリの構成を次の表に示します。

表 7-2　カレントディレクトリの構成（SEI 起点・カスタマイズ）

ディレクトリ	説明
c:¥temp¥jaxws¥works¥annotations	カレントディレクトリです。
┣ server¥	Web サービスの開発で使用します。

ディレクトリ		説明
┃	┣ META-INF¥	EAR ファイルの META-INF ディレクトリに対応します。
┃	┃ ┗ application.xml	「7.3.4　application.xml を作成する」で作成します。
┃	┣ src¥	Web サービスのソースファイル（*.java）を格納します。「7.3.1　Web サービス実装クラスを作成する」および「7.3.2　Web サービス実装クラスをコンパイルする」で使用します。
┃	┣ WEB-INF¥	WAR ファイルの WEB-INF ディレクトリに対応します。
┃	┃ ┣ web.xml	「7.3.3　web.xml を作成する」で作成します。
┃	┃ ┗ classes¥	コンパイルしたクラスファイル（*.class）を格納します。「7.3.2　Web サービス実装クラスをコンパイルする」で使用します。
┃	┣ annotations_dynamic_generate.ear	「7.3.5　EAR ファイルを作成する」で作成します。
┃	┗ annotations_dynamic_generate.war	
┗	client¥	Web サービスクライアントの開発で使用します。
	┣ src¥	Web サービスクライアントのソースファイル（*.java）を格納します。「7.5.1　サービスクラスを生成する」および「7.5.2　Web サービスクライアントの実装クラスを作成する」で使用します。
	┣ classes¥	コンパイルしたクラスファイル（*.class）を格納します。「7.5.3　Web サービスクライアントの実装クラスをコンパイルする」で使用します。
	┣ usrconf.cfg	「7.6.1　Java アプリケーション用オプション定義ファイルを作成する」で作成します。
	┗ usrconf.properties	「7.6.2　Java アプリケーション用ユーザプロパティファイルを作成する」で作成します。

カレントディレクトリのパスは，開発する環境に合わせて変更してください。

なお，以降の説明では，この表に示すディレクトリおよびファイル名を使用します。コマンド実行例やJava ソースなどで背景色付きの太字になっている部分は，この例で使用する指定値や生成される値を示します。構築する環境に合わせて読み替えてください。

また，この章で説明する開発例では，Web サービスと Web サービスクライアントを同じ環境で開発しますが，別の環境で開発することもできます。別の環境で開発する場合は，それぞれの環境に合わせて，カレントディレクトリのパスを読み替えてください。

7.2　開発例の流れ（SEI 起点・カスタマイズ）

この章で説明する開発例では，次の流れでカスタマイズおよび実行します。

Web サービスの開発

1. Web サービス実装クラスを作成する（7.3.1）
2. Web サービス実装クラスをコンパイルする（7.3.2）
3. web.xml を作成する（7.3.3）
4. application.xml を作成する（7.3.4）
5. EAR ファイルを作成する（7.3.5）

デプロイと開始

1. EAR ファイルをデプロイする（7.4.1）
2. Web サービスを開始する（7.4.2）

Web サービスクライアントの開発

1. cjwsimport コマンドを実行し，サービスクラスを生成する（7.5.1）
2. Web サービスクライアントの実装クラスを作成する（7.5.2）
3. Web サービスクライアントの実装クラスをコンパイルする（7.5.3）

Web サービスの実行

1. Java アプリケーション用オプション定義ファイルを作成する（7.6.1）
2. Java アプリケーション用ユーザプロパティファイルを作成する（7.6.2）
3. Web サービスクライアントを実行する（7.6.3）

7.3　Web サービスの開発例（SEI 起点・カスタマイズ）

ここでは，SEI を起点とした場合の Web サービス（カスタマイズあり）の開発例を説明します。

7.3.1　Web サービス実装クラスを作成する

Web サービスの処理を記述した Web サービス実装クラスを作成します。ここでは，受け取った要求メッセージの内容を計算して，応答メッセージを返す Web サービス実装クラスを作成します。

SOAP 1.1 の場合の Web サービス実装クラスの作成例を次に示します。作成した AddNumbersImpl.java は，UTF-8 形式で c:¥temp¥jaxws¥works¥annotations¥server¥src¥com¥sample¥ディレクトリに保存します。

```
package com.sample;

@javax.jws.WebService(name = "TestJaxWs",
    targetNamespace = "http://example.org/sample",
    serviceName = "TestJaxWsService", portName = "testJaxWs")
@javax.xml.ws.BindingType(javax.xml.ws.soap.SOAPBinding.SOAP11HTTP_BINDING)
@javax.jws.soap.SOAPBinding(style = javax.jws.soap.SOAPBinding.Style.DOCUMENT, use = javax.j
ws.soap.SOAPBinding.Use.LITERAL)
public class AddNumbersImpl{

    @javax.jws.WebMethod(operationName = "jaxWsTest1")
    @javax.jws.WebResult(name = "return")
    public int add( @javax.jws.WebParam(name="num1")int number1, @javax.jws.WebParam(name="n
um2")int number2 ) throws AddNumbersFault{

        if( ( number1 < 0 ) || ( number2 < 0 ) ){
            throw new AddNumbersFault( "Negative number cannot be added!",
                "Numbers: " + number1 + ", " + number2 );
        }
        return number1 + number2;
    }

}
```

また，com.sample.AddNumbersImpl でスローしている例外クラス com.sample.AddNumbersFault を作成します。通常，例外クラスの作成は任意ですが，ここでは例外クラスを作成します。

例外クラスの作成例を次に示します。作成した AddNumbersFault.java は，UTF-8 形式で，c:¥temp¥jaxws¥works¥annotations¥server¥src¥com¥sample¥ディレクトリに保存します。

```
package com.sample;

public class AddNumbersFault extends Exception {

    String detail;

    public AddNumbersFault( String message, String detail ){
```

```
        super( message );
        this.detail = detail;
    }

    public String getDetail(){
        return detail;
    }
}
```

SOAP 1.2 の場合の Web サービス実装クラスの作成例を次に示します。

```
package com.sample;

@javax.jws.WebService(name = "TestJaxWs",
    targetNamespace = "http://example.org/sample",
    serviceName = "TestJaxWsService", portName = "testJaxWs")
@javax.xml.ws.BindingType(javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_BINDING)
@javax.jws.soap.SOAPBinding(style = javax.jws.soap.SOAPBinding.Style.DOCUMENT, use = javax.j
ws.soap.SOAPBinding.Use.LITERAL)
public class AddNumbersImpl{

    @javax.jws.WebMethod(operationName = "jaxWsTest1")
    @javax.jws.WebResult(name = "return")
    public int add( @javax.jws.WebParam(name="num1")int number1, @javax.jws.WebParam(name="n
um2")int number2 ) throws AddNumbersFault{

        if( ( number1 < 0 ) || ( number2 < 0 ) ){
            throw new AddNumbersFault( "Negative number cannot be added!",
                "Numbers: " + number1 + ", " + number2 );
        }
        return number1 + number2;
    }

}
```

7.3.2 Web サービス実装クラスをコンパイルする

javac コマンドを実行して，Web サービス実装クラスをコンパイルします。javac コマンドについては，JDK のドキュメントを参照してください。

javac コマンドの実行例を次に示します。添付ファイル機能を使用した Java プログラムを javac コマンドでコンパイルする場合，javac コマンドの引数に"--add-modules=java.activation"を指定してください。

```
> cd c:¥temp¥jaxws¥works¥annotations¥server¥
> mkdir .¥WEB-INF¥classes¥
> javac -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥javaee¥1100¥lib¥ja
vaee-api.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxp.jar;%
COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar" -d WEB-INF¥classes¥ -s src src¥com¥sample¥AddNumbersI
mpl.java src¥com¥sample¥AddNumbersFault.java
```

javac コマンドが正常に終了すると，コンパイルしたクラスが，c:¥temp¥jaxws¥works¥annotations¥server¥WEB-INF¥classes¥com¥sample¥ディレクトリに出力されます。なお，コンパイルした Web サービス実装クラスに対して hwsgen コマンドを実行すると，事前にエラーチェックができます。hwsgen コマンドについては，「14.1.2　hwsgen コマンド」を，エラーチェックについては，「10.23.1　hwsgen コマンドによるエラーチェックについて」を参照してください。

7.3.3　web.xml を作成する

WAR ファイルの構成要素として必要な web.xml を作成します。

web.xml の作成例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/we
b-app_3_0.xsd">
  <description>Sample web service &quot;annotations_dynamic_generate&quot;</description>
  <display-name>Sample_web_service_annotations_dynamic_generate</display-name>
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/TestJaxWsService</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

バージョン 2.5 の web.xml を作成する場合は，web-app 要素の version 属性を"2.5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"にしてください。

作成した web.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥annotations¥server¥WEB-INF¥ディレクトリに保存します。web.xml の設定項目については，「3.4　web.xml の作成」を参照してください。

7.3.4　application.xml を作成する

EAR ファイルの構成要素として必要な application.xml を作成します。

application.xml の作成例を次に示します。なお，Web サービスとして application.xml に設定する項目はありません。

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/application_6.xsd">

  <description>Sample application &quot;annotations_dynamic_generate&quot;</description>
  <display-name>Sample_application_annotations_dynamic_generate</display-name>
  <module>
    <web>
      <web-uri>annotations_dynamic_generate.war</web-uri>
      <context-root>annotations_dynamic_generate</context-root>
    </web>
  </module>
</application>
```

バージョン 5 の application.xml を作成する場合は，application 要素の version 属性を"5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/application_5.xsd"にしてください。

作成した application.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥annotations¥server¥META-INF¥ディレクトリに保存します。application.xml を作成するときの注意事項については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「5.2.2　application.xml 編集時の注意事項」を参照してください。

7.3.5　EAR ファイルを作成する

jar コマンドを使用して，これまでに作成したファイルを含めた EAR ファイルを作成します。

```
> cd c:¥temp¥jaxws¥works¥annotations¥server¥
> jar cvf annotations_dynamic_generate.war .¥WEB-INF
> jar cvf annotations_dynamic_generate.ear .¥annotations_dynamic_generate.war .¥META-INF¥application.xml
```

jar コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥annotations¥server¥ディレクトリに annotations_dynamic_generate.ear が作成されます。

jar コマンドについては，JDK のドキュメントを参照してください。

7.4　デプロイと開始の例（SEI 起点・カスタマイズ）

ここでは，SEI を起点とした場合のデプロイと開始の例を説明します。

7.4.1　EAR ファイルをデプロイする

cjimportapp コマンドを使用して，作成した EAR ファイルを J2EE サーバにデプロイします。

デプロイの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥annotations¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjimportapp" jaxwsserver -nameserver corbaname::testserver
:900 -f annotations_dynamic_generate.ear
```

cjimportapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションをデプロイ（インポート）する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.3　J2EE アプリケーションのインポート」を参照してください。

7.4.2　Web サービスを開始する

cjstartapp コマンドを使用して，Web サービスを開始します。

Web サービスを開始する例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥annotations¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjstartapp" jaxwsserver -nameserver corbaname::testserver:
900 -name Sample_application_annotations_dynamic_generate
```

cjstartapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartapp（J2EE アプリケーションの開始）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションを開始する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.1　J2EE アプリケーションの開始」を参照してください。

7.5　Web サービスクライアントの開発例（SEI 起点・カスタマイズ）

ここでは，SEI を起点とした場合の Web サービスクライアントをカスタマイズする例について説明します。

7.5.1　サービスクラスを生成する

cjwsimport コマンドを実行すると，サービスクラスなど，Web サービスクライアントの開発に必要な Java ソースが生成されます。cjwsimport コマンドについては，「14.1.1　cjwsimport コマンド」を参照してください。

cjwsimport コマンドの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥annotations¥client¥
> mkdir src¥
> mkdir classes¥
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" -s src -d classes http://webhost:8085/annotations_dynamic_generate/TestJaxWsService?wsdl
```

cjwsimport コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥annotations¥client¥src¥com¥example¥sample¥ディレクトリに，Java ソースが生成されます。なお，com¥example¥sample¥（パッケージに対応するディレクトリパス）は，名前空間 URI の記述によって変わります。名前空間 URI とパッケージとのマッピングについては，「15.1.1　名前空間からパッケージ名へのマッピング」を参照してください。

生成物の一覧を次の表に示します。

表 7-3　サービスクラス生成時の生成物（SEI 起点・カスタマイズ）

ファイル名	説明
JaxWsTest1.java	WSDL 定義の「オペレーション」の要求メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
JaxWsTest1Response.java	WSDL 定義の「オペレーション」の応答メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
ObjectFactory.java	JAXB 2.2 仕様の ObjectFactory クラスです。
package-info.java	package-info.java ファイルです。
TestJaxWs.java	WSDL 定義の「サービス」に対応する SEI です。
TestJaxWsService.java	サービスクラスです。
AddNumbersFault.java	AddNumbersFault に対応する JavaBean クラスです。
AddNumbersFault_Exception.java	フォルト bean のラッパ例外クラスです。

ファイル名の JaxWsTest1，TestJaxWs，および TestJaxWsService は，オペレーションのローカル名，ポートタイプのローカル名，およびサービスのローカル名の記述によって変わります。オペレーションの

ローカル名，ポートタイプのローカル名，およびサービスのローカル名のマッピングについては，次の個所を参照してください。

- 「15.1.2　ポートタイプから SEI 名へのマッピング」
- 「15.1.3　オペレーションからメソッド名へのマッピング」
- 「15.1.4　メッセージのパートからパラメタおよび戻り値へのマッピング（wrapper スタイルの場合）」
- 「15.1.5　メッセージのパートからパラメタおよび戻り値へのマッピング（non-wrapper スタイルの場合）」

7.5.2　Web サービスクライアントの実装クラスを作成する

Web サービスを利用する Web サービスクライアントの実装クラスを作成します。

Web サービスに対して 1 回の呼び出しをする Web サービスクライアントの作成例を次に示します。

```
package org.example.sample.client;

import org.example.sample.TestJaxWs;
import org.example.sample.TestJaxWsService;
import org.example.sample.AddNumbersFault_Exception;

public class TestClient {
    public static void main( String[] args ) {
        try {
            TestJaxWsService service = new TestJaxWsService();
            TestJaxWs port = service.getTestJaxWs();

            int returnValue = port.jaxWsTest1( 205, 103 );

            System.out.println( "[RESULT] " + returnValue );
        }
        catch( AddNumbersFault_Exception e ){
            e.printStackTrace();
        }
    }

}
```

作成した TestClient.java は，UTF-8 形式で c:¥temp¥jaxws¥works¥annotations¥client¥src¥com¥example¥sample¥client¥ディレクトリに保存します。

なお，org.example.sample，TestJaxWs，TestJaxWsService，および jaxWsTest1 は，生成された Java ソースのパッケージ名，クラス名，およびクラス内のメソッド名によって変わります。異なる構成の Web サービスを開発する場合には，パッケージ名，クラス名，およびクラス内のメソッド名の記述を見直す必要があります。

7.5.3　Web サービスクライアントの実装クラスをコンパイルする

javac コマンドを使用して，作成した Web サービスクライアントをコンパイルします。

コンパイルの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥annotations¥client¥
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥ja
vaee¥1100¥lib¥javaee-api.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;.¥classes" -d .¥classes
src¥org¥example¥sample¥client¥TestClient.java
```

javac コマンドが正常に終了すると，
c:¥temp¥jaxws¥works¥annotations¥client¥classes¥org¥example¥sample¥client¥ディレクトリに，
TestClient.class が生成されます。

javac コマンドについては，JDK のドキュメントを参照してください。

7.6　Web サービスの実行例（SEI 起点・カスタマイズ）

ここでは，SEI を起点とした場合の Web サービスクライアントをカスタマイズする実行例について説明します。

7.6.1　Java アプリケーション用オプション定義ファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用オプション定義ファイル（usrconf.cfg）を作成します。

Java アプリケーション用オプション定義ファイルの作成例を次に示します。

```
add.class.path=<Application Serverのインストールディレクトリ>¥jaxws¥lib¥cjjaxws.jar
add.class.path=.¥classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home=<Application Serverのインストールディレクトリ>
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

<Application Server のインストールディレクトリ>の部分は，Application Server をインストールしているパスを絶対パスで指定します。

作成した Java アプリケーション用オプション定義ファイルは，c:¥temp¥jaxws¥works¥annotations¥client¥ディレクトリに保存します。Java アプリケーション用オプション定義ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.1　usrconf.cfg（Java アプリケーション用オプション定義ファイル）」を参照してください。

7.6.2　Java アプリケーション用ユーザプロパティファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用ユーザプロパティファイルを作成します。

ここでは特に設定を変更しないため，c:¥temp¥jaxws¥works¥annotations¥client¥ディレクトリに usrconf.properties という名称の空のファイルを作成します。Java アプリケーション用ユーザプロパティファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.2　usrconf.properties（Java アプリケーション用ユーザプロパティファイル）」を参照してください。

7.6.3　Web サービスクライアントを実行する

cjclstartap コマンドを使用して，Web サービスクライアントを実行します。

Web サービスクライアントの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥annotations¥client¥
> "%COSMINEXUS_HOME%¥CC¥client¥bin¥cjclstartap"  org.example.sample.client.TestClient
```

cjclstartap コマンドが正常に終了すると，Web サービスクライアントの実行結果が表示されます。実行結果の表示例を次に示します。

```
KDJE40053-I The cjclstartap command will now start. (directory for the user definition file
= c:¥temp¥jaxws¥works¥annotations¥client, PID = 2636)
[RESULT] 308
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

イタリック体になっている個所は，実行したタイミングや環境によって変わります。

cjclstartap コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjclstartap（Java アプリケーションの開始)」を参照してください。

8 SEI を起点とした開発の例（EJB の Web サービスの場合）

この章では，SEI を起点とした EJB の Web サービスを開発する場合の例を説明します。

8.1　開発例の構成（SEI 起点・EJB の Web サービス）

この章で説明する開発例では，SEI を起点とした Web サービスを開発します。なお，ここではスタブベースでの開発例を説明しますが，ディスパッチベースや API ベースでも開発できます。

開発する Web サービスの構成を次の表に示します。

表 8–1　Web サービスの構成（SEI 起点）

項番	項目		値
1	デプロイする J2EE サーバの名称		jaxwsserver
2	Web サーバのホスト名とポート番号		webhost:8085
3	ネーミングサーバの URL		corbaname::testserver:900
4	コンテキストルート		statelessjava_dynamic_generate
5	スタイル		document/literal/wrapped
6	名前空間 URI		http://sample.com
7	ポートタイプ	個数	1
8		ローカル名	AddNumbersImpl
9	オペレーション	個数	1
10		ローカル名	add
11	サービス	個数	1
12		ローカル名	AddNumbersImplService
13	ポート	個数	1
14		ローカル名	AddNumbersImplPort
15	Web サービス実装クラス		com.sample.AddNumbersImpl
16	Web サービス実装クラスで公開するメソッド	個数	1
17		メソッド名	add
18	Web サービス実装内のメソッドでスローする例外	個数	1
19		クラス名	com.sample.AddNumbersFault

Web サービス開発時のカレントディレクトリの構成を次の表に示します。

表 8–2　カレントディレクトリの構成（SEI 起点）

ディレクトリ	説明
c:¥temp¥jaxws¥works¥statelessjava	カレントディレクトリです。
┣ server¥	Web サービスの開発で使用します。

ディレクトリ			説明
┃	┣	META-INF¥	EAR ファイルの META-INF ディレクトリに対応します。
┃	┃	┗ application.xml	「8.3.3　application.xml を作成する」で作成します。
┃	┣	src¥	Web サービスのソースファイル（*.java）を格納します。「8.3.1　Web サービス実装クラスを作成する」および「8.3.2　Web サービス実装クラスをコンパイルする」で使用します。
┃	┣	jar¥	コンパイルしたクラスファイル（*.class）を格納します。「8.3.1　Web サービス実装クラスを作成する」および「8.3.2　Web サービス実装クラスをコンパイルする」で使用します。
┃	┃	┗ WEB-INF¥	EJB JAR ファイルの WEB-INF ディレクトリに対応します。
┃	┃	┃ ┗ wsdl¥	「8.3.4　WSDL ファイルを作成する（任意）」で作成します。
┃	┣	statelessjava_dynamic_generate.ear	「8.3.5　EAR ファイルを作成する」で作成します。
┃	┗	statelessjava_dynamic_generate.war	
┗	client¥		Web サービスクライアントの開発で使用します。
	┣	src¥	Web サービスクライアントのソースファイル（*.java）を格納します。「8.5.1　サービスクラスを生成する」および「8.5.2　Web サービスクライアントの実装クラスを作成する」で使用します。
	┣	classes¥	コンパイルしたクラスファイル（*.class）を格納します。「8.5.3　Web サービスクライアントの実装クラスをコンパイルする」で使用します。
	┣	usrconf.cfg	「8.6.1　Java アプリケーション用オプション定義ファイルを作成する」で作成します。
	┗	usrconf.properties	「8.6.2　Java アプリケーション用ユーザプロパティファイルを作成する」で作成します。

カレントディレクトリのパスは，開発する環境に合わせて変更してください。

なお，以降の説明では，この表に示すディレクトリおよびファイル名を使用します。コマンド実行例やJava ソースなどで背景色付きの太字になっている部分は，この例で使用する指定値や生成される値を示します。構築する環境に合わせて読み替えてください。

また，この章で説明する開発例では，Web サービスと Web サービスクライアントを同じ環境で開発しますが，別の環境で開発することもできます。別の環境で開発する場合は，それぞれの環境に合わせて，カレントディレクトリのパスを読み替えてください。

8.2　開発例の流れ（SEI 起点・EJB の Web サービス）

この章で説明する開発例では，次の流れで開発および実行します。

Web サービスの開発

1. Web サービス実装クラスを作成する（8.3.1）
2. Web サービス実装クラスをコンパイルする（8.3.2）
3. application.xml を作成する（8.3.3）
4. WSDL ファイルを作成する（任意）（8.3.4）
5. EAR ファイルを作成する（8.3.5）

デプロイと開始

1. EAR ファイルをデプロイする（8.4.1）
2. Web サービスを開始する（8.4.2）

Web サービスクライアントの開発

1. cjwsimport コマンドを実行し，サービスクラスを生成する（8.5.1）
2. Web サービスクライアントの実装クラスを作成する（8.5.2）
3. Web サービスクライアントの実装クラスをコンパイルする（8.5.3）

Web サービスの実行

1. Java アプリケーション用オプション定義ファイルを作成する（8.6.1）
2. Java アプリケーション用ユーザプロパティファイルを作成する（8.6.2）
3. Web サービスクライアントを実行する（8.6.3）

8.3　Web サービスの開発例（SEI 起点・EJB の Web サービス）

ここでは，SEI を起点とした場合の EJB の Web サービスの開発例を説明します。

8.3.1　Web サービス実装クラスを作成する

Web サービスの処理を記述した Web サービス実装クラスを新規に作成します。

ここでは，受け取った要求メッセージの内容を計算して，応答メッセージを返す Web サービス実装クラスを作成します。作成した AddNumbersImpl.java は，UTF-8 形式で c:¥temp¥jaxws¥works¥statelessjava¥server¥src¥com¥sample¥ディレクトリに保存してください。

```
package com.sample;

@javax.ejb.Stateless
@javax.jws.WebService
public class AddNumbersImpl{

    public int add( int number1, int number2 ) throws AddNumbersFault{

        if( ( number1 < 0 ) || ( number2 < 0 ) ){
            throw new AddNumbersFault( ”Negative number cannot be added!”,
                ”Numbers: ” + number1 + ”, ” + number2 );
        }
        return number1 + number2;
    }

}
```

また，com.sample.AddNumbersImpl でスローしている例外クラス com.sample.AddNumbersFault を作成します。通常，例外クラスの作成は任意ですが，ここでは例外クラスを作成します。

例外クラスの作成例を次に示します。作成した AddNumbersFault.java は，UTF-8 形式で，c:¥temp¥jaxws¥works¥statelessjava¥server¥src¥com¥sample¥ディレクトリに保存してください。

```
package com.sample;

public class AddNumbersFault extends Exception {

    String detail;

    public AddNumbersFault( String message, String detail ){
        super( message );
        this.detail = detail;
    }

    public String getDetail(){
        return detail;
    }
}
```

8.3.2　Web サービス実装クラスをコンパイルする

javac コマンドを実行して，Web サービス実装クラスをコンパイルします。javac コマンドについては，JDK のドキュメントを参照してください。

javac コマンドの実行例を次に示します。添付ファイル機能を使用した Java プログラムを javac コマンドでコンパイルする場合，javac コマンドの引数に"--add-modules=java.activation"を指定してください。

```
> cd c:¥temp¥jaxws¥works¥statelessjava¥server¥
> mkdir .¥WEB-INF¥classes¥
> javac -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥javaee¥1100¥lib¥ja
vaee-api.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxp.jar;%
COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar" -d jar¥ -s src src¥com¥sample¥AddNumbersImpl.java src
¥com¥sample¥AddNumbersFault.java
```

javac コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥statelessjava¥server¥WEB-INF¥classes¥com¥sample¥ディレクトリに出力されます。なお，コンパイルした Web サービス実装クラスに対して hwsgen コマンドを実行すると，事前にエラーチェックができます。hwsgen コマンドについては，「14.1.2　hwsgen コマンド」を，エラーチェックについては，「10.23.1　hwsgen コマンドによるエラーチェックについて」を参照してください。

8.3.3　application.xml を作成する

EAR ファイルの構成要素として必要な application.xml を作成します。

application.xml の作成例を次に示します。なお，Web サービスとして application.xml に設定する項目はありません。

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/ap
plication_6.xsd">

  <description>Sample application &quot;statelessjava_dynamic_generate&quot;</description>
  <display-name>Sample_application_statelessjava_dynamic_generate</display-name>
  <module>
    <ejb>statelessjava_dynamic_generate.jar</ejb>
  </module>
</application>
```

バージョン 5 の application.xml を作成する場合は，application 要素の version 属性を"5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/application_5.xsd"にしてください。

作成した application.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥statelessjava¥server¥META-INF¥ディレクトリに保存します。application.xml を作成するときの注意事項については，マニュアル「ア

プリケーションサーバ アプリケーション開発ガイド」の「5.2.2　application.xml 編集時の注意事項」を参照してください。

8.3.4　WSDL ファイルを作成する（任意）

SEI 起点の開発では WSDL ファイルの作成は任意ですが，作成した場合は EAR ファイルに含めます。ここでは，hwsgen コマンドの WSDL 生成機能を実行することで，コンパイル済みの Java ソースから WSDL ファイルを作成する例を説明します。hwsgen コマンドについては，「14.1.2　hwsgen コマンド」を参照してください。

hwsgen コマンドの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥statelessjava¥server¥
> mkdir .¥jar¥META-INF¥wsdl¥
> mkdir .¥temporary
> "%COSMINEXUS_HOME%¥jaxws¥bin¥hwsgen.bat" -r .¥jar¥META-INF¥wsdl -d .¥temporary -cp .¥jar com.sample.AddNumbersImpl
> rmdir /S /Q .¥temporary
```

hwsgen コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥statelessjava¥META-INF¥wsdl¥ディレクトリに，リソースファイルが生成されます。生成物の一覧を次の表に示します。

表 8-3　hwsgen コマンド実行時の生成物

ファイル名	説明
AddNumbersImplService.wsdl	指定した Java ソースに対応する WSDL ファイルです。
AddNumbersImplService_schema1.xsd	WSDL ファイルから参照される XML Schema 定義です。

なお，c:¥temp¥jaxws¥works¥statelessjava¥temporary¥ディレクトリに生成されるファイルは必要ないため削除してください。

8.3.5　EAR ファイルを作成する

jar コマンドを使用して，これまでに作成したファイルを含めた EAR ファイルを作成します。

EAR ファイルの作成例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥statelessjava¥server¥
> jar cvf statelessjava_dynamic_generate.jar -C jar com
> jar cvf statelessjava_dynamic_generate.ear .¥statelessjava_dynamic_generate.jar .¥META-INF¥application.xml
```

jar コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥statelessjava¥server¥ディレクトリに statelessjava_dynamic_generate.ear が作成されます。

jar コマンドについては，JDK のドキュメントを参照してください。

8.4　デプロイと開始の例（SEI 起点・EJB の Web サービス）

ここでは，SEI を起点とした場合のデプロイと開始の例を説明します。

8.4.1　EAR ファイルをデプロイする

cjimportapp コマンドを使用して，作成した EAR ファイルを J2EE サーバにデプロイします。

デプロイの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥statelessjava¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjimportapp" jaxwsserver -nameserver corbaname::testserver
:900 -f statelessjava_dynamic_generate.ear
```

cjimportapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションをデプロイ（インポート）する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.3　J2EE アプリケーションのインポート」を参照してください。

8.4.2　Web サービスを開始する

cjstartapp コマンドを使用して，Web サービスを開始します。

Web サービスを開始する例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥statelessjava¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjstartapp" jaxwsserver -nameserver corbaname::testserver:
900 -name Sample_application_statelessjava_dynamic_generate
```

cjstartapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartapp（J2EE アプリケーションの開始）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションを開始する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.1　J2EE アプリケーションの開始」を参照してください。

8.5　Web サービスクライアントの開発例（SEI 起点・EJB の Web サービス）

ここでは，SEI を起点とした場合の Web サービスクライアントの開発例を説明します。

8.5.1　サービスクラスを生成する

cjwsimport コマンドを実行すると，サービスクラスなど，Web サービスクライアントの開発に必要な Java ソースが生成されます。cjwsimport コマンドについては，「14.1.1　cjwsimport コマンド」を参照してください。

cjwsimport コマンドの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥statelessjava¥client¥
> mkdir src¥
> mkdir classes¥
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" -s src -d classes http://webhost:8085/statelessjava_dynamic_generate/AddNumbersImplService?wsdl
```

cjwsimport コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥statelessjava¥client¥src¥com¥sample¥ディレクトリに，Java ソースが生成されます。なお，com¥sample¥（パッケージに対応するディレクトリパス）は，名前空間 URI の記述によって変わります。名前空間 URI とパッケージとのマッピングについては，「15.1.1　名前空間からパッケージ名へのマッピング」を参照してください。

生成物の一覧を次の表に示します。

表 8-4　サービスクラス生成時の生成物（SEI 起点）

ファイル名	説明
Add.java	WSDL 定義の「オペレーション」の要求メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
AddResponse.java	WSDL 定義の「オペレーション」の応答メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
ObjectFactory.java	JAXB 2.2 仕様の ObjectFactory クラスです。
package-info.java	package-info.java ファイルです。
AddNumbersImpl.java	WSDL 定義の「サービス」に対応する SEI です。
AddNumbersImplService.java	サービスクラスです。
AddNumbersFault.java	AddNumbersFault に対応する JavaBean クラスです。
AddNumbersFault_Exception.java	フォルト bean のラッパ例外クラスです。

ファイル名の Add，AddNumbersImpl，および AddNumbersImplService は，オペレーションのローカル名，ポートタイプのローカル名，およびサービスのローカル名の記述によって変わります。オペレーショ

ンのローカル名，ポートタイプのローカル名，およびサービスのローカル名のマッピングについては，次の個所を参照してください。

- 「15.1.2　ポートタイプから SEI 名へのマッピング」
- 「15.1.3　オペレーションからメソッド名へのマッピング」
- 「15.1.4　メッセージのパートからパラメタおよび戻り値へのマッピング（wrapper スタイルの場合）」
- 「15.1.5　メッセージのパートからパラメタおよび戻り値へのマッピング（non-wrapper スタイルの場合）」

8.5.2　Web サービスクライアントの実装クラスを作成する

Web サービスを利用する Web サービスクライアントの実装クラスを作成します。

Web サービスに対して 1 回の呼び出しをする Web サービスクライアントの作成例を次に示します。

```
package com.sample.client;

import com.sample.AddNumbersImpl;
import com.sample.AddNumbersImplService;
import com.sample.AddNumbersFault_Exception;

public class TestClient {
    public static void main( String[] args ) {
        try {
            AddNumbersImplService service = new AddNumbersImplService();
            AddNumbersImpl port = service.getAddNumbersImplPort();

            int returnValue = port.add( 205, 103 );

            System.out.println( "[RESULT] " + returnValue );
        }
        catch( AddNumbersFault_Exception e ){
            e.printStackTrace();
        }
    }

}
```

作成した TestClient.java は，UTF-8 形式で
c:¥temp¥jaxws¥works¥statelessjava¥client¥src¥com¥sample¥client¥ディレクトリに保存します。

なお，com.sample，AddNumbersImpl，AddNumbersImplService，AddNumbersImplPort，および add は，生成された Java ソースのパッケージ名，クラス名，およびクラス内のメソッド名によって変わります。異なる構成の Web サービスを開発する場合には，パッケージ名，クラス名，およびクラス内のメソッド名の記述を見直す必要があります。

8.5.3　Web サービスクライアントの実装クラスをコンパイルする

javac コマンドを使用して，作成した Web サービスクライアントをコンパイルします。

コンパイルの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥statelessjava¥client¥
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥ja
vaee¥1100¥lib¥javaee-api.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;.¥classes" -d .¥classes
src¥com¥sample¥client¥TestClient.java
```

javac コマンドが正常に終了すると，
c:¥temp¥jaxws¥works¥statelessjava¥client¥classes¥com¥sample¥client¥ディレクトリに，
TestClient.class が生成されます。

javac コマンドについては，JDK のドキュメントを参照してください。

8.6　Web サービスの実行例（SEI 起点・EJB の Web サービス）

ここでは，SEI を起点とした場合の Web サービスクライアントの実行例を説明します。

8.6.1　Java アプリケーション用オプション定義ファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用オプション定義ファイル（usrconf.cfg）を作成します。

Java アプリケーション用オプション定義ファイルの作成例を次に示します。

```
add.class.path=<Application Serverのインストールディレクトリ>¥jaxws¥lib¥cjjaxws.jar
add.class.path=.¥classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home=<Application Serverのインストールディレクトリ>
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

<Application Server のインストールディレクトリ>の部分は，Application Server をインストールしているパスを絶対パスで指定します。

作成した Java アプリケーション用オプション定義ファイルは，c:¥temp¥jaxws¥works¥statelessjava¥client¥ディレクトリに保存します。Java アプリケーション用オプション定義ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.1　usrconf.cfg（Java アプリケーション用オプション定義ファイル）」を参照してください。

8.6.2　Java アプリケーション用ユーザプロパティファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用ユーザプロパティファイルを作成します。

ここでは特に設定を変更しないため，c:¥temp¥jaxws¥works¥statelessjava¥client¥ディレクトリに usrconf.properties という名称の空のファイルを作成します。Java アプリケーション用ユーザプロパティファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.2　usrconf.properties（Java アプリケーション用ユーザプロパティファイル）」を参照してください。

8.6.3　Web サービスクライアントを実行する

cjclstartap コマンドを使用して，Web サービスクライアントを実行します。

Web サービスクライアントの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥statelessjava¥client¥
> "%COSMINEXUS_HOME%¥CC¥client¥bin¥cjclstartap"  com.sample.client.TestClient
```

cjclstartap コマンドが正常に終了すると，Web サービスクライアントの実行結果が表示されます。実行結果の表示例を次に示します。

```
KDJE40053-I The cjclstartap command will now start. (directory for the user definition file
= c:¥temp¥jaxws¥works¥statelessjava¥client, PID = 2636)
[RESULT] 308
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

イタリック体になっている個所は，実行したタイミングや環境によって変わります。

cjclstartap コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjclstartap（Java アプリケーションの開始)」を参照してください。

9 プロバイダを起点とした開発の例（SAAJ を利用した場合）

この章では，プロバイダを起点とした Web サービスを，SAAJ を利用して開発する場合の例を説明します。

9.1　開発例の構成（プロバイダ起点・SAAJ）

この章で説明する開発例では，Dispatch／Provider で SOAP メッセージを送受信する Web サービスを，SAAJ を利用して開発します。

なお，ここでの例では Web サービスクライアントから社員番号と社員の顔写真の情報が送信され，Web サービスが受信した情報を登録し，登録確認メッセージを返却するものとします。社員番号と登録確認メッセージの Java データ型は java.lang.String，顔写真の Java データ型は javax.activation.DataHandler です。

開発する Web サービスの構成を次の表に示します。

表 9-1　Web サービスの構成（プロバイダ起点・SAAJ）

項番	項目		値
1	デプロイする J2EE サーバの名称		jaxwsserver
2	Web サーバのホスト名とポート番号		webhost:8085
3	ネーミングサーバの URL		corbaname::testserver:900
4	コンテキストルート		dispatch_provider
5	名前空間 URI		http://sample.com
6	サービス	個数	1
7		ローカル名	UserInfoService
8	ポート	個数	1
9		ローカル名	UserInfoPort

Web サービス開発時のカレントディレクトリの構成を次の表に示します。

表 9-2　カレントディレクトリの構成（プロバイダ起点・SAAJ）

ディレクトリ	説明
c:¥temp¥jaxws¥works¥dispatch_provider	カレントディレクトリです。
┣server¥	Web サービスの開発で使用します。
┃┣META-INF¥	EAR ファイルの META-INF ディレクトリに対応します。
┃┃┗application.xml	「9.3.4　application.xml を作成する」で作成します。
┃┣src¥	Web サービスのソースファイル（*.java）を格納します。「9.3.1　プロバイダ実装クラスを作成する」で使用します。
┃┣WEB-INF¥	WAR ファイルの WEB-INF ディレクトリに対応します。

ディレクトリ	説明
┃┃┃┃┃┃┃┃┃┣ web.xml	「9.3.3　web.xml を作成する」で作成します。
┃┃┃┃┃┃┃┃┃┗ classes¥	コンパイルしたクラスファイル（*.class）を格納します。「9.3.2　Java ソースを生成する」で使用します。
┃┃┣ dispatch_provider.ear	「9.3.5　EAR ファイルを作成する」で作成します。
┃┃┗ dispatch_provider.war	
┗ client¥	Web サービスクライアントの開発で使用します。
┃┣ src¥	Web サービスクライアントのソースファイル（*.java）を格納します。「9.5.1　Web サービスクライアントの実装クラスを作成する」で作成します。
┃┣ classes¥	コンパイルしたクラスファイル（*.class）を格納します。「9.5.2　Web サービスクライアントの実装クラスをコンパイルする」で作成します。
┃┣ usrconf.cfg	「9.6.1　Java アプリケーション用オプション定義ファイルを作成する」で作成します。
┃┗ usrconf.properties	「9.6.2　Java アプリケーション用ユーザプロパティファイルを作成する」で作成します。

カレントディレクトリのパスは，開発する環境に合わせて変更してください。

なお，以降の説明では，この表に示すディレクトリおよびファイル名を使用します。コマンド実行例や Java ソースなどで背景色付きの太字になっている部分は，この例で使用する指定値や生成される値を示します。構築する環境に合わせて読み替えてください。

また，この章で説明する開発例では，Web サービスと Web サービスクライアントを同じ環境で開発しますが，別の環境で開発することもできます。別の環境で開発する場合は，それぞれの環境に合わせて，カレントディレクトリのパスを読み替えてください。

9.2　開発例の流れ（プロバイダ起点・SAAJ）

この章で説明する開発例では，次の流れで開発および実行します。

Web サービスの開発

1. プロバイダ実装クラスを作成する（9.3.1）
2. Java ソースを生成する（9.3.2）
3. web.xml を作成する（9.3.3）
4. application.xml を作成する（9.3.4）
5. EAR ファイルを作成する（9.3.5）

デプロイと開始

1. EAR ファイルをデプロイする（9.4.1）
2. Web サービスを開始する（9.4.2）

Web サービスクライアントの開発

1. Web サービスクライアントの実装クラスを作成する（9.5.1）
2. Web サービスクライアントの実装クラスをコンパイルする（9.5.2）

Web サービスの実行

1. Java アプリケーション用オプション定義ファイルを作成する（9.6.1）
2. Java アプリケーション用ユーザプロパティファイルを作成する（9.6.2）
3. Web サービスクライアントを実行する（9.6.3）

9.3　Web サービスの開発例（プロバイダ起点・SAAJ）

ここでは，プロバイダを起点とし，SAAJ を利用した場合の Web サービスの開発例を説明します。

9.3.1　プロバイダ実装クラスを作成する

プロバイダ実装クラス com.sample.UserInfoImpl を作成する例を次に示します。作成した com.sample.UserInfoImpl は，UTF-8 形式で c:¥temp¥jaxws¥works¥dispatch_provider¥server¥src¥com¥sample¥ディレクトリに保存してください。

```
package com.sample;

import java.util.Iterator;
import javax.xml.namespace.QName;
import javax.xml.soap.AttachmentPart;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPBodyElement;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.Provider;
@javax.xml.ws.WebServiceProvider(serviceName="UserInfoService")
@javax.xml.ws.ServiceMode(value=javax.xml.ws.Service.Mode.MESSAGE)
public class UserInfoImpl implements Provider<SOAPMessage>{

    public SOAPMessage invoke( SOAPMessage request ){

        // 応答メッセージ
        SOAPMessage response = null;
        // 添付ファイル(顔写真)
        AttachmentPart attachment = null;

        try {
            // 要求メッセージからデータ取得
            // 社員番号を取得
            SOAPBody soapBody = request.getSOAPBody();
            SOAPBodyElement reqRoot =
                (SOAPBodyElement)soapBody.getChildElements().next();
            Iterator number_iterator = reqRoot.getChildElements();
            String number =

((SOAPElement)number_iterator.next()).getFirstChild().getNodeValue();

            // 顔写真を取得
            Iterator attachment_iterator = request.getAttachments();
            while(attachment_iterator.hasNext()){
                attachment = (AttachmentPart)attachment_iterator.next();
            }

            // 顔写真の登録処理など取得した添付ファイルに対して
            // ほかに必要な処理があれば実装する
```

```
            // 応答メッセージの生成
            response = MessageFactory.newInstance().createMessage();
            SOAPBody resSoapBody = response.getSOAPBody();
            SOAPBodyElement resRoot = resSoapBody.addBodyElement(
                new QName("http://sample.com", "result"));
            SOAPElement soapElement = resRoot.addChildElement(
                new QName("http://sample.com", "value"));
            // 登録確認メッセージを設定
            if(null ==  attachment){
                soapElement.addTextNode("Failure(no image).");
            } else {
                soapElement.addTextNode("Success.");
            }

        } catch (SOAPException e) {
            e.printStackTrace();
        }

        return response;
    }
}
```

SOAP 1.2 の場合は，作成したプロバイダ実装クラスに対して javax.xml.ws.BindingType アノテーションを付与してください。設定する値は，SOAP 1.2/HTTP バインディングを意味する"http://www.w3.org/2003/05/soap/bindings/HTTP/"にします。javax.xml.ws.BindingType アノテーションを付与する例を次に示します。

```
package com.sample;

import java.util.Iterator;
import javax.xml.namespace.QName;
import javax.xml.soap.AttachmentPart;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPBodyElement;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.Provider;

@javax.xml.ws.BindingType("http://www.w3.org/2003/05/soap/bindings/HTTP/")
@javax.xml.ws.WebServiceProvider
@javax.xml.ws.ServiceMode(value=javax.xml.ws.Service.Mode.MESSAGE)
public class UserInfoImpl mplements Provider<SOAPMessage>{

    public SOAPMessage invoke( SOAPMessage request ){

（以降はSOAP1.1の場合と同じ）
```

なお，javax.xml.ws.BindingType アノテーションの値には，"http://www.w3.org/2003/05/soap/bindings/HTTP/"の代わりに定数値フィールド javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_BINDING を設定することもできます。定数値フィールドを設定した場合の例については，「5.3.1　Web サービス実装クラスを作成する」を参照してください。

9.3.2　Java ソースを生成する

作成したプロバイダ実装クラス com.sample.UserInfoImpl をコンパイルする例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥dispatch_provider¥server¥
> mkdir WEB-INF¥classes¥
> javac -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥javaee¥1100¥lib¥ja
vaee-api.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxp.jar;%
COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar" -d WEB-INF¥classes¥ -s src src¥com¥sample¥UserInfoImp
l.java
```

コンパイルが正常に終了すると，c:¥temp¥jaxws¥works¥dispatch_provider¥server¥WEB-INF¥classes¥com¥sample¥ディレクトリに，UserInfoImpl.class が生成されます。

9.3.3　web.xml を作成する

WAR ファイルの構成要素として必要な web.xml を作成します。

web.xml の作成例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/we
b-app_3_0.xsd">
  <description>Sample web service &quot;dispatch_provider&quot;</description>
  <display-name>Sample_web_service_dispatch_provider</display-name>
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/UserInfoService</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

バージョン 2.5 の web.xml を作成する場合は，web-app 要素の version 属性を"2.5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"にしてください。

作成した web.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥dispatch_provider¥server¥WEB-INF ディレクトリに保存します。web.xml の設定項目については，「3.4　web.xml の作成」を参照してください。

9.3.4　application.xml を作成する

EAR ファイルの構成要素として必要な application.xml を作成します。application.xml の作成例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/application_6.xsd">

  <description>Sample application &quot;dispatch_provider&quot;</description>
  <display-name>Sample_application_dispatch_provider</display-name>
  <module>
    <web>
      <web-uri>dispatch_provider.war</web-uri>
      <context-root>dispatch_provider</context-root>
    </web>
  </module>
</application>
```

バージョン 5 の application.xml を作成する場合は，application 要素の version 属性を"5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/application_5.xsd"にしてください。

作成した application.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥dispatch_provider¥server¥META-INF ディレクトリに保存します。application.xml を作成するときの注意事項については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「5.2.2　application.xml 編集時の注意事項」を参照してください。

9.3.5　EAR ファイルを作成する

jar コマンドを使用して，これまでに作成したファイルを含めた EAR ファイルを作成します。

EAR ファイルの作成例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥dispatch_provider¥server¥
> jar cvf dispatch_provider.war .¥WEB-INF
> jar cvf dispatch_provider.ear .¥dispatch_provider.war .¥META-INF¥application.xml
```

jar コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥dispatch_provider¥server¥ディレクトリに dispatch_provider.ear が作成されます。

jar コマンドについては，JDK のドキュメントを参照してください。

9.4　デプロイと開始の例（プロバイダ起点・SAAJ）

ここでは，プロバイダを起点とした場合のデプロイと開始の例を説明します。

9.4.1　EAR ファイルをデプロイする

cjimportapp コマンドを使用して，作成した EAR ファイルを J2EE サーバにデプロイします。

デプロイの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥dispatch_provider¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjimportapp" jaxwsserver -nameserver corbaname::testserver
:900 -f dispatch_provider.ear
```

cjimportapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションをデプロイ（インポート）する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.3　J2EE アプリケーションのインポート」を参照してください。

9.4.2　Web サービスを開始する

cjstartapp コマンドを使用して，Web サービスを開始します。

Web サービスを開始する例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥dispatch_provider¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjstartapp" jaxwsserver -nameserver corbaname::testserver:
900 -name Sample_application_dispatch_provider
```

cjstartapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartapp（J2EE アプリケーションの開始）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションを開始する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.1　J2EE アプリケーションの開始」を参照してください。

9.5　Web サービスクライアントの開発例（プロバイダ起点・SAAJ）

ここでは，プロバイダを起点とし，SAAJ を利用した場合の Web サービスクライアントの開発例を説明します。

9.5.1　Web サービスクライアントの実装クラスを作成する

Web サービスを利用する Web サービスクライアントの実装クラスを作成します。

Web サービスに対して 1 回の呼び出しをする，ディスパッチベースの Web サービスクライアント com.sample.client.TestClient の作成例を次に示します。

なお，ディスパッチベースの Web サービスクライアントでは，SOAP バインディングのバージョンを明示する必要があるため，例では SOAP 1.1/HTTP バインディングを指定しています。SOAP 1.2 の場合は，SOAPBinding.SOAP11HTTP_BINDING の部分を SOAPBinding.SOAP12HTTP_BINDING に読み替えてください。

```
package com.sample.client;

import java.io.File;
import java.util.Iterator;
import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.ws.soap.SOAPBinding;
import javax.xml.soap.AttachmentPart;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPBodyElement;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Service;

public class TestClient {
    public static void main( String[] args ) {
        // サービス生成
        QName port = new QName( "http://sample.com", "UserInfoPort" );
        Service service = Service.create(
            new QName("http://sample.com", "UserInfoService"));
        String serviceURL = "http://webhost:8085/dispatch_provider/UserInfoService";

        // サービスにポートを追加
        service.addPort( port, SOAPBinding.SOAP11HTTP_BINDING, serviceURL );

        // Dispatchオブジェクト生成
        Dispatch<SOAPMessage> dispatch = service.createDispatch(
            port, SOAPMessage.class, Service.Mode.MESSAGE );

        // 要求メッセージ
```

```
        SOAPMessage request = null;

        try{
            // 要求メッセージの生成
            request = MessageFactory.newInstance().createMessage();
            SOAPBody reqSoapBody = request.getSOAPBody();

            // 社員番号を設定
            SOAPBodyElement requestRoot= soapBody.addBodyElement(
                new QName("http://sample.com", "number"));
            SOAPElement soapElement = requestRoot.addChildElement(
                new QName("http://sample.com", "value"));
            soapElement.addTextNode( "1234" );

            // 添付ファイル(顔写真)を設定
            String filePath = "C:¥¥attachment.jpg";
            FileDataSource fds = new FileDataSource(filePath);
            AttachmentPart apPart =
                request.createAttachmentPart(new DataHandler(fds));
            request.addAttachmentPart(apPart);

            // SOAPメッセージの送受信
            SOAPMessage response = dispatch.invoke( request );

            // 応答メッセージからデータを取得
            SOAPBody resSoapBody = response.getSOAPBody();
            SOAPBodyElement resRoot =
                (SOAPBodyElement)resSoapBody.getChildElements().next();
            Iterator iterator = resRoot.getChildElements();
            String result =


((SOAPElement)iterator.next()).getFirstChild().getNodeValue();

            // 登録確認メッセージの表示
            System.out.println( "[RESULT] " + result );
        } catch( SOAPException e ) {
            e.printStackTrace();
      }
   }
}
```

作成した TestClient.java は，UTF-8 形式で
c:¥temp¥jaxws¥works¥dispatch_provider¥client¥src¥com¥sample¥client¥ディレクトリに保存します。

9.5.2 Web サービスクライアントの実装クラスをコンパイルする

javac コマンドを使用して，作成した Web サービスクライアントをコンパイルします。

コンパイルの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥dispatch_provider¥client¥
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥ja
```

```
vaee¥1100¥lib¥javaee-api.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;.¥classes” -d .¥classes
src¥com¥sample¥client¥TestClient.java
```

javac コマンドが正常に終了すると，
c:¥temp¥jaxws¥works¥dispatch_provider¥client¥classes¥com¥sample¥client¥ディレクトリに，
TestClient.class が生成されます。

javac コマンドについては，JDK のドキュメントを参照してください。

9.6　Web サービスの実行例（プロバイダ起点・SAAJ）

ここでは，プロバイダを起点とし，SAAJ を利用した場合の Web サービスの実行例を説明します。

9.6.1　Java アプリケーション用オプション定義ファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用オプション定義ファイル（usrconf.cfg）を作成します。

Java アプリケーション用オプション定義ファイルの作成例を次に示します。

```
add.class.path=<Application Serverのインストールディレクトリ>¥jaxws¥lib¥cjjaxws.jar
add.class.path=.¥classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home=<Application Serverのインストールディレクトリ>
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

<Application Server のインストールディレクトリ>の部分は，Application Server をインストールしているパスを絶対パスで指定します。<PRF ID>の部分は，PRF デーモンの識別子を指定します。

作成した Java アプリケーション用オプション定義ファイルは，c:¥temp¥jaxws¥works¥dispatch_provider¥client¥ディレクトリに保存します。Java アプリケーション用オプション定義ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.1　usrconf.cfg（Java アプリケーション用オプション定義ファイル）」を参照してください。

9.6.2　Java アプリケーション用ユーザプロパティファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用ユーザプロパティファイルを作成します。

ここでは特に設定を変更しないため，c:¥temp¥jaxws¥works¥dispatch_provider¥client¥ディレクトリに usrconf.properties という名称の空のファイルを作成します。Java アプリケーション用ユーザプロパティファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.2　usrconf.properties（Java アプリケーション用ユーザプロパティファイル）」を参照してください。

9.6.3　Web サービスクライアントを実行する

cjclstartap コマンドを使用して，Web サービスクライアントを実行します。

Web サービスクライアントの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥dispatch_provider¥client¥
> "%COSMINEXUS_HOME%¥CC¥client¥bin¥cjclstartap"  com.sample.client.TestClient
```

cjclstartap コマンドが正常に終了すると，Web サービスクライアントの実行結果が表示されます。実行結果の表示例を次に示します。

```
KDJE40053-I The cjclstartap command will now start. (directory for the user definition file
= c:¥temp¥jaxws¥works¥dispatch_provider¥client, PID = 2636)
[RESULT] Success.
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

イタリック体になっている個所は，実行したタイミングや環境によって変わります。

cjclstartap コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjclstartap（Java アプリケーションの開始)」を参照してください。

10 JAX-WS 機能の設定と動作

この章では，SOAP Web サービスを開発，運用するときの JAX-WS 機能の各種設定と，運用に当たって理解が必要な JAX-WS エンジンの動作について説明します。

10.1　動作定義ファイル

ログやタイムアウトの設定などは動作定義ファイルに記述します。動作定義ファイルは，次の 2 種類があります。

- **共通定義ファイル**
  システム共通の動作を設定するための定義ファイルです。一つだけあります。
- **プロセス別の定義ファイル**
  プロセス固有の動作を設定する場合に作成する定義ファイルです。固有の設定が必要なプロセスごとに作成します。例えば，J2EE サーバごとに設定を変更したい場合や，Web サービスクライアントに設定を変更したい場合に作成します。

また，一部の定義については，Web サービスクライアントにメッセージコンテキストを取得して定義できます。メッセージコンテキストとして指定できる定義については，「19.2.5(1)　メッセージコンテキストのプロパティのサポート範囲」を参照してください。

ここでは，動作定義ファイルの記述規則，および各定義ファイルの設定について説明します。

10.1.1　動作定義ファイルの記述規則

動作定義ファイルの記述形式および記述規則について説明します。また，動作定義の優先度を示します。

(1)　記述形式

動作定義ファイルは，次のようにキーを指定します。

<キー名称>=<値>

(2)　記述規則

動作定義ファイルは，次に示す規則に従って記述してください。

- 改行までが<値>になります。
- #で始まる行はコメントと見なされます。
- <値>の後ろにコメントは追加できません。追加した場合，コメントまでが値と解釈されます。
- 記載する文字は Java の仕様に従って，ISO 8859-1 文字エンコーディングを使用してください。2 バイト文字などは不正な文字列に解釈されるので，native2ascii コマンドで変換してください。native2ascii コマンドについては，JDK のドキュメントを参照してください。
- <値>にはスペースも指定できます。
- <キー名称>と=の間，および=と<値>の間にスペースを入れた場合，スペースが取り除かれて解釈されます。

- 設定できる<キー名称>以外の<キー名称>を設定した場合，その<キー名称>は使用されません（警告やエラーは出力されません）。
- 値がない行を指定した場合，デフォルト値が仮定されます。
- キー名称は大文字／小文字を区別します。

(3) 動作定義の優先度

動作定義の優先順位は次のとおりです。

1. Web サービスクライアント（メッセージコンテキスト）に定義
2. プロセス別の定義ファイル
3. 共通定義ファイル

Web サービスクライアントに定義する場合の例として，Web サービスクライアントに接続タイムアウト値を設定するコードを示します。

```
// setConnectTimeout()
int timeout = 60000;
Map<String, Object> ctxt = ((BindingProvider)port).getRequestContext();
ctxt.put("com.cosminexus.jaxws.connect.timeout", timeout);
```

10.1.2 共通定義ファイルの設定項目

共通定義ファイルを使用して，システム共通の動作定義を設定します。共通定義ファイルのファイル名，保存ディレクトリ名，および設定項目について説明します。

(1) ファイル名

共通定義ファイルのファイル名を示します。

cjwconf.properties

(2) 保存先ディレクトリ

共通定義ファイルの保存先ディレクトリを示します。保存先ディレクトリは固定です。

<Application Server のインストールディレクトリ>¥jaxws¥conf

(3) 設定項目

設定するキー名称と指定内容の一覧を次の表に示します。

表 10–1　共通定義ファイルの設定項目

項番	設定項目	キー名称	指定内容	デフォルト値	コンテキスト
1	稼働ログの出力レベル	com.cosminexus.jaxws.logger.runtime.message.level	稼働ログの出力レベルを指定します。 ERROR，WARN，INFO，DEBUG，NONE のどれかを指定します。各指定値に対応した出力内容については，「39.3.4　ログの重要度と出力条件」を参照してください。	INFO	−
2	稼働ログの面数	com.cosminexus.jaxws.logger.runtime.message.file_num	稼働ログの面数を指定します。 数字（1～16）を指定します。	2	−
3	稼働ログの容量	com.cosminexus.jaxws.logger.runtime.message.file_size	稼働ログの容量を指定します。 4096～16777216 の数字（単位：バイト）を指定します。	2097152	−
4	保守ログの出力	com.cosminexus.jaxws.logger.runtime.maintenance.level	保守ログを出力するかどうかを指定します。 ALL を指定した場合 　保守ログが出力されます。 NONE を指定した場合 　保守ログが出力されません。	ALL	−
5	保守ログの面数	com.cosminexus.jaxws.logger.runtime.maintenance.file_num	保守ログの面数を指定します。 1～16 の数字を指定します。	2	−
6	保守ログの容量	com.cosminexus.jaxws.logger.runtime.maintenance.file_size	保守ログの容量を指定します。 4096～16777216 の数字（単位：バイト）を指定します。	16777216	−
7	例外ログの出力レベル	com.cosminexus.jaxws.logger.runtime.exception.level	例外ログの出力レベルを指定します。 ERROR，WARN，INFO，DEBUG，NONE のどれかを指定します。各指定値に対応した出力内容については，「39.3.4　ログの重要度と出力条件」を参照してください。	INFO	−
8	例外ログの面数	com.cosminexus.jaxws.logger.runtime.exception.file_num	例外ログの面数を指定します。1～16 の数字を指定します。	2	−
9	例外ログの容量	com.cosminexus.jaxws.logger.runtime.exception.file_size	例外ログの容量を指定します。4096～16777216 の数字（単位：バイト）を指定します。	16777216	−
10	通信ログの出力レベル （Web サービスクライアント側）	com.cosminexus.jaxws.logger.runtime.transport.client_dump	Web サービスクライアント側での通信ログの出力レベルを指定します。	ERROR_HEADER	−

項番	設定項目	キー名称	指定内容	デフォルト値	コンテキスト
			NONE を指定した場合 Web サービスクライアントでは通信ログが出力されません。 ALL を指定した場合 Web サービスクライアントで送受信したメッセージが常に通信ログに出力されます。 HEADER を指定した場合 Web サービスクライアントで受信したメッセージの HTTP ヘッダが常に通信ログに出力されます。 ERROR_HEADER を指定した場合 SOAPFault を受信した場合に，受信したメッセージの HTTP ヘッダが通信ログに出力されます。 **注意事項** ALL を指定した場合，送受信するメッセージの長さによっては java.lang.OutOfMemoryError 例外が発生することがあります。その場合は JVM のヒープサイズを調整してください。		
11	通信ログの出力レベル （Web サービス側）	com.cosminexus.jaxws.logger.runtime.transport.server_dump	Web サービス側での通信ログの出力レベルを指定します。 NONE を指定した場合 Web サービス側では通信ログが出力されません。 ALL を指定した場合 Web サービス側で送受信したメッセージが常に通信ログに出力されます。 HEADER を指定した場合 Web サービス側で受信したメッセージの HTTP ヘッダが常に通信ログに出力されます。 ERROR_HEADER SOAPFault を送信する場合に，受信したメッセージの HTTP ヘッダが通信ログに出力されます。 なお，受信時のメッセージを出力する場合は，HTTP のリクエスト情報も出力されます。	ERROR_HEADER	－

項番	設定項目	キー名称	指定内容	デフォルト値	コンテキスト
			注意事項 ALL を指定した場合，送受信するメッセージの長さによっては java.lang.OutOfMemoryError 例外が発生することがあります。その場合は JavaVM のヒープサイズを調整してください。		
12	通信ログの面数	com.cosminexus.jaxws.logger.runtime.transport.file_num	通信ログの面数を指定します。1～16 の数字を指定します。	2	－
13	通信ログの容量	com.cosminexus.jaxws.logger.runtime.transport.file_size	通信ログの容量を指定します。4096～16777216 の数字（単位：バイト）を指定します。	16777216	－
14	通信ログの文字エンコーディング	com.cosminexus.jaxws.logger.runtime.transport.encoding	通信ログの文字エンコーディングを指定します。J2SE 6.0 でサポートされている文字エンコーディングについては，J2SE 6.0 のドキュメントを参照してください。 "DEFAULT"を指定した場合，通信ログの文字エンコーディングはデフォルトのプラットフォームエンコーディングとなります。	DEFAULT	－
15	稼働ログの出力レベル（cjwsimport）	com.cosminexus.jaxws.logger.cjwsimport.message.level	cjwsimport コマンドの稼働ログの出力レベルを指定します。ERROR，WARN，INFO，DEBUG，NONE のどれかを指定します。各指定値に対応した出力内容については，「39.3.4　ログの重要度と出力条件」を参照してください。	INFO	－
16	稼働ログの面数（cjwsimport）	com.cosminexus.jaxws.logger.cjwsimport.message.file_num	cjwsimport コマンドの稼働ログの面数を指定します。1～16 の数字を指定します。	2	－
17	稼働ログの容量（cjwsimport）	com.cosminexus.jaxws.logger.cjwsimport.message.file_size	cjwsimport コマンドの稼働ログの容量を指定します。4096～16777216 の数字（単位：バイト）を指定します。	2097152	－
18	例外ログの出力レベル（cjwsimport）	com.cosminexus.jaxws.logger.cjwsimport.exception.level	cjwsimport コマンドの例外ログの出力レベルを指定します。ERROR，WARN，INFO，DEBUG，NONE のどれかを指定します。各指定値に対応した出力内容については，「39.3.4　ログの重要度と出力条件」を参照してください。	INFO	－

項番	設定項目	キー名称	指定内容	デフォルト値	コンテキスト
19	例外ログの面数（cjwsimport）	com.cosminexus.jaxws.logger.cjwsimport.exception.file_num	cjwsimport コマンドの例外ログの面数を指定します。1～16 の数字を指定します。	2	−
20	例外ログの容量（cjwsimport）	com.cosminexus.jaxws.logger.cjwsimport.exception.file_size	cjwsimport コマンドの例外ログの容量を指定します。4096～16777216 の数字（単位：バイト）を指定します。	16777216	−
21	保守ログの出力（cjwsimport）	com.cosminexus.jaxws.logger.cjwsimport.maintenance.level	cjwsimport コマンドの保守ログを出力するかどうかを指定します。 ALL を指定した場合 保守ログが出力されます。 NONE を指定した場合 保守ログが出力されません。	ALL	−
22	保守ログの面数（cjwsimport）	com.cosminexus.jaxws.logger.cjwsimport.maintenance.file_num	cjwsimport コマンドの保守ログの面数を指定します。1～16 の数字を指定します。	2	−
23	保守ログの容量（cjwsimport）	com.cosminexus.jaxws.logger.cjwsimport.maintenance.file_size	cjwsimport コマンドの保守ログの容量を指定します。4096～16777216 の数字（単位：バイト）を指定します。	16777216	−
24	SOAPAction がない場合の動作オプション	com.cosminexus.jaxws.fault_omit_soapaction	SOAPAction がない場合の動作オプションを指定します。 true を指定した場合 SOAPAction ヘッダがない場合，SOAP フォルトメッセージを返します。 false を指定した場合 SOAPAction ヘッダがない場合，SOAPAction ヘッダの値が空文字として動作します。	true	−
25	クライアントソケットの接続タイムアウト値	com.cosminexus.jaxws.connect.timeout	クライアントソケットの接続タイムアウト値を指定します。 このプロパティで指定したタイムアウト値は，Web サービス呼び出し時と，Web サービス呼び出し前の javax.xml.ws.Service クラス生成で発生するメタデータ（WSDL）取得時に適用されます。 0～2147483647 の数字（単位：ミリ秒）を指定します。0 を指定した場合，タイムアウトされません。 OS の TCP 接続に関連する設定を変更している場合，OS の設定値が優先されることがあります。	60000	○※

項番	設定項目	キー名称	指定内容	デフォルト値	コンテキスト
26	クライアントソケットの読み込みタイムアウト値	com.cosminexus.jaxws.request.timeout	クライアントソケットの読み込みタイムアウト値を指定します。 このプロパティで指定したタイムアウト値は，Web サービス呼び出し時と，Web サービス呼び出し前の javax.xml.ws.Service クラス生成で発生するメタデータ（WSDL）取得時に適用されます。 0～2147483647 の数字（単位：ミリ秒）を指定します。0 を指定した場合，タイムアウトされません。 OS の TCP 接続に関連する設定を変更している場合，OS の設定値が優先されることがあります。	300000	○※
27	ベーシック認証のユーザ ID	javax.xml.ws.security.auth.username	HTTP 接続で使用するベーシック認証のユーザ ID を指定します。 このプロパティで指定したユーザ ID は，Web サービス呼び出し時と，Web サービス呼び出し前の javax.xml.ws.Service クラス生成で発生するメタデータ（WSDL）取得時に適用されます。	なし	○※
28	ベーシック認証のパスワード	javax.xml.ws.security.auth.password	HTTP 接続で使用するベーシック認証のパスワードを指定します。 このプロパティで指定したパスワードは，Web サービス呼び出し時と，Web サービス呼び出し前の javax.xml.ws.Service クラス生成で発生するメタデータ（WSDL）取得時に適用されます。 javax.xml.ws.security.auth.username を指定していない場合，有効になりません。	なし	○※
29	HTTP セッションの維持の有無	javax.xml.ws.session.maintain	HTTP セッションの維持の有無を指定します。HTTP セッションは同じポートオブジェクトを使用している範囲で維持されます。 true の場合 セッションが維持されます。 false の場合 セッションが維持されません。	false	○※
30	ハンドラチェイン設定ファイルの厳密な検証	com.cosminexus.jaxws.validation.handlerchain.strict	ハンドラチェイン設定ファイルの検証を厳密に行うかどうかを指定します。ハンドラチェイン設定ファイルの	false	－

項番	設定項目	キー名称	指定内容	デフォルト値	コンテキスト
			javaee:handler 要素の子要素として javaee:handler-name 要素が記述されていないとエラーになります。		
31	HTTP リクエストに対する WSDL の発行	com.cosminexus.jaxws.security.publish_wsdl	Web サービスの URL に対して，クエリストリングが?wsdl，または?WSDL である HTTP GET リクエストが送信された場合，その Web サービスの WSDL を発行するかどうかを指定します。 true を指定した場合 WSDL が発行されます。 false を指定した場合 WSDL が発行されないで，405 Method Not Allowed が返されます。	true	−
32	HTTP リクエストに対する Web サービスの情報表示	com.cosminexus.jaxws.security.display_webservice_info	Web サービスに対して，GET による HTTP リクエストが到着したとき，レスポンスとして，Web サービスの情報を表示するかどうかを指定します。 true を指定した場合 そのリクエスト URL に対応する Web サービスの情報が表示されます。 false を指定した場合 405 Method Not Allowed が返されます。	false	−
33	Web サービスで発生した Java 例外の伝搬	com.cosminexus.jaxws.fault.SOAPFaultBuilder.captureStackTrace	Web サービスで発生した Java 例外を Web サービスクライアントに伝搬するかどうかを指定します。 true を指定した場合 例外が伝搬されます。 false を指定した場合 例外が伝搬されません。	false	−
34	ログの出力先ディレクトリ	com.cosminexus.jaxws.tool.log.directory	cjwsimport コマンドが出力するログの出力先ディレクトリを指定します。	<Application Server のインストールディレクトリ>/jaxws/logs	−
35	プロキシサーバの認証ユーザ ID	com.cosminexus.jaxws.http.proxyUser	プロキシサーバの認証ユーザ ID を指定します。Web サービスクライアントが，プロキシサーバを使用して外部のネット	なし	−

項番	設定項目	キー名称	指定内容	デフォルト値	コンテキスト
			ワークにある Web サービスを呼び出す場合に，必要に応じて指定してください。 このプロパティで指定したユーザ ID は，Web サービス呼び出し時と，Web サービス呼び出し前の javax.xml.ws.Service クラス生成で発生するメタデータ（WSDL）取得時に適用されます。		
36	プロキシサーバの認証ユーザ ID に対応するパスワード	com.cosminexus.jaxws.http.proxyPassword	プロキシサーバの認証ユーザ ID に対応するパスワードを指定します。Web サービスクライアントが，プロキシサーバを使用して外部のネットワークにある Web サービスを呼び出す場合に，必要に応じて指定してください。 このプロパティで指定したパスワードは，Web サービス呼び出し時と，Web サービス呼び出し前の javax.xml.ws.Service クラス生成で発生するメタデータ（WSDL）取得時に適用されます。	なし	−
37	SSL プロトコルによる接続で使用するプロキシサーバの認証ユーザ ID	com.cosminexus.jaxws.https.proxyUser	SSL プロトコルによる接続で使用する，プロキシサーバの認証ユーザ ID を指定します。Web サービスクライアントが，プロキシサーバを使用して外部のネットワークにある Web サービスを呼び出す場合で，SSL プロトコルによる接続のときに，必要に応じて指定してください。 このプロパティで指定したユーザ ID は，Web サービス呼び出し時と，Web サービス呼び出し前の javax.xml.ws.Service クラス生成で発生するメタデータ（WSDL）取得時に適用されます。	なし	−
38	SSL プロトコルによる接続で使用するプロキシサーバの認証ユーザ ID に対応するパスワード	com.cosminexus.jaxws.https.proxyPassword	SSL プロトコルによる接続で使用する，プロキシサーバの認証ユーザ ID に対応するパスワードを指定します。Web サービスクライアントが，プロキシサーバを使用して外部のネットワークにある Web サービスを呼び出す場合で，SSL プロトコルによる接続のときに，必要に応じて指定してください。このプロパティで指定したユーザ ID は，Web サービス呼び出し時と，Web サービス呼び出し前の javax.xml.ws.Service クラス生成で発生するメタデータ（WSDL）取得時に適用されます。	なし	−

項番	設定項目	キー名称	指定内容	デフォルト値	コンテキスト
39	soap12:binding 要素の transport 属性の指定	com.cosminexus.jaxws.publish_wsdl.soap12binding	SOAP1.2 の場合に，クエリストリングが?wsdl または?WSDL である HTTP GET リクエストで発行された WSDL で，wsdl:binding 要素の子要素である soap12:binding 要素の transport 属性に，http://schemas.xmlsoap.org/soap/http を設定するかどうかを指定します。 DEFAULT soap12:binding/@transport 属性値を "http://www.w3.org/2003/05/soap/bindings/HTTP/"にします。 WSI_BP20_TRANSPORT soap12:binding/@transport 属性値を "http://schemas.xmlsoap.org/soap/http"にします。	DEFAULT	－
40	SSL 接続におけるホスト名検証の有無	com.cosminexus.xml.ws.client.http.HostnameVerificationProperty	Web サービスクライアントが SSL プロトコルによる接続を行う場合，次に示す検証を省略するかどうかを指定します。 **検証内容：エンドポイントアドレスに含まれるホスト名と，証明書のホスト名が一致しているかどうか。** true 検証を省略します。 false 検証を省略しません。 このプロパティで指定した SSL 接続におけるホスト名検証の有無は，Web サービス呼び出し時と，Web サービス呼び出し前の javax.xml.ws.Service クラス生成で発生するメタデータ（WSDL）取得時に適用されます。	false	○※

（凡例）

○：メッセージコンテキストとして設定できることを示します。

－：メッセージコンテキストとして設定できないことを示します。

注※

メッセージコンテキストへの指定が有効になるのは Web サービス呼び出し時だけで，Web サービス呼び出し前の javax.xml.ws.Service クラス生成で発生するメタデータ（WSDL）取得時には適用されません。

メタデータ取得時にベーシック認証の情報，または SSL 接続におけるホスト名検証の有無を設定する場合は，共通定義ファイルまたはプロセス別の定義ファイルに記述するか，別途 WSDL をローカルマシンにダウンロードして使用してください（ローカルマシンにある WSDL を使用する場合は，メタデータ取得時にリモートマシンへの接続が発生しません）。WSDL から別途インポートされる WSDL がある場合は，インポートされる WSDL もローカルマシン上にダウンロードしてください。

また，このプロパティにメッセージコンテキストで true または false を設定する場合は，java.lang.String 型の文字列で設定してください。java.lang.String 型の文字列以外で設定した場合の動作は保証されません。設定方法については，「19.2.5(2)(k)　com.cosminexus.xml.ws.client.http.HostnameVerificationProperty プロパティの設定方法」を参照してください。数字を設定する場合は，int 型または java.lang.Integer 型で設定してください。int 型または java.lang.Integer 型の数字以外で設定した場合の動作は保証されません。

(4)　設定を変更する場合

Web サービス側の設定を変更する場合

プロセス別の定義ファイルを使用していないすべての J2EE サーバを停止してから，共通定義ファイルの設定を変更してください。プロセス別の定義ファイルについては，「10.1.3　プロセス別の定義ファイルの設定」を参照してください。

ログ関連の設定を変更する場合，必要に応じてログを退避してから設定を変更してください。

Web サービスクライアント側の設定を変更する場合

すべての J2EE サーバ（Web サービスクライアントを J2EE アプリケーションとした場合），またはすべての Java アプリケーションを停止してから，共通定義ファイルの設定を変更してください。

ログ関連の設定を変更する場合，必要に応じてログを退避してから設定を変更してください。

10.1.3　プロセス別の定義ファイルの設定

プロセス別に固有の定義をする場合に，プロセス別の定義ファイルを作成します。

プロセス別の定義ファイルのファイル名，および保存先のディレクトリ名は任意です。システムプロパティで保存先のパスを指定することで，プロセス別の定義が有効になります。プロセス別の定義ファイルの指定例を次に示します。

```
com.cosminexus.jaxws.confpath=d:/tmp/example.properties
```

プロセス別の定義を変更する場合，対象としているプロセス（J2EE アプリケーションまたは Java アプリケーション）を停止してから，プロセス別の定義ファイルの定義を変更してください。

ログ関連の定義を変更する場合，必要に応じてログを退避してから，定義を変更してください。

10.2　JAX-WS エンジンの動作

Application Server の JAX-WS エンジンの動作，および Application Server の JAX-WS エンジンのサポート範囲について説明します。

10.2.1　JAX-WS エンジンの動作とサポート範囲

Web サービス側と Web サービスクライアント側でやり取りされる SOAP メッセージは，JAX-WS エンジンの動作によってマーシャル／アンマーシャルされます。

ここでは，Web サービス側および Web サービスクライアント側の JAX-WS エンジンとサポート範囲について説明します。

(1)　Web サービス側の JAX-WS エンジンの動作とサポート範囲

Web サービス側の JAX-WS エンジンの動作について説明し，Web サービスを利用する Web サービスクライアントのサポート範囲について示します。

(a)　Web サービス側の JAX-WS エンジンの動作

Web サービス側の JAX-WS エンジンは，次に示すような流れで動作します。

- Web サービスクライアント側から POST HTTP メソッドによって SOAP 要求メッセージを受け付け，アンマーシャルして Java オブジェクトに変換する。
- 対象となる Web サービス実装クラスまたはプロバイダ実装クラスを見つけ出し（ディスカバリ），オペレーションに対応するメソッドを呼び出す（ディスパッチ）。
- 対象となる Web サービス実装クラスまたはプロバイダ実装クラスから SOAP 応答メッセージやフォルトメッセージを表現する Java オブジェクトを受け取り，マーシャルして SOAP 応答メッセージまたはフォルトメッセージとして呼び出し元に返す。

ディスカバリとディスパッチについては，「10.2.2　ディスカバリとディスパッチ」を参照してください。

また，Web サービス側の JAX-WS エンジンは，HTTP GET メソッドによって Web サービスのメタデータである WSDL を要求された場合，WAR ファイルに WSDL がなければ自動的に生成して返します。メタデータの発行については，「10.6　メタデータの発行」を参照してください。

なお，POST でも GET でもない HTTP メソッドで Web サービス側の JAX-WS エンジンが呼び出された場合，HTTP ステータスコード 405 Method Not Allowed を返します。

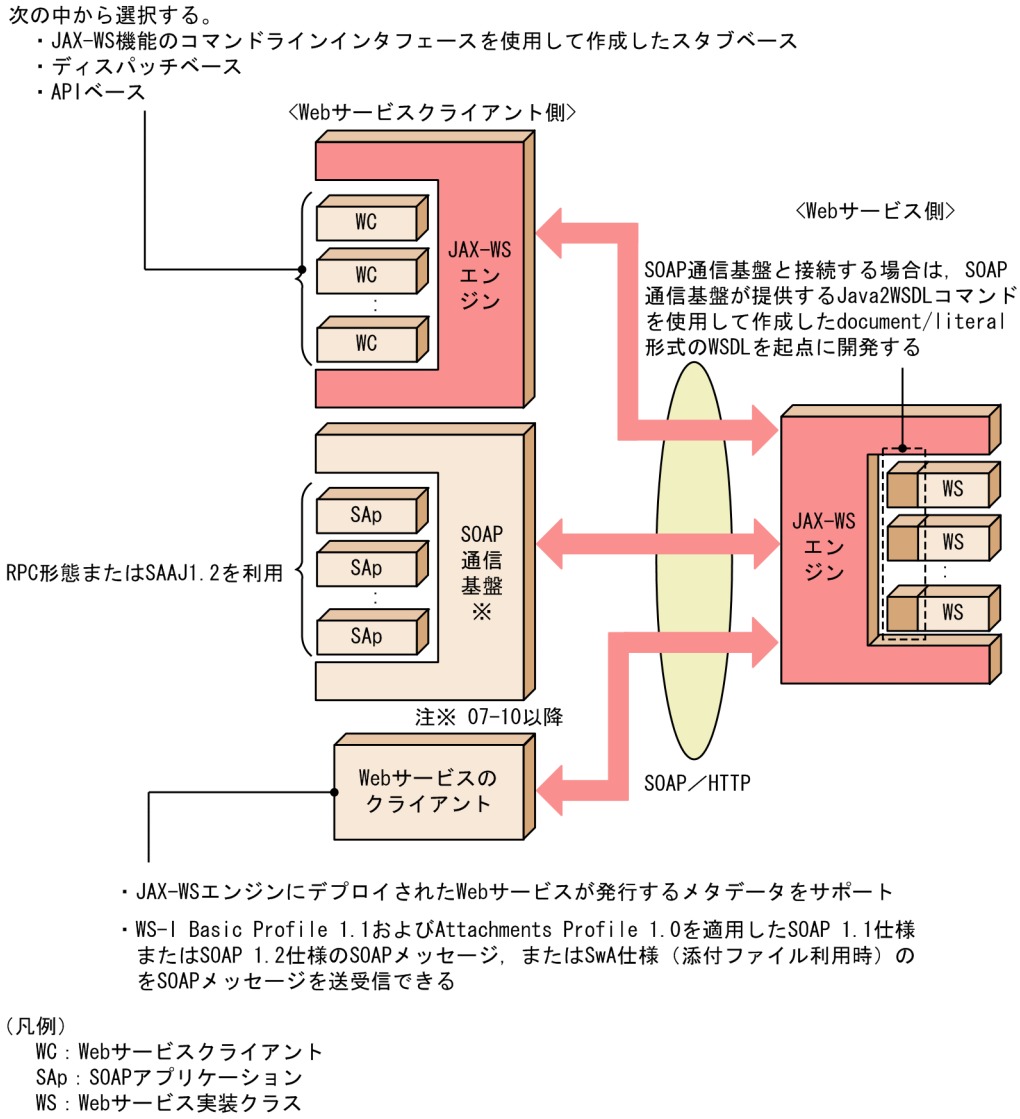
(b)　Web サービス側の JAX-WS エンジンのサポート範囲

Web サービス側の JAX-WS エンジンと Web サービスクライアントの関係について説明します。

Web サービス実装クラスの場合

Web サービス実装クラスの場合の，Web サービス側の JAX-WS エンジンと Web サービスクライアントの関係を次に示します。

図 10-1　Web サービス側の JAX-WS エンジンと Web サービスクライアントの関係（Web サービス実装クラスの場合）



Web サービス側の JAX-WS エンジンが受信できるメッセージ，および接続元の Web サービスクライアントの条件を次に示します。

- Application Server の JAX-WS 機能で動作する Web サービスクライアント

  Application Server の JAX-WS 機能が提供しているコマンドで開発し，Application Server の JAX-WS 機能で動作する Web サービスクライアントを利用できます。接続先の Application Server の JAX-WS 機能が以前のバージョンの場合，そのバージョンでサポートしている機能だけを使用できます。

- SOAP 通信基盤で動作する，RPC 形態の SOAP アプリケーションのクライアント SOAP アプリケーション開発支援機能で開発し，SOAP 通信基盤で動作する RPC 形態の SOAP アプリケーションのクライアントを利用できます。

  開発時の SOAP アプリケーション開発支援機能および SOAP 通信基盤のバージョンは，07-10 以降である必要があります。また，この場合の Web サービスは，SOAP 通信基盤の Java2WSDL コマンドで生成した，document/literal 形式の WSDL を起点として開発している必要があります。

- そのほかの Web サービスクライアント

  Application Server の JAX-WS 機能で動作する Web サービスが発行するメタデータ（WSDL）をサポートし，WS-I Basic Profile 1.1，および Attachments Profile 1.0 を適用した次のどれかの仕様に従った SOAP メッセージ※を送受信できる Web サービスクライアントを利用できます。

  - SOAP 1.1 仕様
  - SOAP 1.2 仕様
  - SwA 仕様（wsi:swaRef 形式の添付ファイルを利用する場合）
  - MTOM/XOP 仕様（MTOM/XOP 仕様形式の添付ファイルを利用する場合）

  WSDL のサポート範囲については，「20.1　WSDL 1.1 仕様のサポート範囲」を参照してください。

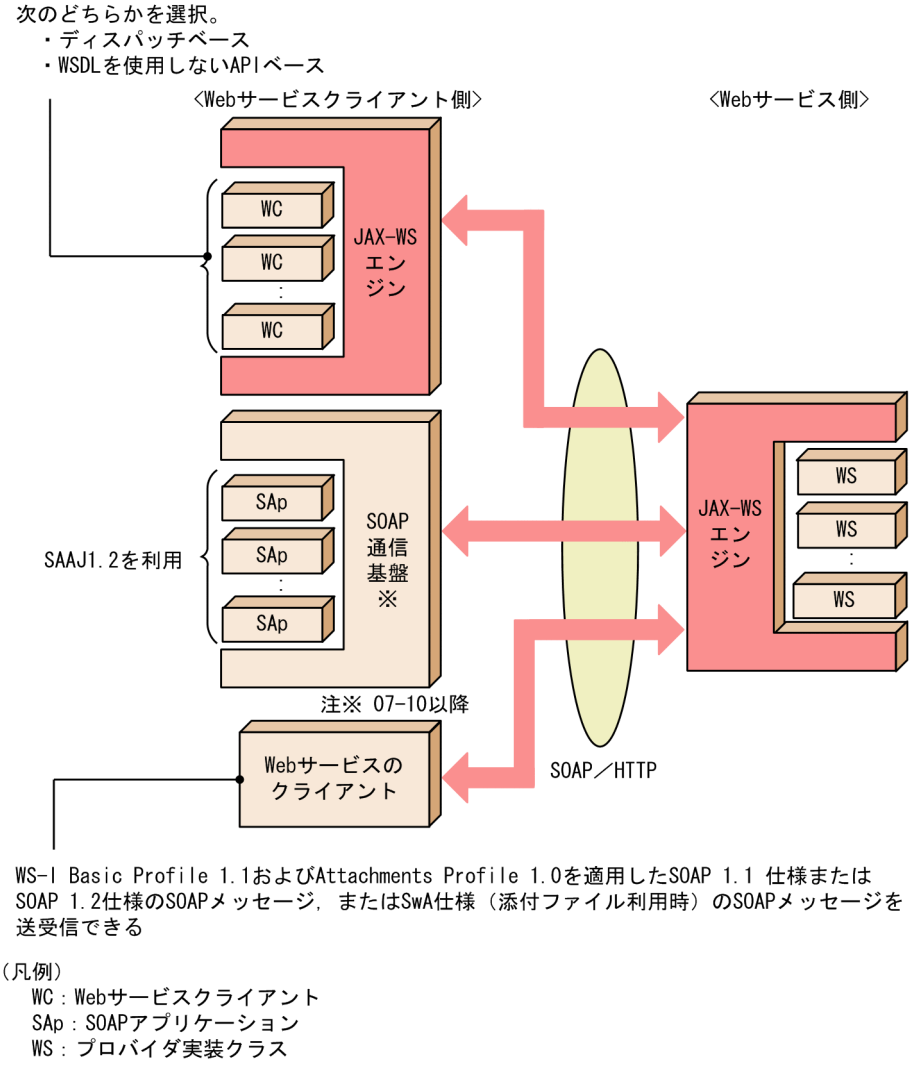
  注※

  標準仕様の性質から，WS-I Basic Profile 1.1 および Attachments Profile 1.0 を適用した SOAP 1.1 仕様，SOAP 1.2 仕様，SwA 仕様，または MTOM/XOP 仕様でもまだあいまいな部分が残ります。したがって，相互接続性について十分に検証してから運用してください。

プロバイダ実装クラスの場合

プロバイダ実装クラスの場合の，Web サービス側の JAX-WS エンジンと Web サービスクライアントの関係を次に示します。

図 10-2　Web サービス側の JAX-WS エンジンと Web サービスクライアントの関係（プロバイダ実装クラスの場合）



Web サービス側の JAX-WS エンジンが受信できるメッセージ，および接続元の Web サービスクライアントの条件を次に示します。

- Application Server の JAX-WS 機能で動作する Web サービスクライアント

  Application Server の JAX-WS 機能が提供している API で開発し，Application Server の JAX-WS 機能で動作する Web サービスクライアントを利用できます。接続先の Application Server の JAX-WS 機能が以前のバージョンの場合，そのバージョンでサポートしている機能だけを使用できます。
- Application Server の JAX-WS 機能で動作するディスパッチベースの Web サービスクライアント
- SAAJ 1.2 仕様を利用した SOAP アプリケーションのクライアント

  SOAP アプリケーション開発支援機能で開発し，SOAP 通信基盤で動作する SAAJ 1.2 仕様を利用した SOAP アプリケーションのクライアントを利用できます。

  開発時の SOAP アプリケーション開発支援機能および SOAP 通信基盤のバージョンは，07-10 以降である必要があります。また，SOAP アプリケーション開発支援機能と Application Server の JAX-WS

機能の両方でサポートする範囲の SOAP メッセージを送受信する SOAP アプリケーションである必要があります。

- そのほかの Web サービスクライアント

  WS-I Basic Profile 1.1 および Attachments Profile 1.0 を適用した，次のどれかの仕様に従った SOAP メッセージ※を送受信できる Web サービスクライアントを利用できます。

  - SOAP 1.1 仕様
  - SOAP 1.2 仕様
  - SwA 仕様（wsi:swaRef 形式の添付ファイルを利用する場合）

  注※

  標準仕様の性質から，WS-I Basic Profile 1.1 および Attachments Profile 1.0 を適用した SOAP 1.1 仕様，SOAP 1.2 仕様，または SwA 仕様でもまだあいまいな部分が残ります。したがって，相互接続性について十分に検証してから運用してください。プロバイダ実装クラスの場合，送受信できる SOAP メッセージの自由度が大きくなるため，特に注意してください。

(2) Web サービスクライアント側の JAX-WS エンジンの動作とサポート範囲

Web サービスクライアント側の JAX-WS エンジンの動作について説明し，Web サービスクライアントから利用できる Web サービスのサポート範囲について示します。

(a) Web サービスクライアント側の JAX-WS エンジンの動作

Web サービスクライアント側の JAX-WS エンジンは，次に示すような流れで動作します。

- Web サービスクライアントから，JAX-WS API を介して SOAP 要求メッセージを表現する Java オブジェクトを受け取る。
- 受け取った Java オブジェクトをマーシャルして，SOAP 要求メッセージとして送信する。
- 呼び出し先から SOAP 応答メッセージやフォルトメッセージを受信し，アンマーシャルして Web サービスクライアントに返す。

Web サービスクライアントは，生成されたクラスまたは JAX-WS API を介して JAX-WS エンジンにアクセスするため，JAX-WS エンジンを意識する必要はありません。

また，生成されたクラスおよび JAX-WS API は，JAX-WS 2.2 仕様に基づくため，Web サービスクライアントの実装者は，標準仕様以外のインタフェースを考慮する必要はありません。Web サービスの呼び出し（SOAP 要求メッセージの送信），および SOAP 応答メッセージおよびフォルトメッセージの受信は，標準仕様のインタフェースのサポート範囲内で行われます。

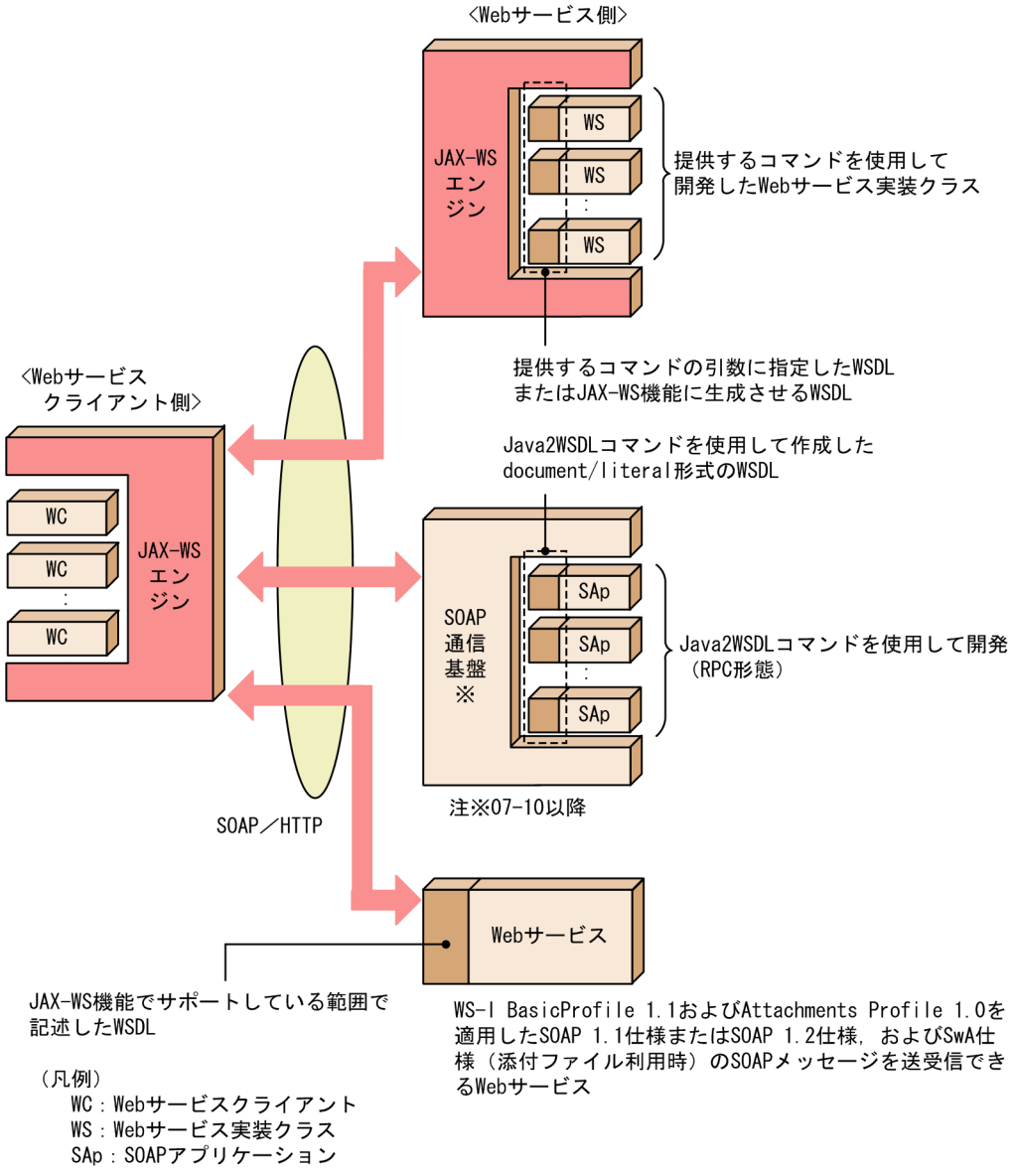
(b) Web サービスクライアント側の JAX-WS エンジンのサポート範囲

Web サービスクライアント側の JAX-WS エンジンと Web サービスの関係について説明します。

スタブベースの Web サービスクライアントの場合

スタブベースの Web サービスクライアントの場合の，Web サービスクライアント側の JAX-WS エンジンと Web サービスの関係を次に示します。

図 10-3　Web サービスクライアント側の JAX-WS エンジンと Web サービスの関係（スタブベースの場合）



Web サービスクライアント側の JAX-WS エンジンが呼び出せる Web サービスの条件を次に示します。

- Application Server の JAX-WS 機能が提供しているコマンドで開発した Web サービス実装クラス

  Application Server の JAX-WS 機能が提供しているコマンドで開発し，JAX-WS エンジン上にデプロイされた Web サービス実装クラスを呼び出せます。接続先の Application Server の JAX-WS 機能が以前のバージョンの場合，そのバージョンでサポートしている機能だけを使用できます。

- SOAP 通信基盤で動作する RPC 形態の SOAP アプリケーション

SOAP アプリケーション開発支援機能で開発し，SOAP 通信基盤にデプロイされた RPC 形態の SOAP アプリケーションを呼び出せます。

開発時の SOAP アプリケーション開発支援機能および SOAP 通信基盤のバージョンは，07-10 以降である必要があります。また，SOAP 通信基盤の Java2WSDL コマンドで生成した document/literal 形式の SOAP アプリケーションである必要があります。

- そのほかの Web サービス

  Application Server の JAX-WS 機能がサポートする範囲で記述された WSDL をメタデータとして公開し，WS-I Basic Profile 1.1 および Attachments Profile 1.0 を適用した次のどれかの仕様に従った SOAP メッセージ※を送受信できる Web サービスクライアントを利用できます。

  - SOAP 1.1 仕様
  - SOAP 1.2 仕様
  - SwA 仕様（wsi:swaRef 形式の添付ファイルを利用する場合）
  - MTOM/XOP 仕様（MTOM/XOP 仕様形式の添付ファイルを利用する場合）

  WSDL のサポート範囲については，「20.1　WSDL 1.1 仕様のサポート範囲」を参照してください。

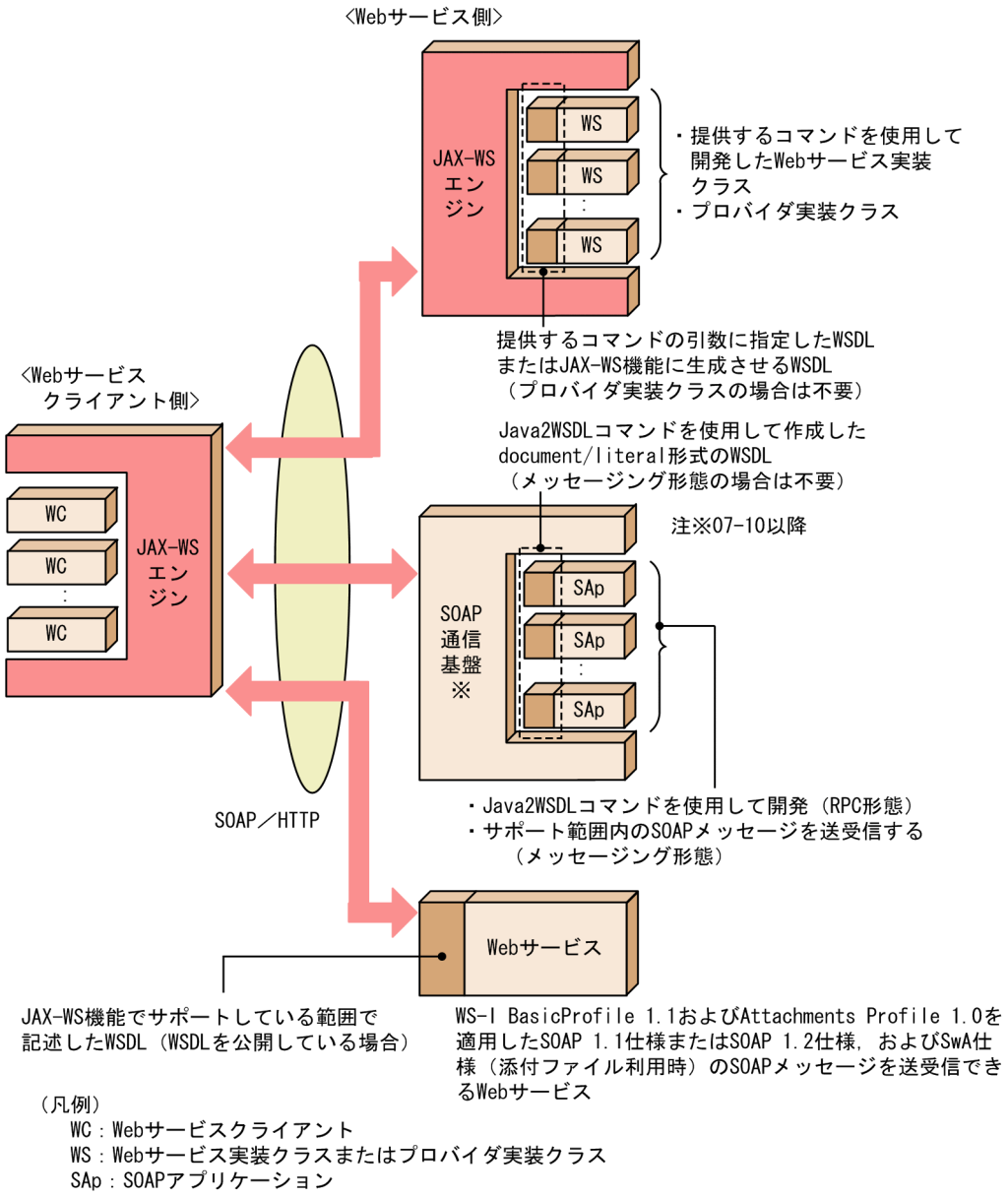
  注※

  標準仕様の性質から，WS-I Basic Profile 1.1 および Attachments Profile 1.0 を適用した SOAP 1.1 仕様，SOAP 1.2 仕様，SwA 仕様，または MTOM/XOP 仕様でもまだあいまいな部分が残ります。したがって，相互接続性について十分に検証してから運用してください。

ディスパッチベースの Web サービスクライアントの場合

ディスパッチベースの Web サービスクライアントの場合の，Web サービスクライアント側の JAX-WS エンジンと Web サービスの関係を次に示します。

図 10-4　Web サービスクライアント側の JAX-WS エンジンと Web サービスの関係（ディスパッチベースの場合）



Web サービスクライアント側の JAX-WS エンジンが呼び出せる Web サービスの条件を次に示します。

- Application Server の JAX-WS 機能が提供しているコマンドで開発した Web サービス実装クラス

  Application Server の JAX-WS 機能が提供しているコマンドで開発し，JAX-WS エンジン上にデプロイされた Web サービス実装クラスを呼び出せます。Application Server の JAX-WS 機能が以前のバージョンの場合，そのバージョンでサポートしている機能だけを使用できます。

- Application Server の JAX-WS 機能が提供しているコマンドで開発したプロバイダ実装クラス

  Application Server の JAX-WS 機能が提供しているコマンドで開発し，JAX-WS エンジン上にデプロイされたプロバイダ実装クラスを呼び出せます。

- SOAP 通信基盤で動作する RPC 形態の SOAP アプリケーション

SOAP アプリケーション開発支援機能で開発し，SOAP 通信基盤にデプロイされた RPC 形態の SOAP アプリケーションを呼び出せます。

開発時の SOAP アプリケーション開発支援機能および SOAP 通信基盤のバージョンは，07-10 以降である必要があります。また，SOAP 通信基盤の Java2WSDL コマンドで生成した document/literal 形式の SOAP アプリケーションである必要があります。

- SOAP 通信基盤で動作するメッセージング形態の SOAP アプリケーション

  SOAP アプリケーション開発支援機能で開発し，バージョン 07-10 以降の SOAP 通信基盤でデプロイしたメッセージング形態の SOAP アプリケーションのクライアントを利用できます。

  SOAP アプリケーション開発支援機能と Application Server の JAX-WS 機能の両方でサポートする範囲の SOAP メッセージを送受信する SOAP アプリケーションである必要があります。

- そのほかの Web サービス

  Application Server の JAX-WS 機能がサポートする範囲で記述された WSDL をメタデータとして公開し，WS-I Basic Profile 1.1 および Attachments Profile 1.0 を適用した次のどれかの仕様に従った SOAP メッセージ※を送受信できる Web サービスクライアントを利用できます。

  - SOAP 1.1 仕様
  - SOAP 1.2 仕様
  - SwA 仕様（wsi:swaRef 形式の添付ファイルを利用する場合）

  WSDL のサポート範囲については，「20.1　WSDL 1.1 仕様のサポート範囲」を参照してください。

  注※

  標準仕様の性質から，WS-I Basic Profile 1.1 および Attachments Profile 1.0 を適用した SOAP 1.1 仕様，SOAP 1.2 仕様，または SwA 仕様でもまだあいまいな部分が残ります。したがって，相互接続性について十分に検証してから運用してください。WSDL をメタデータとして公開していない場合，送受信できる SOAP メッセージの自由度が大きくなるので，特に注意してください。

(3)　配列または java.util.List を利用する場合の注意事項

SOAP メッセージを送信する場合，配列や java.util.List オブジェクトの要素数が 0 で空の状態と，null である状態は区別されないので，注意してください。

「10.2.1(3)(a)　SOAP メッセージを送信する場合」および「10.2.1(3)(b)　SOAP メッセージを受信する場合」では，JAX-WS エンジンの動作について説明します。また，特に Web サービス側も Web サービスクライアント側も Application Server の JAX-WS 機能を使用する場合の動作について「10.2.1(3)(c) Web サービス側も Web サービスクライアント側も Application Server の JAX-WS 機能を使用する場合」で説明します。

説明は次のメソッドを例にします。

```
@WebMethod
public List<String> test( List<Integer> param )
```

(a) SOAP メッセージを送信する場合

Web サービスクライアントの実装クラスが上記のメソッドを次のように呼び出した場合のリクエストメッセージについて説明します。

- 第 1 引数に null を与えてメソッドを呼び出した
- 第 1 引数に要素数が 0 である java.util.List の具象クラスのオブジェクトを与えてメソッドを呼び出した

この場合に，Web サービスクライアント側の JAX-WS エンジンが送信するリクエストメッセージの抜粋を次に示します。

```
<test/>
```

どちらの条件でメソッドを呼び出した場合も，param パラメタに対応する要素はリクエストメッセージには出現しません。Web サービスの実装クラスが戻り値を返す場合のレスポンスメッセージについても同様です。

(b) SOAP メッセージを受信する場合

次の表に示す条件のどれかが満たされている場合，「10.2.1(3)(a) SOAP メッセージを送信する場合」で説明したメッセージを受信した Web サービスクライアントや Web サービスの実装クラスは，要素数が 0 で空の状態である，配列や java.util.List の具象クラスのオブジェクトを受け取ります。

表 10–2 空の配列または java.util.List の具象クラスのオブジェクトを受け取る条件

項番	条件
1	Web サービスクライアントの実装クラスで java.util.List オブジェクトを操作する場合
2	WSDL を起点として開発した Web サービスの実装クラスで java.util.List オブジェクトを操作する場合
3	SEI を起点として開発した Web サービスの実装クラスで，WSDL のオペレーションに対応するメソッド（WebMethod アノテーションでアノテートされたメソッド）の引数に直接出現する配列や java.util.List を操作する場合

なお，SEI を起点として開発した Web サービス実装クラスで，WSDL のオペレーションに対応するメソッドの引数に出現する JavaBeans クラスが持つ配列や java.util.List のプロパティは，その JavaBeans クラスの実装に依存します。プロパティに対応する要素がリクエストメッセージにない場合，JavaBeans クラスは，null である配列や java.util.List の具象クラスのオブジェクトを受け取ります。

(c) Web サービス側も Web サービスクライアント側も Application Server の JAX-WS 機能を使用する場合

Application Server の JAX-WS エンジンの動作は，「10.2.1(3)(a) SOAP メッセージを送信する場合」および「10.2.1(3)(b) SOAP メッセージを受信する場合」で説明したとおりです。特に Web サービスクライアントおよび Web サービスの両方が Application Server 上にある場合，表 10-2 に示す条件を満たす Web サービスクライアントの実装クラスや Web サービスの実装クラスで，Web サービスクライア

ントおよびWebサービスのどちらか一方が配列やjava.util.Listオブジェクトとしてnullを送信しても，もう一方は要素数が0である配列やjava.util.Listの具象クラスを受信するので，注意してください。

10.2.2 ディスカバリとディスパッチ

JAX-WSエンジン上では，SOAPメッセージの送受信を実現するために，Webサービス実装クラスのディスカバリ，およびSOAPメッセージのディスパッチが行われます。

ここでは，Webサービス実装クラスのディスカバリ，SOAPメッセージのディスパッチ，およびフォルトと例外クラスのマッピングについて説明します。

また，インタフェースの透過性についても説明します。

(1) ディスカバリ

Webサービス側のJAX-WSエンジンによって，Webサービスクライアントから要求されたWebサービス実装クラスまたはプロバイダ実装クラスが見つけ出されます。このことを**ディスカバリ**といいます。

ディスカバリでは，SOAP要求メッセージで要求されたURLから，適切なWebサービス実装クラスをマッピングする処理が行われます。ここでは，次のURLが要求された場合のマッピングについて説明します。

http://example.org/fromwsdl/TestJaxWsService

コンテキストルートを"fromwsdl"とした場合，コンテキストルートの後ろの"/TestJaxWsService"（下線部）はパス情報を表します。このパス情報を基に，Webサービス実装クラスまたはプロバイダ実装クラスがマッピングされます。

パス情報とのマッピングについて，Webサービス実装クラスの場合の例を次に示します。

図 10-5　Web サービス実装クラスのディスカバリ（POJO の Web サービス）

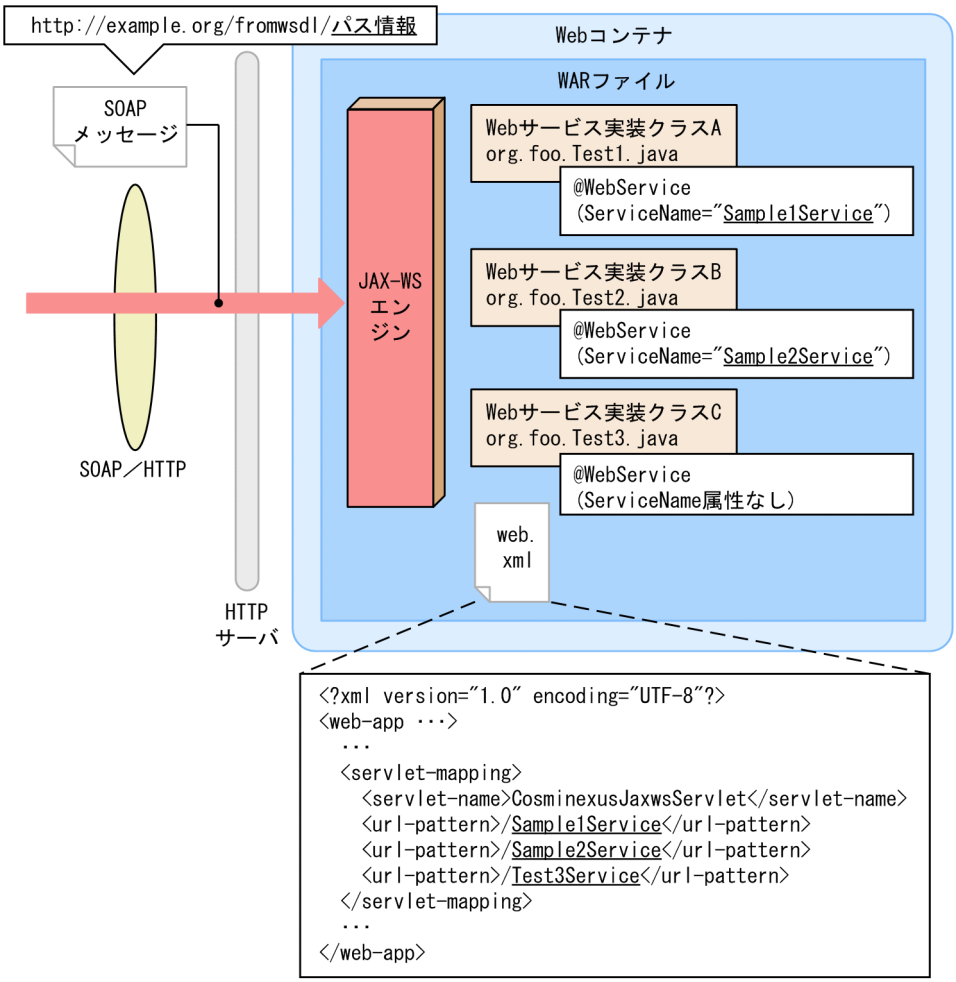
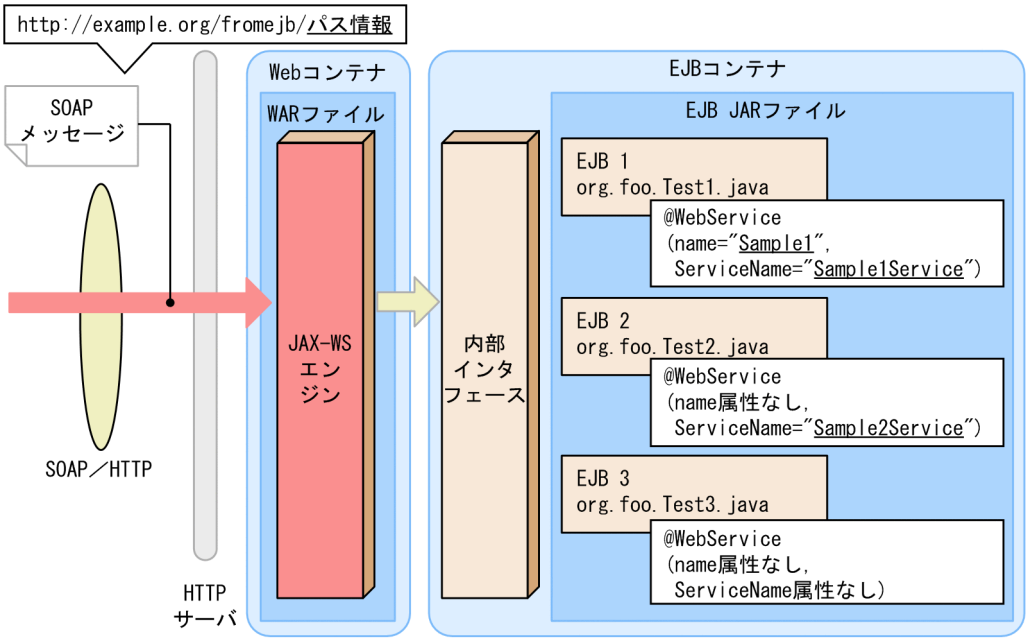


図 10-6　Web サービス実装クラスのディスカバリ（EJB の Web サービス）



Web サービス側の JAX-WS エンジンは，デプロイされている Web サービス実装クラスまたはプロバイダ実装クラスのうち，パス情報に対応するクラスを呼び出します。

POJO の Web サービスの場合，javax.jws.WebService アノテーションまたは javax.xml.ws.WebServiceProvider アノテーションの serviceName 属性が，パス情報から先頭のスラッシュ（/）を取り除いた文字列に等しいものを呼び出します。

EJB の Web サービスの場合，javax.jws.WebService アノテーションの serviceName 属性が，パス情報の先頭のスラッシュ（/）と 2 番目のスラッシュ（/）の間の文字列に等しく，javax.jws.WebService アノテーションの name 属性の値が，パス情報の 2 番目のスラッシュ（/）より後ろの文字列に等しいものを呼び出します。

javax.jws.WebService アノテーションおよび javax.xml.ws.WebServiceProvider アノテーションの serviceName 属性は省略できます。省略されている場合は，JSR-181 仕様に基づいて，Web サービス実装クラスまたはプロバイダ実装クラスのクラス名（単純名）にサフィックスとして"Service"を付加された文字列が，serviceName 属性の値と見なされます。また，javax.jws.WebService アノテーションの name 属性は省略できます。省略されている場合は，JSR-181 仕様に基づいて，Web サービス実装クラスのクラス名（単純名）が name 属性の値と見なされます。

POJO の Web サービスの例でのパス情報と，呼び出される Web サービス実装クラスの対応を示します。

- パス情報が Sample1Service の場合
  Web サービス実装クラス A（org.foo.Test1.java）が呼び出されます。
- パス情報が Sample2Service の場合
  Web サービス実装クラス B（org.foo.Test2.java）が呼び出されます。
- パス情報が Test3Service の場合
  Web サービス実装クラス C（org.foo.Test3.java）が呼び出されます。

EJB の Web サービスの例でのパス情報と，呼び出される Web サービス実装クラスの対応を示します。

- パス情報が/Sample1Service/Sample1 の場合
  EJB 1（org.foo.Test1.java）が呼び出されます。
- パス情報が/Sample2Service/Test2 の場合
  EJB 2（org.foo.Test2.java）が呼び出されます。
- パス情報が/Test3Service/Test3 の場合
  EJB 3（org.foo.Test3.java）が呼び出されます。

このマッピングは，cosminexus-jaxws.xml を記述することでカスタマイズできます。次に示す cosminexus-jaxws.xml の記述例を基に，マッピングのカスタマイズについて説明します。

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns='http://java.sun.com/xml/ns/jax-ws/ri/runtime'>
    <endpoint
        name="test1"
```

```
        implementation="org.foo.Test1"
        url-pattern="/test1"
    />
    <endpoint
        name="test2"
        implementation="org.foo.Test2"
        url-pattern="/test2"
    />
    <endpoint
        name="test3"
        implementation="org.foo.Test3"
        url-pattern="/test3"
    />
</endpoints>
```

この例でのパス情報と，呼び出される Web サービス実装クラスの対応を示します。

- パス情報が test1 の場合

  Web サービス実装クラス A（org.foo.Test1.java）が呼び出されます。
- パス情報が test2 の場合

  Web サービス実装クラス B（org.foo.Test2.java）が呼び出されます。
- パス情報が test3 の場合

  Web サービス実装クラス C（org.foo.Test3.java）が呼び出されます。

ただし，cosminexus-jaxws.xml の url-pattern 属性と，web.xml の url-pattern 要素は，1 対 1 で対応している必要があります。したがって，この例のマッピングをカスタマイズした場合は，web.xml でも記述を変更する必要があります。web.xml の記述例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
  ...
  <servlet-mapping>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/test1</url-pattern>
    <url-pattern>/test2</url-pattern>
    <url-pattern>/test3</url-pattern>
  </servlet-mapping>
  ...
</web-app>
```

cosminexus-jaxws.xml のカスタマイズ方法については，「10.3　cosminexus-jaxws.xml によるカスタマイズ」を参照してください。

(2)　SOAP メッセージのディスパッチ

Web サービス側の JAX-WS エンジンでは，発見した対象が Web サービス実装クラスの場合，受信した SOAP メッセージの内容に応じてオペレーションに対応するメソッドが呼び出され，実行されます。発見した対象がプロバイダ実装クラスの場合，アンマーシャルした SOAP メッセージはプロバイダ実装クラスが指定するオブジェクトに変換され，invoke()メソッドが呼び出さます。

このことを SOAP メッセージの**ディスパッチ**といいます。

SOAP メッセージは，WS-I Basic Profile 1.1 および Attachments Profile 1.0 を適用した，次のどれかの仕様に準拠していなければなりません。

- SOAP 1.1 仕様
- SOAP 1.2 仕様
- SwA 仕様（wsi:swaRef 形式の添付ファイルを利用する場合）
- MTOM/XOP 仕様（MTOM/XOP 仕様形式の添付ファイルを利用する場合）

SOAP 1.1 仕様の場合の SOAP メッセージの例を次に示します。なお，添付ファイルを付けない SOAP メッセージの例を示します。添付ファイル付き SOAP メッセージについては，「28.4　添付ファイル付き SOAP メッセージ（wsi:swaRef 形式）」を参照してください。

```
POST http://sample.org/fromjava/AddNumbersImplService HTTP/1.1
SOAPAction: ""
Content-Type: text/xml;charset="utf-8"
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2

<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:add xmlns:ns2="http://sample.com/">
      <arg0>256</arg0>
      <arg1>103</arg1>
    </ns2:add>
  </S:Body>
</S:Envelope>
```

メッセージ先頭に HTTP リクエストライン（POST の行）および HTTP リクエストのヘッダフィールド（SOAPAction，Content-Type，および Accept の行）が続きます。そのあとに空白行が入り，SOAP メッセージが続きます。この SOAP メッセージの内容を基に，適切な SEI が呼び出されて処理されます。

HTTP リクエストのヘッダフィールドには，空文字（""）が設定された SOAPAction ヘッダが必要です。SOAPAction ヘッダに値が設定されていたとしても，JAX-WS エンジンによって無視されます。SOAPAction ヘッダの値が引用符「"」で囲まれていない場合，エラーメッセージが出力されます（KDJW3022-W）。また，SOAPAction ヘッダが HTTP リクエストに含まれていない場合の動作については，動作定義ファイルの設定によって異なります。動作定義ファイル設定については，「10.1　動作定義ファイル」を参照してください。

SOAP 1.2 仕様の場合の SOAP メッセージの例を次に示します。なお，添付ファイルを付けない SOAP メッセージの例を示します。添付ファイル付き SOAP メッセージについては，「28.4　添付ファイル付き SOAP メッセージ（wsi:swaRef 形式）」を参照してください。

```
POST http://sample.org/fromjava/AddNumbersImplService HTTP/1.1
Content-Type: application/soap+xml;charset="utf-8";action=""
Accept: application/soap+xml, multipart/related, text/html, image/gif, image/jpeg, *; q=.2,
*/*; q=.2
```

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Body>
    <ns2:add xmlns:ns2="http://sample.com/">
      <arg0>256</arg0>
      <arg1>103</arg1>
    </ns2:add>
  </S:Body>
</S:Envelope>
```

SOAP 1.2 仕様の場合，SOAPAction ヘッダは，HTTP リクエストのヘッダフィールドに出現しても無視されます。また，action パラメタは省略できます。

10.3　cosminexus-jaxws.xml によるカスタマイズ

cosminexus-jaxws.xml は，POJO の Web サービスの URL と Web サービス実装クラスまたはプロバイダ実装クラスのマッピングをカスタマイズするための DD です。Web サービス実装クラスまたはプロバイダ実装クラスを見つけ出す処理（ディスカバリ）をカスタマイズしたい場合，または複数の URL に対して単一の Web サービス実装クラスおよびプロバイダ実装クラスを割り当てたい場合に作成します。

マッピングをカスタマイズしない場合，cosminexus-jaxws.xml は不要です。また，cosminexus-jaxws.xml を EJB JAR ファイルに含めても，EJB の Web サービスには適用されません。

マッピングのカスタマイズについては，「10.2.2(1)　ディスカバリ」を参照してください。

マッピングをカスタマイズする場合，cosminexus-jaxws.xml が正しく記述されているか注意してください。cosminexus-jaxws.xml は Web サービスの初期化時に読み込まれます。したがって，cosminexus-jaxws.xml の読み込みでエラーが発生した場合，Web サービスの初期化に失敗します。Web サービスの初期化については，「10.9.1　Web サービスの初期化」を参照してください。

ここでは，cosminexus-jaxws.xml のファイル名と格納先，および書式について説明します。

10.3.1　cosminexus-jaxws.xml のファイル名と格納先

cosminexus-jaxws.xml は，WAR ファイルに含めます。cosminexus-jaxws.xml を含める場合は，WEB-INF ディレクトリ直下に，cosminexus-jaxws.xml という名称で含めてください。ファイル名と格納先を次に示します。

<WAR のルート>/WEB-INF/cosminexus-jaxws.xml

10.3.2　cosminexus-jaxws.xml の書式

cosminexus-jaxws.xml のフォーマットとエンコーディングを次に示します。このフォーマットとエンコーディング以外で記述した場合の動作は保証されません。

- フォーマット：XML バージョン 1.0
- エンコーディング：UTF-8

cosminexus-jaxws.xml で指定できる要素を次の表に示します。

表 10–3　cosminexus-jaxws.xml の要素一覧

要素名	指定個数	説明
jaxwsdd:endpoints 要素	1 個	ルート要素です。

要素名		指定個数	説明
└	jaxwsdd:endpoint 要素	1 個以上	URL と，Web サービス実装クラスまたはプロバイダ実装クラスのマッピングを指定します。

各要素および属性について説明します。

(1) jaxwsdd:endpoints 要素（cosminexus-jaxws.xml）

jaxwsdd:endpoints 要素は，cosminexus-jaxws.xml のルート要素です。属性は持ちません。

(2) jaxwsdd:endpoint 要素（cosminexus-jaxws.xml）

jaxwsdd:endpoint 要素には，URL と，Web サービス実装クラスまたはプロバイダ実装クラスのマッピングを記述します。1 個の jaxwsdd:endpoint 要素に 1 個のマッピングを記述します。Web サービスが複数のポートを持つような場合（一つの Web サービスが複数の URL で提供される場合）や，同じ WAR ファイル内に複数の Web サービスがある場合は，ポートや Web サービスに対応した数の jaxwsdd:endpoint 要素を記述する必要があります。

jaxwsdd:endpoint 要素の属性の一覧を次の表に示します。

表 10–4　jaxwsdd:endpoint 要素の属性一覧

項番	属性名	必須	説明
1	name	○	jaxwsdd:endpoint 要素を区別するための名前を指定します。
2	implementation	○	Web サービス実装クラスまたはプロバイダ実装クラスを指定します。
3	port	×	Web サービス実装クラスまたはプロバイダ実装クラスと対応づけるポートを指定します。javax.jws.WebService アノテーション，または javax.xml.ws.WebServiceProvider アノテーションの設定よりも優先されます。一つの Web サービス実装クラスまたはプロバイダ実装クラスを複数の URL にマッピングする場合に指定します。
4	url-pattern	○	Web サービス実装クラスまたはプロバイダ実装クラスと対応づける URL を指定します。web.xml の url-pattern 要素と対応しています。

（凡例）

○：指定が必須であることを示します。

×：指定が必須でないことを示します。

(a) name 属性（cosminexus-jaxws.xml）

jaxwsdd:endpoint 要素を区別するための名前を空文字列以外の文字列（Java で扱える文字列）で指定します。

空文字列を指定した場合，デプロイ時に KDJW20031-E のメッセージが出力されます。同じ cosminexus-jaxws.xml 内では一意の値にする必要があります。同じ name 属性値を持つ jaxwsdd:endpoint 要素が複数ある場合，KDJW40007-W のメッセージが出力されます。

(b)　implementation 属性（cosminexus-jaxws.xml）

javax.jws.WebService アノテーションを持つクラスのクラス名を指定します。

空文字列を指定した場合，デプロイ時に KDJW20031-E のメッセージが出力されます。また，存在しないクラスを指定した場合，デプロイ時に KDJW20014-E のメッセージが出力されます。

複数の URL に対して同じ Web サービス実装クラスまたはプロバイダ実装クラスをマッピングする場合は，port 属性が一意になるように記述してください。特に Web サービス実装クラスの場合，port 属性の記述がないときや port 属性値が一意でないとき，メタデータとして不正な WSDL が発行されます。

(c)　port 属性（cosminexus-jaxws.xml）

メタデータとして発行する WSDL のポート名（wsdl:port 要素の name 属性の値）の QName を指定します。メタデータの発行については，「10.6　メタデータの発行」を参照してください。

この属性は省略できます。属性を省略した場合や空文字列が指定された場合，implementation 属性で指定したクラスの javax.jws.WebService アノテーション，または javax.xml.ws.WebServiceProvider アノテーションの portName 属性の値が使用されます。portName 属性が省略されている場合は，JSR-181 仕様に従って，Web サービス実装クラスまたはプロバイダ実装クラスの単純名にサフィックスとして"Port"を付加した文字列が使用されます。

複数の URL に対して同じ Web サービス実装クラスをマッピングする場合，port 属性を指定しないとメタデータが正常に発行されません。

(d)　url-pattern 属性（cosminexus-jaxws.xml）

implementation 属性で指定した Web サービス実装クラスまたはプロバイダ実装クラスと対応づけるパス情報を指定します。url-pattern 属性で指定した値に基づいて，ディスカバリが行われます。ディスカバリについては，「10.2.2(1)　ディスカバリ」を参照してください。

パス情報は，明確な値でなければなりません（アスタリスクなどのワイルドカードは使用できません）。また，web.xml の url-pattern 要素の値と 1 対 1 になるように指定する必要があります。

例えば，url-pattern 属性に"/path1"を指定した場合，"/path1"へのリクエストに対して，implementation 属性で指定したクラスがマッピングされます。

同じ cosminexus-jaxws.xml 内では一意の値にする必要があります。同じ url-pattern 属性値を持つ jaxwsdd:endpoint 要素が複数ある場合，KDJW40009-W のメッセージが出力されます。この場合，同じ url-pattern 属性値を持つ jaxwsdd:endpoint 要素の中で，最初の jaxwsdd:endpoint 要素に記述されたマッピングだけが有効になります。

(3)　cosminexus-jaxws.xml 使用時の設定例

次の Web サービスに対応した DD の記述例を示します。

表 10-5　Web サービス実装クラスと対応する URL の例

項番	Web サービス実装クラス	URL
1	com.sample.AddNumbersImplA	/test1
2	com.sample.AddNumbersImplB	/test2，/test3

web.xml の例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/we
b-app_3_0.xsd">
  <description>Sample web service &quot;fromwsdl&quot;</description>
  <display-name>Sample_web_service_fromwsdl</display-name>
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/test1</url-pattern>
    <url-pattern>/test2</url-pattern>
    <url-pattern>/test3</url-pattern>
  </servlet-mapping>
</web-app>
```

バージョン 2.5 の web.xml を作成する場合は，web-app 要素の version 属性を"2.5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"にしてください。

cosminexus-jaxws.xml の例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns='http://java.sun.com/xml/ns/jax-ws/ri/runtime'>
    <endpoint
        name="test1"
        implementation="com.sample.AddNumbersImplA"
        url-pattern="/test1"
    />
    <endpoint
        name="test2"
        implementation="com.sample.AddNumbersImplB"
```

```
        url-pattern="/test2"
        port="{http://sample.com}port1"
    />
    <endpoint
        name="test3"
        implementation="com.sample.AddNumbersImplB"
        url-pattern="/test3"
        port="{http://sample.com}port2"
    />
</endpoints>
```

バージョン 2.5 の web.xml を作成する場合は，web-app 要素の version 属性を"2.5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"にしてください。

10.4　フォルトと例外の処理

JAX-WS エンジンは，Web サービス側および Web サービスクライアント側とも，JAX-WS 2.2 仕様に基づいてフォルトと例外をバインディングします。POJO と EJB の両方の Web サービスで動作します。

JAX-WS エンジンでのフォルトおよび例外の処理について説明します。

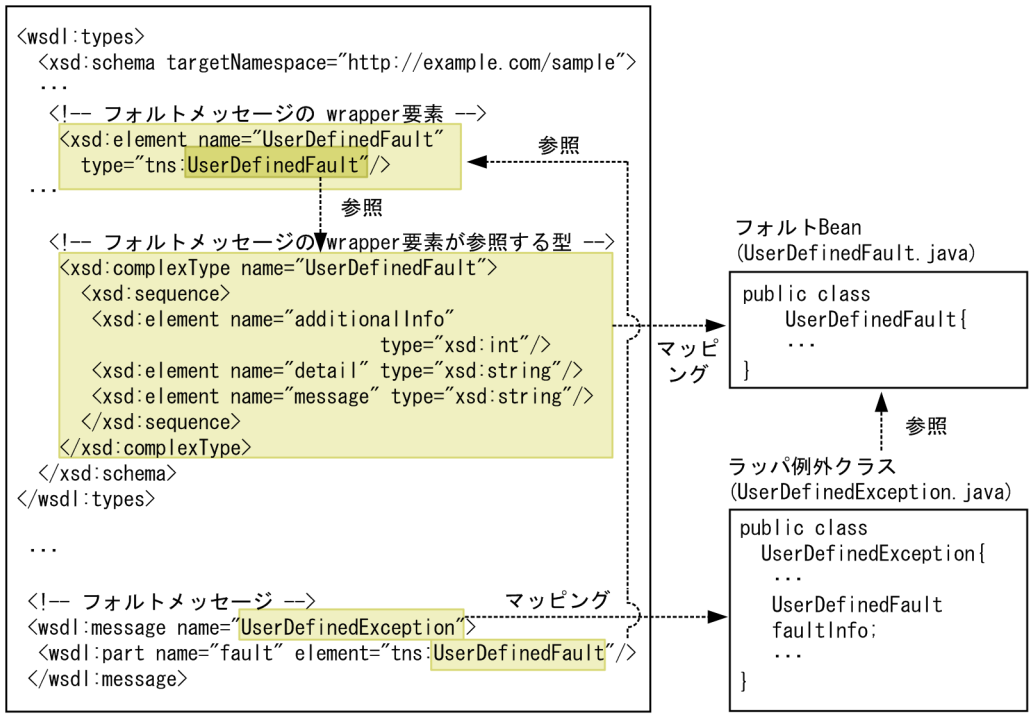
10.4.1　Web サービス側のフォルトおよび例外の処理

Web サービス側の JAX-WS エンジンでのフォルトおよび例外の処理について説明します。なお，Web サービスがプロバイダ実装クラスで実装されている場合，この処理は実行されません。

(1)　サービス固有例外の処理

WSDL のフォルトと Java の例外は，JAX-WS 2.2 仕様に従ってマッピングされます。WSDL のフォルトと Java の例外クラスのマッピング例を次の図に示します。

図 10–7　WSDL のフォルトと Java の例外クラスのマッピング例



マッピング例では，UserDefinedFault フォルトがフォルト bean（com.example.sample.UserDefinedFault）と，ラッパ例外クラス（com.example.sample.UserDefinedException）にマッピングされていることがわかります。

フォルトと例外クラスのマッピングについては，「15.1.7　フォルトから例外クラスへのマッピング」，および「16.1.7　Java のラッパ例外クラスからフォルトへのマッピング」を参照してください。

Web サービス側の JAX-WS エンジンによって，ラッパ例外クラスは次の表のように SOAP フォルトにバインディングされます。

表 10-6　ラッパ例外クラスのバインディング

項番	SOAP フォルトの子要素		内容
	SOAP 1.1 仕様	SOAP 1.2 仕様	
1	faultcode	soapenv12:Code	SOAP 1.1 仕様 QName soapenv:server で固定です。 SOAP 1.2 仕様 QName soapenv12:Receiver で固定です。
2	faultstring	soapenv12:Reason	ラッパ例外クラスに対して getMessage メソッドを実行した結果になります。
3	faultactor	soapenv12:Role	ありません。
4	detail	soapenv12:Detail	フォルト bean をマーシャルした結果になります。

ラッパ例外クラスを Web サービス実装クラスでスローする例を示します。

```
//フォルトbeanを生成し，マーシャルすべき情報を設定する
UserDefinedFault fault = new UserDefinedFault();
fault.additionalInfo = 257;
fault.detail = ”Failed by some reason.”;
fault.message = ”Contact your administrator.”;

//ラッパ例外クラスをスロー
throw new UserDefinedException(
  ”Something happens.”, fault );
```

送信される SOAP 1.1 仕様の SOAP フォルトメッセージの例を示します（実際は，改行およびインデントはありません）。

```
<?xml version=”1.0” encoding=”utf-8”?>
<S:Envelope xmlns:S=”http://schemas.xmlsoap.org/soap/envelope/”>
  <S:Body>
    <S:Fault xmlns:ns3=”http://www.w3.org/2003/05/soap-envelope”>
      <faultcode>S:Server</faultcode>
      <faultstring>Something happens.</faultstring>
      <detail>
        <ns2:UserDefinedFault xmlns:ns2=”http://example.com/sample”>
          <additionalInfo>257</additionalInfo>
          <detail>Failed by some reason.</detail>
          <message>Contact your administrator.</message>
        </ns2:UserDefinedFault>
      </detail>
    </S:Fault>
  </S:Body>
</S:Envelope>
```

注

SOAP フォルトメッセージには，SOAP 1.1 仕様と SOAP 1.2 仕様の名前空間定義が必ず含まれます。

送信される SOAP 1.2 仕様の SOAP フォルトメッセージの例を示します（実際は，改行およびインデントはありません）。

```
<?xml version="1.0" encoding="utf-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Body>
    <S:Fault xmlns:ns3="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Code>
        <S:Value>S:Receiver</S:Value>
      </S:Code>
      <S:Reason>
        <S:Text xml:lang="ja">Something happens.</S:Text>
      </S:Reason>
      <S:Detail>
        <ns2:UserDefinedFault xmlns:ns2="http://example.com/sample">
          <additionalInfo>257</additionalInfo>
          <detail>Failed by some reason.</detail>
          <message>Contact your administrator.</message>
        </ns2:UserDefinedFault>
      </S:Detail>
    </S:Fault>
  </S:Body>
</S:Envelope>
```

注

SOAP フォルトメッセージには，SOAP 1.1 仕様と SOAP 1.2 仕様の名前空間定義が必ず含まれます。

(2) ランタイム例外のバインディング

Web サービス実装クラス内で javax.xml.ws.WebServiceException 以外のランタイム例外がスローされた場合，Web サービス側の JAX-WS エンジンによって，ランタイム例外が SOAP フォルトにバインディングされます（JAX-WS 2.2 仕様に基づいてバインディング）。

ランタイム例外のバインディングの例を次の表に示します。

表 10-7 ランタイム例外のバインディング

項番	SOAP フォルトの子要素		内容
	SOAP 1.1 仕様	SOAP 1.2 仕様	
1	faultcode	soapenv12:Code	SOAP 1.1 仕様 QName soapenv:server で固定です。 SOAP 1.2 仕様 QName soapenv12:Receiver で固定です。
2	faultstring	soapenv12:Reason	ランタイム例外に対して getMessage メソッドを実行した結果になります。

項番	SOAP フォルトの子要素		内容
	SOAP 1.1 仕様	SOAP 1.2 仕様	
3	faultactor	soapenv12:Role	ありません。
4	detail	soapenv12:Detail	ありません。

ランタイム例外のスローの例を示します。

```
//ランタイム例外をスロー
throw new IllegalArgumentException( "Something illegal." );
```

送信される SOAP 1.1 仕様の SOAP フォルトメッセージの例を示します（実際は，改行およびインデントはありません)。

```
<?xml version="1.0" encoding="utf-8"?>
<S:Envelope xmlns:S= "http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <S:Fault xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
      <faultcode>S:Server</faultcode>
      <faultstring>Something illegal.</faultstring>
    </S:Fault>
  </S:Body>
</S:Envelope
```

注

SOAP フォルトメッセージには，SOAP 1.1 仕様と SOAP 1.2 仕様の名前空間定義が必ず含まれます。

送信される SOAP 1.2 仕様の SOAP フォルトメッセージの例を示します（実際は，改行およびインデントはありません)。

```
<?xml version="1.0" encoding="utf-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Body>
    <S:Fault xmlns:ns3="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Code>
        <S:Value>S:Receiver</S:Value>
      </S:Code>
      <S:Reason>
        <S:Text xml:lang="ja">Something illegal.</S:Text>
      </S:Reason>
    </S:Fault>
  </S:Body>
</S:Envelope>
```

注

SOAP フォルトメッセージには，SOAP 1.1 仕様と SOAP 1.2 仕様の名前空間定義が必ず含まれます。

(3) javax.xml.ws.WebServiceException のバインディング

Web サービス実装クラスまたはプロバイダ実装クラス内で javax.xml.ws.soap.SOAPFaultException 以外の javax.xml.ws.WebServiceException がスローされた場合，Web サービス側の JAX-WS エンジンによって，javax.xml.ws.WebServiceException が SOAP フォルトにバインディングされます（JAX-WS 2.2 仕様に基づいてバインディング）。

javax.xml.ws.WebServiceException のバインディングの例を次の表に示します。

表 10–8 javax.xml.ws.WebServiceException のバインディング

項番	SOAP フォルトの子要素		内容
	SOAP 1.1 仕様	SOAP 1.2 仕様	
1	faultcode	soapenv12:Code	SOAP 1.1 仕様 QName soapenv:server で固定です。 SOAP 1.2 仕様 QName soapenv12:Receiver で固定です。
2	faultstring	soapenv12:Reason	javax.xml.ws.soap.SOAPFaultException に対して getMessage メソッドを実行した結果になります。SOAP 1.2 の場合，xml:lang 属性には JavaVM のデフォルトのロケールが設定されます。
3	faultactor	soapenv12:Role	ありません。
4	detail	soapenv12:Detail	ありません。

javax.xml.ws.WebServiceException のスローの例を示します。

```
//javax.xml.ws.WebServiceExceptionをスロー
throw new javax.xml.ws.WebServiceException( "Web Service Exception." );
```

送信される SOAP 1.1 仕様の SOAP フォルトメッセージの例を示します（実際は，改行およびインデントはありません）。

```
<?xml version="1.0" encoding="utf-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <S:Fault xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
      <faultcode>S:Server</faultcode>
      <faultstring>Web Service Exception.</faultstring>
    </S:Fault>
  </S:Body>
</S:Envelope>
```

注

SOAP フォルトメッセージには，SOAP 1.1 仕様と SOAP 1.2 仕様の名前空間定義が必ず含まれます。

送信される SOAP 1.2 仕様の SOAP フォルトメッセージの例を示します（実際は，改行およびインデントはありません）。

```
<?xml version="1.0" encoding="utf-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Body>
    <S:Fault xmlns:ns3="http://schemas.xmlsoap.org/soap/envelope/>
      <S:Code>
        <S:Value>S:Receiver</S:Value>
      </S:Code>
      <S:Reason>
        <S:Text xml:lang="ja">Something illegal.</S:Text>
      </S:Reason>
    </S:Fault>
  </S:Body>
</S:Envelope>
```

注

SOAP フォルトメッセージには，SOAP 1.1 仕様と SOAP 1.2 仕様の名前空間定義が必ず含まれます。

(4) javax.xml.ws.soap.SOAPFaultException のバインディング

Web サービス実装クラスまたはプロバイダ実装クラス内で，javax.xml.ws.soap.SOAPFaultException がスローされた場合，Web サービス側の JAX-WS エンジンによって，javax.xml.ws.soap.SOAPFaultException が SOAP フォルトにバインディングされます（JAX-WS 2.2 仕様に基づいてバインディング）。

javax.xml.ws.soap.SOAPFaultException のバインディングの例を次の表に示します。

表 10-9 javax.xml.ws.soap.SOAPFaultException のバインディング

項番	SOAP フォルトの子要素		内容
	SOAP 1.1 仕様	SOAP 1.2 仕様	
1	faultcode	soapenv12:Code	SOAP 1.1 仕様 getFault().getFaultCodeAsQName メソッドの結果になります。 ただし，null の場合は QName soapenv:server で固定です。 SOAP 1.2 仕様 QName soapenv12:Sender で固定です。soapenv12:Code の子要素の soapenv12:Subcode が結果を持ちます。
2	faultstring	soapenv12:Reason	getFaultReasonText メソッドの結果になります。 ただし，null の場合は getMessage メソッドを実行した結果になります。
3	faultactor	soapenv12:Role	getFault().getFaultRole メソッドの結果になります。 ただし，null の場合はありません。

項番	SOAP フォルトの子要素		内容
	SOAP 1.1 仕様	SOAP 1.2 仕様	
4	detail	soapenv12:Detail	getFault().getDetail メソッドを実行した結果をマーシャルした結果になります。 ただし，null の場合はありません。

SOAP 1.1 仕様の場合の，javax.xml.ws.soap.SOAPFaultException のスローの例を示します。

```
SOAPFault soapFault = ...;
soapFault.setFaultCode( new QName( "http://sample.org", "UserDefined" ) );
soapFault.setFaultActor( "http://example.com/sample" );
soapFault.setFaultString( "SOAPFaultException happens." );
Detail detail = soapFault.addDetail();
SOAPElement soapElement = detail.addChildElement( new QName( "", "detailTest" ) );
soapElement.addTextNode( "TEST." );

//javax.xml.ws.soap.SOAPFaultExceptionをスロー
throw new SOAPFaultException( soapFault );
```

SOAP 1.2 仕様の場合の，javax.xml.ws.soap.SOAPFaultException のスローの例を示します。

```
SOAPFactory soapFactory = SOAPFactory.newInstance( SOAPConstants.SOAP_1_2_PROTOCOL );
SOAPFault soapFault = soapFactory.createFault();
soapFault.appendFaultSubcode( new QName( "http://sample.org", "UserDefined" ) );
soapFault.setFaultRole( "http://example.com/sample" );
soapFault.addFaultReasonText( "SOAPFaultException happens.", Locale.getDefault() );
Detail detail = soapFault.addDetail();
SOAPElement soapElement = detail.addChildElement( new QName( "", "detailTest" ) );
soapElement.addTextNode( "TEST." );

//javax.xml.ws.soap.SOAPFaultExceptionをスロー
throw new SOAPFaultException( soapFault );
```

送信される SOAP 1.1 仕様の SOAP フォルトメッセージの例を示します（実際は，改行およびインデントはありません）。

```
<?xml version="1.0" encoding="utf-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <S:Fault xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
      <faultcode xmlns:ns0="http://sample.org">ns0:UserDefined</faultcode>
      <faultstring>SOAPFaultException happens.</faultstring>
      <faultactor>http://example.com/sample</faultactor>
      <detail><detailTest>TEST.</detailTest></detail>
    </S:Fault>
  </S:Body>
</S:Envelope>
```

注

SOAP フォルトメッセージには，SOAP 1.1 仕様と SOAP 1.2 仕様の名前空間定義が必ず含まれます。

送信される SOAP 1.2 仕様の SOAP フォルトメッセージの例を示します（実際は，改行およびインデントはありません）。

```
<?xml version="1.0" encoding="utf-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Body>
    <S:Fault xmlns:ns3="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Code>
        <S:Value>S:Sender</S:Value>
        <S:Subcode>
          <S:Value xmlns:ns0="http://sample.org">ns0:UserDefined</S:Value>
        </S:Subcode>
      </S:Code>
      <S:Reason>
        <S:Text xml:lang="ja">SOAPFaultException happens.</S:Text>
      </S:Reason>
      <S:Role>http://example.com/sample</S:Role>
      <S:Detail>
        <detailTest>TEST.</detailTest>
      </S:Detail>
    </S:Fault>
  </S:Body>
</S:Envelope>
```

注

SOAP フォルトメッセージには，SOAP 1.1 仕様と SOAP 1.2 仕様の名前空間定義が必ず含まれます。

10.4.2 Web サービスクライアント側のフォルトの処理

wsdl:fault 要素に対応する SOAP フォルト，および wsdl:fault 要素に対応しない SOAP フォルトについてそれぞれ説明します。

- wsdl:fault 要素に対応する SOAP フォルト

  Web サービスクライアント側の JAX-WS エンジンが，「10.4.1(1) サービス固有例外の処理」に従ってマーシャルされた SOAP フォルトメッセージを受け取った場合，元のフォルト bean とラッパ例外にアンマーシャルされ，Web サービスクライアントにスローされます。つまり，Web サービスでスローされたサービス固有例外は，透過的に Web サービスクライアントへ送信されます（この動作は，JAX-WS 2.2 仕様に従っています）。

- wsdl:fault 要素に対応しない SOAP フォルト

  Web サービスクライアント側の JAX-WS エンジンが，wsdl:fault 要素に対応しない SOAP フォルトメッセージを受け取った場合，javax.xml.ws.soap.SOAPFaultException にアンマーシャルされ，Web サービスクライアントにスローされます（この動作は，JAX-WS 2.2 仕様に従っています）。

  SOAP 1.2 の場合，SOAP フォルトが soapenv12:Text 要素を複数持つとき，SOAPFaultException オブジェクトが持つ SOAPFault オブジェクトには Reason Text が一つだけ設定されます。次に示す個所の値（文字列）およびロケールが設定されます。

  - 値：最後の soapenv12:Text 要素の値

- ロケール：JavaVM のデフォルトのロケール

10.4.3　Java 例外の伝搬

Web サービスおよび Web サービスクライアントの両方が，Application Server の JAX-WS エンジン上にデプロイされている場合，Web サービスで発生した Java 例外を Web サービスクライアントに伝搬できます。POJO と EJB の両方の Web サービスで動作します。ここでは，Java 例外の伝搬方法と動作について説明します。

注意事項

Java 例外の伝搬は，SOAP 1.1 仕様や，JAX-WS 2.2 仕様で定められた機能ではないため，Application Server の JAX-WS 機能以外の Web サービス基盤製品と接続した場合に，意図しない動作をしたり，通信が失敗したりするおそれがあります。また，スタックトレースには，Web サービスの実装の内部情報（システムの設定情報や個人情報を扱っているのであればそのような情報など）も含まれることがあります。したがって，実運用でこの機能を使用することを想定して，Web サービスを実装することはお勧めしません。開発時に必要に応じて使用してください。

(1)　Java 例外の伝搬方法

Web サービスで発生した Java 例外を伝搬するには，動作定義ファイルで com.cosminexus.jaxws.fault.SOAPFaultBuilder.captureStackTrace プロパティに true を設定します。

com.cosminexus.jaxws.fault.SOAPFaultBuilder.captureStackTrace プロパティについては，「10.1.2　共通定義ファイルの設定項目」を参照してください。

(2)　Java 例外の伝搬時の動作（Web サービス側）

Web サービス側での Java 例外の伝搬時に，「10.4.1(2)」の「表 10-7 の項番 4」，「10.4.1(3)」の「表 10-8 の項番 4」および「10.4.1(4)」の「表 10-9 の項番 4」にある要素の子要素として，Application Server の JAX-WS 機能独自の{http://jax-ws.dev.java.net/}exception 要素が追加され，発生した Java 例外のスタックトレースがマーシャルされます。

また，「10.4.1(1)」の「表 10-6 の項番 4」にバインドされるフォルト bean が存在しない場合，Web サービス側での Java 例外の伝搬時に，「10.4.1(1)」の「表 10-6 の項番 4」にある要素の子要素として，Application Server の JAX-WS 機能独自の{http://jax-ws.dev.java.net/}exception 要素が追加され，発生した Java 例外のスタックトレースがマーシャルされます。

ランタイム例外のスローの例を示します。

```
//ランタイム例外をスロー
catch( NullPointerException ){
  throw new IllegalArgumentException( "Something illegal.", e );
}
```

このようにスローされた場合に，送信される SOAP 1.1 仕様の SOAP フォルトメッセージの例を示します（実際は，改行およびインデントはありません）。

```
<?xml version="1.0" encoding="utf-8"?>
<S:Envelope xmlns:S= "http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <S:Fault xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
      <faultcode>S:Server</faultcode>
      <faultstring>Something illegal.</faultstring>
      <detail>
        <ns2:exception xmlns:ns2="http://jax-ws.dev.java.net/"
          class="java.lang.IllegalArgumentException"
          note="To disable this feature, set com.cosminexus.jaxws.fault.SOAPFaultBuilder.cap
tureStackTrace property to false">
          <message>Something illegal</message>
          <ns2:stackTrace>
            <ns2:frame class="com.example.sample.TestJaxWsImpl" file="TestJaxWsImpl.java" li
ne="32" method="jaxWsTest1"/>
            <ns2:frame class="sun.reflect.NativeMethodAccessorImpl" file="NativeMethodAccess
orImpl.java" line="native" method="invoke0"/>
            <ns2:frame class="sun.reflect.NativeMethodAccessorImpl" file="NativeMethodAccess
orImpl.java" line="39" method="invoke"/>
            ...
            <ns2:frame class="java.lang.Thread" file="Thread.java" line="595" method="run"/>
          </ns2:stackTrace>
          <ns2:cause class="java.lang.NullPointerException"
              note="To disable this feature, set com.cosminexus.jaxws.fault.SOAPFaultBuilder
.captureStackTrace property to false">
          <message>Something null.</message>
          <ns2:stackTrace>
            <ns2:frame class="com.example.sample.TestJaxWsImpl" file="TestJaxWsImpl.java" li
ne="32" method="jaxWsTest1"/>
            <ns2:frame class="sun.reflect.NativeMethodAccessorImpl" file="NativeMethodAccess
orImpl.java" line="native" method="invoke0"/>
            ...
            <ns2:frame class="sun.reflect.DelegatingMethodAccessorImpl" file="DelegatingMeth
odAccessorImpl.java" line="25" method="invoke"/>
            <ns2:frame class="java.lang.reflect.Method" file="Method.java" line="585" method
="invoke"/>
            <ns2:frame class="org.apache.tomcat.util.threads.ThreadPool$ControlRunnable" fil
e="ThreadPool.java" line="1510" method="run"/>
            <ns2:frame class="java.lang.Thread" file="Thread.java" line="595" method="run"/>
          </ns2:stackTrace>
          </ns2:cause>
        </ns2:exception>
      </detail>
    </S:Fault>
  </S:Body>
</S:Envelope>
```

送信される SOAP 1.2 仕様の SOAP フォルトメッセージの例を示します（実際は，改行およびインデントはありません）。

```
<?xml version="1.0" encoding="utf-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
```

```
  <S:Body>
    <S:Fault xmlns:ns3="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Code>
        <S:Value>S:Receiver</S:Value>
      </S:Code>
      <S:Reason>
        <S:Text xml:lang="ja">Something illegal.</S:Text>
      </S:Reason>
      <S:Detail>
        <detailTest>TEST.</detailTest>
        <ns2:exception xmlns:ns2="http://jax-ws.dev.java.net/"
           class="javax.xml.ws.soap.SOAPFaultException"
           note="To disable this feature, set com.cosminexus.jaxws.fault.SOAPFaultBuilder.ca
ptureStackTrace property to false">
           ...
            (SOAP 1.1仕様のSOAPフォルトと同じ)
           ...
        </ns2:exception>
      </S:Detail>
    </S:Fault>
  </S:Body>
</S:Envelope>
```

(3) Java 例外の伝搬時の動作（Web サービスクライアント側）

Web サービスクライアント側での Java 例外の伝搬時に，Web サービスクライアントにスローするラッパ例外クラス，または javax.xml.ws.soap.SOAPFaultException の cause に，{http://jax-ws.dev.java.net/}exception 要素の情報からアンマーシャルした com.cosminexus.xml.ws.developer.ServerSideException が設定されます。

Web サービスクライアントは，javax.xml.ws.soap.SOAPFaultException の getCause メソッドを実行することで，Web サービス側で発生した例外のスタックトレースを含む com.cosminexus.xml.ws.developer.ServerSideException を取得できます。

10.4.4 例外をフォルトにバインディングする場合の HTTP ステータスコード

Web サービス側の JAX-WS エンジンが例外をフォルトにバインディングして Web サービスクライアントへ返す場合，HTTP レスポンスのステータスコードに"500 Internal Server Error"が設定されます。POJO と EJB の両方の Web サービスで動作します。

10.4.5 エラーページをカスタマイズする場合の注意事項

Web サービス側の JAX-WS エンジンが動作する J2EE サーバや，Web サービスを含む WAR ファイルでエラーページをカスタマイズしている場合，HTTP ステータスコード 500 はカスタマイズしないでください。Web サービス側の JAX-WS エンジンは，フォルトを Web サービスクライアントに返す場合，HTTP ステータスコードに「500 Internal Server Error」を設定します。このため，HTTP ステータス

コード 500 をカスタマイズすると，SOAP フォルトではない不正な SOAP メッセージが Web サービスクライアントに送信されてしまいます。

10.5　インタフェースの透過性

Web サービスと Web サービスクライアントは，疎な関係にあり，相互のインタフェースは WSDL の定義内容だけです。Web サービス側は，WSDL を通じて Web サービスのインタフェース情報（メタデータ）を公開し，Web サービスクライアント側はそのメタデータを使用して SOAP メッセージを生成し，送受信します。

Web サービスおよび Web サービスクライアントの両方が Application Server の JAX-WS エンジンで動作する場合も，WSDL 以外のインタフェース情報は交換されません。

Java インタフェースの透過性はないため，SEI を起点として Web サービスを開発した場合，Web サービス側と Web サービスクライアント側でメソッドシグネチャが異なることがあります。

ここでは，生成前の Java メソッドと生成後の Java メソッドの例を基に，メソッドシグネチャの違いについて説明します。

10.5.1　配列のパラメタを持つ場合

次に示す Java メソッドを持つ SEI を起点に，Web サービスを開発することを想定します。この Java メソッドは，配列（int 型）のパラメタを持ちます。

```
@WebMethod
public void test1( int[] param1 );
```

この場合にマッピングされる WSDL の一部を次に示します。

```
...
<types>
  <xsd:schema targetNamespace="http://cosminexus.com/jaxws">

    <xs:element name="test1" type="tns:test1"/>

    <xs:element name="test1Response" type="tns:test1Response"/>

    <xs:complexType name="test1">
      <xs:element name="arg0" type="xs:int" nillable="true"
        minOccurs="0" maxOccurs="unbounded" />
    </xs:complexType>

    <xs:complexType name="test1Response">
      <xs:sequence/>
    </xs:complexType>
  </xs:element>
</types>
<message name="test1">
  <part name="parameters" element="tns:test1"/>
</message>

<message name="test1Response">
```

```
  <part name="parameters" element="tns:test1Response"/>
</message>

<portType ...>
  <operation name="test1">
    <input message="tns:test1"/>
    <output message="tns:test1Response"/>
  </operation>
  ...
</portType>
...
```

パラメタは，maxOccurs="unbounded"の wrapper 子要素にマッピングされます。

この WSDL を指定して cjwsimport コマンドを実行すると，生成されるサービスクラスの Java メソッドは次のようになります。

```
@WebMethod
public String test1( java.util.List<Integer> arg0 );
```

maxOccurs="unbounded"の wrapper 子要素は，java.util.List クラスにマッピングされます。また，この場合，xsd:int 型は，java.lang.Integer クラスにマッピングされます。

10.5.2　OUT パラメタを一つだけ持つ場合

次に示す Java メソッドを持つ SEI を起点に，Web サービスを開発することを想定します。この Java メソッドは，OUT パラメタを一つだけ持ち，戻り値は持ちません。

```
@WebMethod
public void test1( @WebParam(mode=WebParam.Mode.OUT) Holder<String> param1 );
```

この場合にマッピングされる WSDL の一部を次に示します。

```
...
<types>
  <xsd:schema targetNamespace="http://cosminexus.com/jaxws">

    <xs:element name="test1" type="tns:test1"/>

    <xs:element name="test1Response" type="tns:test1Response"/>

    <xs:complexType name="test1">
      <xs:sequence/>
    </xs:complexType>

    <xs:complexType name="test1Response">
      <xs:sequence>
        <xs:element name="arg0" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
```

```
  </xs:element>
</types>
<message name="test1">
  <part name="parameters" element="tns:test1"/>
</message>

<message name="test1Response">
  <part name="parameters" element="tns:test1Response"/>
</message>

<portType ...>
  <operation name="test1">
    <input message="tns:test1"/>
    <output message="tns:test1Response"/>
  </operation>
  ...
</portType>
...
```

OUT パラメタは，wsdl:output 要素から参照される wrapper 子要素にマッピングされます。

この WSDL を指定して cjwsimport コマンドを実行すると，生成されるサービスクラスの Java メソッドは次のようになります。

```
@WebMethod
public String test1();
```

wsdl:output 要素から参照される wrapper 子要素が 1 個だけなので，その wrapper 子要素は戻り値にマッピングされます。

10.5.3 non-wrapper スタイルの配列

次に示す Java メソッドを持つ SEI を起点に，Web サービスを開発することを想定します。この Java メソッドは，non-wrapper スタイルで，java.lang.String クラスの配列のパラメタを持ちます。

```
@WebMethod
@javax.jws.soap.SOAPBinding(
  parameterStyle=javax.jws.soap.SOAPBinding.ParameterStyle.BARE)
public String test1( String[] param1 );
```

この場合にマッピングされる WSDL の一部を次に示します。

```
...
<types>
  <xsd:schema targetNamespace="http://jaxb.dev.java.net/array">
    <xsd:complexType name="stringArray" final="#all">
      <xsd:sequence>
        <xsd:element name="item" type="xsd:string" minOccurs="0"
          maxOccurs="unbounded" nillable="true" />
      </xsd:sequence>
```

```
    </xsd:complexType>
  </xsd:schema>

  <xsd:schema targetNamespace="http://cosminexus.com/jaxws"
    xmlns:ns1="http://jaxb.dev.java.net/array">
    <xsd:element name="test1" nillable="true" type="ns1:stringArray"/>
    <xsd:element name="test1Response" nillable="true" type="xsd:string"/>
  </xsd:schema>
</types>

<message name="test1">
  <part name="test1" element="tns:test1" />
</message>

<message name="test1Response">
  <part name="test1Response" element="tns:test1Response" />
</message>

<portType ...>
  <operation name="test1">
    <input message="tns:test1" />
    <output message="tns:test1Response" />
  </operation>
  ...
</portType>
...
```

パラメタは，{http://jaxb.dev.java.net/array}stringArray 型の wrapper 子要素にマッピングされます。

この WSDL を指定して cjwsimport コマンドを実行すると，生成されるサービスクラスの Java メソッドは次のようになります。

```
@WebMethod
@javax.jws.soap.SOAPBinding(
  parameterStyle=javax.jws.soap.SOAPBinding.ParameterStyle.BARE)
public String test1(net.java.dev.jaxb.array.StringArray test1);
```

{http://jaxb.dev.java.net/array}stringArray 型の wrapper 子要素は，
net.java.dev.jaxb.array.StringArray クラスにマッピングされます。

10.6　メタデータの発行

Web サービス側の JAX-WS エンジンは，要求に応じて Web サービス（Web サービス実装クラスまたはプロバイダ実装クラス）のメタデータを記述した WSDL ファイルを発行できます。発行された WSDL ファイルは，cjwsimport コマンドを使用して Web サービス，および Web サービスクライアントの開発に必要な Java コードを生成するときに利用できます。

ここでは，メタデータの発行を利用するに当たって，注意が必要な点について説明します。

10.6.1　メタデータの発行条件

POJO と EJB の Web サービスのそれぞれの発行条件を説明します。

(1)　POJO の Web サービスの場合

POJO の Web サービスのメタデータの発行条件を次の表に示します。Web サービス側の JAX-WS エンジンが，次の表に示す条件をすべて満たした HTTP リクエストを受信したときにメタデータが発行されます。

表 10–10　POJO の Web サービスのメタデータの発行に必要な HTTP リクエスト

項番	項目		条件
1	HTTP メソッド		GET メソッド
2	URL	スキーマ	http または https
3		ホスト名（:ポート番号）	メタデータの発行を要求する Web サービスが存在するホスト名（およびポート番号）
4		コンテキストパス	メタデータの発行を要求する Web サービスが含まれる Web アプリケーションのコンテキストパス
5		Web サービス名	メタデータの発行を要求する Web サービス（Web サービス実装クラスまたはプロバイダ実装クラスのサービス名）
6		クエリストリング	"wsdl"または"WSDL"（文字の大小は区別されます）

要求された URL に対応する Web サービスに関連づけられた WSDL ファイルが，HTTP レスポンスのコンテンツとして要求元に発行されます。リクエスト URL には，次に示すようなクエリストリングを付加する必要があります。

GET http://sample.com:8085/fromjava/AddNumbersImplService?wsdl HTTP/1.1

GET http://sample.com:8085/fromjava/AddNumbersImplService?WSDL HTTP/1.1

(2) EJB の Web サービスの場合

EJB の Web サービスのメタデータの発行条件を次の表に示します。Web サービス側の JAX-WS エンジンが，次の表に示す条件をすべて満たした HTTP リクエストを受信したときにメタデータが発行されます。

表 10–11 EJB の Web サービスのメタデータの発行に必要な HTTP リクエスト

項番	項目		条件
1	HTTP メソッド		GET メソッド
2	URL	スキーマ	http または https
3		ホスト名（:ポート番号）	メタデータの発行を要求する Web サービスが存在するホスト名（およびポート番号）
4		コンテキストパス	メタデータの発行を要求する Web サービスが含まれる Web アプリケーションのコンテキストパス
5		Web サービス名	メタデータの発行を要求する Web サービス（Web サービス実装クラスのサービス名）
6		EJB のクラス名	メタデータの発行を要求する Web サービスの EJB のクラス名
7		クエリストリング	"wsdl"または"WSDL"（文字の大小は区別されます）

要求された URL に対応する Web サービスに関連づけられた WSDL ファイルが，HTTP レスポンスのコンテンツとして要求元に発行されます。リクエスト URL には，次に示すようなクエリストリングを付加する必要があります。

GET http://sample.com:8085/statelessjava/AddNumbersImplService/AddNumbersImpl?wsdl HTTP/1.1

GET http://sample.com:8085/statelessjava/AddNumbersImplService/AddNumbersImpl?WSDL HTTP/1.1

10.6.2 発行されるメタデータ

リクエスト時の条件と，発行されるメタデータの対応を次の表に示します。

表 10–12 リクエストの条件と発行されるメタデータの対応

項番	リクエスト時の条件	発行されるメタデータ	Web サービスの適用可否	
			POJO	EJB※
1	javax.jws.WebService アノテーション（Web サービス実装クラスの場合），または javax.xml.ws.WebServiceProvider アノテー	wsdlLocation 属性に指定された場所にある WSDL ファイルが返されます。	○	○

項番	リクエスト時の条件	発行されるメタデータ	Web サービスの適用可否	
			POJO	EJB※
	ション（プロバイダ実装クラスの場合）に wsdlLocation 属性がある場合			
2	javax.jws.WebService アノテーション（Web サービス実装クラスの場合），または javax.xml.ws.WebServiceProvider アノテーション（プロバイダ実装クラスの場合）に wsdlLocation 属性の指定はないが，デプロイされた WAR ファイルの WEB-INF/wsdl ディレクトリに wsdl:service 要素を持つ WSDL ファイルがある場合	デプロイされた Web アプリケーションの WEB-INF/wsdl ディレクトリにある WSDL ファイルが返されます。	○	−
3	WSDL が Web アプリケーション内にない場合 （項番 1 および項番 2 以外の場合）	対象が Web サービス実装クラスの場合は，WSDL が新たに生成されて返されます。 対象がプロバイダ実装クラスの場合は，メタデータは発行されません。	○	○

（凡例）

○：適用されます。

−：適用されません。

注※

EJB の Web サービスの場合，Web サービス実装クラスだけに適用されます。

メタデータが発行される前提条件は，Web サービスのアプリケーションがエラーもなく，正常にデプロイされ実行が開始されている状態です。

メタデータの発行に関する注意事項を示します。

- wsdlLocation 属性には，「WEB-INF/wsdl」または「META-INF/wsdl」から始まる WSDL ファイルへの相対パスを指定してください。wsdlLocation 属性に指定された WSDL ファイルが，WEB-INF/wsdl ディレクトリまたは META-INF/wsdl ディレクトリにない場合，または WSDL の構文として不正な場合は，Web サービスのデプロイ時にエラーとなります。
- wsdlLocation 属性に指定された WSDL ファイルに wsdl:service 要素が含まれていない場合は，Web サービスのデプロイ時にエラーとなります。
- 項番 1，項番 2 の場合であっても，WEB-INF/wsdl ディレクトリまたは META-INF/wsdl ディレクトリ以下にある WSDL ファイルが次のどちらかに該当すると，Web サービスのデプロイ時にエラーとなります。
  - wsdl:service 要素を持つファイルが複数ある。
  - wsdl:portType 要素を持つファイルが複数ある（Web サービス実装クラスの場合）。
- 項番 1，項番 2 の場合であっても，その WSDL ファイルを基に，要求された Web アプリケーションの WSDL として内容を更新したものを返します。

プロバイダ実装クラスの場合，基本的に WSDL を持つ必要がありません。また，このため，標準仕様でも javax.xml.ws.WebServiceProvider アノテーションと WSDL のマッピング規則は定義されていません。そのため，メタデータを発行したい場合，作成した WSDL を WAR ファイルに適切に含めておく必要があります（Web サービス実装クラスの場合とは異なり，JAX-WS エンジンによって自動生成されません）。また，WAR ファイルに cosminexus-jaxws.xml を含めない場合，javax.xml.ws.WebServiceProvider アノテーションの portName 属性と targetNamespace 属性は必須です。WSDL の定義内容に従って，適切な値を設定してください。cosminexus-jaxws.xml については，「10.3　cosminexus-jaxws.xml によるカスタマイズ」を参照してください。

10.6.3　WSDL の更新

Web サービス側の JAX-WS エンジンは，Web サービスがデプロイされるときに，Web サービス実装クラスであれば必要に応じて WSDL ファイルを自動生成します。自動生成しない場合でも，WAR ファイルに含まれる WSDL を基に，次の情報を反映した WSDL を返します。

- ヘッダ情報

  WSDL ファイルが公開されたバージョン，および日時をヘッダ情報として追加します。追加されるヘッダ情報の例を次に示します。

  ```
  <!-- Published by Cosminexus JAX-WS 0900 (2012.01.01 00:00). -->
  ```

- ロケーション情報

  soap:address 要素の location 属性や，xsd:include 要素の schemaLocation 属性などを更新します。

  例えば，soap:address 要素の location 属性の値が"REPLACE_WITH_ACTUAL_URL"のような WSDL でも，WAR ファイルに含めることで，soap:address 要素の location 属性の値は適切な URL に更新されます。

  更新後のロケーション情報の例を次に示します。

  - ホスト名：sample.com
  - コンテキストルート：/fromjava
  - Web サービス呼び出し URL：/AddNumbersImplService

  このような場合，更新後の URL は，"http://sample.com/fromjava/AddNumbersImplService"となります。

10.6.4　メタデータ発行の有効／無効

メタデータの発行の有効／無効は，com.cosminexus.jaxws.security.publish_wsdl プロパティの値で指定できます。POJO と EJB の両方の Web サービスで指定できます。com.cosminexus.jaxws.security.publish_wsdl プロパティについては，「10.1.2　共通定義ファイルの設定項目」を参照してください。

10.6.5 WAR ファイルに複数の Web サービス実装クラスまたはプロバイダ実装クラスを含む場合の注意

メタデータは Web サービス実装クラスまたはプロバイダ実装クラス単位で取得します。WAR ファイルに複数の Web サービス実装クラスまたはプロバイダ実装クラスを含む場合，Web サービス側の JAX-WS エンジンで新たに生成して返す WSDL 定義の wsdl:binding 要素および wsdl:port 要素は，リクエストで指定した URL の Web サービス実装クラスまたはプロバイダ実装クラスに対応した wsdl:binding 要素と wsdl:port 要素だけです。WAR ファイルに含まれるほかの Web サービス実装クラスまたはプロバイダ実装クラスに対応した wsdl:binding 要素と wsdl:port 要素は含まれません。

WAR ファイル中のすべての Web サービス実装クラスまたはプロバイダ実装クラスに対応した wsdl:binding 要素と wsdl:port 要素を含むメタデータを公開したい場合，Web サービス（アプリケーション）開発者が適切なメタデータをあらかじめ作成し，WEB-INF/wsdl ディレクトリに含める必要があります。

10.6.6 WSDL 定義または XML Schema をインポート／インクルードしている場合の注意

WSDL 定義または XML Schema をインポート，インクルードしている場合，Web サービスの開発で使用した WSDL 定義や XML Schema をそのまま WAR ファイルに含めると，正常にメタデータが発行できなくなることがあります。

インポートまたはインクルード対象のファイルを WAR ファイルに含める場合，インポート元またはインクルード元の WSDL 定義や XML Schema に含まれるパス情報を確認し，次に示す方法でパス情報を適切に修正する必要があります。

- 相対パスの記述を含む WSDL 定義や XML Schema を WAR ファイルに含める場合
  インポート，インクルード対象のファイルを WAR ファイルの WEB-INF/wsdl ディレクトリ以下に含め，インポート元およびインクルード元の WSDL 定義や XML Schema の location 属性，schemaLocation 属性に指定したパス情報を，環境に合わせて適切な相対パスに修正します。
- リモートの URL の記述を含む WSDL 定義や XML Schema を WAR ファイルに含める場合
  WSDL 定義を取得する Web サービスクライアントからアクセスできる URL の場合，修正は不要です。アクセスできない URL の場合，インポート，インクルード対象のファイルを WAR ファイルの WEB-INF/wsdl ディレクトリ以下に含め，インポート元およびインクルード元の WSDL 定義や XML Schema の location 属性，schemaLocation 属性に指定したパス情報を，環境に合わせて適切な相対パスに修正します。
- ローカルの URL の記述を含む WSDL 定義や XML Schema を WAR ファイルに含める場合
  インポート，インクルード対象のファイルを WAR ファイルの WEB-INF/wsdl ディレクトリ以下に含め，インポート元およびインクルード元の WSDL 定義の location 属性，schemaLocation 属性に指定したパス情報を，環境に合わせて適切な相対パスに修正します。

10.6.7　WSDL の soap12:binding 要素の transport 属性についての注意

自動生成されて発行される SOAP 1.2 仕様の WSDL で，soap12:binding 要素の transport 属性は，デフォルトでは次の URL です。

http://www.w3.org/2003/05/soap/bindings/HTTP/

次の URL に変更したい場合は，動作定義ファイルに定義を追加してください。

変更したい URL

http://schemas.xmlsoap.org/soap/http

追加する定義の内容

```
com.cosminexus.jaxws.publish_wsdl.soap12binding=WSI_BP20_TRANSPORT
```

10.7　Web サービスの情報表示

Web サービス実装クラス，またはプロバイダ実装クラスを呼び出す URL をブラウザ上で実行すると，Web サービスの情報を表示できます。

10.7.1　表示される Web サービスの情報

Web サービスの情報を表示する場合，GET メソッドで URL を指定します。HTTP の GET メソッドを使用して，表示される Web サービスの情報を次の表に示します。

表 10–13　表示される Web サービスの情報

EndPoint	Information
Service Name： 　サービスの QName Port Name： 　ポートの QName	Address： 　Web サービスを呼び出すための URL WSDL： 　Web サービスの WSDL を取得するための URL 　メタデータの発行については，「10.6　メタデータの発行」を参照してください。 Implementation Class： 　Web サービス実装クラス名，またはプロバイダ実装クラス名

10.7.2　Web サービスの情報を表示する方法

Web サービスの情報を表示する場合，次の例に示すような URL を含む HTTP リクエストを GET メソッドで Web サービス側の JAX-WS エンジンに送信します。

http://sample.com/fromjava/AddNumbersImplService

この URL では，次の情報を指定しています。

- sample.com：ホスト名
- /fromjava：コンテキストルート
- /AddNumbersImplService：Web サービスの呼び出し

また，この機能は，com.cosminexus.jaxws.security.display_webservice_info プロパティで，有効／無効を指定できます。com.cosminexus.jaxws.security.display_webservice_info プロパティについては，「10.1.2　共通定義ファイルの設定項目」を参照してください。

10.8　使用できる HTTP メソッド

HTTP メソッドとして POST が使用できます。また，メタデータの発行および Web サービスの情報表示では GET が使用できます。POJO と EJB の両方の Web サービスで使用できます。

次に示す HTTP リクエストメソッドが Web サービス側の JAX-WS エンジンに到着した場合，HTTP ステータスコード"405 Method Not Allowed"が返されます。

- GET（メタデータの発行および Web サービスの情報表示の場合を除く）
- DELETE
- HEAD
- OPTION
- PUT
- TRACE

10.9　Web サービスの初期化と破棄

Web サービスの初期化と破棄について説明します。POJO と EJB の両方の Web サービスで動作します。

10.9.1　Web サービスの初期化

Web サービス側の JAX-WS エンジンでは，J2EE アプリケーションが J2EE サーバによって開始されるときに，Web サービスを初期化します。具体的には次の処理が実行されます。

- Web サービスのディスカバリをするための，URL と，Web サービス実装クラスまたはプロバイダ実装クラスとのマッピング情報の生成処理
- POJO の Web サービスの実行に必要なリクエスト bean，レスポンス bean，およびフォルト bean の JavaBeans クラスが WAR ファイルに含まれない場合，または EJB の Web サービスの実行に必要なリクエスト bean，レスポンス bean，およびフォルト bean の JavaBeans クラスが EAR ファイルに含まれない場合，JAX-WS エンジンがこれらの JavaBeans クラスを動的に生成
- メタデータを発行するための WSDL の確認処理（Web サービス実装クラスの場合）
- Web サービス実装クラスまたはプロバイダ実装クラスにハンドラが関連づけられている場合のハンドラチェイン設定ファイルの読み込みと，ハンドラの初期化処理
  （javax.annotation.PostConstruct アノテーションでアノテートされたメソッドがある場合はそのメソッドの呼び出しも含む）

Web サービスの初期化の開始時に，ログおよび標準出力に KDJW40001-I のメッセージが出力されます。Web サービスの初期化が正常に終了した場合，ログおよび標準出力に KDJW40003-I のメッセージが出力されます。Web サービスの初期化でエラーが発生し，初期化に失敗した場合，ログおよび標準エラー出力に KDJW40002-E のメッセージとエラーの原因となったエラーメッセージが出力されます。

Web サービスの初期化でエラーが発生した場合，ログにエラーが出力されますが，J2EE アプリケーションの開始自体は継続され，正常終了します。ただし，初期化に失敗しているので，デプロイされた Web サービスは動作しません。この場合，KDJW40002-E のメッセージが出力されたかどうかを確認し，エラーが出力された場合には問題を修正し，再度 Web サービスをデプロイしてください。

J2EE サーバの開始時に，デプロイされている J2EE アプリケーションが自動的に開始される場合など，どの J2EE アプリケーションに含まれる Web サービスの初期化に失敗したのか，調査が困難なことがあります。そのようなときは，「39.3.1　ログの種類」で示すログ，および J2EE サーバの稼働ログを確認することで，J2EE アプリケーション名とコンテキストルート名を特定できます。

"com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener"を含む，KDJE39103-E メッセージの出力内容を確認する例を次に示します。

```
0095 2009/12/18 13:48:36.471     HEJB            0125FEFA 004413EE KDJE39103-E          An
 exception javax.xml.ws.WebServiceException was raised in notification of the listener class
```

```
 com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener. (J2EE application =
Sample_application_fromwsdl, context root = /fromjava)
```

上記のようなログが出力されている場合，J2EE アプリケーション"Sample_application_fromwsdl"に含まれる，コンテキストルート名"fromjava"に関連づけられた WAR ファイル内の Web サービスの初期化が失敗しています。

ユーザプログラムに問題がある場合，基本的に KDJE39103-E のメッセージのエラーが発生します。ただし，必要に応じて次のエラーメッセージについても確認してください。

- KDJE39100-E
- KDJE39101-E
- KDJE39102-E

J2EE サーバの稼働ログについては，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「5.2　アプリケーションサーバのログ」を参照してください。また，上記のメッセージについては，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」の「7.2　KDJE30000 から KDJE39999 までのメッセージ」を参照してください。

10.9.2　Web サービスの破棄

Web サービス側の JAX-WS エンジンでは，J2EE アプリケーションが J2EE サーバによって終了されるときに，Web サービスを破棄します。具体的には次の処理が実行されます。

- Web サービスのディスカバリをするための，URL と，Web サービス実装クラスまたはプロバイダ実装クラスとのマッピング情報の削除処理
- Web サービス実装クラスまたはプロバイダ実装クラスにハンドラが関連づけられている場合のハンドラの破棄処理

  (javax.annotation.PreDestory アノテーションでアノテートされたメソッドがある場合はそのメソッドの呼び出しも含む)

Web サービスの破棄の開始時に，ログおよび標準出力に KDJW40004-I のメッセージが出力されます。Web サービスの破棄が正常に終了した場合，ログおよび標準出力に KDJW40006-I のメッセージが出力されます。Web サービスの破棄でエラーが発生し，破棄に失敗した場合，ログおよび標準エラー出力に KDJW40015-E のメッセージとエラーの原因となったエラーメッセージが出力されます。

Web サービスの破棄でエラーが発生した場合，ログにエラーが出力されますが，J2EE アプリケーションの終了処理自体は継続され，正常終了します。この場合，KDJW40015-E のメッセージが出力されたかどうかを確認し，エラーが出力された場合には問題を修正してください。

10.10　プロキシサーバ経由の接続

Web サービスクライアントからプロキシサーバを経由して，外部のネットワークにある Web サービスを利用できます。

ここでは，プロキシサーバを経由して外部に接続する場合に必要なプロパティの設定について説明します。

10.10.1　プロパティ値の指定

プロキシサーバを経由して Web サービスにアクセスするには，JavaVM のプロパティ，または Application Server の JAX-WS 機能固有のプロパティを指定して，プロキシサーバの情報を設定します。プロキシサーバ経由で接続するためのプロパティと指定内容を次の表に示します。

表 10–14　プロキシサーバ経由で接続するためのプロパティ

項番	プロパティ	指定内容	非 SSL の場合	SSL の場合
1	http.proxyHost※1	プロキシサーバのホスト名または IP アドレスを指定します。 空文字を指定した場合は，プロキシサーバに接続されません。	○	×
2	http.proxyPort※1	プロキシサーバのポート番号を設定します。 http.proxyHost が適切に設定されていた場合に，http.proxyPort に空文字を指定したときは，http.proxyHost に指定されているホストの 80 番ポートにアクセスされます。 http.proxyHost を指定していない場合は，http.proxyPort を指定しても，プロキシサーバに接続されません。	△	×
3	com.cosminexus.jaxws.http.proxyUser※2	プロキシサーバの認証ユーザ ID を設定します。 http.proxyHost プロパティ，および http.proxyPort プロパティが適切に設定されていた場合に，com.cosminexus.jaxws.http.proxyUser プロパティに空文字を指定したときは，匿名でプロキシサーバに接続します。	△	×
4	com.cosminexus.jaxws.http.proxyPassword※2	プロキシサーバの認証ユーザ ID に対応するパスワードを設定します。 次のプロパティが適切に設定されていた場合に，com.cosminexus.jaxws.http.proxyPassword プロパティに空文字を指定したときは，パスワードを設定しないでプロキシサーバに接続します。 • http.proxyHost • http.proxyPort • com.cosminexus.jaxws.http.proxyUser	△	×

項番	プロパティ	指定内容	非 SSL の場合	SSL の場合
5	https.proxyHost※1	SSL プロトコルによる接続※3 で使用するプロキシサーバのホスト名か IP アドレスを設定します。 SSL プロトコルによる接続でプロキシサーバを使用する場合は，必ず設定してください。なお，空文字を指定した場合は，プロキシサーバに接続されません。	×	○
6	https.proxyPort※1	SSL プロトコルによる接続※3 で使用するプロキシサーバのポート番号を設定します。なお，https.proxyHost が適切に設定されていた場合に，https.proxyPort に空文字を指定したときは，https.proxyHost に指定されているホストの 443 番ポートにアクセスされます。 https.proxyHost を指定していない場合は，https.proxyPort を設定しても，プロキシサーバに接続されません。	×	△
7	com.cosminexus.jaxws.https.proxyUser※2	SSL プロトコルによる接続※3 で使用するプロキシサーバの認証ユーザ ID を設定します。 https.proxyHost プロパティ，および https.proxyPort プロパティが適切に設定されていた場合に，com.cosminexus.jaxws.https.proxyUser プロパティに空文字を指定したときは，匿名でプロキシサーバに接続します。	×	△
8	com.cosminexus.jaxws.https.proxyPassword※2	SSL プロトコルによる接続※3 で使用するプロキシサーバの認証ユーザ ID に対応するパスワードを設定します。 次のプロパティが適切に設定されていた場合に，com.cosminexus.jaxws.https.proxyPassword プロパティに空文字を指定したときは，パスワードを設定しないでプロキシサーバに接続します。 • https.proxyHost • https.proxyPort • com.cosminexus.jaxws.https.proxyUser	×	△
9	http.nonProxyHosts※1	プロキシサーバを利用しないホスト名群を必要に応じて指定します。 このプロパティで指定したホストに接続する場合は，http.proxyHost または https.proxyHost に指定したプロキシサーバは経由されません。ホストを複数指定する場合は，「|」で区切って設定します。ホスト名とホスト名の間には，「|」以外の文字（空白など）は指定できません。	△	△

（凡例）

○：プロパティの指定が必須であることを示します。

△：必要に応じてプロパティを指定することを示します。

×：プロパティの指定が不要であることを示します。

注※1

JavaVM で標準でサポートされているシステムプロパティです。JavaVM のシステムプロパティについては，JavaVM のドキュメントを参照してください。

注※2

Application Server の JAX-WS 機能固有のプロパティです。このプロパティは簡易的なプロパティです。細かな制御をしたい場合，Web サービスクライアントで J2SE 6.0 標準の java.net.Authenticator クラスを利用した実装をすることをお勧めします。詳細については，「10.10.3　JAX-WS 機能固有のプロパティを利用しない場合」を参照してください。

注※3

SSL プロトコルによる接続については，「10.11　SSL プロトコルによる接続」を参照してください。

10.10.2　プロパティの設定方法

Application Server の JAX-WS 機能固有のプロパティは，動作定義ファイルに設定します。動作定義ファイルの設定方法については「10.1　動作定義ファイル」を参照してください。JAX-WS 機能固有のプロパティを利用する場合，「10.10.4　JAX-WS 機能固有のプロパティを利用する場合の注意事項」の内容に注意してください。

JavaVM のシステムプロパティの設定方法は，Web サービスクライアントの実行形態によって異なります。

- Web サービスクライアントをコマンドで実行する場合

  Web サービスクライアントをコマンド（cjclstartap）で実行する場合は，Java アプリケーション用ユーザプロパティファイル（usrconf.properties）に，JavaVM のプロパティを設定します。

- Web サービスクライアントを J2EE サーバ上で実行する場合

  Web サービスクライアントを J2EE サーバ上で実行する場合は，J2EE サーバ用ユーザプロパティファイル（usrconf.properties）に，JavaVM のプロパティを設定します。

プロパティの設定例を示します。

```
http.proxyHost=10.209.15.79
http.proxyPort=3128
https.proxyHost=10.209.15.79
https.proxyPort=3128
http.nonProxyHosts=10.209.15.80|www.xxx.co.jp
```

プロパティの設定を追加する位置についての決まりはありません。

10.10.3　JAX-WS 機能固有のプロパティを利用しない場合

JAX-WS 機能固有のプロパティは簡易的なプロパティです。そのため，細かな制御をしたい場合，Web サービスクライアントで J2SE 6.0 標準の java.net.Authenticator クラスを利用した実装をすることをお勧めします。詳細については J2SE 6 のドキュメントを参照してください。java.net.Authenticator クラスを利用した実装例を次に示します。

```
java.net.Authenticator.setDefault( new java.net.Authenticator(){

  // getPasswordAuthenticationメソッドをオーバーライドする
  public java.net.PasswordAuthentication getPasswordAuthentication() {
```

```
    // ユーザ名を設定する
    String userName = ...

    // パスワードを設定する
    char[] password = ...

    // PasswordAuthenticationを生成する。
    java.net.PasswordAuthentication auth =
        new java.net.PasswordAuthentication( userName, password );

    return auth;
  }
} );
```

10.10.4　JAX-WS 機能固有のプロパティを利用する場合の注意事項

Application Server は，Java SE 標準の java.net.Authenticator クラスの setDefault()メソッドを利用して，JAX-WS 機能固有のプロパティの値を JavaVM に設定します。そのため，次の点に注意してください。

- 有効範囲

  Web サービスクライアントが動作するプロセス全体（Web サービスクライアントが J2EE サーバ上で動作する場合は，J2EE サーバ全体）で有効となり，Application Server 以外の HTTP 接続にも影響します。Application Server 以外の HTTP 接続に JAX-WS 機能固有のプロパティによるプロキシの設定を適用したくない場合，JAX-WS 機能固有のプロパティではなく，ユーザプログラム（Web サービスクライアント）で，java.net.Authenticator クラスの setDefault()メソッドを利用した実装をする必要があります。詳細については，「10.10.3　JAX-WS 機能固有のプロパティを利用しない場合」を参照してください。

- 競合

  JAX-WS 機能固有のプロパティを利用する場合，Web サービスクライアントが動作するプロセスでは，Application Server 以外が java.net.Authenticator クラスの setDefault()メソッドを呼び出さないようにしてください。ライブラリなどでほかの製品を利用している場合は，特に注意してください。java.net.Authenticator クラスの setDefault()メソッドを呼び出すタイミングによっては設定が競合し，動作が不正になるおそれがあります。ユーザプログラムやライブラリなど，ほかの製品が java.net.Authenticator クラスの setDefault()メソッドを呼び出す場合，JAX-WS 機能固有のプロパティを利用しないでください。

- セキュリティ例外

  java.net.Authenticator クラスの setDefault()メソッドが java.lang.SecurityException をスローする場合，KDJW10025-W のメッセージがログに出力され，処理が続行されます。必要に応じて，詳細メッセージを確認し，エラーの要因を取り除いてください。

10.11　SSL プロトコルによる接続

Web サービスクライアントから，SSL プロトコルに対応した Web サービスに接続できます。

ここでは，SSL プロトコルによる接続に必要なプロパティの設定について説明します。

10.11.1　プロパティ値の指定

SSL プロトコルで Web サービスにアクセスするには，JDK にサポートされているプロパティに値を指定して，SSL プロトコルの情報を設定します。SSL プロトコルによる接続をするためのプロパティ，および指定内容を次の表に示します。

表 10–15　SSL プロトコルによる接続をするためのプロパティ

項番	プロパティ	指定内容
1	javax.net.ssl.trustStore	トラストストアを指定します。
2	javax.net.ssl.trustStorePassword	トラストストアのパスワードを指定します。

これらのプロパティは必要に応じて指定してください。なお，トラストストアを指定しない場合は，<JDK インストールディレクトリ>/lib/security/jssecacerts などのデフォルト値が使用されます。

JDK のプロパティについては，JDK のドキュメントを参照してください。

10.11.2　プロパティの設定方法

プロパティの指定値を有効にするために，プロパティをシステムプロパティに設定します。プロパティの設定方法は，Web サービスクライアントの実行形態によって異なります。

- Web サービスクライアントをコマンドで実行する場合
  Web サービスクライアントをコマンド（cjclstartap）で実行する場合は，Java アプリケーション用ユーザプロパティファイル（usrconf.properties）に，JavaVM のプロパティを設定します。
- Web サービスクライアントを J2EE サーバ上で実行する場合
  Web サービスクライアントを J2EE サーバ上で実行する場合は，J2EE サーバ用ユーザプロパティファイル（usrconf.properties）に，JavaVM のプロパティを設定します。

プロパティの設定例を示します。

```
javax.net.ssl.trustStore=<トラストストア>
javax.net.ssl.trustStorePassword=<トラストストアのパスワード>
```

プロパティの設定を追加する位置についての決まりはありません。

10.11.3　ホスト名検証についての注意事項

Web サービスクライアントから，SSL プロトコルに対応した Web サービスに接続する場合，エンドポイントアドレスに含まれるホスト名と，証明書のホスト名が一致しているかどうかが検証されます。使用される HostnameVerifier は JDK のデフォルトの実装です。JDK のデフォルトの HostnameVerifier の動作については JDK のドキュメントを参照してください。

動作定義ファイル，またはメッセージコンテキストで設定を行うことでホスト名検証が行われないようにすることができます。ホスト名検証が行われないようにするためのプロパティ，および指定内容を次の表に示します。

表 10–16　ホスト名検証が行われないようにするためのプロパティ

項番	プロパティ	指定内容	必須
1	com.cosminexus.xml.ws.client.http.HostnameVerificationProperty	ホスト名検証を省略する場合，true を指定します。省略しない場合，false を指定します。	△

（凡例）

△：必要に応じてプロパティを指定することを示します。

ホスト名検証が行われないようにする場合の注意事項

- メッセージコンテキストへの指定が有効になるのは，Web サービス呼び出し時だけで，Web サービス呼び出し前の javax.xml.ws.Service クラス生成で発生するメタデータ（WSDL）取得時には適用されません。

  メタデータ取得時にホスト名検証の有無を設定する場合は，共通定義ファイルまたはプロセス別の定義ファイルに記述するか，別途 WSDL をローカルマシンにダウンロードして使用してください（ローカルマシンにある WSDL を使用する場合は，メタデータ取得時にリモートマシンへの接続が発生しません）。WSDL から別途インポートされる WSDL が存在する場合は，インポートされる WSDL もローカルマシン上にダウンロードしてください。

- 同じプロセスで動作する複数の Web サービスクライアントによって，ホスト名検証の有無が異なる場合は，プロセス別の定義ファイルや共通定義ファイルにプロパティを含めないで，メッセージコンテキストだけに含めるようにしてください。

  同様に，同じシステムで動作する複数のプロセスによって，ホスト名検証の有無が異なる場合は，共通定義ファイルにプロパティを含めないで，プロセス別の定義ファイルかメッセージコンテキストだけに含めるようにしてください。

プロパティを動作定義ファイルに設定する方法については，「10.1.2　共通定義ファイルの設定項目」を参照してください。メッセージコンテキストに設定する方法については，「19.2.5　メッセージコンテキストの使用」を参照してください。

10.12　ベーシック認証による接続

Web サービスクライアントから，ベーシック認証に対応した Web サービスに接続できます。

ここでは，ベーシック認証による接続に必要なプロパティの設定について説明します。

10.12.1　プロパティ値の指定

ベーシック認証で Web サービスにアクセスするには，動作定義ファイル，またはメッセージコンテキストに値を指定します。ベーシック認証による接続をするためのプロパティ，および指定内容を次の表に示します。

表 10–17　ベーシック認証による接続をするためのプロパティ

項番	プロパティ	指定内容	必須
1	javax.xml.ws.security.auth.username	ユーザ ID を設定します。	○
2	javax.xml.ws.security.auth.password	パスワードを設定します。	○

（凡例）

○：プロパティの指定が必須であることを示します。

ベーシック認証による接続をする場合の注意事項

- メッセージコンテキストへの指定が有効になるのは Web サービス呼び出し時だけで，Web サービス呼び出し前の javax.xml.ws.Service クラス生成で発生するメタデータ（WSDL）取得時には適用されません。

  メタデータ取得時にベーシック認証の情報を設定する場合は，共通定義ファイルまたはプロセス別の定義ファイルに記述するか，別途 WSDL をローカルマシンにダウンロードして使用してください（ローカルマシンにある WSDL を使用する場合は，メタデータ取得時にリモートマシンへの接続が発生しません）。WSDL から別途インポートされる WSDL が存在する場合は，インポートされる WSDL もローカルマシン上にダウンロードしてください。

- 同じプロセスで動作する複数の Web サービスクライアントによって，ベーシック認証を行うか，行わないかが異なる場合は，プロセス別の定義ファイルや共通定義ファイルにこれらのプロパティを含めないで，メッセージコンテキストだけに含めるようにしてください。

  同様に，同じシステムで動作する複数のプロセスによって，ベーシック認証を行うか，行わないかが異なる場合は，共通定義ファイルにこれらのプロパティを含めないで，プロセス別の定義ファイルかメッセージコンテキストだけに含めるようにしてください。

10.12.2　プロパティの設定方法

プロパティを動作定義ファイルに設定する方法については，「10.1.2　共通定義ファイルの設定項目」を参照してください。メッセージコンテキストに設定する方法については，「19.2.5　メッセージコンテキストの使用」を参照してください。

10.13　SOAP のバージョン選択

ここでは，Web サービス，および Web サービスアプリケーションの開発時に必要となる，SOAP のバージョン選択について説明します。

10.13.1　SOAP のバージョン選択（Web サービス開発時）

WSDL 起点，SEI 起点，およびプロバイダ起点の Web サービス開発時のバージョン選択方法について説明します。

(1)　WSDL 起点の場合

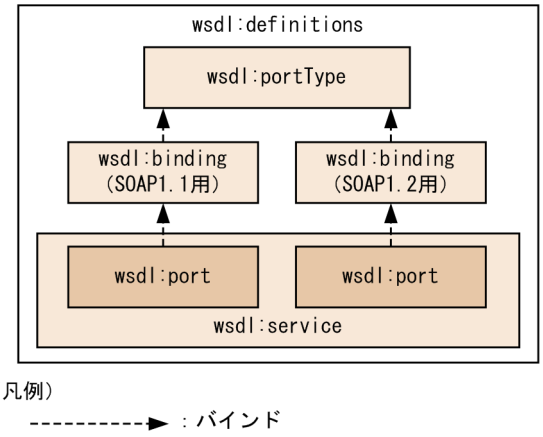
SOAP 1.1 仕様または SOAP 1.2 仕様のどちらのメッセージを送受信するのかを，WSDL に記述するバインディングで決定します。

cjwsimport コマンドで生成した Web サービス実装クラスのスケルトンはバインディング単位のため，SOAP 1.1 仕様または SOAP 1.2 仕様のどちらか専用となります。実行時に動的に変更することはできません。

一方，WSDL には複数のバインディングを記述できるので，一つの WSDL に，SOAP 1.1 仕様および SOAP 1.2 仕様のポートを混在させることができます。

SOAP 1.1 仕様および SOAP 1.2 仕様のポートを混在させる例を次に示します。

図 10–8　ポートを混在させる例



これは，一つのポートタイプを SOAP 1.1 用と SOAP 1.2 用にそれぞれバインドする例です。一つのポートタイプで，SOAP 1.1 仕様および SOAP 1.2 仕様のどちらの形式のメッセージも受け付けられます。

(2)　SEI 起点の場合

SOAP 1.1 仕様または SOAP 1.2 仕様のどちらにバインドするのかを，アノテーションで指定します。Web サービス実装クラスの単位で指定してください。実行時に動的に変更することはできません。

アノテーションは省略できます。省略した場合は，SOAP 1.1 仕様になります。

(3) プロバイダ起点の場合

SOAP 1.1 仕様または SOAP 1.2 仕様のどちらにバインドするのかを，アノテーションで指定します。プロバイダ実装クラスの単位で指定してください。実行時に動的に変更することはできません。

10.13.2 SOAP のバージョン選択（Web サービスクライアント開発時）

Web サービスクライアントは，Web サービスのメタデータ（WSDL，または接続しようとしている Web サービスに依存する情報など）に基づいて開発します。メタデータで定義される情報に基づき，適切なバージョンの SOAP で通信する Web サービスクライアントを実装してください。

スタブベース，ディスパッチベース，および API ベースの Web サービスクライアント開発時の，SOAP のバージョン選択方法について説明します。

(1) スタブベースの場合

cjwsimport コマンドを実行すると，WSDL の定義に基づいて適切なスタブを自動生成します。そのため，SOAP 1.1 仕様または SOAP 1.2 仕様のどちらで通信するのかを指定する必要はありません。

なお，ポートは必ず一つのバインディングにバインドされます。そのため，サービスクラスに含まれるポート取得メソッドは，SOAP 1.1 仕様または SOAP 1.2 仕様のどちらか専用となります。

(2) ディスパッチベースの場合

Web サービスのメタデータに基づいて適切なバージョンの SOAP を選択し，API で指定してください。

(3) API ベースの場合

Web サービスのメタデータに基づいて適切なバージョンの SOAP を選択し，API で指定してください。

10.13.3 SOAP のバージョン選択（実行時）

Web サービス，および Web サービスクライアント実行時の SOAP のバージョン選択方法について説明します。

(1) Web サービスの場合

Web サービス実装クラスおよびプロバイダ実装クラスは，開発時の選択に基づいて，SOAP 1.1 仕様または SOAP 1.2 仕様のどちらか専用となります。そのため，受信できるメッセージは，SOAP 1.1 仕様また

は SOAP 1.2 仕様のどちらかです。両方のバージョンの SOAP に対応するメッセージを受信したい場合は，複数の Web サービス実装クラスまたはプロバイダ実装クラスを定義・実装する必要があります。

(2) Web サービスクライアントの場合

スタブベースの Web サービスクライアントは，SOAP 1.1 仕様または SOAP 1.2 仕様のどちらかのメッセージを送信します。開発時の誤りで次のどちらかの状態になった場合，SOAP のバージョンが一致しない SOAP メッセージを送信することになり，Web サービス側でエラーが発生します。

- Web サービスが SOAP 1.2 仕様のバインディングの場合に SOAP 1.1 仕様用のスタブを生成している
- Web サービスが SOAP 1.1 仕様のバインディングの場合に SOAP 1.2 仕様用のスタブを生成している

ディスパッチベースや API ベースの Web サービスクライアントでは，SOAP のバージョンを API で指定します。指定値を設定ファイルで変更できるようにするなど，実装方法によってはどちらの SOAP のバージョンのメッセージを送信するかを動的に変更できます。

10.14　コマンドラインを利用したクライアントアプリケーションの実行

Web サービスでは，コマンドラインで動作する Java アプリケーションを，クライアントアプリケーションとして利用できます。

ここでは，コマンドラインによるクライアントアプリケーションを利用するときに必要な設定，コマンドラインの指定例，および注意事項について説明します。

10.14.1　コマンドライン利用時の設定

コマンドラインによるクライアントアプリケーションを利用するために必要な設定について説明します。

(1)　Java アプリケーション用オプション定義ファイルの設定

クライアントアプリケーションを実行するカレントディレクトリに，Java アプリケーション用オプション定義ファイルを作成し，次に示すキーと値を追加してください。

- add.class.path=<クライアント Web サービスのクラスファイルが格納された JAR またはディレクトリのパス>
- add.class.path=<Application Server のインストールディレクトリ>/jaxws/lib/cjjaxws.jar
- add.jvm.arg=-Dcosminexus.home=<Application Server のインストールディレクトリ>
- add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF 識別子>※
- ejb.client.log.directory=<ログファイルを出力するディレクトリパス>

注※

cprfstart コマンドで指定した PRF 識別子と同じ識別子を指定してください。指定しない場合のデフォルト値は，PRF_ID です。

(2)　Java アプリケーション用ユーザプロパティファイルの設定

クライアントアプリケーションを実行するカレントディレクトリに，Java アプリケーション用ユーザプロパティファイルを作成し，必要に応じて設定してください。

(3)　パスの追加

環境変数 PATH に，次に示すパスを追加してください。

<Application Server のインストールディレクトリ>/PRF/bin

10.14.2　コマンドラインの実行

Web サービスクライアントをコマンドラインから実行する場合の指定例を示します。

(1)　PRF デーモンの起動

PRF 識別子として，デフォルト値を使用する場合の指定例を次に示します。

<Application Server のインストールディレクトリ>/PRF/bin/cprfstart

PRF 識別子として，デフォルト値以外を使用する場合の指定例を次に示します。

<Application Server のインストールディレクトリ>/PRF/bin/cprfstart <prfid>※

注※

<prfid>は PRF 識別子を示します。PRF 識別子は，Java アプリケーション用オプション定義ファイルに指定した PRF 識別子と同じ識別子を指定してください。

(2)　アプリケーションのクライアントの実行

Java アプリケーションの開始コマンド（cjclstartap）で実行します。実行例を次に示します。

cjclstartap localhost.testMain

10.14.3　コマンドライン利用時の注意事項

コマンドラインを利用するときの注意事項について説明します。

- ログは，Java アプリケーション用オプション定義ファイルの ejb.client.log.directory プロパティに指定したファイルパスに出力されます。ログの出力については，「39.3.3(2)　コマンドラインインタフェースで Web サービスクライアントを利用する場合」を参照してください。
- コマンドラインを利用している場合で，PRF トレースを取得するときは，アプリケーションを実行する前に PRF デーモンを起動しておく必要があります。PRF デーモンを起動していない場合は，PRF トレースが取得されないまま処理が続行されます。

10.15　HTTP ステータスコード

サービス側 JAX-WS エンジンが返す HTTP ステータスコードの一覧を次の表に示します。

表 10–18　HTTP ステータスコード一覧

項番	HTTP ステータスコード	HTTP ステータスコードを返す条件
1	200 OK	Request-Response オペレーション型の Web サービスの呼び出しが正常終了した場合。
2	202 Accepted	one-way オペレーション型の Web サービスの呼び出し，またはアドレッシング機能を利用している場合に，非同期呼び出しに対応した Web サービスの呼び出しが正常終了したとき。
3	404 Not Found	メタデータの発行が有効な場合に，クエリストリングの形式が不正であるとき。この場合，HTTP ステータスコードが「405 Method Not Allowed」となることがあります。 メタデータの発行については，「10.6　メタデータの発行」を参照してください。
4	405 Method Not Allowed	次のどれかの場合。 • 使用できない HTTP メソッドが到着した場合 使用できる HTTP メソッドについては，「10.8　使用できる HTTP メソッド」を参照してください。 • メタデータの発行が無効な場合に，メタデータの発行を要求する HTTP リクエストを受け取ったとき メタデータの発行については，「10.6　メタデータの発行」を参照してください。 • Web サービスの情報表示が無効な場合に，Web サービスの情報表示を要求する HTTP リクエストを受け取ったとき Web サービスの情報表示については，「10.7　Web サービスの情報表示」を参照してください。
5	415 Unsupported Media Type	Content-Type HTTP ヘッダが存在しないか，または不正な場合。 この場合，HTTP ステータスコードが「500 Internal Server Error」となることがあります。
6	500 Internal Server Error	上記を除くエラーが発生した場合。Web サービスの実行結果が SOAP フォルトとなる場合もこの HTTP ステータスコードが返ります。

10.16 HTTP ヘッダ

JAX-WS エンジンの HTTP ヘッダについて説明します。

Content-Type ヘッダの action パラメタについて

JAX-WS エンジンは，Content-Type ヘッダの action パラメタの値を引用符「"」で囲んで，HTTP リクエストおよび HTTP レスポンスの Content-Type ヘッダの action パラメタの値として設定します。すでに引用符「"」で囲まれている値は，そのまま Content-Type ヘッダの action パラメタの値として設定します。

Web サービス側でも Web サービスクライアント側でも同様に動作します。

10.17　HTTP リクエストボディの gzip 圧縮

HTTP リクエストボディを gzip 圧縮すると，Web サービスクライアントと Web コンテナ間の HTTP リクエスト通信に掛かる時間を削減できます。HTTP リクエストボディを圧縮するには，Web サービスクライアントからリクエストメッセージを送信する際に，gzip 形式で圧縮された HTTP リクエストボディを送信することを示す HTTP ヘッダを付ける必要があります。クライアントアプリケーションで HTTP ヘッダを付ける処理を実装してください。

クライアントアプリケーションでの実装例を次に示します。

```
Map<String, List<String>> httpHeaders =
  ( Map<String, List<String>> )context.get( MessageContext.HTTP_REQUEST_HEADERS );
if( null == httpHeaders ){
  httpHeaders = new HashMap<String, List<String>>();
}
List<String> contentEncondings = httpHeaders.get( "Content-Encoding" );
if( null == contentEncondings ){
  contentEncondings = new ArrayList<String>();
}
contentEncondings.add( "gzip" );
httpHeaders.put( "Content-Encoding", contentEncondings );
context.put( MessageContext.HTTP_REQUEST_HEADERS, httpHeaders );
```

10.18　HTTP レスポンス圧縮機能との連携

HTTP レスポンスボディを gzip 圧縮して Web コンテナと Web サービスクライアント間の HTTP レスポンス通信に掛かる時間を削減する Application Server の機能を HTTP レスポンス圧縮機能といいます。

JAX-WS エンジンは，HTTP レスポンス圧縮機能と連携できます。HTTP レスポンス圧縮機能と連携するには，Web サービスクライアントからリクエストメッセージを送信する際に，gzip 形式で圧縮された HTTP レスポンスを受信できることを示す HTTP ヘッダを付ける必要があります。クライアントアプリケーションで HTTP ヘッダを付ける処理を実装してください。

クライアントアプリケーションでの実装例を次に示します。

```
Map<String, List<String>> httpHeaders =
  ( Map<String, List<String>> )context.get( MessageContext.HTTP_REQUEST_HEADERS );
if( null == httpHeaders ){
  httpHeaders = new HashMap<String, List<String>>();
}
List<String> acceptEncondings = httpHeaders.get( "Accept-Encoding" );
if( null == acceptEncondings ){
  acceptEncondings = new ArrayList<String>();
}
acceptEncondings.add( "gzip" );
httpHeaders.put( "Accept-Encoding", acceptEncondings );
context.put( MessageContext.HTTP_REQUEST_HEADERS, httpHeaders );
```

10.19　EJB の Web サービス呼び出し

EJB を Web サービスとして呼び出せる条件と使用できる機能について説明します。

EJB の条件を次に示します。

- EJB のバージョン
  EJB 3.0 以降で Web サービスとして呼び出せます。
- EJB の種類
  ステートレスセッション Bean の EJB を Web サービスとして呼び出せます。
- インタフェース
  EJB Web サービスでは，ビジネスインタフェースを用意する必要はありません。ホームインタフェースを併用できます。
  EJB Web サービスが，ビジネスインタフェース，ホームインタフェース，およびコンポーネントインタフェースを持つ場合，Web サービス経由ではこれらのインタフェースを介してメソッドを呼び出すことはできません。
  EJB Web サービスがビジネスインタフェースを持つ場合，EJB Web サービスは EJB のローカル呼び出しができます。EJB Web サービスがビジネスインタフェースを持たない場合，EJB Web サービスは Web サービスとして呼び出せますが，EJB として呼び出すことはできません。
  ホームインタフェースは javax.ejb.RemoteHome アノテーションまたは javax.ejb.LocalHome アノテーションによって指定される場合だけ Web サービスとして呼び出せます。DD によって指定される場合は Web サービスとして呼び出せません。

10.19.1　EJB の Web サービス呼び出しでの EJB 機能

EJB を Web サービスとして呼び出す際に，同時に使用できる EJB の機能を次に示します。ただし，EJB の Web サービス実装クラスだけで使用でき，ハンドラチェインでは使用できません。

- インターセプタの使用
- CMT および BMT のトランザクション管理
- javax.annotation.security.PermitAll アノテーションおよび javax.annotation.security.DenyAll アノテーションによるアクセス管理
- リソース接続
- javax.annotation.Resource アノテーションによる Web サービスコンテキストのインジェクション
  Web サービスコンテキストのインジェクションについては，「10.21.2　Web サービスコンテキストのインジェクション」を参照してください。
- Timer Service

注意事項

タイムアウトメソッドは，サービスメソッドとして公開できません。タイムアウトメソッドを public 以外の修飾子にするか，または exclude 要素の要素値が"true"の javax.jws.WebMethod アノテーションをアノテートしてサービスメソッドから除外してください。

EJB を Web サービスとして呼び出す際に，同時に使用できない EJB の機能を次に示します。

- クライアントからのトランザクションコンテキストの引き継ぎ
- クライアントからのセキュリティコンテキストの引き継ぎ

10.19.2 EJB の Web サービス呼び出しで使用できるアプリケーションサーバ機能

JAX-WS は，EJB を Web サービスとして呼び出す際に，cosminexus.xml で設定できるアプリケーションサーバの機能を同時に使用できます。使用できるアプリケーションサーバの機能を次に示します。

- 同時実行スレッド数の設定
- 実行待ちキューサイズの設定
- セキュリティロールの設定

これらの機能を使用するには，WAR ファイル名を設定します。WAR ファイル名の設定については「3.5.4(3)　設定用 WAR ファイル名」を参照してください。なお，これらの cosminexus.xml の設定項目以外でも，EJB Web サービス以外の Web アプリケーションでは利用できます。

10.20　sun.net.www.http.HttpClient によるリクエスト再送抑止

Web サービスクライアント側の JAX-WS エンジンは，JDK の HTTP クライアント実装を利用して通信しています。JDK の HTTP クライアント実装は，RFC 2616 に反して HTTP 通信でエラーが発生し，サーバから正しいレスポンスを受け取れなかった場合，一度だけリクエストを再送します。JDK のシステムプロパティを使用するとリクエストの再送を抑止できます。

プロパティ値の指定

JDK の HTTP クライアント実装によるリクエスト再送を抑止するには，システムプロパティに sun.net.http.retryPost=false を指定します。このシステムプロパティは，JDK 6 以降で標準でサポートされているプロパティです。システムプロパティについては，JDK のドキュメントを参照してください。

プロパティの設定方法

SSL プロトコルによる接続をするためのシステムプロパティと同じ設定方法です。設定方法については，「10.11.2　プロパティの設定方法」を参照してください。

10.21　インジェクション

サービスクラスおよびポートのインジェクションと javax.xml.ws.WebServiceRef アノテーション，および Web サービスコンテキストのインジェクションについて説明します。

10.21.1　サービスクラスおよびポートのインジェクション

J2EE サーバ上で動作する Web サービスクライアントの，次に示すフィールドおよびメソッドに javax.xml.ws.WebServiceRef アノテーションを指定すると，J2EE サーバが，Web サービスクライアントのインスタンス生成時にサービスクラスおよびポートの生成とインジェクションを行います。javax.xml.ws.WebServiceRef アノテーションについては，「19.3　アノテーションのサポート範囲」を参照してください。

- サービスクラス型およびポート型のフィールド（static および final を除く）
- サービスクラス型およびポート型を引数とする setter メソッド（static および final を除く）

javax.xml.ws.WebServiceRef アノテーションによるインジェクションの利用には，次のような利点があります。

- アプリケーションの開発では，javax.xml.ws.WebServiceRef アノテーションによるインジェクションを利用しない場合に比べコーディング量を削減できるため，Web サービスクライアントの作成が容易になります。
- J2EE アプリケーションの開始時に Web サービスクライアントのインスタンスを生成すると，ポートのインジェクションを利用しない場合に比べ，Web サービスクライアント実行時の性能を改善できます。

Web サービスクライアントのインスタンスの生成については，「10.21.1(2)　Web サービスクライアントのインスタンス生成」を参照してください。

注意事項

- javax.xml.ws.WebServiceRef アノテーションを指定できるのは，Web サービスクライアントをサーブレットまたは EJB として実装する場合だけです。これ以外のアプリケーションとして実装するときは指定できません。例えば，コマンドラインアプリケーションの Web サービスクライアントで，サービスクラスやポートをインジェクトすることはできません。
- サービスクラスおよびポートのインスタンスは，J2EE アプリケーションの開始時に一度だけ生成します。生成するインスタンスの数は，javax.xml.ws.WebServiceRef アノテーションを指定したサービスクラスおよびポートごとに一つだけです。
- サービスクラスおよびポートのインジェクションは，Web サービスクライアントのインスタンスを生成するごとに行います。
- Web サービスクライアントを EJB として実装する場合，Stateless Session Bean でのプーリングを有効に設定することで，J2EE アプリケーションの開始時に複数の Web サービスクライアントのインスタンスを生成しておくことができます。この場合，javax.xml.ws.WebServiceRef アノテー

ションを指定したサービスクラスおよびポートのインスタンスを一つ生成して，それを Web サービスクライアントの各インスタンスにインジェクトします。Web サービスクライアントのインスタンス生成については，「10.21.1(2)　Web サービスクライアントのインスタンス生成」を参照してください。

- Web サービス A から Web サービス B を呼び出す構成で，javax.xml.ws.WebServiceRef アノテーションを指定して Web サービス B のサービスクラスやポートを，Web サービス A にインジェクトすることはできません。
- Web サービスクライアントが参照するクラスで javax.xml.ws.WebServiceRef アノテーションは指定できません。
- javax.xml.ws.WebServiceRef アノテーションによってサービスクラスやポートのインジェクションを行う J2EE アプリケーションを，リロード機能を使用して入れ替えることはできません。入れ替える場合は，入れ替え前の J2EE アプリケーションを停止・削除してから，入れ替える J2EE アプリケーションをインポート・開始してください。リロード機能の詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」を参照してください。

(1)　javax.xml.ws.WebServiceRef アノテーションの指定例

javax.xml.ws.WebServiceRef アノテーションの指定例を次に示します。

- フィールドへの指定例

```
...
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.ws.WebServiceRef;

import com.sample.AddNumbersImpl;
import com.sample.AddNumbersImplService;

public class TestClient extends HttpServlet {
    // サービスクラスの例
    // サービスクラス型のフィールドにサービスクラスのインスタンスをインジェクトする
    @WebServiceRef
    private AddNumbersImplService service;

    // ポートの例
    // ポート型のフィールドにポートのインスタンスをインジェクトする
    @WebServiceRef(AddNumbersImplService.class)
    private AddNumbersImpl port;

    @Override
    public void init() {
        // Webサービスクライアントの実行前にアプリケーションサーバがサービスクラスと
        // ポートをインジェクトするので，以下の処理は行う必要がない
        //service = new AddNumbersImplService();
        //port = service.getAddNumbersImplPort();
    }
```

```
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException {
        ...
```

- setter メソッドへの指定例

```
...
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.ws.WebServiceRef;

import com.sample.AddNumbersImpl;
import com.sample.AddNumbersImplService;

public class TestClient extends HttpServlet {
    private AddNumbersImplService service;
    private AddNumbersImpl port;

    @Override
    public void init() {
        // アプリケーション開始時にアプリケーションサーバがsetterメソッドを使用して
        // サービスクラスとポートをインジェクトするので，以下の処理は行う必要がない
        //service = new AddNumbersImplService();
        //port = service.getAddNumbersImplPort();
    }

    // サービスクラスの例
    @WebServiceRef
    public void setAddNumbersImplService(AddNumbersImplService service) {
        // 引数serviceにサービスクラスのインスタンスをインジェクトする
        this.service = service;
    }

    // ポートの例
    @WebServiceRef(AddNumbersImplService.class)
    public void setAddNumbersImpl(AddNumbersImpl port) {
        // 引数portにポートのインスタンスをインジェクトする
        this.port = port;
    }

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException {
        ...
```

(2) Web サービスクライアントのインスタンス生成

J2EE アプリケーションの開始時に Web サービスクライアントのインスタンスを生成するには，次のように設定します。

サーブレットの場合

Web サービスクライアントが含まれる WAR ファイルの web.xml に load-on-startup 要素を指定します。なお，load-on-startup 要素を指定しないときは，最初の Web アプリケーション実行時にインスタンスを生成します。詳細については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」を参照してください。

EJB の場合

Stateless Session Bean でのプーリングを有効に設定します。J2EE アプリケーション開始時に，Stateless Session Bean でのプーリングの最小値分だけインスタンス生成を実行します。なお，Stateless Session Bean でのプーリングを使用しないときは，最初の J2EE アプリケーション実行時にインスタンスを生成します。詳細については，「アプリケーションサーバ 機能解説 基本・開発編(EJB コンテナ)」を参照してください。

注意事項

Web サービスクライアントと接続先の Web サービスが，同じ J2EE サーバ上にデプロイされている環境では，J2EE サーバの起動時にインジェクションに失敗します（KDJW40043-E)。

インジェクションが失敗する場合の対策方法を次に示します。

- javax.xml.ws.WebServiceRef アノテーションの wsdlLocation 要素に，相対パスまたは絶対パスでローカルに格納した WSDL 文書を指定してください。
- J2EE サーバを停止する前に，Web サービスクライアントが含まれる J2EE アプリケーションを停止してください。J2EE サーバを再起動したあとに，Web サービスクライアントを開始してください。
- Web サービスと Web サービスクライアントを別の J2EE サーバにデプロイして，Web サービスをデプロイした J2EE サーバを起動したあとに Web サービスクライアントをデプロイした J2EE サーバを起動してください。
- カタログ機能を使用して，サービスクラスまたはポートのインジェクションでローカルに格納した WSDL 文書を参照するように設定してください。

(3) ハンドラフレームワークの使用

javax.xml.ws.WebServiceRef アノテーションを指定してインジェクトしたサービスクラスまたはポートでハンドラフレームワークを使用する場合，API を使用してハンドラチェインを設定します。ハンドラチェインの設定については，「36.9.2 Web サービスクライアント側のハンドラチェインの設定」を参照してください。なお，ハンドラチェインの設定は，ポートに対して一度だけ実行すればいいので，Web サービスクライアントの初期化処理での実行をお勧めします。Web サービスを呼び出すごとに設定する必要はありません。初期化処理を実行するメソッドを次に示します。

Web サービスクライアントをサーブレットとして実装する場合

init メソッドまたは javax.annotation.PostConstruct アノテーションを指定したメソッド

EJB として実装する場合

javax.annotation.PostConstruct アノテーションを指定したメソッド

(4) フィーチャの有効化

ポート型のフィールドや，フィールドに対応する setter メソッドに，javax.xml.ws.WebServiceRef アノテーションとフィーチャに対応するアノテーションを同時に指定すると，インジェクションを行うポートのフィーチャを有効化できます。ただし，フィーチャに対応するアノテーションを指定したフィールドまたはフィールドに対応する setter メソッドに，javax.xml.ws.WebServiceRef アノテーションを指定しない場合は，フィーチャは有効になりません。また，サービスクラス型のフィールドやフィールドに対応する setter メソッドに対しては，フィーチャに対応するアノテーションを指定できません。

フィーチャを有効化するときに，ポートに指定できるアノテーションを次に示します。各アノテーションについては，「16.2　Java から WSDL へのマッピングのカスタマイズ」を参照してください。

- javax.xml.ws.soap.Addressing
- javax.xml.ws.soap.MTOM
- com.sun.xml.ws.developer.StreamingAttachment

インジェクションを行うポートで MTOM/XOP 仕様形式の添付ファイルを利用できるように，フィーチャを有効化する場合の指定例を次に示します。

- フィールドへの指定例

```
...
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.ws.WebServiceRef;

import com.sample.AddNumbersImpl;
import com.sample.AddNumbersImplService;

public class TestClient extends HttpServlet {
    // インジェクションを行うポートでMTOM/XOP仕様形式の添付ファイルを有効化
    @MTOM
    @WebServiceRef(AddNumberImplService.class)
    private AddNumbersImpl port;

    @Override
    public void init() {
        ...
    }

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException {
        ...
```

- setter メソッドへの指定例

```
...
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.ws.WebServiceRef;

import com.sample.AddNumbersImpl;
import com.sample.AddNumbersImplService;

public class TestClient extends HttpServlet {
    private AddNumbersImpl port;

    @Override
    public void init() {
        ...
    }

    // インジェクションを行うポートでMTOM/XOP仕様形式の添付ファイルを有効化
    @MTOM
    @WebServiceRef(AddNumberImplService.class)
    public void setAddNumbersImpl(AddNumbersImpl port) {
        this.port = port;
    }

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException {
        ...
```

(5) 要求コンテキストのプロパティ変更

インジェクションを行うポートの要求コンテキストのプロパティを変更する際は，Web サービスクライアントの初期化処理での実行をお勧めします。初期化処理を実行するメソッドを次に示します。

サーブレットとして実装する場合

init メソッドまたは javax.annotation.PostConstruct アノテーションを指定したメソッド

EJB として実装する場合

javax.annotation.PostConstruct アノテーションを指定したメソッド

注意事項

複数スレッドでポートを共有する Web サービスクライアントで，複数スレッドの動作中にポートの要求コンテキストのプロパティを変更すると，通信が失敗したり，不正な SOAP メッセージが送信されたりします。このため，複数スレッドで共有するポートの要求コンテキストのプロパティの変更は，複数スレッドが動作する前に実行する必要があります。

サーブレットとして実装した Web サービスクライアントの，ポートの要求コンテキストのプロパティを変更する例を次に示します。

```
...
import java.util.Map;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.WebServiceRef;

import com.sample.AddNumbersImpl;
import com.sample.AddNumbersImplService;

public class TestClient extends HttpServlet {

    @WebServiceRef(AddNumbersImplService.class)
    AddNumbersImpl port;

    @Override
    public void init() {
        // 初期化処理で，要求コンテキストを設定する
        Map<String, Object> context = ((BindingProvider)port).getRequestContext();
        context.put("com.cosminexus.jaxws.connect.timeout", 60000);
    }
    ...
}
```

10.21.2　Web サービスコンテキストのインジェクション

javax.xml.ws.WebServiceContext インタフェースは，JAX-WS 2.2 仕様 5.3 節に記載されているサービス API の一つです。Web サービスコンテキストは，javax.annotation.Resource アノテーションを用いて，リソースをインジェクトして使用します。

Web サービス実装クラスまたはプロバイダ実装クラスの，javax.xml.ws.WebServiceContext 型のフィールドまたは javax.xml.ws.WebServiceContext 型を引数とする setter メソッドに，javax.annotation.Resource アノテーションを指定します。これによって，指定したフィールドまたは指定した setter メソッドに対応するフィールドに，処理中のリクエストに関連する情報をインジェクトします。javax.xml.ws.WebServiceContext インタフェースの getMessageContext メソッドでメッセージコンテキストを取得することで，メッセージコンテキストの情報にアクセスできます。ただし，EJB の Web サービス実装クラスに EJB としてアクセスする場合，getMessageContext メソッドではメッセージコンテキストを取得できません。また，次のこともできます。

- インバウンド時のサービス側ハンドラで追加した APPLICATION スコープのユーザ定義メッセージコンテキストプロパティを，Web サービス実装クラスまたはプロバイダ実装クラスから参照できます。
- Web サービス実装クラスまたはプロバイダ実装クラスで追加したユーザ定義メッセージコンテキストプロパティを，アウトバウンド時のサービス側ハンドラから参照できます。

getMessageContext メソッドについては，「19.2.3(2)　javax.xml.ws.WebServiceContext インタフェース」を，メッセージコンテキストについては，「19.2.5　メッセージコンテキストの使用」を参照してください。

(1) javax.annotation.Resource アノテーションの指定

Web サービスコンテキストのインジェクションで javax.annotation.Resource アノテーションを使用する場合，Web サービス実装クラスまたはプロバイダ実装クラス（実装クラスの親クラスを含む）の次に示すフィールドまたはメソッドに javax.annotation.Resource アノテーションを指定できます。このフィールドとメソッド以外に指定した場合の動作は保証されません。また，フィールドとフィールドに対応する setter メソッドに，同時に指定することはできません。

- javax.xml.ws.WebServiceContext 型のフィールド（static および final のフィールドを除く）
- javax.xml.ws.WebServiceContext 型を引数とする公開対象にならない setter メソッド※（static および final のメソッドを除く）

  注※

  非 public の setter メソッド，または exclude 要素が true の javax.jws.WebMethod アノテーションが指定されている setter メソッドです。

また，Web サービスコンテキストのインジェクションで javax.annotation.Resource アノテーションを使用する場合，javax.annotation.Resource アノテーションで要素は指定できません。指定した場合の動作は保証されません。

javax.annotation.Resource アノテーションの指定例を次に示します。

- フィールドへの指定例

```
import javax.annotation.Resource;
import javax.jws.WebService;
import javax.servlet.ServletContext;
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.WebServiceContext;

@WebService
public class AddNumbersImpl {
    // wsContextフィールドに現在処理中のリクエスト関連情報をインジェクトする
    @Resource
    private WebServiceContext wsContext;

    public int add(int number1, int number2) throws AddNumbersFault {
        // メッセージコンテキストを取得する
        MessageContext mContext = wsContext.getMessageContext();
        // プロパティを取得する
        ServletContext sContext = (ServletContext)mContext.get(MessageContext.SERVLET_CON
TEXT);
        ...
    }
    ...
}
```

- setter メソッドへの指定例

```
import javax.annotation.Resource;
import javax.jws.WebService;
import javax.xml.ws.handler.MessageContext;
```

```
import javax.xml.ws.WebServiceContext;

@WebService
public class AddNumbersImpl {
    // setterメソッドに対応するwsContextフィールドに現在処理中のリクエスト関連情報をインジェクトする
    private WebServiceContext wsContext;

    @Resource
    private void setWebServiceContext(WebServiceContext wsContext) {
        this.wsContext = wsContext;
    }
    public int add(int number1, int number2) throws AddNumbersFault {
        // メッセージコンテキストを取得する
        MessageContext mContext = wsContext.getMessageContext();
        // プロパティを設定する
        mContext.put("userPropKey", "userPropValue");
        ...
    }
    ...
}
```

(2) ユーザ定義メッセージコンテキストプロパティ追加時の注意事項

ユーザ定義メッセージコンテキストプロパティを追加するときの注意事項について説明します。

(a) インバウンド時のサービス側ハンドラでプロパティを追加する場合

インバウンド時のサービス側ハンドラで追加したユーザ定義メッセージコンテキストプロパティを，Web サービス実装クラスまたはプロバイダ実装クラスから参照する場合，サービス側ハンドラではユーザ定義メッセージコンテキストプロパティを設定したあとに，javax.xml.ws.handler.MessageContext インタフェースの API である setScope（java.lang.String name, MessageContext.Scope scope）メソッドを使用して，ユーザ定義プロパティを APPLICATION スコープとして設定する必要があります。

なお，インバウンド時のサービス側ハンドラで追加したユーザ定義メッセージコンテキストプロパティを，アウトバウンド時のサービス側ハンドラから参照する場合，インバウンド時のサービス側ハンドラでは，ユーザ定義メッセージコンテキストプロパティの設定だけを実施し，setScope（java.lang.String name, MessageContext.Scope scope）メソッドを使用してスコープの設定をする必要はありません。setScope（java.lang.String name, MessageContext.Scope scope）メソッドについては，「19.2.4(8) javax.xml.ws.handler.MessageContext インタフェース」を参照してください。

インバウンド時のサービス側ハンドラで，ユーザ定義メッセージコンテキストプロパティを追加する場合の例を次に示します。

```
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;

public class ServiceSOAPHandlerImpl implements SOAPHandler<SOAPMessageContext> {

    public boolean handleMessage(SOAPMessageContext smContext) {
```

```
        // メッセージ方向の取得
        boolean outbound = (boolean)smContext.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);
        if(outbound) {
           // アウトバウンド処理
           ...
        } else {
           // インバウンド処理

           // プロパティをkey値(”userPropKey”), value値(”userPropValue”)で
           // 設定する
           smContext.put(”userPropKey”, ”userPropValue”);
           // プロパティのスコープをAPPLICATIONスコープで設定する
           smContext.setScope(”userPropKey”, MessageContext.Scope.APPLICATION);
           ...
        }
        ...
    }
    ...
}
```

(b) Web サービス実装クラスまたはプロバイダ実装クラスでプロパティを追加する場合

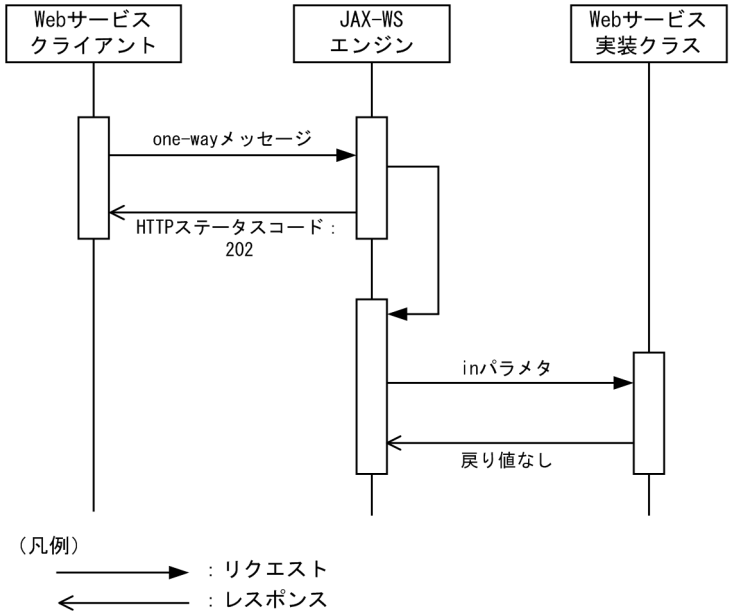
Web サービス実装クラスまたはプロバイダ実装クラスで追加したユーザ定義メッセージコンテキストプロパティを，アウトバウンド時のサービス側ハンドラから参照する場合，Web サービス実装クラスまたはプロバイダ実装クラスでは，ユーザ定義メッセージコンテキストプロパティの設定だけを実施してください。setScope（java.lang.String name, MessageContext.Scope scope）メソッドを使用してスコープの設定はしないでください。setScope(java.lang.String name, MessageContext.Scope scope)メソッドを使用した場合の動作は保証されません。

10.22 one-way オペレーション

one-way オペレーションは一方向の通信なので応答の SOAP メッセージは存在しません。しかし，one-way メッセージの転送プロトコルには HTTP を使用するため，Web サービスクライアントは HTTP ボディに SOAP メッセージを含まない HTTP レスポンスが返るのを待ちます。一方，サービス側 JAX-WS エンジンは，one-way オペレーションの Web サービス実装クラスの場合，Web サービス実装クラスのメソッドを呼び出す前に，Web サービスクライアントに HTTP ステータスコード"202 Accepted"の HTTP レスポンスを返します。

one-way オペレーションのシーケンスを次の図に示します。

図 10–9 one-way オペレーションのシーケンス



10.22.1 one-way オペレーションの注意事項

- プロバイダ実装クラスの場合，null を返すと，invoke メソッドの呼び出し終了後に HTTP ステータスコード"202 Accepted"の HTTP レスポンスを返します。
- WSS（Web Services - Security），WS-RM 1.2 機能，およびアドレッシング機能を使用する場合，one-way オペレーションはサポートしていません。また，同時に使用した場合の動作は保証されません。
- one-way オペレーションは応答の SOAP メッセージが存在しないため，Web サービス実装クラスの one-way オペレーションのメソッドの実装では，例外を明示的にスローしないようにしてください。
- サービス側 JAX-WS エンジンが，不正な SOAP メッセージを受信してアンマーシャルに失敗した場合，HTTP ステータスコード"500 Internal Server Error"を設定した SOAP フォルトを Web サービスクライアントに送信します。
- one-way オペレーションを使用する Web サービスクライアントは，HTTP ステータスコード"200 OK"および"202 Accepted"以外を受信すると javax.xml.ws.WebServiceException をスローします。

10.23　ラッパ bean の動的生成機能

ラッパ bean の動的生成機能とは，ラッパ bean（リクエスト bean とレスポンス bean），およびフォルト bean の JavaBeans クラスを JAX-WS エンジンが動的に生成する機能です。SEI を起点として開発した Web サービスで使用できます。Web サービスクライアントや，WSDL を起点とする Web サービスでは，ラッパ bean の動的生成機能はサポートしていません。

ここでは，ラッパ bean の動的生成機能のエラーチェックおよび性能について説明します。

10.23.1　hwsgen コマンドによるエラーチェックについて

SEI を起点とした Web サービスを開発する場合，javac コマンドで Web サービス実装クラスをコンパイルします。ただし，javac コマンドのコンパイルでは，次に示すマッピングのエラーチェックはされません。

- デフォルトマッピング
  詳細については，「1.4.2(1)(a)　デフォルトマッピング」を参照してください。
- マッピングのカスタマイズ（アノテーション）
  詳細については，「1.4.2(1)(b)　マッピングのカスタマイズ」を参照してください。

このため，Web サービス実装クラスに JAX-WS 機能でサポートしていない定義内容が含まれると，Web サービスの開始に失敗したり，Web サービスの開始に成功してもメタデータの取得や SOAP 通信に失敗したりする場合があります。これらの失敗を回避するには，javac コマンドでコンパイルした Web サービス実装クラスに対して hwsgen コマンドを実行し，事前にエラーチェックをする必要があります。なお，Web サービスの開始に失敗すると，ログおよび標準エラー出力に，次のエラーメッセージが出力されます。

- KDJW00003-E
- KDJW00005-E
- KDJW00006-E
- KDJW00007-E
- KDJW00008-E
- KDJW20026-E
- KDJW20027-E
- KDJW20041-E
- KDJW20042-E
- KDJW40011-E
- KDJW40013-E

10.23.2　性能について

Web サービスの起動性能は，JAX-WS エンジンが JavaBeans クラスをオンメモリ中に動的生成するオーバーヘッドが掛かるため，JavaBeans クラスを静的に生成した場合に比べると劣化します。しかし，WAR ファイルから読み込む必要がないため，ファイル読み込みのオーバーヘッドは軽減します。

また，JAX-WS エンジンが動的に生成する JavaBeans クラスは，hwsgen コマンドが静的に生成する JavaBeans クラスと同一のため，Web サービス開始後の SOAP 通信では，JavaBeans クラスを動的または静的のどちらで生成しても，性能は変わりません。

10.24　MIME Multipart/Related 構造の SOAP メッセージの受信

JAX-WS エンジンは，1048576 バイト以上のサイズの MIME ボディを含む MIME Multipart/Related 構造の SOAP メッセージを受信する際に，SOAP メッセージに含まれる MIME ボディを一時ファイルに出力することで，Java ヒープサイズの制約を受けることなく大きいサイズの SOAP メッセージを受信します。

一時ファイルは，ルートパートと添付ファイルパートの MIME ボディごとに出力します。なお，SOAP メッセージに添付ファイルが含まれない場合(ルートパートだけの場合)には，ルートパートの MIME ボディだけを一時ファイルに出力します。

MIME Multipart/Related 構造の SOAP メッセージに含まれる添付ファイルは，wsi:swaRef 形式の添付ファイル，MTOM/XOP 仕様形式の添付ファイルまたは SAAJ1.3 仕様の API を利用して扱えます。

10.24.1　一時ファイル

(1)　命名規則

JAX-WS エンジンが出力する一時ファイルのファイル名は，接頭辞に"MIME"，接尾辞に".tmp"が付いた名前となります。この名前は JAX-WS エンジンが自動で付与するため，任意のファイル名には変更できません。

JAX-WS エンジンが出力する一時ファイルのファイル名の例を次に示します。

```
MIME6838906861691549713.tmp
```

異なるプロセスの cjclstartap コマンドで Web サービスクライアントを実行する場合，一時ファイルのファイル名が重複することがあります。そのため，1048576 バイト以上のサイズの MIME ボディを含む MIME Multipart/Related 構造の SOAP メッセージを受信する場合は，cjclstartap コマンドのプロセスごとに一時ファイルの出力先を変える必要があります。システムプロパティの"java.io.tmpdir"キーで一時ファイルの出力先を指定してください。

(2)　出力と削除

一時ファイルは，1048576 バイト以上のサイズの MIME ボディであるルートパートを受信したとき，および 1048576 バイト以上のサイズの MIME ボディである添付ファイルパートを受信し操作したときに，Java のデフォルト一時ファイルディレクトリ(システムプロパティにある"java.io.tmpdir"キーに対応する値)へ出力されます。

ルートパートの MIME ボディに対応する一時ファイルは，SOAP メッセージをアンマーシャルしたあと JAX-WS エンジンによって削除されます。また，添付ファイルパートの MIME ボディに対応する一時ファイルは，添付ファイルをクローズするときに削除されます。javax.activation.DataHandler 型にマッピングされる添付ファイルを使用する場合は，javax.activation.DataHandler オブジェクトに対して

instanceof 演算子を用いて com.sun.xml.ws.developer.StreamingDataHandler クラスであるか判別したあと，com.sun.xml.ws.developer.StreamingDataHandler クラスにキャストしてから com.sun.xml.ws.developer.StreamingDataHandler#close()メソッドでクローズする必要があります。

MIME Multipart/Related 構造の SOAP メッセージを受信している場合，コネクションの切断や JVM がダウンしたとき，一時ファイルが残ることがあります。一時ファイルが残った場合，一時ファイルを削除してください。

(3) 見積もり方法

JAX-WS エンジンが出力する一時ファイルは，受信する MIME Multipart/Related 構造の SOAP メッセージに含まれる MIME ボディと同じサイズです。一時ファイルの最大ディスク使用量は次に示す算出式で求めることができます。

- 一時ファイルの最大ディスク使用量（Web サービス）

  一時ファイルの最大ディスク使用量＝<受信するMIMEボディの最大サイズ>×<受信するMIMEボディの最大数>×<Webサービスの同時実行数>

- 一時ファイルの最大ディスク使用量（Web サービスクライアント）

  一時ファイルの最大ディスク使用量＝<受信するMIMEボディの最大サイズ>×<受信するMIMEボディの最大数>×<Webサービスの呼び出し回数>

11 RESTful Web サービス開発のポイント

この章では，RESTful Web サービス（Web リソース）開発の作業ごとに，あらかじめ理解しておく必要がある点と注意事項について説明します。

11.1　ルートリソースクラスの作成

ルートリソースクラスは，コンパイルした Java クラスファイル（*.class）として作成します。必要に応じてサブリソースクラスや例外マッピングプロバイダも同様に作成します。

コンパイルした Java クラスファイル（*.class）は，WAR ファイルを構成するディレクトリに含めます。次に示すどちらか，または両方に含めてください。

- classes ディレクトリ以下
- lib ディレクトリ以下に含まれる JAR ファイル内

WAR ファイルには，少なくとも一つ以上のルートリソースクラスが含まれる必要があります。格納先については，「11.3.1　WAR ファイルの構成」を参照してください。

注意事項

通常，Web リソースはステートレスです。ルートリソースクラスのフィールドを，複数回の Web リソースの呼び出しで共有しないでください。

11.2　web.xml の作成

ここでは，Web リソースで使用する WAR ファイルに含まれる web.xml について説明します。

ファイル名称は"web.xml"とし，WAR ファイルを構成する WEB-INF ディレクトリの直下に格納します。web.xml の格納は必須です。

Web サービスの実行に必要な定義および web.xml の例について説明します。

11.2.1　Web サービスの実行に必要な定義

web.xml は，次に示す条件を満たすように作成してください。

- バージョン

  web.xml のバージョンは，2.5 以上にしてください。

- サーブレット

  web-app 要素に，次に示す servlet 要素を含めてください。

```
<servlet>
 <servlet-name>CosminexusJaxrsServlet</servlet-name>
  <servlet-class>
   com.cosminexus.jersey.spi.container.servlet.ServletContainer
  </servlet-class>
</servlet>
```

- サーブレット初期化パラメタ

  servlet 要素に，次に示す init-param 要素を必要に応じて含めてください。init-param 要素は大文字小文字を区別します。

```
<init-param>
 <param-name>
  com.sun.jersey.config.property.packages
 </param-name>
 <param-value>
  セミコロン，コンマ，またはスペースで区切られたパッケージ名のリスト
 </param-value>
</init-param>
```

  servlet 要素に init-param 要素を含める場合，init-param 要素で指定したパッケージとそのサブパッケージに含まれる Web リソースを公開します。servlet 要素に init-param 要素を含めない場合，WAR に含まれるすべての Web リソースを公開します。

注意事項

- com.sun.jersey.config.property.packages の<param-value>に，*（アスタリスク）は使用できません。

- アスタリスクを使用した場合，または間違ったパッケージ名を指定した場合，エラーになります（KDJJ10020-E）。HTTP エラーコードには 500 が返ります。
- com.sun.jersey.config.property.packages の<param-value>に指定したパッケージとそのサブパッケージに含まれるクラスに，JDK8 から導入されたラムダ式は使用できません。

- サーブレットマッピング

  web-app 要素以下に servlet-mapping 要素を記述してください。

  servlet-mapping 要素の記述例を次に示します。

```
<servlet-mapping>
 <servlet-name>CosminexusJaxrsServlet</servlet-name>
 <url-pattern>/*</url-pattern>
</servlet-mapping>
```

11.2.2 web.xml の例

web.xml の例を次に示します。

```
<web-app version="3.0"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/we
b-app_3_0.xsd">
 <servlet>
  <servlet-name>CosminexusJaxrsServlet</servlet-name>
  <servlet-class>
   com.cosminexus.jersey.spi.container.servlet.ServletContainer
  </servlet-class>
  <init-param>
   <param-name>com.sun.jersey.config.property.packages</param-name>
   <param-value>org.test.resources1;org.test.resources2</param-value>
  </init-param>
 </servlet>
 <servlet-mapping>
  <servlet-name>CosminexusJaxrsServlet</servlet-name>
  <url-pattern>/*</url-pattern>
 </servlet-mapping>
</web-app>
```

バージョン 2.5 の web.xml を作成する場合は，web-app 要素の version 属性を"2.5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"にしてください。

この例で，WAR ファイルに org.test.resources1.ResourceA, org.test.resources1.ResourceB, org.test.resources2.ResourceC，および org.test.resources3.ResourceD を含んでいると仮定すると，公開される Web リソースは次のとおりです。

- ResourceA
- ResourceB
- ResourceC

11.3　アーカイブの作成

WAR ファイルの構成および EAR ファイルの作成について説明します。

11.3.1　WAR ファイルの構成

Web リソースで使用する WAR ファイルは，次の表に示す構成にする必要があります。

表 11–1　WAR ファイルの構成

ディレクトリ	備考
/	－
┣ META-INF/	－
┃ ┗ MANIFEST.MF	－
┗ WEB-INF/	－
┣ web.xml	作成した web.xml です。
┣ classes/	コンパイルした Java クラスを格納します。
┗ lib/	コンパイルした Java クラスを含む JAR ファイルを格納します。

（凡例）

－：説明や補足事項が特にないことを示します。

11.3.2　EAR ファイルの作成

Web リソースを J2EE サーバにデプロイするには，作成した WAR ファイルを含めた EAR ファイルを作成します。EAR ファイルを作成するには，application.xml が必要になります。

EAR ファイルの構成については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「1.4.2　アーカイブ形式の J2EE アプリケーション」を参照してください。

11.4 RESTful Web サービス用クライアント API を使用するクライアントの実装

ここでは，RESTful Web サービス用クライアント API を利用して実装するクライアント（以降，Web リソースクライアントと呼びます）について説明します。java.net.URL や java.net.HttpURLConnection など，標準的な Java API を利用してクライアントを実装する場合は，JDK のドキュメントを参照してください。

Web リソースクライアントの形態に制限はありません。そのため，例えば次に示す Web サービスを開発できます。

- Java アプリケーション（Java アプリケーションから Web リソースを呼び出す）
- JSP（JSP から Web リソースを呼び出す）
- サーブレット（サーブレットから Web リソースを呼び出す）
- EJB（EJB から，Web リソースを呼び出す）
- SOAP Web サービス（SOAP Web サービスから Web リソースを呼び出す）
- Web リソース（ルートリソースクラスやサブリソースクラスから，さらに別の Web リソースを呼び出す）

Web リソースクライアントは SOAP Web サービスを呼び出すクライアントとは異なり，Web リソースクライアントを実装する前に，コマンドを実行してスタブなどの Java ソースをあらかじめ作成する必要はありません。RESTful Web サービス用クライアント API の仕様に従って，実装してください。RESTful Web サービス用クライアント API の仕様については「25. RESTful Web サービス用クライアント API のサポート範囲」を参照してください。

11.4.1 Web リソースクライアントのユースケース

ここでは，RESTful Web サービス用クライアント API の基本的なユースケースについて説明します。Web リソースを呼び出すには，次の三つの方法があります。

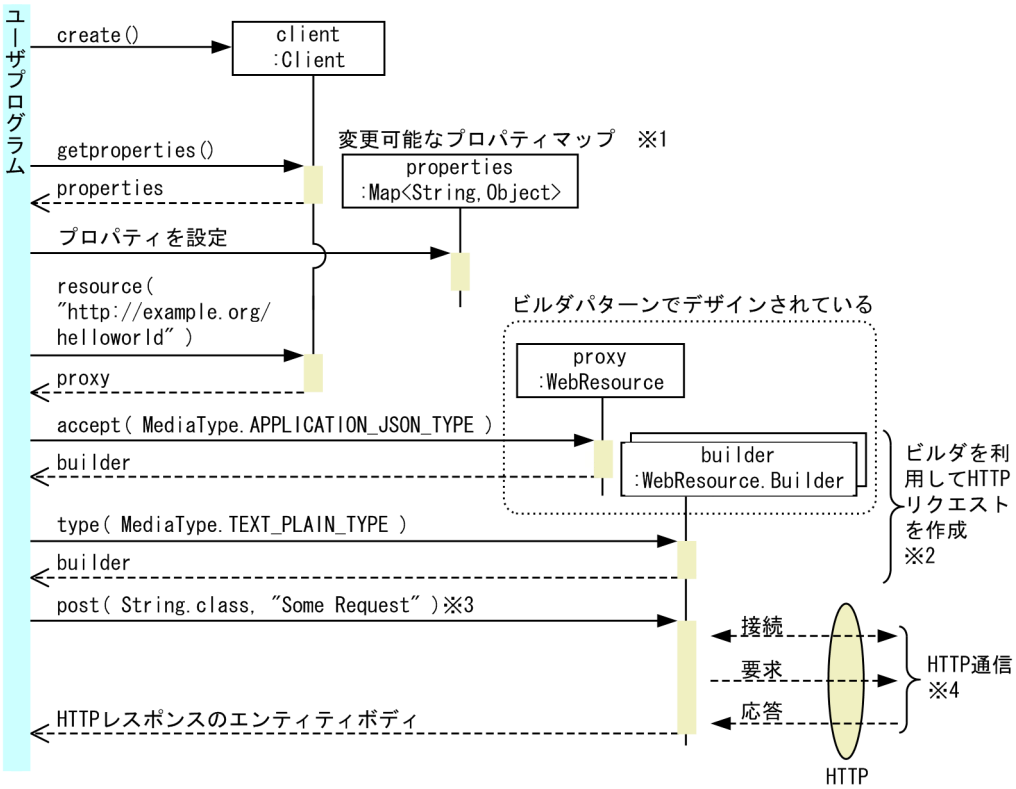
- Java 型を指定して HTTP リクエストおよび HTTP レスポンスを送受信する方法
- Java 型を指定して HTTP リクエストを送信し，汎用型（ClientResponse）で HTTP レスポンスを受信する方法
- 汎用型（ClientRequest および ClientResponse）で HTTP リクエストおよび HTTP レスポンスを送受信する方法

それぞれのパターンについて次に説明します。

(1) Java 型を指定して HTTP リクエストおよび HTTP レスポンスを送受信する

Java 型を指定して HTTP リクエストおよび HTTP レスポンスを送受信する場合のユースケースを次の図に示します。

図 11-1 Java 型を指定して HTTP リクエストおよび HTTP レスポンスを送受信する場合のユースケース



注※1

マップに含まれるプロパティはさまざまな方法で参照・変更できます。詳細は「11.4.3 プロパティとフィーチャの設定」を参照してください。

注※2

クライアント API のさまざまなメソッドを利用してリクエストを生成できます。詳細は「11.4.3 プロパティとフィーチャの設定」を参照してください。

注※3

HTTP リクエストのエンティティボディを文字列の形式で，HTTP POST メソッドを使用して送信し，同様に文字列の形式で HTTP レスポンスのエンティティボディを受信します。

クライアント API では DELETE，GET，HEAD，OPTIONS，PUT の各 HTTP メソッドに対応するメソッドもあります。詳細は「11.4.3 プロパティとフィーチャの設定」を参照してください。

注※4

HttpURLConnection が行います。JAX-RS エンジンは HTTP メッセージの作成まで行い，HTTP 通信については Java SE の HttpURLConnection に委譲しています。

図 11-1 に対応したコーディング例は次のとおりです。

```
Client client = Client.create();
Map<String, Object> properties = client.getProperties();
properties.put(ClientConfig.PROPERTY_READ_TIMEOUT, 10000);
WebResource proxy = client.resource( "http://example.org/helloworld" );
WebResource.Builder builder = proxy.accept( MediaType.APPLICATION_JSON_TYPE );
builder = builder.type( MediaType.TEXT_PLAIN_TYPE );
String response = builder.post( String.class, "Some Request" );
```

この例での処理の流れについて説明します。

1. Client クラスの create()スタティックメソッドを使用して Client オブジェクトを生成します。
2. Client クラスの getProperties()メソッドで取得した変更可能なプロパティマップにプロパティを設定します。なお，プロパティは別の方法でも設定できます。プロパティとフィーチャの詳細については「11.4.3　プロパティとフィーチャの設定」を参照してください。
3. Client クラスの resource()メソッドを呼び出して WebResource オブジェクトを生成し，WebResource オブジェクトのメソッドを呼び出して HTTP リクエストを作成していきます。WebResource クラスはビルダパターンでデザインされており，HTTP リクエストを生成する各種メソッドが含まれています。
4. WebResource.Builder クラスの post()メソッドを呼び出して HTTP 通信を行います。WebResource クラスには，DELETE，GET，HEAD，OPTIONS，PUT の各 HTTP メソッドに対応するメソッドもあります。

(2)　Java 型を指定して HTTP リクエストを送信し，汎用型（ClientResponse）で HTTP レスポンスを受信する

受信する HTTP レスポンスは汎用型，つまり ClientResponse オブジェクトでも取得できます。ClientResponse オブジェクトで取得した場合，ユーザは受信した HTTP レスポンスの各種情報（HTTP ヘッダ，エンティティボディ，およびステータスコード）を取得できます。

エンティティボディは，getEntity()メソッドを使用することで，Java 型を指定して取得できます。ClientResponse クラスでサポートしているメソッドについては「25.1　クライアント API のインタフェースおよびクラスのサポート範囲」を参照してください。

受信する HTTP レスポンスを ClientResponse オブジェクトで取得する方法は次のとおりです。

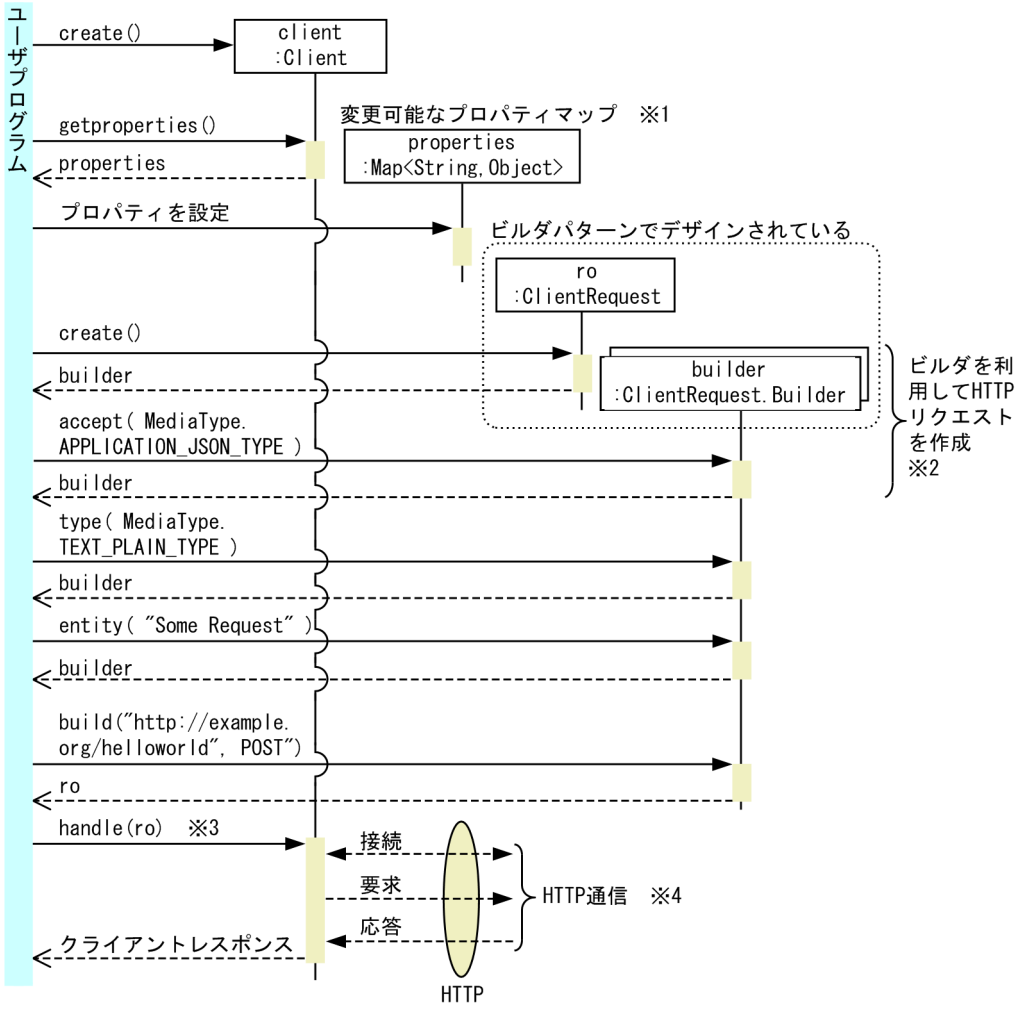
WebResource オブジェクトを使用して HTTP リクエストを生成する場合は，HTTP レスポンスの Java 型を指定するパラメタに ClientResponse.class を指定してください。「11.4.1(1)　Java 型を指定して HTTP リクエストおよび HTTP レスポンスを送受信する」で示したコーディング例では，String.class の代わりに ClientResponse.class を指定してください。

Client クラスを使用して HTTP リクエストを直接送信する場合は，HTTP レスポンスは常に ClientResponse です。Client クラスの詳細については，「11.4.1(3)　汎用型で HTTP リクエストおよび HTTP レスポンスを送受信する」を参照してください。

(3)　汎用型で HTTP リクエストおよび HTTP レスポンスを送受信する

汎用型（ClientRequest および ClientResponse）で HTTP リクエストおよび HTTP レスポンスを送受信する場合のユースケースを次の図に示します。

図 11-2　汎用型で HTTP リクエストおよび HTTP レスポンスを送受信する場合のユースケース



注※1

マップに含まれるプロパティはさまざまな方法で参照・変更できます。詳細は「11.4.3　プロパティとフィーチャの設定」を参照してください。

注※2

クライアント API のさまざまなメソッドを利用してリクエストを生成できます。詳細は「11.4.3　プロパティとフィーチャの設定」を参照してください。

注※3

HTTP リクエストのエンティティボディを文字列の形式で，HTTP POST メソッドを使用して送信し，同様に文字列の形式で HTTP レスポンスのエンティティボディを受信します。

クライアント API では DELETE，GET，HEAD，OPTIONS，PUT の各 HTTP メソッドに対応するメソッドもあります。詳細は「11.4.3　プロパティとフィーチャの設定」を参照してください。

注※4

HttpURLConnection が行います。JAX-RS エンジンは HTTP メッセージの作成まで行い，HTTP 通信については Java SE の HttpURLConnection に委譲しています。

上記の図に対応したコーディング例は次のとおりです。

```
Client client = Client.create();
Map<String, Object> properties = client.getProperties();
properties.put(ClientConfig.PROPERTY_READ_TIMEOUT, 10000);
ClientRequest ro;
ClientRequest.Builder builder = ClientRequest.create();
builder.accept( MediaType.APPLICATION_JSON_TYPE );
builder.type( MediaType.TEXT_PLAIN_TYPE );
builder.entity("Some Request");
ro = builder.build(new URI("http://example.org/helloworld"), "POST");
ClientResponse clientResponse = client.handle(ro);
//The actual response in the form of String
String response = clientResponse.getEntity(String.class);
```

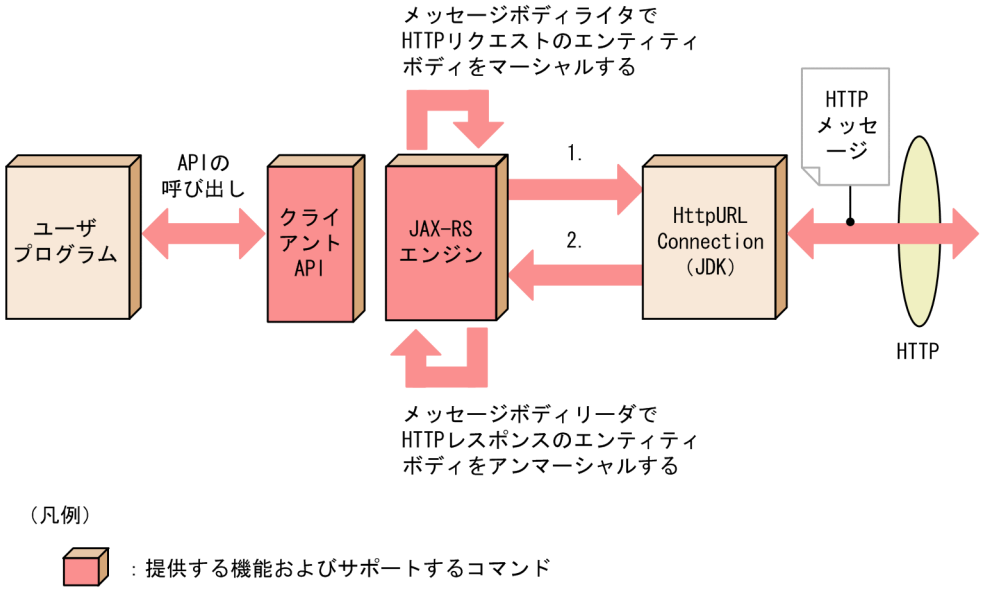
この例での処理の流れについて説明します。

1. Client クラスの create()スタティックメソッドを使用して Client オブジェクトを生成します。
2. Client クラスの getProperties()メソッドで取得した変更可能なプロパティマップにプロパティを設定します。プロパティとフィーチャの詳細については，「11.4.3　プロパティとフィーチャの設定」を参照してください。
3. ClientRequest クラスの create()メソッドを呼び出して ClientRequest.Builder オブジェクトを生成し，ClientRequest.Builder オブジェクトのメソッドを呼び出して HTTP リクエストを作成していきます。ClientRequest.Builder クラスはビルダパターンでデザインされており，HTTP リクエストを生成する各種メソッドが含まれています。
4. ClientRequest.Builder クラスの build()メソッドを呼び出し ClientRequest オブジェクトを生成します。次に Client クラスの handle()メソッドを呼び出して HTTP 通信を行います。

11.4.2　RESTful Web サービス用クライアント API の仕組み

RESTful Web サービス用クライアント API は，HttpURLConnection/HttpsURLConnection クラスをラップしており，実際の HTTP 通信については JDK が行っています。なお，以降では特に明記のないかぎり，HttpURLConnection/HttpsURLConnection をまとめて HttpURLConnection と呼びます。

仕組みについて次の図に示します。



図中の 1.および 2.では次の処理を行っています。

1.

- 接続タイムアウトなどのプロパティを設定します。
- HTTP ヘッダを追加します。
- HTTP リクエストのエンティティボディを，HttpURLConnection から取得した出力ストリームを介して送信します。

2.

- HTTP ヘッダを取得します。
- HTTP レスポンスのエンティティボディを，HttpURLConnection から取得した入力ストリームを介して受信します。

(1)　RESTful Web サービス用クライアント API とユーザプログラムの関係

RESTful Web サービス用クライアント API は，エンティティボディリーダやエンティティボディライタを使用して，Java オブジェクトや HTTP リクエストを相互にマーシャル・アンマーシャルしています。このため，ユーザプログラムが入力ストリームや出力ストリームを直接操作する必要はありません。

(2)　RESTful Web サービス用クライアント API と JDK の関係

RESTful Web サービス用クライアント API は，トランスポート層の処理を HttpURLConnection に委譲しています。また，プロパティや HTTP ヘッダを設定するためのさまざまなビルトインメソッドを提供していますが，ビルトインメソッドに指定した値はそのまま HttpURLConnection に設定します。そのため，実際の HTTP 通信やビルトインメソッドに指定した値を使用した処理は JDK が行います。

11.4.3　プロパティとフィーチャの設定

Client オブジェクトにはプロパティとフィーチャを格納した変更可能なプロパティマップが含まれています。

これ以降，プロパティとフィーチャは区別しないで「プロパティ」と総称します。

(1)　プロパティマップの初期化

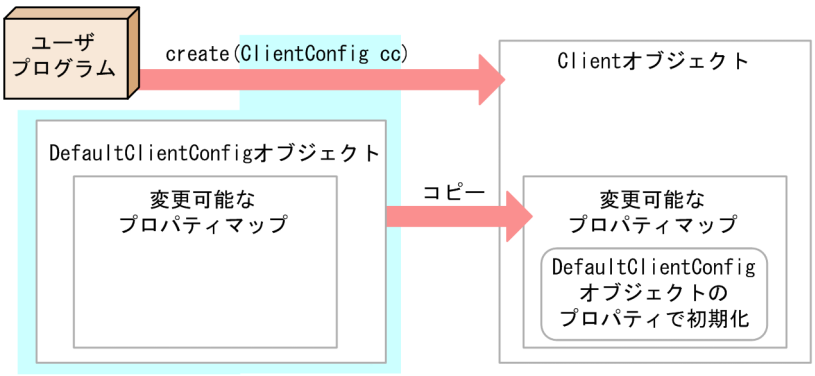
変更可能なプロパティマップは，Client オブジェクトの生成時に初期化されます。プロパティマップを初期化する方法について次に説明します。

方法 1

DefaultClientConfig オブジェクトをパラメタに指定して，Client クラスの create(ClientConfig cc) スタティックメソッドを呼び出し，Client オブジェクトを生成します。この場合，Client オブジェクトの変更可能なプロパティマップは，DefaultClientConfig オブジェクトが持つプロパティによって初期化※されます。

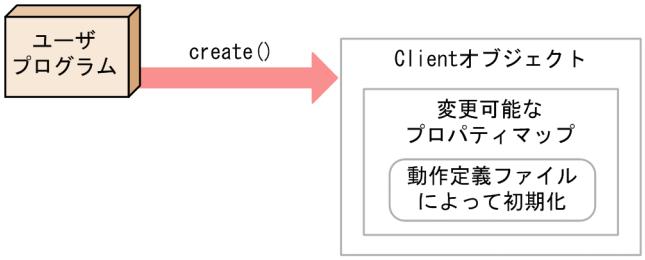
注※

DefaultClientConfig オブジェクトにプロパティが含まれれば，その値で初期化されます。
DefaultClientConfig オブジェクトに含まれないプロパティは，動作定義ファイルの設定で初期化されます。



方法 2

Client クラスの create()スタティックメソッドで Client オブジェクトを生成します。この場合，Client オブジェクトの変更可能なプロパティマップは，動作定義ファイルによって初期化されます。



動作定義ファイルの詳細については，「13.1　動作定義ファイル」を参照してください。

(2)　変更可能なプロパティマップの構造

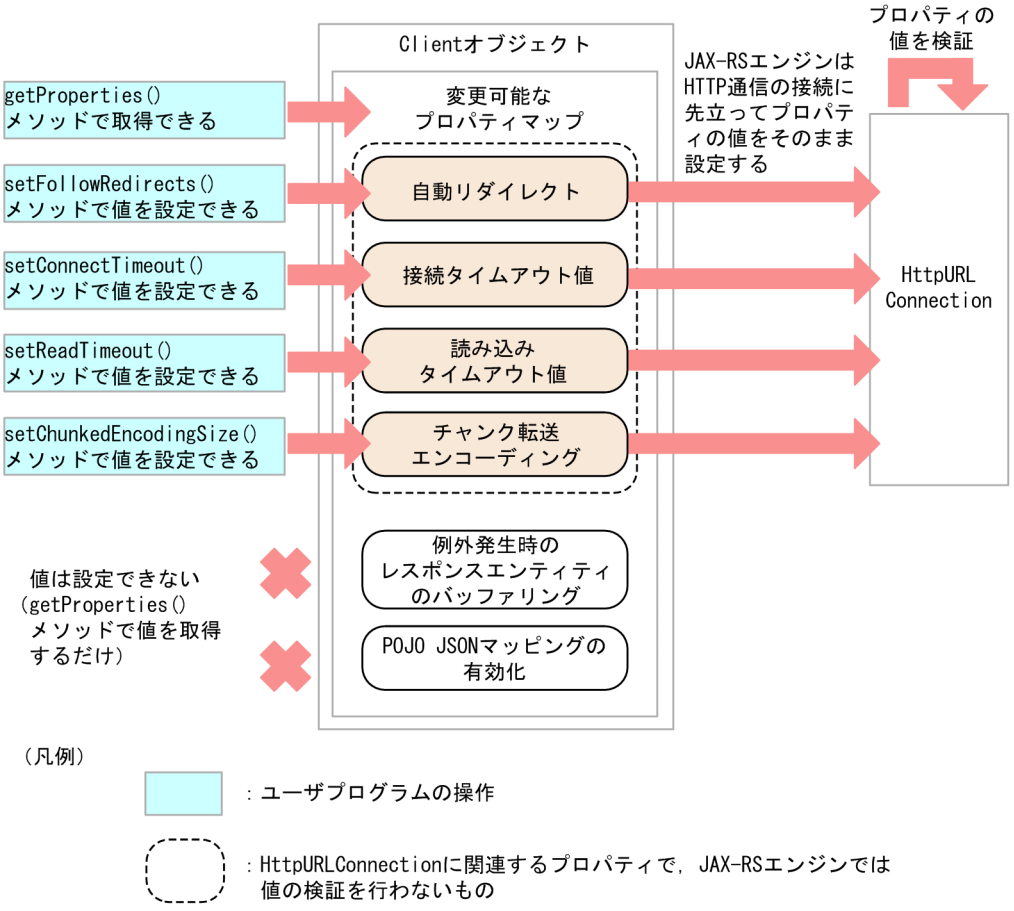
HttpURLConnection クラスに関連する四つのプロパティは，Client クラスの getProperties()メソッドで取得したプロパティマップを変更したり，プロパティそれぞれに対応する setter メソッドを使用して設定したりできます。

なお，HttpURLConnection に関連する四つのプロパティ以外は，対応する setter メソッドがありません。プロパティを設定する場合は Client クラスの getProperties()メソッドで取得したプロパティマップを変更してください。

変更可能なプロパティマップの構造について次の図に示します。

なお，サポートしているプロパティの一覧については「25.1.1　サポートするプロパティとフィーチャ」を参照してください。

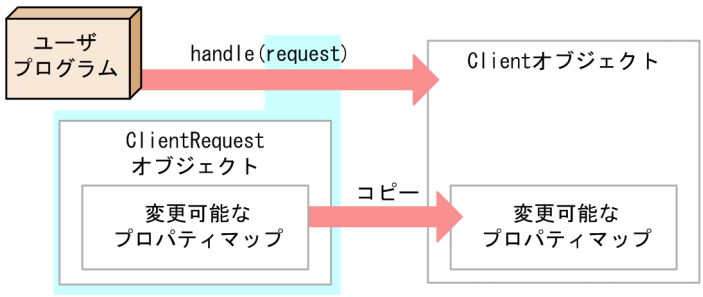
図 11-3　変更可能なプロパティマップの構造



HTTP 通信に先立って，JAX-RS エンジンは図中の HttpURLConnection クラスに関連する四つのプロパティの値をそのまま HttpURLConnectioin にコピーします（値の検証はしません）。したがって，値が不正な場合，HTTP 通信の実行前または実行中に HttpURLConnection クラスが例外をスローすることがあります。なお，値が null の場合はコピーされないで，無視されます。

ClientRequest クラスにも同じ形式の変更可能なプロパティマップがあります。したがって，Client クラスの handle()メソッドを使用して HTTP 通信を行う場合，ユーザプログラムは ClientRequest オブジェクトを介してプロパティを設定することもできます。仕組みを次の図で説明します。

図 11-4　変更可能なプロパティマップのコピーの仕組み



JAX-RS エンジンは，同じプロパティが Client オブジェクトのプロパティマップに存在しない場合だけ，ClientRequest オブジェクトのプロパティマップに含まれるプロパティを Client オブジェクトのプロパティマップにコピーします。

11.4.4　HTTP ヘッダの設定

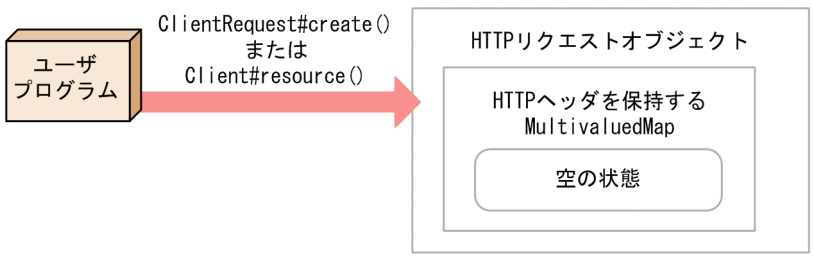
クライアント API には，HTTP リクエストオブジェクトに HTTP ヘッダを設定するためのさまざまなメソッドを提供しています。サポートしているメソッドの詳細については「25.1　クライアント API のインタフェースおよびクラスのサポート範囲」を参照してください。なお，この説明での HTTP リクエストオブジェクトとは，次のどちらかを指します。

- WebResource オブジェクトまたは ClientRequest オブジェクト
- WebResource.Builder オブジェクトまたは ClientRequest.Builder オブジェクト

HTTP リクエストオブジェクトには，HTTP ヘッダと対応する値を保持する MultivaluedMap オブジェクトが含まれています。

HTTP ヘッダを保持する MultivaluedMap オブジェクトがどのように初期化されるのかを次の図で説明します。

図 11-5　MultivaluedMap オブジェクトの初期化の仕組み

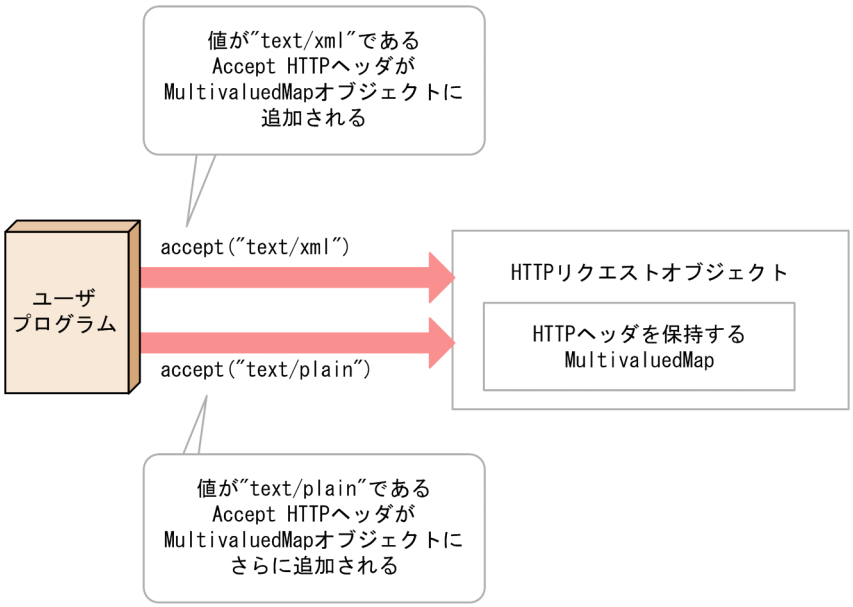


この図では，ユーザプログラムは ClientRequest クラスの create()スタティックメソッドまたは Client クラスの resource()メソッドを使用して HTTP リクエストオブジェクトを生成しています。

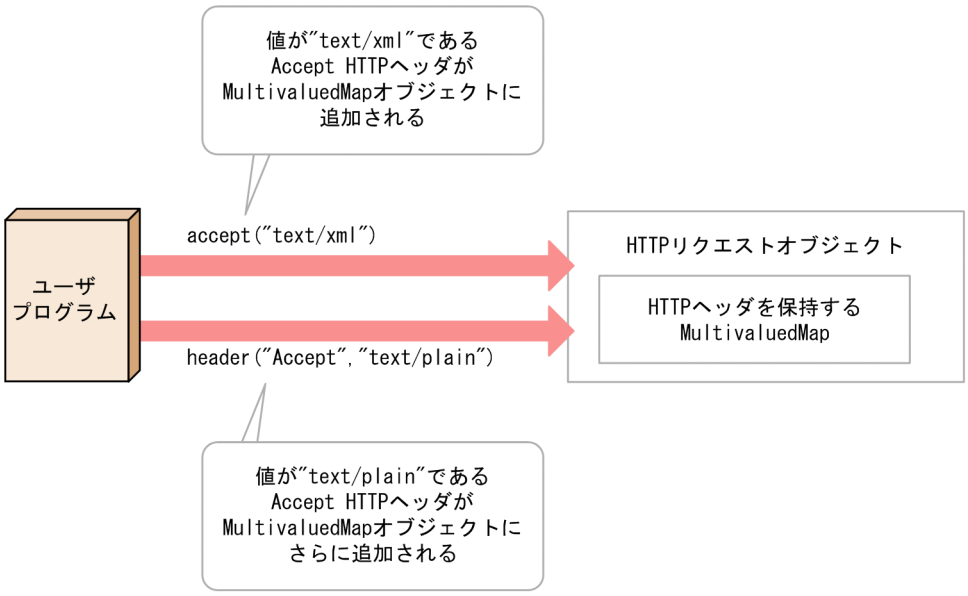
この場合，MultivaluedMap オブジェクトは空の状態で初期化されます。

MultivaluedMap オブジェクトに HTTP ヘッダを設定する例を次の図に示します。

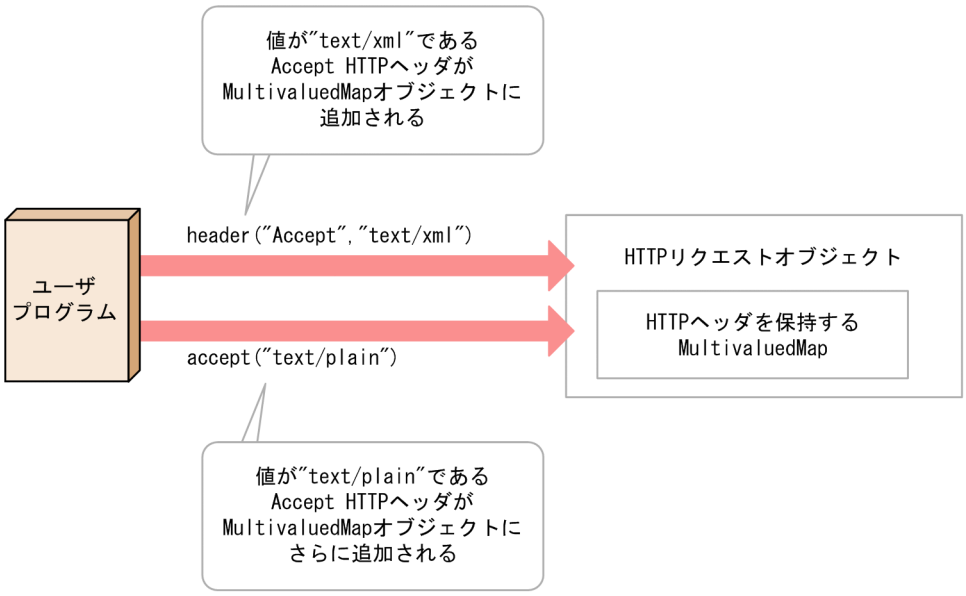
例 1



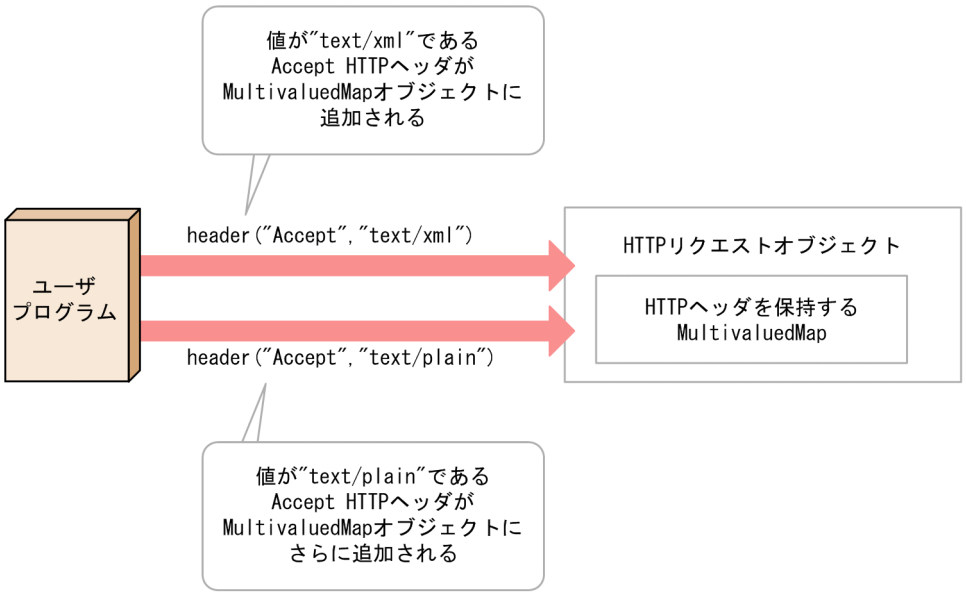
例 2



例 3



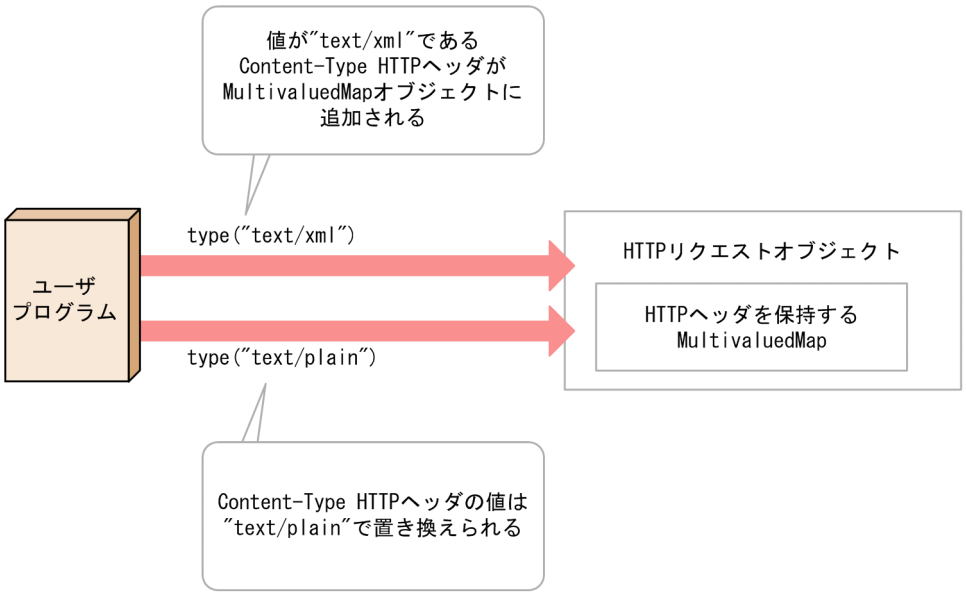
例 4



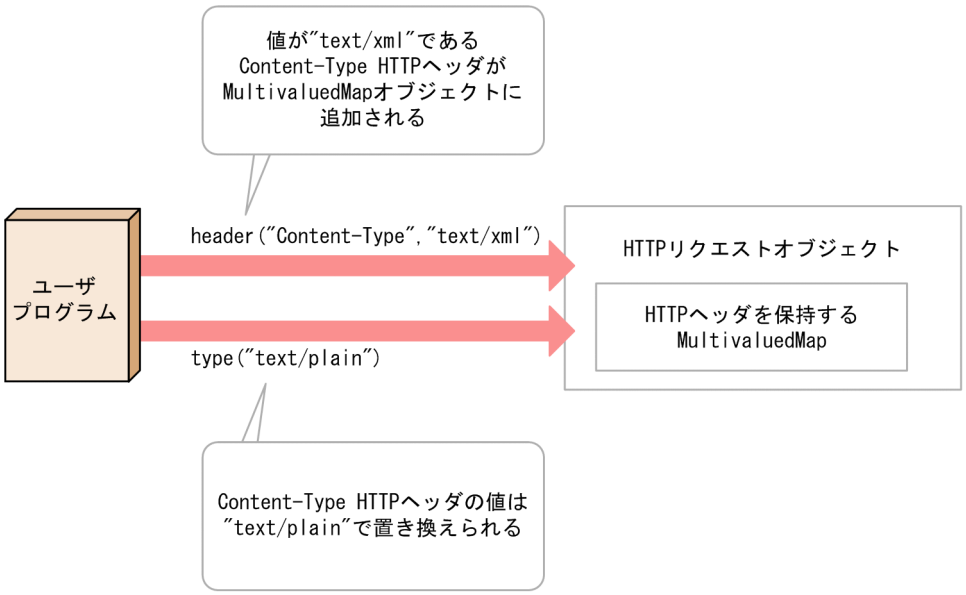
例 1～例 4 のどの方法でも，ユーザプログラムは，"text/xml"と"text/plain"を値とする Accept HTTP ヘッダを持つ HTTP リクエストオブジェクトを生成できます。Accept-Language HTTP ヘッダと Cookie HTTP ヘッダについても同様です。なお，Accept-Language HTTP ヘッダと Cookie HTTP ヘッダの場合は accept メソッドをそれぞれ acceptLanguage()メソッドまたは cookie()メソッドに読み替えてください。

Content-Type HTTP ヘッダの場合，例 1 と例 3 の動作は次に示すとおりになります。

例 1

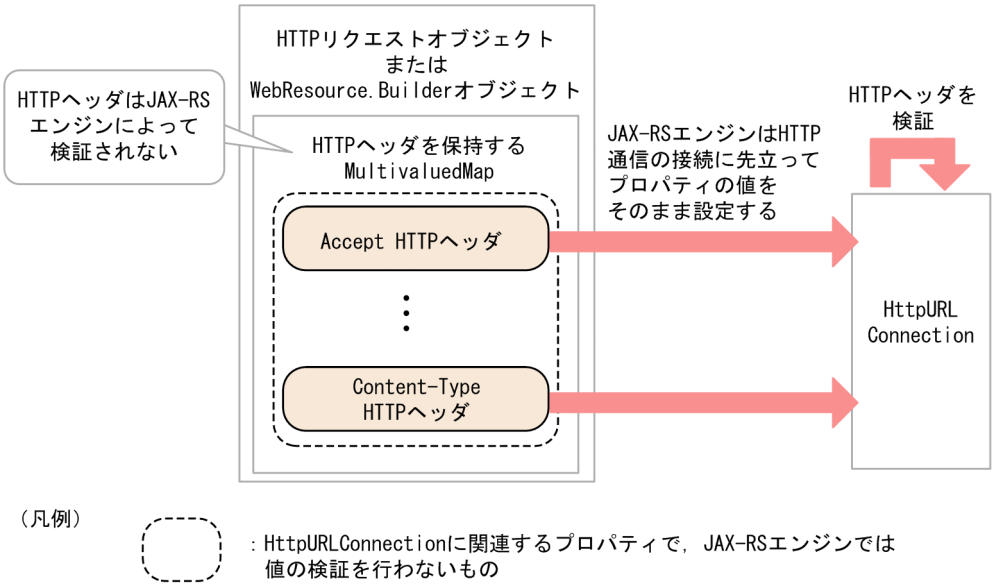


例 3



どちらの場合でも，HTTP リクエストオブジェクトに設定される Content-Type HTTP ヘッダの値は"text/plain"です。なお，Content-Type HTTP ヘッダは値を一つしか持てないため，例 2 と例 4 の方法は利用できません。

HTTP 通信に先立って，HTTP リクエストオブジェクトに追加されたすべての HTTP ヘッダとその値は，HttpURLConnection オブジェクトに設定されます。仕組みを次の図で説明します。



それぞれの HTTP ヘッダには，HTTP リクエストオブジェクトに含まれる対応する HTTP ヘッダの値が設定されます。なお，この場合，HTTP リクエストオブジェクトに含まれる対応する HTTP ヘッダの値が null でなければ，toString()メソッドの戻り値に設定されます。JAX-RS エンジンは HTTP ヘッダの値を検証しません。標準仕様に従って HTTP ヘッダの値を HTTP リクエストオブジェクトに設定してください。

なお，HTTP 通信を行うまでに，特定の HTTP ヘッダを HTTP リクエストオブジェクトに設定しなかった場合の動作は，HTTP ヘッダを HttpURLConnection オブジェクトに設定しなかった場合と同様です。

11.4.5　注意事項

Web リソースクライアント実装時の注意事項について説明します。

(1)　オブジェクトの再利用

Client オブジェクトの生成には処理コストが掛かるので，一度生成した Client オブジェクトは再利用することをお勧めします。WebResource オブジェクトを複数回生成したり，Client クラスのメソッドを利用して Web リソースを複数回呼び出したりする場合に，Client オブジェクトを複数回生成する必要はありません。

同様に，WebResource の生成にも処理コストが掛かるので，一度生成した WebResource オブジェクトは再利用することをお勧めします。同じ Web リソース（URL）に対する HTTP リクエストやビルダを複数回生成する場合に，WebResource オブジェクトを複数回生成する必要はありません。

ただし，Client クラスの設定を行うメソッドや，オブジェクトを破棄する Client クラスのメソッドはスレッドセーフではありません。次の内容に注意してください。

- 複数スレッドで Client オブジェクトを共有する場合，共有する Client オブジェクトに対する設定は，複数スレッドが動作する前に実行してください。

- Client オブジェクトの破棄は，複数スレッドの動作が完了したあとに実行してください。

複数スレッドの動作中にこれらの操作を行った場合，通信が失敗したり，不正な HTTP リクエストが送信されたりすることがあります。

Web リソースクライアントをサーブレットや EJB などで実装する場合は，サーブレットや EJB などの初期化メソッドで Client オブジェクトを生成し，必要な設定を行ってください。Client オブジェクトの破棄についても同様に，破棄メソッドで Client オブジェクトの破棄を行ってください。

サーブレットで RESTful Web サービス用クライアント API を利用する例を次に示します。

```
@WebServlet("/example")
public class ClientServlet extends HttpServlet {

  // 共有されるClientオブジェクト
  private Client client = null;

  // 共有されるWebResourceオブジェクト
  private WebResource proxy = null;

  @PostConstruct
  public void postConstruct() {
    // RESTful Webサービス用クライアントAPIを利用するために
    // Clientオブジェクトを生成する
    this.client = Client.create();
    // クライアントの設定: Clientオブジェクトからプロパティバッグを取得する
    Map<String, Object> properties = this.client.getProperties();
    // クライアントの設定: 読み込みタイムアウトを設定する
    properties.put(ClientConfig.PROPERTY_READ_TIMEOUT, 10000);
    //  ClientオブジェクトからWebResourceオブジェクトを生成する
    this.proxy = this.client.resource( "http://..." );
  }

 @PreDestroy
  public void preDestory() {
    //  Clientオブジェクトを破棄する
    if( this.client != null ){
      this.client.destroy();
      this.client = null;
    }
  }

  @Override
  public void doGet( HttpServletRequest request, HttpServletResponse response )
    throws ServletException, IOException{
    ...
    ClientResponse clientResponse = null;
    // Invoke resource
    try {
      // クッキーを設定後，WebリソースからHTTPレスポンスを取得する
      Cookie cookie = new Cookie("cookie", "cookie%20value");
      clientResponse = this.proxy.cookie(cookie).get(ClientResponse.class);
    }catch (Exception e) {
      printStackTrace( e, out );
    }
```

```
    ...
  }
}
```

RESTful Web サービス用クライアント API のスレッドセーフ性の詳細については，「25.16　RESTful Web サービス用クライアント API のスレッドセーフ性」を参照してください。

(2)　プロキシ・SSL 接続・ベーシック認証の設定

プロキシ，SSL 接続，およびベーシック認証の設定については，それぞれ次の個所を参照してください。

- プロキシの設定
  「10.10　プロキシサーバ経由の接続」
- SSL 接続の設定
  「10.11　SSL プロトコルによる接続」
- ベーシック認証の設定
  「10.12　ベーシック認証による接続」

(3)　Windows 環境での注意事項

Web リソースクライアントから大量のリクエストを送信するような環境では，次の例外が記録されることがあります。

```
java.net.BindException: Address already in use: connect  [errno=10048, syscall=select]
```

この例外は，例えばサーブレットとして実装した Web リソースクライアントに対して大量のリクエストが到着したときなどに発生します。

このような場合は，次に示すどちらか，または両方の対策を実施してください。

- OS で使用できるポート番号の範囲を広げる
  例）レジストリの MaxUserPort の設定を見直す
- TIME_WAIT の継続時間を短くする
  例）レジストリの TcpTimedWaitDelay の設定を見直す

ただし，OS のバージョンやエディション，セキュリティ更新プログラムの適用状況によって仕様が異なるため，詳細については各 OS のドキュメントを参照してください。また，これらの設定は OS 全体に影響が及ぶため，注意が必要です。

(4)　リクエスト再送抑止

Web リソースクライアント側の JAX-RS エンジンは，JDK の HTTP クライアント実装を利用して通信しています。JDK の HTTP クライアント実装は，RFC 2616 に反して HTTP 通信でエラーが発生し，サーバから正しいレスポンスを受け取れなかった場合，一度だけリクエストを再送します。JDK のシステムプ

ロパティを使用するとリクエストの再送を抑止できます。詳細は，「10.20　sun.net.www.http.HttpClientによるリクエスト再送抑止」を参照してください。なお，参照する場合は「Web サービスクライアント」を「RESTful Web サービスクライアント」に，「JAX-WS エンジン」を「JAX-RS エンジン」に読み替えてください。

(5)　コマンドラインを利用したクライアントアプリケーションの実行

コマンドラインで動作する Java アプリケーションをクライアントアプリケーションとして利用するときに必要な設定，コマンドラインの指定例，注意事項については，「10.14　コマンドラインを利用したクライアントアプリケーションの実行」を参照してください。なお，RESTful Web サービスクライアントでは，Java アプリケーション用オプション定義ファイルに次のキーと値も追加してください。

```
add.class.path=<インストールディレクトリ>/jaxws/lib/cjjaxrs.jar
```

(6)　通信失敗時の Java 例外

Web サービスとの通信が失敗した場合，次の例外が記録されることがあります。

java.net.ConnectException: ["Connection refused: connect"または"接続を拒否されました"]

このメッセージは，OS や環境によって出力内容が異なる場合があります。

12 RESTful Web サービス開発の例

この章では，RESTful Web サービス（Web リソース）を開発する場合の例を説明します。

12.1　開発例の構成

この章で説明する開発例では，RESTful Web サービス（Web リソース）を開発します。ルートリソースクラスのほか，サブリソースクラスと例外マッピングプロバイダも実装します。

開発する Web リソースの構成を次の表に示します。なお，この開発例は次のディレクトリにサンプルとして提供されています。

<インストールディレクトリ>¥jaxrs¥samples¥tutorial¥

表 12–1　Web リソースの構成

項番	項目	値
1	デプロイする J2EE サーバの名称	jaxrsserver
2	Web サーバのホスト名とポート番号	webhost:8085
3	ネーミングサーバの URL	corbaname::testserver:900
4	コンテキストルート	tutorial
5	ルートリソースクラスのコンテキストパス	root
6	ルートリソースクラス	com.sample.resources.Resource
7	サブリソースクラス	com.sample.resources.SubResource
8	例外マッピングプロバイダ	com.sample.providers.RuntimeExceptionMapper

Web リソース開発時のカレントディレクトリの構成を次の表に示します。

表 12–2　カレントディレクトリの構成

ディレクトリ	説明
c:¥temp¥jaxrs¥works¥tutorial	カレントディレクトリです。
├server¥	Web リソースの開発で使用します。
├├META-INF¥	EAR ファイルの META-INF ディレクトリに対応します。
├└application.xml	「12.3.4　application.xml を作成する」で作成します。
├src¥	Web リソースのソースファイル（*.java）を格納します。「12.3.1　ルートリソースクラスを作成する」で使用します。
├WEB-INF¥	WAR ファイルの WEB-INF ディレクトリに対応します。
├├web.xml	「12.3.3　web.xml を作成する」で作成します。
├└classes¥	コンパイルしたクラスファイル（*.class）を格納します。「12.3.2　Java ソースをコンパイルする」で使用します。

ディレクトリ	説明
│ │ │ │ │ │	
├ tutorial.ear	「12.3.5　EAR ファイルを作成する」で作成します。
└ tutorial.war	
└ client¥	Web リソースクライアントの開発で使用します。
├ src¥	Web リソースクライアントのソースファイル（*.java）を格納します。「12.5.1 Web リソースクライアントの実装クラスを作成する（クライアント API を利用する）」または「12.5.2　Web リソースクライアントの実装クラスを作成する（java.net.HttpURLConnection を利用する）」で作成します。
├ classes¥	コンパイルしたクラスファイル（*.class）を格納します。「12.3.2　Java ソースをコンパイルする」で作成します。
├ usrconf.cfg	「12.6.1　Java アプリケーション用オプション定義ファイルを作成する」で作成します。
└ usrconf.properties	「12.6.2　Java アプリケーション用ユーザプロパティファイルを作成する」で作成します。

カレントディレクトリのパスは，開発する環境に合わせて変更してください。

なお，以降の説明では，この表に示すディレクトリおよびファイル名を使用します。コマンド実行例や Java ソースなどで背景色付きの太字になっている部分は，この例で使用する指定値や生成される値を示します。構築する環境に合わせて読み替えてください。

また，この章で説明する開発例では，Web リソースと Web リソースクライアントを同じ環境で開発しますが，別の環境で開発することもできます。別の環境で開発する場合は，それぞれの環境に合わせて，カレントディレクトリのパスを読み替えてください。

12.2　開発例の流れ

この章で説明する開発例では，次の流れで開発および実行します。

Web リソースの開発

1. ルートリソースクラスを作成する（12.3.1）
2. Java ソースをコンパイルする（12.3.2）
3. web.xml を作成する（12.3.3）
4. application.xml を作成する（12.3.4）
5. EAR ファイルを作成する（12.3.5）

デプロイと開始

1. EAR ファイルをデプロイする（12.4.1）
2. Web サービスを開始する（12.4.2）

Web リソースクライアントの開発

1. Web リソースクライアントの実装クラスを作成する（12.5.1 または 12.5.2）
2. Web リソースクライアントの実装クラスをコンパイルする（12.5.3）

Web リソースの呼び出し

1. Java アプリケーション用オプション定義ファイルを作成する（12.6.1）
2. Java アプリケーション用ユーザプロパティファイルを作成する（12.6.2）
3. Web リソースクライアントを実行する（12.6.3）

12.3 Web リソースの開発例

ここでは，Web リソースの開発例を説明します。

12.3.1 ルートリソースクラスを作成する

ルートリソースクラスを作成します。ここでは，次のリソースメソッド，サブリソースメソッド，サブリソースロケータ，およびインジェクションの対象となるコンストラクタ，フィールド，bean プロパティを持つルートリソースクラスを作成します。

表 12-3 作成するルートリソースクラス

項目		要求メソッド識別子	/root 以下のコンテキストパス	備考
リソースメソッド	resourceMethod1	@GET	−	−
	resourceMethod7	@POST	−	• フォーム値をインジェクトする FormParam アノテーションでアノテートされたパラメタを持ちます。 • MIME タイプ"*/*"の HTTP エンティティを受け取ります。 • MIME タイプ"application/xml"の HTTP エンティティを返します。
サブリソースメソッド	subResourceMethod2	@GET	/getQueryParam	−
	subResourceMethod3	@POST	/getUriInfoAndEntity	エンティティパラメタを持ちます。
	subResourceMethod4	@GET	/getHttpHeaders	HttpHeaders をインジェクトする Context アノテーションでアノテートされたパラメタを持ちます。
	subResourceMethod5	@GET	/getMatrixParam	マトリクスパラメタをインジェクトする MatrixParam アノテーションでアノテートされたパラメタを持ちます。
	subResourceMethod6	@GET	/getCookieParam	Cookie をインジェクトする CookieParam アノテーションでアノテートされたパラメタを持ちます。
	subResourceMethod8	@GET	/getPathParam	パスパラメタをインジェクトする PathParam アノテーションでアノテートされたパラメタを持ちます。

項目		要求メソッド識別子	/root 以下のコンテキストパス	備考
	subResourceMethodThrowingException	@GET	/{path:[A-Z][a-z]+}/exception	例外をスローします。スローされた例外は例外マッピングプロバイダ RuntimeExceptionMapper によって HTTP エンティティにマッピングされます。
	pojoJsonMappingMethod	@POST	/pojoJsonMapping	JSON 形式のデータを処理します。
サブリソースロケータ	subResourceMethod9	−	/subresourceLocator	サブリソースクラス SubResource に処理を委譲します。
コンストラクタ		−	−	UriInfo をインジェクトする Context アノテーションでアノテートされたパラメタを持ちます。
フィールド		−	−	Request をインジェクトする Context アノテーションでアノテートされています。
bean プロパティ		−	−	クエリパラメタをインジェクトする QueryParam アノテーションでアノテートされています。

（凡例）−：該当する内容がないことを示します。

ルートリソースクラスの作成例を次に示します。

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.CookieParam;
import javax.ws.rs.FormParam;
import javax.ws.rs.MatrixParam;
import javax.ws.rs.PathParam;
import javax.ws.rs.QueryParam;
import javax.ws.rs.DefaultValue;
import javax.ws.rs.Encoded;
import javax.ws.rs.Consumes;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.HttpHeaders;
import javax.ws.rs.core.Request;
import javax.ws.rs.core.UriInfo;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;

//サンプル：ルートリソースクラス
//このクラスはURI ”/root”でアクセスされる
@Path(”/root”)
```

```
public class Resource {

    //コンテキスト（javax.ws.rs.core.Requestオブジェクト）の
    //フィールドへのインジェクション
    private @Context
    Request request;

    //setter()メソッドを使用したクエリパラメタのインジェクション
    @QueryParam("queryParam")
    public void setQueryParam(String queryParam) {
        this.queryParam = queryParam;
    }
    //上記のsetter()メソッドでインジェクトされたクエリパラメタを
    //格納するprivateフィールド
    private String queryParam;

    //コンテキスト（javax.ws.rs.core.UriInfoオブジェクト）の
    //コンストラクタパラメタへのインジェクション
    public Resource(@Context UriInfo uriInfo) {
        this.uriInfo = uriInfo;
    }
    //上記のコンストラクタでインジェクトされたUriInfoオブジェクトを
    //格納するprivateフィールド
    private UriInfo uriInfo;

    //サンプル：リソースメソッド1
    //  フィールドにインジェクトされたコンテキストの値
    //  （javax.ws.rs.core.Requestオブジェクト）を返す
    //このメソッドはHTTP GETリクエストを処理する
    @GET
    public String resourceMethod1() {
        String returnString = "RequestMethod: " + request.getMethod();
        return returnString;
    }

    //サンプル：サブリソースメソッド2
    //  setter()メソッドにインジェクトされたクエリパラメタの値を返す
    //このメソッドはHTTP GETリクエストを処理する
    @GET
    //このメソッドはURI "/root/getQueryParam"でアクセスされる
    @Path("getQueryParam")
    public String subResourceMethod2() {
        String returnString = "QueryParameter: " + queryParam;
        return returnString;
    }

    //サンプル：サブリソースメソッド3
    //  コンストラクタパラメタにインジェクトされた
    //  コンテキストの値（javax.ws.rs.core.UriInfoオブジェクト）と，
    //  受信したHTTPリクエストのエンティティボディを返す
    //このメソッドはHTTP POSTリクエストを処理する
    @POST
    //このメソッドはURI "/root/getUriInfoAndEntity"でアクセスされる
    @Path("getUriInfoAndEntity")
    public String subResourceMethod3(
            //エンティティパラメタ（アノテートされていないパラメタ）は
            //HTTPリクエストのエンティティボディからマッピングされる
            String entity) {
```

```
        String returnString = "UriInfo: " + uriInfo.getPath() + "; Entity: " + entity;
        return returnString;
    }

    //サンプル：サブリソースメソッド4
    //  サブリソースメソッドパラメタにインジェクトされた
    //  コンテキストの値（javax.ws.rs.core.HttpHeadersオブジェクト）を返す
    //このメソッドはHTTP GETリクエストを処理する
    @GET
    //このメソッドはURI "/root/getHttpHeaders"でアクセスされる
    @Path("getHttpHeaders")
    public String subResourceMethod4(
            //javax.ws.rs.core.HttpHeadersオブジェクトの
            //サブリソースメソッドパラメタへのインジェクション
            @Context HttpHeaders httpHeaders) {
        String returnString = "HttpHeaders: " +
            httpHeaders.getRequestHeader("header").get(0).toString();
        return returnString;
    }
    //サンプル：サブリソースメソッド5
    //  サブリソースメソッドパラメタにインジェクトされた
    //  マトリクスパラメタの値を返す
    //このメソッドはHTTP GETリクエストを処理する
    @GET
    //このメソッドはURI "/root/getMatrixPara"でアクセスされる
    @Path("getMatrixParam")
    public String subResourceMethod5(
            //デフォルト値はマトリクスパラメタに与えられる
            @DefaultValue("defaultValue")
            //マトリクスパラメタの
            //サブリソースメソッドパラメタへのインジェクション
            @MatrixParam("matrix") String matrixParam) {
        String returnString = "MatrixParam: " + matrixParam;
        return returnString;
    }

    //サンプル：サブリソースメソッド6
    //  サブリソースメソッドパラメタにインジェクトされたCookieの値を返す
    //このメソッドはHTTP GETリクエストを処理する
    @GET
    //このメソッドはURI "/root/getCookieParam"でアクセスされる
    @Path("getCookieParam")
    public String subResourceMethod6(
            //自動URI復号化を無効にする
            @Encoded
            //Cookieのサブリソースメソッドパラメタへのインジェクション
            @CookieParam("cookie") String cookieParam ) {
        String returnString = "CookieParam: " + cookieParam;
        return returnString;
    }
    //サンプル：リソースメソッド7
    //  リソースメソッドパラメタにインジェクトされた値を返す
    //このメソッドはMIMEメディアタイプで定義された内容を使用する
    @Consumes("*/*")
    //このメソッドはMIMEメディアタイプ"application/xml"で定義された
    //内容を使用する
    @Produces("application/xml")
    //このメソッドはHTTP POSTリクエストを処理する
```

```
@POST
public Response resourceMethod7(
        //formパラメタのリソースメソッドパラメタへのインジェクション
        @FormParam("form") String formParam ) {
    ResponseBuilder rb = Response.status(200)
        .entity("<FormParam>" + formParam + "</FormParam>" )
        .type("application/xml");
    return rb.build();
}

//サンプル：サブリソースメソッド8
//  サブリソースメソッドパラメタにインジェクトされた
//  パスパラメタの値を返す
//このメソッドはHTTP GETリクエストを処理する
@GET
//このメソッドはURI "/root/getPathParam/{path:[A-Z][a-z]+}"で
//アクセスされる
//"getPathParam/{path:[A-Z][a-z]+}"は変数を含むURIテンプレートである
//"{path:[A-Z][a-z]+}"は正規表現を使用している
@Path("getPathParam/{path:[A-Z][a-z]+}")
public String subResourceMethod8(
        //パスパラメタの
        //サブリソースメソッドパラメタへのインジェクション
        @PathParam("path") String pathParam ) {
    String returnString = "PathParam: " + pathParam;
    return returnString;
}

//サンプル：サブリソースロケータ9
//サブリソースクラスはURI "/root/subresourceLocator"でアクセスされる
@Path("/subresourceLocator")
public SubResource subResourceMethod9() {
    //サブリソースロケータはHTTPリクエストを処理するために
    //リソースクラスインスタンスを返す
    return new SubResource();
}

//サンプル：サブリソースメソッド10
//  例外をスローする
//スローされた例外は例外マッピングプロバイダによって
//HTTPレスポンスにマッピングされる
//このメソッドはHTTP GETリクエストを処理する
@Path("/exception")
//このメソッドはURI "/root/exception"でアクセスされる
@GET
public String subResourceMethodThrowingException() {
    throw new RuntimeException();
}

//サンプル: サブリソースメソッド11
//  JSON形式のデータを処理する
//このメソッドはHTTP POSTリクエストを処理する
@Path("pojoJsonMapping")
//このメソッドは URI "root/PojoJsonMapping"でアクセスされる
@POST
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public CustomType pojoJsonMappingMethod(CustomType record) {
```

```
        // クライアントから受け取ったrecordオブジェクトの内容を確認する
        if(!record.getName().equals("Old Record Name")){
            throw new RuntimeException();
        }
        List<Integer> oldGrades = new ArrayList<Integer>();
        oldGrades.add(1);
        oldGrades.add(2);
        oldGrades.add(3);
        if(!record.getGrades().equals(oldGrades)){
            throw new RuntimeException();
        }

        // クライアントに送り返すCustomTypeオブジェクトを新規に生成する
        CustomType newRecord = new CustomType();
        newRecord.setName("New Record Name");
        List<Integer> newGrades = new ArrayList<Integer>();
        newGrades.add(5);
        newGrades.add(6);
        newGrades.add(7);
        newRecord.setGrades(newGrades);

        return newRecord;
    }
}
```

作成したルートリソースクラス（Resource.java）は，UTF-8 形式で c:¥temp¥jaxrs¥works¥tutorial¥server¥src¥com¥sample¥resources¥ディレクトリに保存します。

サブリソースクラスの作成例を次に示します。このサブリソースクラスは HTTP GET リクエストを受け付けるリソースメソッドを持ちます。なお，サブリソースクラスの作成は任意です。

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.HeaderParam;

//サンプル：サブリソースクラス
public class SubResource {

    //このメソッドはHTTP GETリクエストを処理する
    @GET
    public String getHandlerForSubResource(
            //サブリソースメソッドパラメタへの
            //ヘッダパラメタのインジェクション
            @HeaderParam("header") String headerParam) {
        return "Header: " + headerParam;
    }
}
```

作成したサブリソースクラス（SubResource.java）は，UTF-8 形式で c:¥temp¥jaxrs¥works¥tutorial¥server¥src¥com¥sample¥resources¥ディレクトリに保存します。

例外マッピングプロバイダの作成例を次に示します。この例外マッピングプロバイダは，HTTP ヘッダ header に”RuntimeException occurs”の値を持つステータスコード 501 の HTTP レスポンスに，ランタイム例外をマッピングします。なお，例外マッピングプロバイダの作成は任意です。

```
package com.sample.providers;

import javax.ws.rs.core.Response;
import javax.ws.rs.ext.Provider;
import javax.ws.rs.ext.ExceptionMapper;

//サンプル：例外マッピングクラス
//このクラスはRuntimeExceptionをHTTPレスポンスにマッピングする
@Provider
public class RuntimeExceptionMapper implements
        ExceptionMapper<RuntimeException> {
    //このメソッドはアプリケーションがランタイム例外を
    //スローしたときに呼び出される
    public Response toResponse(RuntimeException re) {
        //HTTPレスポンスのステータスとして”HTTP/1.1 501 Not Implemented”を
        //設定する
        //HTTPレスポンスのヘッダに”header: RuntimeException occurs”を
        //設定する
        return Response.status(501)
            .header(”header”, ”RuntimeException occurs”).build();
    }
}
```

作成した例外マッピングプロバイダ（RuntimeExceptionMapper.java）は，UTF-8 形式で c:¥temp¥jaxrs¥works¥tutorial¥server¥src¥com¥sample¥providers¥ディレクトリに保存します。

JSON 形式のデータにマッピングされる POJO の作成例を次に示します。なお，JSON マッピング用の POJO の作成は任意です。

```
package com.sample.resources;

import java.util.List;

public class CustomType {

    private String name;
    private List<Integer> grade;

    public CustomType() {
    }
    public CustomType (String name, List<Integer> grades){
        this.name = name;
        this.grade = grades;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
```

```
    public List<Integer> getGrades() {
        return grade;
    }
    public void setGrades(List<Integer> grades) {
        this.grade = grades;
    }
    @Override
    public String toString() {
        return "Record [Name=" + name + ", Grades=" + grade.toString() + "]";
    }
}
```

作成した JSON マッピング用の POJO（CustomType.java）は，UTF-8 形式で c:¥temp¥jaxrs¥works¥tutorial¥server¥src¥com¥sample¥resources¥ディレクトリに保存します。

12.3.2 Java ソースをコンパイルする

javac コマンドを使用して，作成した各 Java ソースをコンパイルします。コンパイルの例を次に示します。

```
> cd c:¥temp¥jaxrs¥works¥tutorial¥server¥
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥jaxrs
¥lib¥cjjaxrs.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxp.j
ar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar;.¥classes" -d WEB-INF¥classes¥ src¥com¥sample¥reso
urces¥Resource.java src¥com¥sample¥resources¥SubResource.java src¥com¥sample¥providers¥Runti
meExceptionMapper.java src¥com¥sample¥resources¥CustomType.java
```

javac コマンドが正常に終了すると，c:¥temp¥jaxrs¥works¥tutorial¥server¥WEB-INF¥classes¥ディレクトリ以下のパッケージ名に対応するサブディレクトリに，クラスファイル (*.class) が生成されます。

javac コマンドについては，JDK のドキュメントを参照してください。

12.3.3 web.xml を作成する

WAR ファイルの構成要素として必要な web.xml を作成します。

web.xml の作成例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/
web-app_3_0.xsd">

    <description>Sample web service &amp;quot;tutorial&amp;quot;</description>
    <display-name>Sample_web_service_jaxrs_tutorial</display-name>
    <servlet>
        <servlet-name>CosminexusJaxrsServlet</servlet-name>
        <servlet-class>
```

```
            com.cosminexus.jersey.spi.container.servlet.ServletContainer
        </servlet-class>
        <init-param>
            <param-name>com.sun.jersey.config.property.packages</param-name>
            <param-value>com.sample.resources;com.sample.providers</param-value>
        </init-param>
        <!-- POJO JSON support web.xml configuration -->
        <init-param>
            <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
            <param-value>true</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>CosminexusJaxrsServlet</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
</web-app>
```

バージョン 2.5 の web.xml を作成する場合は，web-app 要素の version 属性を"2.5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"にしてください。

作成した web.xml は，UTF-8 形式で c:¥temp¥jaxrs¥works¥tutorial¥server¥WEB-INF¥ディレクトリに保存します。web.xml の設定項目については，「3.4　web.xml の作成」を参照してください。

12.3.4　application.xml を作成する

EAR ファイルの構成要素として必要な application.xml を作成します。

application.xml の作成例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/
application_6.xsd">

    <description>Sample application &quot;tutorial&quot;</description>
    <display-name>Sample_application_jaxrs_tutorial</display-name>
    <module>
        <web>
            <web-uri>tutorial.war</web-uri>
            <context-root>tutorial</context-root>
        </web>
    </module>
</application>
```

バージョン 5 の application.xml を作成する場合は，application 要素の version 属性を"5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/application_5.xsd"にしてください。

作成した application.xml は，UTF-8 形式で c:¥temp¥jaxrs¥works¥tutorial¥server¥META-INF¥ディレクトリに保存します。application.xml を作成するときの注意事項については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「5.2.2　application.xml 編集時の注意事項」を参照してください。

12.3.5　EAR ファイルを作成する

jar コマンドを使用して，これまでに作成したファイルを含めた EAR ファイルを作成します。

EAR ファイルの作成例を次に示します。

```
> cd c:¥temp¥jaxrs¥works¥tutorial¥server¥
> jar cvf tutorial.war .¥WEB-INF
> jar cvf tutorial.ear .¥jaxrs_sample.war .¥META-INF¥application.xml
```

jar コマンドが正常に終了すると，c:¥temp¥jaxrs¥works¥tutorial¥server¥ディレクトリに tutorial.ear が作成されます。

jar コマンドについては，JDK のドキュメントを参照してください。

12.4　デプロイと開始の例

ここでは，Web リソースのデプロイと開始の例を説明します。

12.4.1　EAR ファイルをデプロイする

cjimportapp コマンドを使用して，作成した EAR ファイルを J2EE サーバにデプロイします。

デプロイの例を次に示します。

```
> cd c:¥temp¥jaxrs¥works¥tutorial¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjimportapp" jaxrsserver -nameserver corbaname::testserver
:900 -f tutorial.ear
```

cjimportapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート)」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションをデプロイ（インポート）する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.3　J2EE アプリケーションのインポート」を参照してください。

12.4.2　Web サービスを開始する

cjstartapp コマンドを使用して，Web サービスを開始します。

Web サービスを開始する例を次に示します。

```
> cd c:¥temp¥jaxrs¥works¥tutorial¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjstartapp" jaxrsserver -nameserver corbaname::testserver:
900 -name Sample_application_jaxrs_tutorial
```

cjstartapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartapp（J2EE アプリケーションの開始)」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションを開始する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.1　J2EE アプリケーションの開始」を参照してください。

12.5 Web リソースクライアントの開発例

ここでは，Web リソースクライアントの開発例を説明します。Web リソースクライアントは，RESTful Web サービス用クライアント API か，または java.net.URL や java.net.HttpURLConnection などの標準的な Java API を利用して開発します。

12.5.1 Web リソースクライアントの実装クラスを作成する（クライアント API を利用する）

RESTful Web サービス用クライアント API を利用するクライアントの実装クラスを作成します。

例を次に示します。

```
package com.sample.client;

import java.net.URI;
import java.net.URLEncoder;
import java.util.List;
import java.util.ArrayList;
import java.util.Map;

import com.cosminexus.jersey.api.client.Client;
import com.cosminexus.jersey.api.client.WebResource;
import com.cosminexus.jersey.api.client.ClientRequest;
import com.cosminexus.jersey.api.client.ClientResponse;
import com.cosminexus.jersey.api.client.config.ClientConfig;
import com.cosminexus.jersey.api.client.config.DefaultClientConfig;
import com.cosminexus.jersey.api.client.ClientHandlerException;
import com.cosminexus.jersey.api.json.JSONConfiguration;
import javax.ws.rs.core.Cookie;

//サンプル：Webリソースのクライアントの実行
public class SampleClient {

    public static void main(String[] args) {

        final String HOST = args[0];
        final String PORT = args[1];

        SampleClient sampleClient = new SampleClient();

        try{
            sampleClient.demonstration13(HOST, PORT);
            sampleClient.demonstration14(HOST, PORT);
            sampleClient.demonstration15(HOST, PORT);

            System.out.println("¥n----- Successfully Ended -----");

        }catch(Exception e){
            //詳細な例外のメッセージを表示する
            System.out.println(e.getMessage());
```

```
    }
}
private void demonstration13(String HOST, String PORT) {

    System.out.println("¥n Demonstration 13 started.");
    System.out.println(" This demonstrates how to use Client API " +
            "to receive a response as a ClientResponse.");
    System.out.println(" This demonstrates usage of @Encoded at " +
            "@CookieParam. ¥n Automatic URI decoding should be disabled.");

    String url = null;
    Client client = null;
    ClientResponse response = null;

    String responseEntity = "";
    Map<String, List<String>> headers = null;
    int status;

    //Webリソースを呼び出す
    try {
        //呼び出す対象のWebリソースのURIを設定する
        url = new String("http://" + HOST + ":" + PORT+
                "/tutorial/root/getCookieParam");
        Cookie cookie = new Cookie("cookie", "cookie%20value");
        //クライアントAPIを利用するため, Clientオブジェクトを生成する
        client = Client.create();
        //HTTPリクエストを作成して送信しHTTPレスポンスを受信する
        //- ClientオブジェクトからWebResourceオブジェクトを生成する
        //- Cookieヘッダに"cookie=cookie%20value"を設定する
        //- HTTP GETリクエストを送信しClientResponseとして
        //  HTTPレスポンスを受信する
        response = client.resource(url)
                         .cookie(cookie)
                         .get(ClientResponse.class);
        //HTTPレスポンスのヘッダを取得する
        headers = response.getHeaders();
        //HTTPレスポンスのステータスコードを取得する
        status = response.getStatus();
        //HTTPレスポンスのエンティティを取得する
        responseEntity = response.getEntity(String.class);
    }catch (Exception e) {
        System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
        //スタックトレースを表示する
        e.printStackTrace();
        throw new RuntimeException(" Demonstration 13 failed.");
    }
    System.out.println(" The target URL is ¥"" + url + "¥".");
    System.out.println(" The HTTP method  is " + "¥"GET¥"" + ".");
    System.out.println(" Connection and interaction ended successfully.");

    //ステータスコードとエンティティの期待値を設定する
    int statusExpect = 200;
    String responseEntityExpect = "CookieParam: cookie%20value";

    //ステータスコードとエンティティが期待値と同じかチェックする
    if (status == statusExpect
            & responseEntity.equals(responseEntityExpect)) {
        //HTTPレスポンスのヘッダを表示する
```

```
            System.out.println(" Response headers are " + headers.toString() + ".");
            //HTTPレスポンスのエンティティを表示する
            System.out.println(" Response entity is " + responseEntity + ",");
            System.out.println(" which means target resource completed " +
                    "the process described above without any problem.");

            System.out.println(" Demonstration 13 ended successfully.");
        }else {
            System.out.println(" The response is not as expected.");
            throw new RuntimeException(" Demonstration 13 failed.");
        }

    }

    private void demonstration14(String HOST, String PORT) {

        System.out.println("¥n Demonstration 14 started.");
        System.out.println(" This demonstrates how to send a ClientRequest " +
                "and receive a ClientResponse by using " +
                "Client#handle(ClientRequest request).");
        System.out.println(" This demonstrates usage of @Consumes and " +
                "@Produces.");
        URI url = null;
        Client client = null;
        ClientRequest.Builder requestBuilder = null;
        ClientRequest request = null;
        ClientResponse response = null;

        String responseEntity = "";
        Map<String, List<String>> headers = null;
        int status;

        //Webリソースを呼び出す
        try {
            //呼び出す対象のWebリソースのURIを設定する
            url = new URI("http://" + HOST + ":" + PORT+ "/tutorial/root");
            //HTTPリクエストのエンティティを作成する
            String data = URLEncoder.encode("form", "UTF-8") + "="
                + URLEncoder.encode("formValue", "UTF-8");
            //クライアントAPIを利用するため, Clientオブジェクトを生成する
            client = Client.create();
            //ClientRequest.Builderオブジェクトを生成する
            requestBuilder = ClientRequest.create();
            //- HTTPリクエストの"Content-Type"ヘッダに
            //  "application/x-www-form-urlencoded"を設定する
            //- HTTPリクエストのエンティティを設定する
            requestBuilder.type("application/x-www-form-urlencoded")
                          .entity(data);
            //ClientRequest.BuilderからClientRequestを作成する
            request = requestBuilder.build(url, "POST");
            //Client#handle()メソッドを呼び出してHTTP POSTリクエストを送信し
            //ClientResponseとしてHTTPレスポンスを受信する
            response = client.handle(request);
            //HTTPレスポンスのヘッダを取得する
            headers = response.getHeaders();
            //HTTPレスポンスのステータスコードを取得する
            status = response.getStatus();
            //HTTPレスポンスのエンティティを取得する
```

```
            responseEntity = response.getEntity(String.class);
        }catch (ClientHandlerException e) {
            System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
            //スタックトレースを表示する
            e.printStackTrace();
            throw new RuntimeException(" Demonstration 14 failed.");
        }catch (Exception e) {
            System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
            //スタックトレースを表示する
            e.printStackTrace();
            throw new RuntimeException(" Demonstration 14 failed.");
        }
        System.out.println(" The target URL is ¥"" + url + "¥".");
        System.out.println(" The HTTP method  is " + "¥"POST¥"" + ".");
        System.out.println(" Connection and interaction ended successfully.");

        //ステータスコードとエンティティの期待値を設定する
        int statusExpect = 200;
        String responseEntityExpect = "<FormParam>formValue</FormParam>";

        //ステータスコードとエンティティが期待値と同じかチェックする
        if (status == statusExpect
                & responseEntity.equals(responseEntityExpect)) {
            //HTTPレスポンスのヘッダを表示する
            System.out.println(" Response headers are " + headers.toString() + ".");
            //HTTPレスポンスのエンティティを表示する
            System.out.println(" Response entity is " + responseEntity + ",");
            System.out.println(" which means target resource completed " +
                    "the process described above without any problem.");

            System.out.println(" Demonstration 14 ended successfully.");
        }else {
            System.out.println(" The response is not as expected.");
            throw new RuntimeException(" Demonstration 14 failed.");
        }

    }
    private void demonstration15(String HOST, String PORT) {

        System.out.println("¥n Demonstration 15 started.");
        System.out.println(" This demonstrates JSON support of CJR.");
        System.out.println(" This demonstrates POJO and JSON mapping.");

        String url = null;
        Client client = null;
        CustomType response = null;

        //Webリソースを呼び出す
        try {
            //呼び出す対象のWebリソースのURIを設定する
            url = new String("http://" + HOST + ":" + PORT +
                    "/tutorial/root/PojoJsonMapping");
            //JSON POJOマッピングを有効にする設定を行う
            ClientConfig cc = new DefaultClientConfig();
            cc.getFeatures()
              .put(JSONConfiguration.FEATURE_POJO_MAPPING, Boolean.TRUE);
            //クライアントAPIを利用するため, Clientオブジェクトを生成する
            //(設定を有効にするためClientConfigオブジェクトを渡す)
```

```
            client = Client.create(cc);
            //CustomTypeオブジェクトを生成する
            CustomType record = new CustomType();
            record.setName("Old Record Name");
            List<Integer> grades = new ArrayList<Integer>();
            grades.add(1);
            grades.add(2);
            grades.add(3);
            record.setGrades(grades);
            //HTTPリクエストを作成する
            //- ClientオブジェクトからWebResourceオブジェクトを生成する
            //- Content-Typeヘッダに"application/json"を設定する
            //- エンティティにCustomTypeオブジェクトを設定する
            //- HTTP POSTリクエストを送信しCustomTypeとして
            //  HTTPレスポンスを受信する
            response = client.resource(url)
                            .type("application/json")
                            .entity(record)
                            .post(CustomType.class);
        }catch (Exception e) {
            System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
            //スタックトレースを表示する
            e.printStackTrace();
            throw new RuntimeException(" Demonstration 15 failed.");
        }
        System.out.println(" The target URL is ¥"" + url + "¥".");
        System.out.println(" The HTTP method  is " + "¥"POST¥"" + ".");
        System.out.println(" Connection and interaction ended successfully.");

        //エンティティの期待値を設定する
        String responseNameExpect = "New Record Name";
        List<Integer> responseGradesExpect = new ArrayList<Integer>();
        responseGradesExpect.add(5);
        responseGradesExpect.add(6);
        responseGradesExpect.add(7);

        //エンティティが期待値と同じかチェックする
        if (response.getName().equals(responseNameExpect)
                & response.getGrades().equals(responseGradesExpect)) {
            //HTTPレスポンスのエンティティを表示する
            System.out.println(" Response is " + response.toString() + ",");
            System.out.println(" which means target resource completed " +
                    "the process described above without any problem.");

            System.out.println(" Demonstration 15 ended successfully.");
        }else {
            System.out.println(" The response is not as expected.");
            throw new RuntimeException(" Demonstration 15 failed.");
        }

    }
    private static class CustomType {

        private String name;
        private List<Integer> grade;

        public CustomType() {
        }
```

```
        public CustomType (String name, List<Integer> grades){
            this.name = name;
            this.grade = grades;
        }

        public String getName() {
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }

        public List<Integer> getGrades() {
            return grade;
        }

        public void setGrades(List<Integer> grades) {
            this.grade = grades;
        }

        @Override
        public String toString() {
            return "Record [Name=" + name + ", Grades=" + grade.toString() + "]";
        }

    }
}
```

作成した SampleClient.java は，UTF-8 形式で
c:¥temp¥jaxrs¥works¥tutorial¥client¥src¥com¥sample¥client¥ディレクトリに保存します。

12.5.2 Web リソースクライアントの実装クラスを作成する（java.net.HttpURLConnection を利用する）

HttpURLConnection クラスを利用するクライアントの実装クラスを作成します。

例を次に示します。

```
package com.sample.client;

import java.net.URL;
import java.net.URLEncoder;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.BufferedReader;
import java.io.IOException;
import java.util.List;
import java.util.Map;
```

```
//サンプル：Webリソースのクライアントの実行
public class SampleClient {

    public static void main(String[] args) {

        final String HOST = args[0];
        final String PORT = args[1];

        SampleClient sampleClient = new SampleClient();

        try{
            sampleClient.demonstration1(HOST, PORT);
            sampleClient.demonstration2(HOST, PORT);

            System.out.println("¥n----- Successfully Ended -----");

        }catch(Exception e){
            //詳細な例外のメッセージを表示する
            System.out.println(e.getMessage());
        }
    }

    private void demonstration1(String HOST, String PORT) {

        System.out.println("¥n Demonstration 1 started.");
        System.out.println(" This demonstrates injection of " +
                "javax.ws.rs.core.Request instance " +
                "onto resource class field by using @Context.");

        URL url = null;
        HttpURLConnection httpConn = null;

        Map<String, List<String>> headers = null;
        List<String> status = null;
        String responseEntity = "";

        //Webリソースを呼び出す
        try {
            //呼び出す対象のWebリソースのURIを設定する
            url = new URL("http://" + HOST + ":" + PORT+
                    "/tutorial/root");
            //初回接続をセットアップする
            httpConn = (HttpURLConnection) url.openConnection();
            //HTTP()メソッドを"GET"に設定する
            httpConn.setRequestMethod("GET");
            //接続を開始する
            httpConn.connect();
        }catch (MalformedURLException e) {
            System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
            //スタックトレースを表示する
            e.printStackTrace();
            throw new RuntimeException(" Demonstration 1 failed.");
        }catch (IOException e) {
            System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
            //スタックトレースを表示する
            e.printStackTrace();
            throw new RuntimeException(" Demonstration 1 failed.");
```

```
        }catch (Exception e) {
            System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
            //スタックトレースを表示する
            e.printStackTrace();
            throw new RuntimeException(" Demonstration 1 failed.");
        }

        System.out.println(" The target URL is ¥"" + url + "¥".");
        System.out.println(" The HTTP method  is " + "¥"GET¥"" + ".");

        try {
            //HTTPレスポンスのヘッダを取得する
            headers = httpConn.getHeaderFields();
            //ヘッダからHTTPステータスを取得する
            status = headers.get(null);
            //入力ストリームリーダを取得してHTTPレスポンスの
            //エンティティボディを読み込む
            BufferedReader rd = new BufferedReader(new InputStreamReader(
                httpConn.getInputStream()));
            String line = "";
            while ((line = rd.readLine()) != null) {
                responseEntity += line;
            }
            //入力ストリームリーダを閉じる
            rd.close();
            //接続を終了する
            httpConn.disconnect();
        }catch (Exception e) {
            System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
            //スタックトレースを表示する
            e.printStackTrace();
            throw new RuntimeException(" Demonstration 1 failed.");
        }

        System.out.println(" Connection and interaction ended successfully.");

        //ステータスコードとエンティティボディの期待値を設定する
        String statusExpect = "[HTTP/1.1 200 OK]";
        String responseEntityExpect = "RequestMethod: GET";

        //ステータスコードとエンティティボディが期待値と同じかチェックする
        if (status.toString().equals(statusExpect)
                & responseEntity.equals(responseEntityExpect)) {
            //HTTPヘッダを表示する
            System.out.println(" Response headers are " + headers.toString() + ".");
            //エンティティボディを表示する
            System.out.println(" Response entity is " + responseEntity + ",");
            System.out.println(" which means target resource completed " +
                    "the process described above without any problem.");

            System.out.println(" Demonstration 1 ended successfully.");
        }else {
            System.out.println(" The response is not as expected.");
            throw new RuntimeException(" Demonstration 1 failed.");
        }

    private void demonstration2(String HOST, String PORT) {
```

```
System.out.println("¥n Demonstration 2 started.");
System.out.println(" This demonstrates injection of QueryParam " +
        "onto resource bean setter method by using @QueryParam.");

URL url = null;
HttpURLConnection httpConn = null;
String responseEntity = "";
Map<String, List<String>> headers = null;
List<String> status = null;

//Webリソースを呼び出す
try {
    //呼び出す対象のWebリソースのURIを設定する
    //URIにクエリパラメタを付与する

    url = new URL("http://" + HOST + ":" + PORT+
            "/tutorial/root/getQueryParam?queryParam=queryValue");
    //初回接続をセットアップする
    httpConn = (HttpURLConnection) url.openConnection();
    //HTTP()メソッドを"GET"に設定する
    httpConn.setRequestMethod("GET");
    //接続を開始する
    httpConn.connect();
}catch (MalformedURLException e) {
    System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
    //スタックトレースを表示する
    e.printStackTrace();
    throw new RuntimeException(" Demonstration 2 failed.");
}catch (IOException e) {
    System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
    //スタックトレースを表示する
    e.printStackTrace();
    throw new RuntimeException(" Demonstration 2 failed.");
}catch (Exception e) {
    System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
    //スタックトレースを表示する
    e.printStackTrace();
    throw new RuntimeException(" Demonstration 2 failed.");
}

System.out.println(" The target URL is ¥"" + url + "¥".");
System.out.println(" The HTTP method  is " + "¥"GET¥"" + ".");

try {
    //HTTPヘッダを取得する
    headers = httpConn.getHeaderFields();
    //HTTPヘッダからのステータスコードを取得する
    status = headers.get(null);
    //入力ストリームリーダを取得してHTTPレスポンスの
    //エンティティボディを読み込む
    BufferedReader rd = new BufferedReader(new InputStreamReader(
        httpConn.getInputStream()));
    String line = "";
    while ((line = rd.readLine()) != null) {
        responseEntity += line;
    }
    //入力ストリームリーダを閉じる
    rd.close();
```

```
            //接続を終了する
            httpConn.disconnect();
        }catch (Exception e) {
            System.out.println(" ERROR: " + e.getClass() + " was thrown. ");
            //スタックトレースを表示する
            e.printStackTrace();
            throw new RuntimeException(" Demonstration 2 failed.");
        }

        System.out.println(" Connection and interaction ended successfully.");

        //ステータスコードとエンティティボディの期待値を設定する
        String statusExpect = "[HTTP/1.1 200 OK]";
        String responseEntityExpect = "QueryParameter: queryValue";

        //ステータスコードとエンティティボディが期待値と同じかチェックする
        if (status.toString().equals(statusExpect)
                & responseEntity.equals(responseEntityExpect)) {
            //HTTPヘッダを表示する
            System.out.println(" Response headers are " + headers.toString() + ".");
            //エンティティボディを表示する
            System.out.println(" Response entity is " + responseEntity + ",");
            System.out.println(" which means target resource completed " +
                    "the process described above without any problem.");

            System.out.println(" Demonstration 2 ended successfully.");
        }else {
            System.out.println(" The response is not as expected.");
            throw new RuntimeException(" Demonstration 2 failed.");
        }

    }
    }

}
```

作成した SampleClient.java は，UTF-8 形式で
c:¥temp¥jaxrs¥works¥tutorial¥client¥src¥com¥sample¥client¥ディレクトリに保存します。

12.5.3 Web リソースクライアントの実装クラスをコンパイルする

javac コマンドを使用して，作成した Web リソースクライアントをコンパイルします。

コンパイルの例を次に示します。

```
> cd c:¥temp¥jaxrs¥works¥tutorial¥client¥
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥jaxrs
¥lib¥cjjaxrs.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;.¥classes" -d .¥classes src¥com¥samp
le¥client¥SampleClient.java
```

javac コマンドが正常に終了すると，c:¥temp¥jaxrs¥works¥tutorial¥client¥classes¥ディレクトリ以下のパッケージ名に対応するサブディレクトリに，クラスファイル（*.class）が生成されます。

javac コマンドについては，JDK のドキュメントを参照してください。

12.6　Web リソースの呼び出し例

ここでは，Web リソースの呼び出し例を説明します。

12.6.1　Java アプリケーション用オプション定義ファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用オプション定義ファイル（usrconf.cfg）を作成します。

Java アプリケーション用オプション定義ファイルの作成例を次に示します。

```
add.class.path=<Application Serverのインストールディレクトリ>¥jaxrs¥lib¥cjjaxws.jar
add.class.path=.¥classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home=<Application Serverのインストールディレクトリ>
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

<Application Server のインストールディレクトリ>の部分は，Application Server をインストールしているパスを絶対パスで指定します。<PRF ID>の部分は，PRF デーモンの識別子を指定します。

作成した Java アプリケーション用オプション定義ファイルは，c:¥temp¥jaxrs¥works¥tutorial¥client¥ディレクトリに保存します。Java アプリケーション用オプション定義ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.1　usrconf.cfg（Java アプリケーション用オプション定義ファイル)」を参照してください。

12.6.2　Java アプリケーション用ユーザプロパティファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用ユーザプロパティファイルを作成します。

ここでは特に設定を変更しないため，c:¥temp¥jaxrs¥works¥tutorial¥client¥ディレクトリに usrconf.properties という名称の空のファイルを作成します。Java アプリケーション用ユーザプロパティファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.2　usrconf.properties（Java アプリケーション用ユーザプロパティファイル)」を参照してください。

12.6.3　Web リソースクライアントを実行する

cjclstartap コマンドを使用して，Web リソースクライアントを実行します。

Web リソースクライアントの実行例を次に示します。

```
> cd c:¥temp¥jaxrs¥works¥tutorial¥client¥
> "%COSMINEXUS_HOME%¥CC¥client¥bin¥cjclstartap"  com.sample.client. SampleClient webhost 808
5
```

cjclstartap コマンドが正常に終了すると，Web リソースクライアントの実行結果が表示されます。Web リソースクライアントの開発方法ごとの，実行結果の表示例を次に示します。

- クライアント API を利用する場合

```
KDJE40053-I The cjclstartap command will now start. (directory for the user definition fi
le = c:¥temp¥jaxrs¥works¥tutorial¥client, PID = 2636)

Demonstration 13 started.
 This demonstrates how to use Client API to receive a response as a ClientResponse.
 This demonstrates usage of @Encoded at @CookieParam.
 Automatic URI decoding should be disabled.
 The target URL is "http://webhost:8085/tutorial/root/getCookieParam".
 The HTTP method is "GET".
 Connection and interaction ended successfully.
 Response headers are {Transfer-Encoding=[chunked], Date=[Tue, 27 Dec 2011 0
7:59:41 GMT], Content-Type=[text/html], Server=[CosminexusComponentContainer]}.
 Response entity is CookieParam: cookie%20value,
 which means target resource completed the process described above without any problem.
 Demonstration 13 ended successfully.

 Demonstration 14 started.
 This demonstrates how to send a ClientRequest and receive a ClientResponse by using Clie
nt#handle(ClientRequest request).
 This demonstrates usage of @Consumes and @Produces.
 The target URL is "http://webhost:8085/tutorial/root".
 The HTTP method is "POST".
 Connection and interaction ended successfully.
 Response headers are {Transfer-Encoding=[chunked], Date=[Tue, 27 Dec 2011 0
7:59:41 GMT], Content-Type=[application/xml], Server=[CosminexusComponentContainer]}.
 Response entity is <FormParam>formValue</FormParam>,
 which means target resource completed the process described above without any problem.
 Demonstration 14 ended successfully.

 Demonstration 15 started.
 This demonstrates JSON support of CJR.
 This demonstrates POJO and JSON mapping.
 The target URL is "http://webhost:8085/tutorial/root/PojoJsonMapping".
 The HTTP method is "POST".
 Connection and interaction ended successfully.
 Response is Record [Name=New Record Name, Grades=[5, 6, 7]],
 which means target resource completed the process described above without any problem.
 Demonstration 15 ended successfully.

----- Successfully Ended -----
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

- HttpURLConnection を利用する場合

```
KDJE40053-I The cjclstartap command will now start. (directory for the user definition fi
le = c:¥temp¥jaxrs¥works¥tutorial¥client, PID = 2636)
```

```
Demonstration 1 started.
 This demonstrates injection of javax.ws.rs.core.Request instance onto resource class fie
ld by using @Context.
 The target URL is "http://webhost:8085/tutorial/root".
 The HTTP method is "GET".
 Connection and interaction ended successfully.
 Response headers are {null=[HTTP/1.1 200 OK], Transfer-Encoding=[chunked], Date=[Tue, 27
 Dec 2011 0
7:59:41 GMT], Content-Type=[text/html], Server=[CosminexusComponentContainer]}.
 Response entity is RequestMethod: GET,
 which means target resource completed the process described above without any problem.
 Demonstration 1 ended successfully.

 Demonstration 2 started.
 This demonstrates injection of QueryParam onto resource bean setter method by using @Que
ryParam.
 The target URL is "http://webhost:8085/tutorial/root/getQueryParam?queryParam=queryValue
".
 The HTTP method  is "GET".
 Connection and interaction ended successfully.
 Response headers are {null=[HTTP/1.1 200 OK], Transfer-Encoding=[chunked], Date=[Tue, 27
 Dec 2011 0
7:59:41 GMT], Content-Type=[text/html], Server=[CosminexusComponentContainer]}.
 Response entity is QueryParameter: queryValue,
 which means target resource completed the process described above without any problem.
 Demonstration 2 ended successfully.

----- Successfully Ended -----
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

イタリック体になっている個所は，実行したタイミングや環境によって変わります。

cjclstartap コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjclstartap（Java アプリケーションの開始)」を参照してください。

13 JAX-RS 機能の設定と動作

この章では，RESTful Web サービス（Web サービス）を開発，運用するときの JAX-RS 機能の各種設定，Web リソースクライアントからの接続の概要，および運用に当たって理解が必要な JAX-RS エンジンの動作について説明します。

13.1　動作定義ファイル

ログの設定などは動作定義ファイルに記述します。動作定義ファイルは，次の 2 種類があります。

- **共通定義ファイル**
  システム共通の動作を設定するための定義ファイルです。一つだけあります。
- **プロセス別の定義ファイル**
  プロセス固有の動作を設定する場合に作成する定義ファイルです。固有の設定が必要なプロセスごとに作成します。例えば，J2EE サーバごとに設定を変更したい場合や，Web リソースクライアントごとに設定を変更したい場合に作成します。

なお，Web リソース側の一部の定義については，web.xml のフィルタの定義や初期化パラメタで上書きできます。web.xml については，「11.2　web.xml の作成」を参照してください。また，クライアント側の一部の定義はクライアント API で上書きできます。クライアント API で上書きできる定義については，「25.1.1　サポートするプロパティとフィーチャ」を参照してください。

ここでは，動作定義ファイルの記述規則，および各定義ファイルの設定について説明します。

13.1.1　動作定義ファイルの記述規則

動作定義ファイルの記述形式，記述規則および動作定義の優先度は，JAX-WS 機能と同じです。詳細は，「10.1.1　動作定義ファイルの記述規則」を参照してください。

13.1.2　共通定義ファイルの設定項目

共通定義ファイルを使用して，システム共通の動作定義を設定します。共通定義ファイルのファイル名，保存ディレクトリ名，および設定項目について説明します。

(1)　ファイル名

共通定義ファイルのファイル名を示します。

cjrconf.properties

(2)　保存先ディレクトリ

共通定義ファイルの保存先ディレクトリを示します。保存先ディレクトリは固定です。

<Application Server のインストールディレクトリ>¥jaxrs¥conf

(3) 設定項目

設定するキー名称と指定内容の一覧を次の表に示します。

表 13–1 共通定義ファイルの設定項目

項番	設定項目	キー名称	指定内容	デフォルト値
1	稼働ログの出力レベル	com.cosminexus.jaxrs.logger.runtime.message.level	稼働ログの出力レベルを指定します。 ERROR，WARN，INFO，DEBUG，NONE のどれかを指定します。各指定値に対応した出力内容については，「39.3.4 ログの重要度と出力条件」を参照してください。	INFO
2	稼働ログの面数	com.cosminexus.jaxrs.logger.runtime.message.file_num	稼働ログの面数を指定します。 数字（1～16）を指定します。	2
3	稼働ログの容量	com.cosminexus.jaxrs.logger.runtime.message.file_size	稼働ログの容量を指定します。 4096～16777216 の数字（単位：バイト）を指定します。	2097152
4	保守ログの出力	com.cosminexus.jaxrs.logger.runtime.maintenance.level	保守ログを出力するかどうかを指定します。 ALL を指定した場合 保守ログが出力されます。 NONE を指定した場合 保守ログが出力されません。	ALL
5	保守ログの面数	com.cosminexus.jaxrs.logger.runtime.maintenance.file_num	保守ログの面数を指定します。 1～16 の数字を指定します。	2
6	保守ログの容量	com.cosminexus.jaxrs.logger.runtime.maintenance.file_size	保守ログの容量を指定します。 4096～16777216 の数字（単位：バイト）を指定します。	16777216
7	例外ログの出力レベル	com.cosminexus.jaxrs.logger.runtime.exception.level	例外ログの出力レベルを指定します。ERROR，WARN，INFO，DEBUG，NONE のどれかを指定します。各指定値に対応した出力内容については，「39.3.4 ログの重要度と出力条件」を参照してください。	INFO
8	例外ログの面数	com.cosminexus.jaxrs.logger.runtime.exception.file_num	例外ログの面数を指定します。1～16 の数字を指定します。	2
9	例外ログの容量	com.cosminexus.jaxrs.logger.runtime.exception.file_size	例外ログの容量を指定します。4096～16777216 の数字（単位：バイト）を指定します。	16777216
10	通信ログの出力レベル（Web リソース側）※1	com.cosminexus.jaxrs.logger.runtime.transport.server.level	Web リソース側での通信ログの出力レベルを指定します。※2	NONE

項番	設定項目	キー名称	指定内容	デフォルト値
			NONE を指定した場合 　通信ログが出力されません。 ALL を指定した場合 　送受信した HTTP ヘッダとエンティティボディの両方が常に通信ログに出力されます。 HEADER を指定した場合 　送受信したメッセージの HTTP ヘッダが常に通信ログに出力されます。	
11	通信ログの出力レベル（Web リソースクライアント側）	com.cosminexus.jaxrs.logger.runtime.transport.client.level	Web リソースクライアント側での通信ログの出力レベルを指定します。※2 NONE を指定した場合 　通信ログが出力されません。 ALL を指定した場合 　送受信した HTTP ヘッダとエンティティボディの両方が常に通信ログに出力されます。 HEADER を指定した場合 　送受信したメッセージの HTTP ヘッダが常に通信ログに出力されます。	NONE
12	通信ログの面数	com.cosminexus.jaxrs.logger.runtime.transport.file_num	通信ログの面数を指定します。1～16 の数字で指定します。	2
13	通信ログの容量	com.cosminexus.jaxrs.logger.runtime.transport.file_size	通信ログの容量を指定します。4096～16777216 の数字（単位：バイト）を指定します。	16777216
14	通信ログの文字エンコーディング	com.cosminexus.jaxrs.logger.runtime.transport.encoding	通信ログの文字エンコーディングを指定します。J2SE 6.0 でサポートされているエンコーディングについては，J2SE 6.0 のドキュメントを確認してください。 DEFAULT を指定した場合，通信ログのエンコーディングはデフォルトのプラットフォームエンコーディングとなります。	DEFAULT
15	WADL の発行の抑止※1	com.sun.jersey.config.feature.DisableWADL	WADL の発行を抑止するかどうかを指定します。※2 true を指定した場合 　WADL の発行を抑止します。 false を指定した場合 　WADL の発行を抑止しません。	false
16	JSON POJO マッピングの有効化※1，※3	com.sun.jersey.api.json.POJOMappingFeature	JSON POJO マッピングを有効にするかどうかを指定します。※2 true を指定した場合 　JSON POJO マッピングを有効にします。	false

項番	設定項目	キー名称	指定内容	デフォルト値
			false を指定した場合 　JSON POJO マッピングを有効にしません。	
17	自動リダイレクト※3	com.sun.jersey.client.property.followRedirects	HTTP リダイレクト（HTTP ステータスコードが 300 番台の要求）に自動的に従うべきかどうかを設定します。 true を指定した場合 　HTTP リダイレクトに自動的に従います。 false を指定した場合 　HTTP リダイレクトに自動的には従いません。 この項目を指定したときの動作は，Java SE の HttpURLConnection クラスの setInstanceFollowRedirects メソッドで引数にこの項目の設定値を指定して呼び出した場合と同じになります。※2	true
18	クライアントソケットの接続タイムアウト値※3	com.sun.jersey.client.property.connectTimeout	クライアントソケットの接続タイムアウト値を指定します。 このプロパティで指定したタイムアウト値は，Web リソース呼び出し時に適用されます。 0～2147483647 の数字（単位：ミリ秒）を指定します。0 を指定した場合，タイムアウトされません。 OS の TCP 接続に関連する設定を変更している場合，OS の設定値が優先されることがあります。	0
19	クライアントソケットの読み込みタイムアウト値※3	com.sun.jersey.client.property.readTimeout	クライアントソケットの読み込みタイムアウト値を指定します。 このプロパティで指定したタイムアウト値は，Web リソース呼び出し時に適用されます。 0～2147483647 の数字（単位：ミリ秒）を指定します。0 を指定した場合，タイムアウトされません。 OS の TCP 接続に関連する設定を変更している場合，OS の設定値が優先されることがあります。	0
20	チャンク転送エンコーディング※3	com.sun.jersey.client.property.chunkedEncodingSize	チャンク転送エンコーディングを使用するかどうかを，0～2147483647 の数字（単位：バイト）で指定します。0 を指定した場合，デフォルト値が適用されます。 この項目を指定したときの動作は，Java SE の HttpURLConnection クラスの setChunkedStreamingMode メソッドで引数にこの項目の設定値を指定して呼び出した場合と同じになります。	4096

項番	設定項目	キー名称	指定内容	デフォルト値
21	例外発生時のレスポンスエンティティのバッファリング※3	com.sun.jersey.client.property.bufferResponseEntityOnException	UniformInterfaceException が発生した場合，HTTP レスポンスにエンティティがあれば，自動的にバッファリングし，ストリームを閉じるかどうかを指定します。※2 true を指定した場合 HTTP レスポンスのエンティティを自動的にバッファリングし，ストリームを閉じます。 false を指定した場合 HTTP レスポンスのエンティティのストリームは自動的には閉じられません。	true

注※1

Web リソース側では，プロパティに指定した値より，サーブレット初期化パラメタに指定した値が優先されます。

注※2

プロパティの値は大文字と小文字とが区別されません。無効な値が指定されている場合は，デフォルト値と見なされます。

注※3

クライアント側では，プロパティに指定した値より，クライアント API で指定した値が優先されます。

(4)　設定を変更する場合

プロセス別の定義ファイルを使用していないすべての J2EE サーバを停止してから，共通定義ファイルの設定を変更してください。プロセス別の定義ファイルについては，「10.1.3　プロセス別の定義ファイルの設定」を参照してください。

ログ関連の設定を変更する場合，必要に応じてログを退避してから設定を変更してください。

13.1.3　プロセス別の定義ファイルの設定（JAX-RS）

プロセス別に固有の定義をする場合に，プロセス別の定義ファイルを作成します。

プロセス別の定義ファイルのファイル名，および保存先のディレクトリ名は任意です。システムプロパティで保存先のパスを指定することで，プロセス別の定義が有効になります。プロセス別の定義ファイルの指定例を次に示します。

```
com.cosminexus.jaxrs.confpath=d:/tmp/example.properties
```

プロセス別の定義を変更する場合，対象としているプロセス（J2EE アプリケーションまたは Java アプリケーション）を停止してから，プロセス別の定義ファイルの定義を変更してください。

ログ関連の定義を変更する場合，必要に応じてログを退避してから，定義を変更してください。

13.2 JAX-RS エンジンの動作

Application Server の JAX-RS エンジンの動作，および Application Server の JAX-RS エンジンのサポート範囲について説明します。

JAX-RS エンジンは RESTful Web サービス（Web リソース）の通信基盤となるエンジンです。Web リソースクライアントおよび Web リソースでの動作は次のとおりです。

- Web リソースクライアント側の JAX-RS エンジン

  Web リソースクライアントから RESTful Web サービス用クライアント API を介して Java オブジェクトを受け取り，HTTP リクエストを生成します。生成した HTTP リクエストは呼び出し先の Web リソースに送信します。また，Web リソースから HTTP レスポンスを受け取り，Java オブジェクトを生成します。生成した Java オブジェクトは Web リソースクライアントに返します。

- Web リソース側の JAX-RS エンジン

  HTTP リクエストを受け取り，対象となるリソースクラスを見つけ出し（ディスカバリ），そして要求に対応するメソッドを呼び出します（ディスパッチ）。ディスカバリとディスパッチの際には，リソースクラスが持つアノテーションに基づき，必要なインジェクションを実行します。また，JAX-RS エンジンは対象となるリソースクラスから HTTP レスポンスを生成して返します。

13.2.1 ディスカバリとディスパッチ

ここでは，Web リソースのディスカバリ，ディスパッチ，およびフォルトと例外クラスのマッピングについて説明します。

(1) ディスカバリ

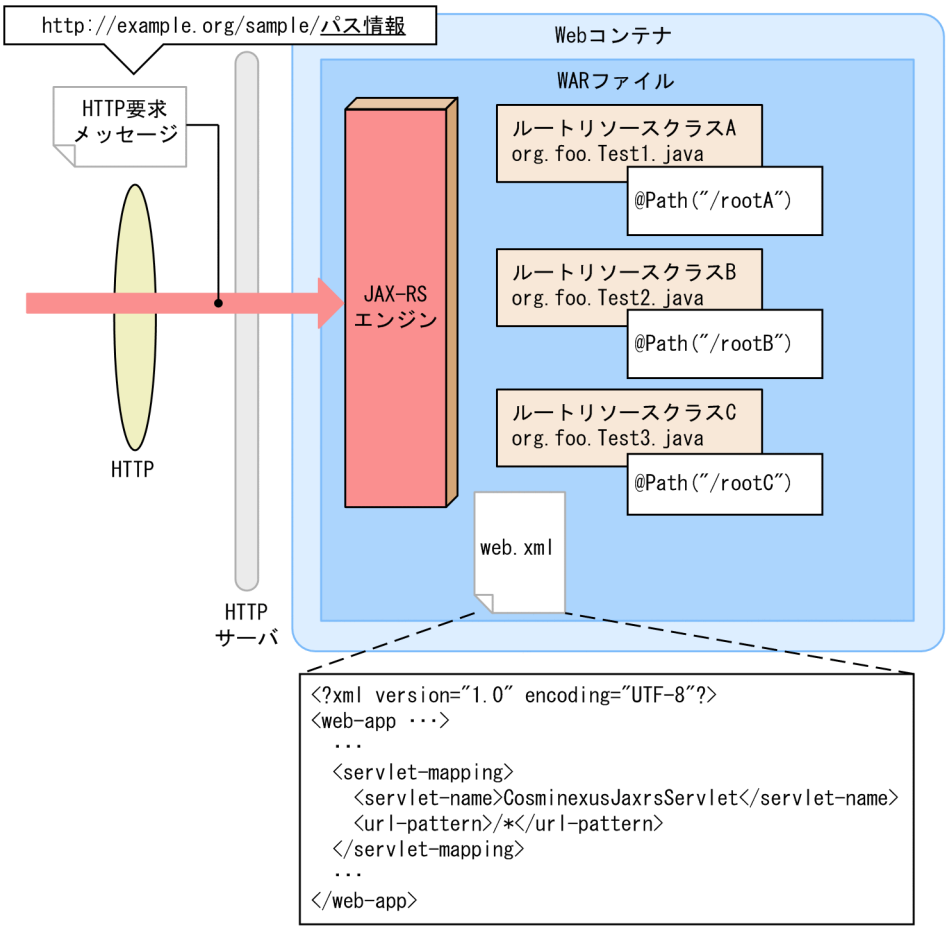
ディスカバリでは，HTTP リクエストで要求された URL から，リソースクラスをマッピングする処理が行われます。ここでは，次の URL が要求された場合のマッピングについて説明します。

http://example.org/sample/rootA

コンテキストルートを"sample"とした場合，コンテキストルートの後ろの"/rootA"（下線部）はパス情報を表します。このパス情報を基に，リソースクラスがマッピングされます。

パス情報とのマッピングの例を次に示します。なお，この例では，リソースクラスとして，サブリソースクラスを使用しないで，ルートリソースクラスだけを使用します。また，各ルートリソースクラスはリソースメソッドだけを持ちます。

図 13-1　ディスカバリ



JAX-RS エンジンは，デプロイされているルートリソースクラスのうち，Path アノテーションの値がパス情報に等しいものを呼び出します。

パス情報と，呼び出されるルートリソースクラスの対応を示します。

- パス情報が rootA の場合
  ルートリソースクラス A（org.foo.Test1.java）が呼び出されます。
- パス情報が rootB の場合
  ルートリソースクラス B（org.foo.Test2.java）が呼び出されます。
- パス情報が rootC の場合
  ルートリソースクラス C（org.foo.Test3.java）が呼び出されます。

なお，サブリソースクラスがデプロイされていて，対応するサブリソースロケータの Path アノテーションの値がパス情報に等しい場合は，そのサブリソースクラスを呼び出します。

(2)　HTTP メッセージのディスパッチ

JAX-RS エンジンは，受信した HTTP メッセージの内容（Content-Type HTTP ヘッダ）と受け入れできるメディアタイプ（Accept HTTP ヘッダ）に応じて，リソースメソッドを呼び出し，実行します。ディ

スパッチは Consumes アノテーションおよび Produces アノテーションで指定されたメディアタイプによって実行されます。

なお，リソースクラスにサブリソースメソッドが含まれている場合は，ディスカバリとディスパッチの判定が同時に行われます。

13.3　メタデータの発行

JAX-RS エンジンは，RESTful Web サービス（Web リソース）の WADL（メタデータ）を自動的に発行します。

ここでは，WADL の発行を利用するに当たって，注意が必要な点について説明します。

13.3.1　メタデータの発行条件

Web リソースの WADL を発行するメソッドは，次に示す二つです。

- HTTP GET メソッド
- HTTP OPTIONS メソッド

HTTP GET メソッドを使用して Web リソースの WADL を発行する条件を次の表に示します。JAX-RS エンジンが，次の表に示す条件をすべて満たした HTTP リクエストを受信したときに WADL が発行されます。

表 13–2　Web リソースのメタデータの発行に必要な HTTP リクエスト（GET）

項番	項目		条件
1	HTTP メソッド		GET メソッド
2	URL	スキーマ	http または https
3		ホスト名（:ポート番号）	メタデータの発行を要求する Web リソースが存在するホスト名（およびポート番号）
4		コンテキストパス	メタデータの発行を要求する Web リソースが含まれる Web アプリケーションのコンテキストパス
5		コンテキストパスの後ろのパス	"application.wadl"

例えば，Web リソースを含む Web アプリケーション（WAR ファイル）のコンテキストルートが，"sample" で，Web アプリケーションが"example.org"というホストで公開されていると仮定すると，URL は「http://example.org/sample/application.wadl」です。この場合，発行される WADL には，web.xml の com.sun.jersey.config.property.packages 初期化パラメタ（init-param 要素）に指定されたすべての Web リソースが含まれます。web.xml に com.sun.jersey.config.property.packages 初期化パラメタ（init-param 要素）が記述されていない場合は，WAR ファイルに含まれるすべての Web リソースが含まれます。web.xml の com.sun.jersey.config.property.packages 初期化パラメタ（init-param 要素）については「11.2　web.xml の作成」を参照してください。

Web リソースの WADL の例を次に示します。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://research.sun.com/wadl/2006/10">
```

```
 <doc xmlns:jersey="http://jersey.dev.java.net/" jersey:generatedBy="Cosminexus JAX-RS 09-00
"/>
 <resources base="http://example.org/sample/">
  <resource path="root">
   <method name="GET" id="resourceMethod">
    <response>
     <representation mediaType="*/*"/>
    </response>
   </method>
   <method name="POST" id="postHandler">
    <request>
     <representation mediaType="*/*">
      <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string"
       style="query" name="form"/>
     </representation>
    </request>
    <response>
     <representation mediaType="text/html"/>
    </response>
   </method>
   <resource path="subresourceMethod">
    <method name="GET" id="subResourceMethod">
     <request>
      <param xmlns:xs="http://www.w3.org/2001/XMLSchema" default="1"
       type="xs:string" style="matrix" name="matrix"/>
      <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string"
       name="cookie"/>
     </request>
     <response>
      <representation mediaType="*/*"/>
     </response>
    </method>
   </resource>
   <resource path="exception">
    <method name="GET" id="subResourceMethodThrowingException">
     <response>
      <representation mediaType="*/*"/>
     </response>
    </method>
   </resource>
   <resource path="subresourceLocator/{id}">
    <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string"
     style="template" name="id"/>
    <method name="GET" id="getHandlerForSubResource">
     <request>
      <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string"
       style="header" name="HeaderKey"/>
     </request>
     <response>
      <representation mediaType="*/*"/>
     </response>
    </method>
   </resource>
  </resource>
 </resources>
</application>
```

HTTP OPTIONS メソッドを使用して Web リソースのメタデータを発行する条件を次の表に示します。Web サービス側の JAX-RS エンジンが，次の表に示す条件をすべて満たした HTTP リクエストを受信したときにメタデータが発行されます。

表 13-3　Web リソースのメタデータの発行に必要な HTTP リクエスト（OPTIONS）

項番	項目		条件
1	HTTP メソッド		OPTIONS メソッド
2	URL	スキーマ	http
3		ホスト名（:ポート番号）	メタデータの発行を要求する Web リソースが存在するホスト名（およびポート番号）
4		コンテキストパス	メタデータの発行を要求する Web リソースが含まれる Web アプリケーションのコンテキストパス
5		Web リソースのパス	Web リソースで使われている Path アノテーションの値

例えば，Web リソース A の Path アノテーションの値が「/rootA」，ホスト名が「example.org」，コンテキストパスが「sample」だとします。この場合，メタデータを発行する URL は「http://example.org/sample/rootA」となります。

このとき，発行される WADL には，要求された Web リソースだけが含まれます。

ポイント

対象となる Web リソースのメソッドが OPTIONS アノテーションを持ち，HTTP OPTIONS 要求を処理できる場合，JAX-RS エンジンは WADL を発行しないで，その Web リソースの HTTP OPTIONS リクエストを処理するメソッドを呼び出します。Web リソースが HTTP OPTIONS リクエストを処理できなければ，JAX-RS エンジンは自動で WADL を生成します。

HTTP OPTIONS メソッドを使用した WADL の例を次に示します。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://research.sun.com/wadl/2006/10">
 <doc xmlns:jersey="http://jersey.dev.java.net/" jersey:generatedBy="Cosminexus JAX-RS 09-00
"/>
 <resources base="http://example.org/sample/">
  <resource path="/root">
   <method name="GET" id="resourceMethod">
    <response>
     <representation mediaType="*/*"/>
    </response>
   </method>
   <method name="POST" id="postHandler">
    <request>
     <representation mediaType="*/*">
      <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string"
       style="query" name="form"/>
     </representation>
```

```
    </request>
    <response>
     <representation mediaType="text/html"/>
    </response>
   </method>
   <resource path="subresourceMethod">
    <method name="GET" id="subResourceMethod">
     <request>
      <param xmlns:xs="http://www.w3.org/2001/XMLSchema" default="1"
       type="xs:string" style="matrix" name="matrix"/>
      <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string"
       name="cookie"/>
     </request>
     <response>
      <representation mediaType="*/*"/>
     </response>
    </method>
   </resource>
   <resource path="exception">
    <method name="GET" id="subResourceMethodThrowingException">
     <response>
      <representation mediaType="*/*"/>
     </response>
    </method>
   </resource>
   <resource path="subresourceLocator/{id}">
    <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string"
     style="template" name="id"/>
    <method name="GET" id="getHandlerForSubResource">
     <request>
      <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string"
       style="header" name="HeaderKey"/>
     </request>
     <response>
      <representation mediaType="*/*"/>
     </response>
    </method>
   </resource>
  </resource>
 </resources>
</application>
```

com.sun.jersey.config.feature.DisableWADL プロパティに true または false を設定することで，WADL の発行の有無を選択できます。このプロパティは次の個所に指定します。

- cjrconf.properties ファイル
- web.xml のサーブレット初期化パラメタ

どちらも指定された場合，サーブレット初期化パラメタが優先されます。

WADL の発行の有無を次の表に示します。

表 13–4　WADL の発行の有無

項番	サーブレット初期化パラメタ	値	JAX-RS エンジンの動作
1	com.sun.jersey.config.feature.DisableWADL	true	WADL を発行しません。 値の大文字と小文字を区別しません。
		false	WADL を発行します。 値の大文字と小文字を区別しません。
		true または false 以外の値	共通定義ファイル（cjrconf.properties）で取得したプロパティの値が使用されて，WADL を発行します。なお，web.xml で指定した値は無視されます。

13.4　プロキシサーバ経由の接続

Web リソースクライアントからプロキシサーバを経由して，外部のネットワークにある Web リソースを利用できます。

ここでは，プロキシサーバを経由して外部に接続する場合に必要なプロパティの設定について説明します。

13.4.1　プロパティ値の指定

プロキシサーバを経由して Web リソースにアクセスするには，JavaVM のプロパティを指定して，プロキシサーバの情報を設定します。プロキシサーバ経由で接続するための JavaVM のプロパティと指定内容を次の表に示します。なお，JavaVM のシステムプロパティの詳細については，JavaVM のドキュメントを参照してください。

表 13–5　プロキシサーバ経由で接続するための JavaVM のプロパティ

項番	JavaVM のプロパティ	指定内容	非 SSL の場合	SSL の場合
1	http.proxyHost	プロキシサーバのホスト名または IP アドレスを指定します。 空文字を指定した場合は，プロキシサーバに接続されません。	○	×
2	http.proxyPort	プロキシサーバのポート番号を設定します。 http.proxyHost が適切に設定されていた場合に，http.proxyPort に空文字を指定したときは，http.proxyHost に指定されているホストの 80 番ポートにアクセスされます。 http.proxyHost を指定していない場合は，http.proxyPort を指定しても，プロキシサーバに接続されません。	△	×
3	https.proxyHost	SSL プロトコルによる接続※で使用するプロキシサーバのホスト名か IP アドレスを設定します。 SSL プロトコルによる接続でプロキシサーバを使用する場合は，必ず設定してください。なお，空文字を指定した場合は，プロキシサーバに接続されません。	×	○
4	https.proxyPort	SSL プロトコルによる接続※で使用するプロキシサーバのポート番号を設定します。なお，https.proxyHost が適切に設定されていた場合に，https.proxyPort に空文字を指定したときは，https.proxyHost に指定されているホストの 443 番ポートにアクセスされます。 https.proxyHost を指定していない場合は，https.proxyPort を設定しても，プロキシサーバに接続されません。	×	△
5	http.nonProxyHosts	プロキシサーバを利用しないホスト名群を必要に応じて指定します。	△	△

項番	JavaVM のプロパティ	指定内容	非 SSL の場合	SSL の場合
		このプロパティで指定したホストに接続する場合は，http.proxyHost または https.proxyHost に指定したプロキシサーバは経由されません。ホストを複数指定する場合は，「|」で区切って設定します。ホスト名とホスト名の間には，「|」以外の文字（空白など）は指定できません。		

（凡例）

○：プロパティの指定が必須であることを示します。

△：必要に応じてプロパティを指定することを示します。

×：プロパティの指定が不要であることを示します。

注※

SSL プロトコルによる接続については，「10.11　SSL プロトコルによる接続」を参照してください。

13.4.2　プロパティの設定方法

JavaVM のシステムプロパティの設定方法については，「10.10.2　プロパティの設定方法」を参照してください。

13.4.3　匿名でないプロキシサーバを経由して接続する場合

匿名でないプロキシサーバを経由して Web リソースにアクセスする場合は，Web リソースクライアントで J2SE 6.0 標準の java.net.Authenticator クラスを利用して実装することをお勧めします。詳細については J2SE 6 のドキュメントを参照してください。

java.net.Authenticator クラスを利用した実装例については「10.10.3　JAX-WS 機能固有のプロパティを利用しない場合」を参照してください。

13.5　SSL プロトコルによる接続

Web リソースクライアントから，SSL プロトコルに対応した Web リソースに接続できます。

SSL プロトコルで Web リソースにアクセスするには，JDK にサポートされているプロパティに値を指定して，SSL プロトコルの情報を設定します。SSL プロトコルによる接続で使用するプロパティ，および指定内容を次の表に示します。

表 13-6　SSL プロトコルによる接続で使用するプロパティ

項番	プロパティ	指定内容
1	javax.net.ssl.trustStore	トラストストアを指定します。
2	javax.net.ssl.trustStorePassword	トラストストアのパスワードを指定します。

これらのプロパティは必要に応じて指定してください。なお，トラストストアを指定しない場合は，<JDK インストールディレクトリ>/lib/security/jssecacerts などのデフォルト値が使用されます。

JDK のプロパティについては，JDK のドキュメントを参照してください。プロパティの設定方法や注意事項については「10.11.2　プロパティの設定方法」および「10.11.3　ホスト名検証についての注意事項」を参照してください。

13.6　ベーシック認証による接続

Web リソースクライアントから，ベーシック認証に対応した Web リソースに接続できます。

ここでは，ベーシック認証による接続に必要な実装について説明します。

13.6.1　ベーシック認証による接続に必要な実装

ベーシック認証で Web リソースにアクセスするには，必要な HTTP ヘッダを追加する処理を実装してください。RESTful Web サービス用クライアント API を利用する場合の実装例を次に示します。

```
// ベーシック認証のユーザIDとパスワード
String username = ...
String password = ...
// Clientオブジェクトを生成する
Client client = Client.create();
// Authorization HTTPヘッダを持つHTTPリクエストを生成し
// Webリソースへポストする
client.resource( ”http://example.org/helloworld” )
 .header( HttpHeaders.AUTHORIZATION,
    ”Basic ” + encode(username + ”:” + password))
 .post( String.class, ”Some Request” );
...
String encode( String value ){
 String encoded;
 // Base64アルゴリズムでvalueパラメタをエンコードし
 // 結果をencodedパラメタに設定する
 ...
 return encoded;
}
```

13.7　トラブルシュート

ここでは，Web リソースおよび Web リソースクライアントのトラブルシュートで特に注意が必要な点について説明します。一般的な注意事項や，障害の種類や対策，ログについては，「39.　障害対策」を参照してください。

また，それぞれの具体的な説明については「17.　Web リソースとプロバイダ」および「25.　RESTful Web サービス用クライアント API のサポート範囲」を参照してください。

13.7.1　Web リソース初期化時の構文チェック（KDJJ20003-W と KDJJ10006-E）

ルートリソースクラスと例外マッピングプロバイダは，Web アプリケーション（WAR ファイル）に含まれる Web リソースが最初に呼び出される際に JAX-RS エンジンによって初期化されます。サブリソースクラスは対応するサブリソースロケータが初期化します。

ここでは初期化時の構文チェックで誤りが検出された場合について説明します。

(1)　重大な誤りがある場合

リソースクラスに public コンストラクタが一つも宣言されていない場合など，初期化を完了できないような重大な構文の誤りを検出したとき，エラーメッセージがログに出力されます（KDJJ10006-E※）。ルートリソースクラスと例外マッピングプロバイダでは HTTP ステータスコード 500 の HTTP レスポンスが返され，J2EE サーバに java.lang.RuntimeException がスローされます。J2EE サーバへの例外スローについては，「13.7.4　J2EE サーバへの例外のスロー」を参照してください。サブリソースクラスでは，例外マッピングプロバイダで処理可能な java.lang.RuntimeException がスローされます。

注※

追加の情報がある場合はほかのエラーメッセージも出力されます。

KDJJ10006-E のエラーメッセージには詳細情報（サブメッセージ）のリストが含まれます。サブメッセージは，それぞれ構文の誤りの具体的な内容です。サブメッセージを参照して構文の誤りを取り除いてください。KDJJ10006-E のエラーメッセージの注意事項を次に示します。

- 構文の誤りが複数ある場合は，すべての誤りを取り除いてください。
- Web アプリケーション（WAR ファイル）にルートリソースクラス，サブリソースクラス，または例外マッピングプロバイダが複数含まれている場合，どれか一つに対してでも KDJJ10006-E のエラーメッセージが出力されているとき，ほかのどのルートリソースクラス，サブリソースクラス，および例外マッピングプロバイダも利用できません。KDJJ10006-E のエラーメッセージが出力されなくなるまで，Web アプリケーション（WAR ファイル）に含まれるルートリソースクラス，サブリソースクラス，または例外マッピングプロバイダのすべてを確認してください。

(2)　軽微な誤りがある場合

HTTP GET リクエストを受け付けるリソースメソッドの戻り値が void である場合など，軽微な構文の誤りを検出したとき，ほかに重大な誤りがなければ警告メッセージがログに出力され（KDJJ20003-W），ほかに一つでも重大な誤りがあればエラーメッセージがログに出力されます（KDJJ10006-E）。

KDJJ10006-E のエラーメッセージがログに出力される場合については「13.7.1(1)　重大な誤りがある場合」を参照してください。

KDJJ20003-W の警告メッセージがログに出力される場合は，処理はそのまま続行され，軽微な構文の誤りが検出されたメソッドを除いて正常に初期化されます。軽微な構文の誤りが検出されたメソッドについては，KDJJ20003-W の警告メッセージが出力されなくなるまで，構文を確認してください。

13.7.2　受信した HTTP リクエストの処理で検出されるエラー

HTTP リクエストに対し，ディスパッチするリソースメソッドが一つもない場合など，HTTP リクエストを処理できないとき，例外マッピングプロバイダで処理できる javax.ws.rs.WebApplicationException またはその他の例外がスローされます。追加の情報がある場合はエラーメッセージも出力されます。

13.7.3　例外マッピングプロバイダで処理できる例外

例外マッピングプロバイダで処理できる例外がスローされる場合，例外マッピングプロバイダを作成し，Web アプリケーション（WAR ファイル）に適切に含めれば，その例外の送信する HTTP レスポンスへのマッピングをカスタマイズできます。例外マッピングプロバイダの処理については，「17.1.8　例外ハンドリング」を参照してください。

13.7.4　J2EE サーバへの例外のスロー

Web リソース側の JAX-RS エンジンは，JAX-RS 仕様が規定するサーブレットベースの仕組みをサポートしています。このため，JAX-RS エンジンで発生する例外は必要に応じて，Web リソースを含む Web アプリケーション（WAR ファイル）がデプロイされている J2EE サーバにもスローされます。

Web リソースの運用中に障害が発生した場合は，JAX-RS 機能のログファイルのほか，J2EE サーバのログも確認してください。

13.7.5　クライアント API の使用時に発生する例外（KDJJ18888-E）

クライアント API の使用中に ClientHandlerException または UniformInterfaceException が発生した場合，ログには KDJJ18888-E のエラーメッセージが出力されます。

14 コマンド

SOAP Web サービスに必要な SEI や JavaBean クラスは，cjwsimport コマンドで生成できます。また，hwsgen コマンドを使用すると，コンパイル済みの Java ソースから WSDL を生成できます。

この章では，cjwsimport コマンドおよび hwsgen コマンドの使用方法について説明します。

14.1.1 cjwsimport コマンド

cjwsimport コマンドは，JAX-WS 2.2 仕様で規定されているマッピング規則に従い，WSDL ファイルから Java にマッピングするコマンドです。cjwsimport コマンドを実行すると，WSDL ファイルから Web サービスおよび Web サービスクライアントの実装に必要な Java ソースが生成され，コンパイルされます。

ここでは，cjwsimport コマンドを実行するときの形式や指定内容について説明します。

(1) 形式

cjwsimport コマンドの指定形式を次に示します。

```
cjwsimport［オプション群］<WSDLファイルのURLまたはファイルパス>
```

指定例

- ローカルにある WSDL ファイルを相対パス（wsdl/input.wsdl）で指定
  cjwsimport -d generated wsdl/input.wsdl
- ローカルにある WSDL ファイルを絶対パス（/tmp/wsdl/input.wsdl）で指定
  cjwsimport -d generated /tmp/wsdl/input.wsdl
- ローカルにある WSDL ファイルを URL（file:///tmp/wsdl/input.wsdl）で指定
  cjwsimport -d generated file:///tmp/wsdl/input.wsdl
- リモートにある WSDL ファイルを URL（http://example.com:8080/fromjava/test?wsdl）で指定
  cjwsimport -d generated http://example.com:8080/fromjava/test?wsdl

cjwsimport コマンド実行時の注意

cjwsimport コマンドは，「%」，「&」および「^」の文字を含んだディレクトリをカレントディレクトリとして実行することはできません。「%」，「&」および「^」の文字を含んだディレクトリをカレントディレクトリとして実行した場合の動作は保証されません。

WSDL ファイル指定時の注意

- 引数には WSDL ファイルのファイルパス（相対パスまたは絶対パス）または WSDL への URL を一つ指定します。WSDL ファイル以外のファイルを指定した場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51200-E）。
- WSDL をファイルパスで指定する場合，「&」および「^」を含んだ文字をファイルパスに使用しないでください。使用した場合の動作は保証されません。また，空白を含むファイルパスを指定する場合，ファイルパス全体を引用符（"）で囲んでください。ファイルパス全体を引用符で囲まない場合の動作は保証されません。

- WSDL を URL で指定する場合は，RFC 2396 仕様で規定されている文字，および RFC 2396 仕様の規則に従った文字列を使用してください。さらに，文字列は RFC 2396 仕様の規則に従って，UTF-8 でパーセントエンコーディングする必要があります。RFC 2396 仕様の規則に従わない，またはエンコードしない文字や文字列を指定した場合の動作は保証されません。
- WSDL を jar プロトコルで指定することはできません。jar プロトコルで指定した場合の動作は保証されません。
- WSDL ファイルをファイルパスまたは WSDL への URL で指定する場合は，正しいファイルパスまたは URL を指定してください。WSDL のファイルパスまたは WSDL への URL の指定を誤り，WSDL ファイルが見つからない場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51180-E または KDJW51189-E）。
- 引数に指定する WSDL ファイルの拡張子は任意です（.wsdl 以外の拡張子を指定してもかまいません）。
- 大文字，小文字は区別されません。
- 指定する文字列の長さに制限はありません。ただし，OS の制限を超えた場合はエラーになります。
- アクセス権限がない WSDL ファイルを指定した場合，JDK のエラーとなり処理が終了されます。

(2) オプション一覧

cjwsimport コマンドのオプション群には，次の表に示すオプションを指定できます。

表 14–1 cjwsimport コマンドのオプション一覧

オプション	設定項目	指定内容
-d <ディレクトリ>	クラスファイルの出力先ディレクトリ	コンパイル済みクラスファイル（*.class）の出力先ディレクトリを指定します。 指定できる値については，表外の「-d オプション／-s オプション指定時の注意事項」を参照してください。 このオプションを指定しない場合は，カレントディレクトリに出力されます。
-keep	なし	ソースファイル（*.java）を生成する場合に指定します。
-s <ディレクトリ>	ソースファイルの出力先ディレクトリ	ソースファイル（*.java）を出力する場合の出力先ディレクトリを指定します。 指定できる値については，表外の「-d オプション／-s オプション指定時の注意事項」を参照してください。 出力先ディレクトリは，-d オプションおよび-s オプションの指定によって変わります。オプションの指定と出力先については，表 14-3 を参照してください。
-verbose	なし	コマンド実行時の詳細な処理経過を出力する場合に指定します。
-b <パス>	外部バインディングファイルのパス	外部バインディングファイルを使用する場合に，外部バインディングファイルのパスを指定します。 指定できる値については，表外の「-b オプション指定時の注意事項」を参照してください。
-p <パッケージ>	Java コードのパッケージ名	Java ソースのパッケージ名を指定します。

オプション	設定項目	指定内容
		このオプションを指定した場合，バインディング宣言で指定したパッケージ名のカスタマイズや，標準仕様で定義されたデフォルトのパッケージ名の生成アルゴリズムは上書きされます。
-generateService	Web サービス実装クラスの生成	Web サービス実装クラス（スケルトンクラス）を生成する場合に指定します。 生成されるファイルについては，「14.1.1(4)　生成されるファイル」を参照してください。
-help	なし	ヘルプを表示する場合に指定します。 このオプションを指定した場合，-version を除くすべてのオプションの指定が無視され，ヘルプが表示されて終了します。
-version	なし	バージョン情報を表示する場合に指定します。 このオプションを指定した場合，ほかのオプションの指定が無視され，バージョン情報が表示されて終了します。
-wsdllocation	javax.xml.ws.WebServiceClient アノテーションの，wsdlLocation 要素に設定する値	javax.xml.ws.WebServiceClient アノテーションの，wsdlLocation 要素に設定する値を指定します。
-catalog <ファイル>	カタログファイルのパス	カタログ機能を利用する場合に指定します。指定できる値については表外の「-catalog オプション指定時の注意事項」を参照してください。

ファイル生成時のディレクトリの作成

cjwsimport コマンドを実行すると，指定した出力先ディレクトリに，生成されるファイルのパッケージ名に対応するディレクトリが作成され，そのディレクトリにファイルが出力されます。

WSDL ファイル（test.wsdl）の「要求メッセージの wrapper 要素が参照する型」の名前空間 URI に，「http://example.com/sample」が記述されている場合のコマンド指定例および出力先（リクエスト bean の場合）を示します。

- コマンド指定例

  cjwsimport -d ./output -keep input/test.wsdl

- 出力先（リクエスト bean）

  リクエスト bean のコンパイル済みクラスファイルとソースファイルは，次のディレクトリに出力されます。

  ./output/com/example/sample/

オプションに指定できる値や，指定を省略した場合の動作など，オプションの指定値についての注意事項を次に示します。

オプション共通の注意事項

全オプション共通の注意事項を示します。

- オプション群と引数の指定順序は任意です。また，各オプションの指定順序も任意です。

- 引数を持つオプションは，必ず引数を指定してください。引数を指定しない場合，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51001-E）。
- -catalog オプション以外は，同じオプションを重複して指定した場合，最後に指定したオプションが有効になります。
- 大文字，小文字を区別します。
- 指定する文字列の長さに制限はありません。ただし，OS の制限を超えた場合はエラーになります。
- パスを指定するオプションには，「&」および「^」を含んだ文字列を使用しないでください。使用した場合の動作は保証されません。また，空白を含むパスを指定する場合，パス全体を引用符「"」で囲んでください。パス全体を引用符で囲まない場合の動作は保証されません。
- 指定できないオプションを指定した場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51001-E）。

-d オプション／-s オプション指定時の注意事項

-d オプションおよび-s オプションの指定値についての注意事項を示します。

- 指定値の大文字，小文字は区別されません。
- 指定した出力先ディレクトリがない場合，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51181-E）。
- 出力先ディレクトリに誤ってファイルを指定した場合，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51182-E）。
- アクセス権限がない WSDL ファイルを指定した場合，JDK のエラーとなり処理が終了されます。

-d／-s／-keep オプションの指定有無とファイルの出力先

-d オプション，-s オプション，および-keep オプションの指定によって，コンパイル済みクラスファイルと，ソースファイルの出力先ディレクトリが異なります。

オプションの指定有無とコンパイル済みクラスファイルの出力先ディレクトリを次の表に示します。

表 14–2　オプションの指定有無とコンパイル済みクラスファイルの出力先ディレクトリ

オプションの指定有無			ソースファイルの出力有無と出力先ディレクトリ
-d	-s	-keep	
○	−	−	-d オプションで指定したディレクトリに出力されます。
×	−	−	カレントディレクトリに出力されます。

（凡例）

○：オプションが指定されていることを示します。

×：オプションが指定されていないことを示します。

−：オプションの指定有無は，出力先ディレクトリに影響しないことを示します。

オプションの指定有無とソースファイルの出力先ディレクトリを次の表に示します。

表 14–3　オプションの指定有無とソースファイルの出力先ディレクトリ

オプションの指定有無			ソースファイルの出力有無と出力先ディレクトリ
-d	-s	-keep	
○	○	−	-s オプションで指定したディレクトリに出力されます。
○	×	○	-d オプションで指定したディレクトリに出力されます。
○	×	×	出力されません。
×	○	−	-s オプションで指定したディレクトリに出力されます。
×	×	○	カレントディレクトリに出力されます。
×	×	×	出力されません。

（凡例）

○：オプションが指定されていることを示します。

×：オプションが指定されていないことを示します。

−：オプションの指定有無は，出力先ディレクトリに影響しないことを示します。

-b オプション指定時の注意事項

-b オプションの指定値についての注意事項を示します。

- 指定値の大文字，小文字は区別されません。
- 外部バインディングファイルはファイルパスで指定してください。URL 形式で指定した場合の動作は保証されません。
- 指定した外部バインディングファイルがない場合，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51184-E）。
- 外部バインディングファイル以外のファイルを指定した場合，動作は保証されません。
- 誤ってディレクトリを指定した場合，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51185-E）。
- アクセス権限がない WSDL ファイルを指定した場合，JDK のエラーとなり処理が終了されます。
- 指定した外部バインディングファイルがカスタマイズ対象とする WSDL ファイルと，cjwsimport コマンドの引数に指定したカスタマイズ対象の WSDL ファイルは同じファイルを指定してください。同じでない場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51190-E）。
- 外部バインディングファイルの拡張子の指定は任意です。

-p オプション指定時の注意事項

-p オプションの指定値についての注意事項を示します。

- パッケージ名は，半角英数字（0～9，A～Z，a～z），アンダースコア（_），およびピリオド（.）で指定します。それ以外の文字を指定した場合，動作は保証されません。

- "xxx.yyy.zzz"のようにピリオド（.）で区切られた各ラベル（"xxx"，"yyy"，"zzz"）は，Java の識別子に使用できる文字列を指定してください。使用できない文字を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます。

-wsdllocation オプション指定時の注意事項

- -wsdllocation オプションは，URI 形式で指定してください。それ以外の形式で指定した場合の動作は保証されません。
- -wsdllocation オプションで指定する値に jar プロトコルは指定できません。指定した場合の動作は保証されません。WSDL を示す URL として jar プロトコルを指定する場合は，サービスクラスの生成時に URL をパラメタに取るコンストラクタを使用してください。

-catalog オプション指定時の注意事項

-catalog オプションの指定値についての注意事項を示します。

- -catalog オプションを重複して指定した場合の動作は保証されません。
- 指定値の大文字，小文字は区別されません。
- カタログファイルはファイルパスで指定してください。また，指定形式は，java.io.File クラスの仕様に従います。URL 形式で指定した場合の動作は保証されません。
- カタログファイルのファイルパスは，半角英数字（0～9，A～Z，a～z），空白，ピリオド（.），アンダースコア（_），コロン（:），スラッシュ（/），および¥で指定します。それ以外の文字を指定した場合，動作は保証されません。
- カタログファイルは任意のファイル名を設定できます。
- 指定したカタログファイルが存在しない場合，標準エラー出力とログに警告メッセージが出力され，カタログ機能が無効な状態で処理が続行されます（KDJW51219-W）。
- カタログファイル以外のファイルを指定した場合，動作は保証されません。
- 誤ってディレクトリを指定した場合，標準エラー出力とログに警告メッセージが出力され，カタログ機能が無効な状態で処理が続行されます（KDJW51220-W）。
- サポートしていない構文で記述したカタログファイルを指定した場合，標準エラー出力とログに警告メッセージが出力され，カタログ機能が無効な状態で処理が続行されます（KDJW51221-W）。
- アクセス権限のないカタログファイルを指定した場合，JDK のエラーとなりカタログ機能が無効な状態で処理が続行されます。

無視される値に指定した場合の動作について

コマンドの指定値のうち，指定しても無視されるとしているものに値を指定した場合，あとの処理でエラーになる可能性があります。

(3) WSIMPORT_OPTS 環境変数の指定

WSIMPORT_OPTS 環境変数にオプション文字列を指定すると，cjwsimport コマンドを起動する java コマンドにオプションを追加できます。デフォルトの設定では，WSIMPORT_OPTS 環境変数には何も指定されていないので，必要に応じて任意の文字列を指定してください。

例えば，WSIMPORT_OPTS 環境変数を利用して，SSL 通信に必要な JDK のシステムプロパティを指定し，従来 HTTPS を利用しないと WSDL にアクセスできない WSDL に対しても，cjwsimport コマンドを実行できます。例を次に示します。

```
> set WSIMPORT_OPTS=-Djavax.net.ssl.trustStore=<トラストストア> -Djavax.net.ssl.trustStorePassword=<トラストストアのパスワード>
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" https://securehost:443/fromwsdl/TestJaxWsService?wsdl
```

(4) 生成されるファイル

cjwsimport コマンドの実行時に生成されるファイルを次の表に示します。

表 14-4 cjwsimport コマンドの生成ファイル一覧

項番	Java コード	内容	-generateService オプションによる出力	
			指定あり	指定なし
1	リクエスト bean クラス	要求メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。 指定した WSDL ファイルが wrapper スタイルでない場合は出力されません。	○	○
2	レスポンス bean クラス	応答メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。 指定した WSDL ファイルが wrapper スタイルでない場合は出力されません。	○	○
3	フォルト bean クラス	フォルトメッセージが参照する型に対応する JavaBean クラスです。 指定した WSDL ファイルでフォルトが定義されていない場合は出力されません。	○	○
4	ラッパ例外クラス	フォルト bean クラスのラッパ例外クラスです。	○	○
5	ObjectFactory クラス	JAXB 2.2 仕様のファクトリクラスです。	○	○
6	JAXB 2.2 仕様のそのほかのクラス	JAXB 2.2 仕様のそのほかのクラスです。XML Schema 仕様の構文で記述した各種要素，型に対応する Java インタフェースおよび Java クラスです。	○	○
7	package-info.java	パッケージ情報です。	○	○
8	SEI	サービスエンドポイントインタフェースです。	○	○
9	スケルトンクラス	SEI を実装（implements）したスケルトンクラスです。このクラスに実装を追加します。	○	×
10	サービスクラス	Web サービスにアクセスするためのクラスです。	×	○

（凡例）

○：ファイルが出力されることを示します。

×：ファイルが出力されないことを示します。

ファイル生成時の注意事項

生成されるファイルの出力先ディレクトリにスケルトンクラスと同名のファイルがある場合は，標準エラー出力とログに警告メッセージが出力されます（KDJW51203-W）。このとき，スケルトンクラスは出力されないで，処理は続行されます。

また，生成されるファイルの出力先ディレクトリに Web サービス実装クラス以外の同名のファイルがある場合，ファイルが上書きされます。

Javadoc のヘッダ情報の出力

生成されるファイルでは，Application Server に関する情報が，Javadoc としてヘッダ情報に出力されます。

(5) 処理中の動作

cjwsimport コマンドを実行すると，標準出力とログにコマンド実行を示すメッセージ（KDJW50001-I）が出力され，Java ソースの生成処理および Java ソースのコンパイル処理が行われます。それぞれの処理について，次に示します。

Java ソースの生成処理

Java ソースの生成開始時には，標準出力とログに生成開始を示すメッセージが出力されます（KDJW50004-I）。Java ソースの生成に失敗した場合，エラーの原因となったエラーメッセージが標準エラー出力とログに出力されます（KDJW50005-E）。

Java ソースのコンパイル処理

Java ソースのコンパイル開始時には，標準出力とログにコンパイル開始を示すメッセージが出力されます（KDJW50006-I）。Java ソースのコンパイルに失敗した場合は，エラーの原因となったエラーメッセージが標準エラー出力とログに出力されます（KDJW50007-E）。

(6) 終了コード

cjwsimport コマンドの終了コードを示します。

0：正常終了

処理の途中で続行できないエラーが検出されなければ，標準出力とログに終了したことを示すメッセージが表示され，処理が終了されます（KDJW50002-I）。

1：異常終了

処理の途中で続行できないエラーが一つでも検出された場合は，標準出力とログにエラーメッセージが表示され，処理が終了されます（KDJW50003-E）。異常終了時の対処については，「14.1.1(7) 異常終了時の対処」を参照してください。

処理の途中で続行できる軽微なエラーが検出された場合は，警告メッセージが出力され，処理が続行されます。

なお，設定した出力レベル（重要度）によって，ログが出力されない場合があります。ログの出力レベルの設定については，「10.1.2　共通定義ファイルの設定項目」を参照してください。

(7)　異常終了時の対処

cjwsimport コマンドの実行時に異常終了した場合，エラーメッセージが出力され，処理が終了されます。この場合，出力されたエラーの要因を取り除き，cjwsimport コマンドを再実行します。

複数のエラーがある場合でも，最初に検出されたエラーが表示されます。この場合，cjwsimport コマンドを繰り返し実行して，表示されたエラーの要因を一つずつ取り除いてください。

14.1.2　hwsgen コマンド

hwsgen コマンドは，サービス実装クラスのクラスファイルを基に，Web サービスのデプロイに必要な Java コード（JavaBean クラス）と，リソースファイル（WSDL および XSD）を生成するコマンドです。なお，サービス実装クラスには，サービス実装クラスが参照しているクラスや SEI も含むものとします。

ここでは，hwsgen コマンドを実行するときの形式や指定内容について説明します。

(1)　形式

hwsgen コマンドの指定形式を次に示します。

```
hwsgen［オプション群］サービス実装クラスの完全修飾名
```

指定例

- デプロイ前に WSDL を確認する場合

  hwsgen -wsdl -cp . com.example.UserInfoImpl
- 既存の Web サービスから Web サービス（サービス名が MyService）の Java コードおよびリソースファイルを生成する場合

  hwsgen -servicename {http://example.com/}MyService -cp . com.example.UserInfoImpl
- 既存の Web サービスから Web サービス（ポート名が MyServicePort）の Java コードおよびリソースファイルを生成する場合

  hwsgen -portname {http://example.com/}MyServicePort -cp . com.example.UserInfoImpl

hwsgen コマンド実行時の注意

hwsgen コマンドは，「&」および「^」の文字を含んだディレクトリをカレントディレクトリとして実行することはできません。「&」および「^」の文字を含んだディレクトリをカレントディレクトリとして実行した場合の動作は保証されません。

次に示すディレクトリ以下に，hwsgen コマンドが生成する以外のソースファイルが存在しないことを確認してください。

- ソースファイルを出力する場合：ソースファイルの出力先ディレクトリ
- ソースファイルを出力しない場合：hwsgen コマンドが使用する作業ディレクトリ

ソースファイルの出力先と作業ディレクトリについては，「-d／-s／-keep オプションの組み合わせと出力先ディレクトリ／作業ディレクトリ」を参照してください。

サービス実装クラス指定時の注意

- サービス実装クラスのソースファイルは，クラスファイルとは別のディレクトリに配置してください。ソースファイルがクラスファイルと同じディレクトリに配置されていると，エラーが発生するおそれがあります。
- 引数には，サービス実装クラスのクラスファイルを完全修飾名で指定します。
- クラスが有効に扱われたのかは，警告メッセージで確認してください。
- インナークラスであるサービス実装クラスは，引数に指定しないでください。指定した場合の動作は保証されません。
- 上記以外の SEI および Web サービス実装クラスに関する注意事項は，「16.1　Java から WSDL へのデフォルトマッピング」および「16.2　Java から WSDL へのマッピングのカスタマイズ」を参照してください。

(2)　オプション一覧

hwsgen コマンドのオプション群には，次の表に示すオプションを指定できます。

表 14–5　hwsgen コマンドのオプション一覧

オプション	設定項目	指定内容
-d <ディレクトリ>	コンパイル済みクラスファイルの出力先ディレクトリのパス	コンパイル済みクラスファイル（*.class）の出力先ディレクトリを指定します。 指定できる値については，表外の「-d／-s／-r オプション指定時の注意事項」を参照してください。 ほかのオプションが指定されている場合，このオプションで指定したディレクトリにコンパイル済みクラスファイル以外のファイルが出力されることがあります。詳細は，表外の「-d／-s／-keep オプションの組み合わせと出力先ディレクトリ／作業ディレクトリ」および「-d／-r／-servicename／-portname／-wsdl オプションの組み合わせと出力先ディレクトリ」を参照してください。
-s <ディレクトリ>	ソースファイルの出力先ディレクトリのパス	ソースファイル（*.java）を出力する場合の出力先ディレクトリを指定します。
-r <ディレクトリ>	リソースファイルの出力先ディレクトリのパス	リソースファイル（*.wsdl および*.xsd）を出力する場合の出力先ディレクトリを指定します。
-keep	なし	ソースファイル（*.java）を保持するかどうかを指定します。

オプション	設定項目	指定内容
-wsdl	なし	リソースファイル（*.wsdl および*.xsd）を出力するかどうかを指定します。
-servicename <サービス名>	サービス名	変更後のサービス名を指定します。
-portname <ポート名>	ポート名	変更後のポート名を指定します。
-classpath <クラスパス>	サービス実装クラスが含まれるクラスパス	引数に指定するサービス実装クラスが含まれるクラスパスを指定します。
-cp <クラスパス>		
-verbose	なし	コマンド実行時の詳細な処理経過を出力する場合に指定します。
-help	なし	ヘルプを表示する場合に指定します。 このオプションを指定した場合，-version を除くすべてのオプションの指定が無視され，ヘルプが表示されて終了します。
-version	なし	バージョン情報を表示する場合に指定します。 このオプションを指定した場合，ほかのオプションの指定が無視され，バージョン情報が表示されて終了します。

オプションに指定できる値や，指定を省略した場合の動作など，オプションの指定値についての注意事項を次に示します。

オプション共通の注意事項

全オプション共通の注意事項を示します。

- オプション群と引数の指定順序は任意です。また，各オプションの指定順序も任意です。
- 引数を持つオプションは，必ず引数を指定してください。
- 同じオプションを重複して指定した場合は，最後に指定したオプションが有効になります。ただし，不正な値を指定したオプションがある場合は，エラーが発生することがあります。
- 大文字，小文字は区別されます。
- 指定する文字列の長さに制限はありません。ただし，OS の制限を超えた場合はエラーになります。
- オプションに空白を含むパスを指定する場合，パス全体を引用符（"）で囲んでください。パス全体を引用符で囲まない場合の動作は保証されません。

-d／-s／-r オプション指定時の注意事項

-d オプション，-s オプション，および-r オプションの指定値についての注意事項を示します。

- 指定値の大文字，小文字は区別されません。
- -d オプションに指定する出力先ディレクトリのパスには，次の文字は使用しないでください。使用した場合の動作は保証されません。

  # % & ^ ; ` { } [ ]

- -r オプションに指定する出力先ディレクトリのパスには，次の文字は使用しないでください。使用した場合の動作は保証されません。

  % & ^ ` { } [ ]

- -s オプションに指定する出力先ディレクトリのパスには，次の文字は使用しないでください。使用した場合の動作は保証されません。

  & ^

- アクセス権限がないディレクトリを指定した場合，JDK のエラーとなり処理が終了されます。

-wsdl オプション指定時の注意事項

SOAP バインディングのバージョンは，hwsgen コマンドでは-wsdl オプションで次のように指定してください。記述例を次に示します。

- SOAP バインディングのバージョンが SOAP 1.1 の場合

```
hwsgen -wsdl -cp . com.example.UserInfoImpl
または
hwsgen -wsdl:soap1.1 -cp . com.example.UserInfoImpl
```

- SOAP バインディングのバージョンが SOAP 1.2 の場合

```
hwsgen -wsdl:Xsoap1.2 -extension -cp . com.example.UserInfoImpl
```

-servername オプションと-portname オプションは,hwsgen コマンドでは-wsdl オプションと組み合わせて使用してください。

-servicename オプション指定時の注意事項

-servicename オプションは，QName 形式で記述します。-servicename オプションの記述例を次に示します。

```
{名前空間URI}サービス名
```

名前空間

- 名前空間 URI は，括弧（{ }）で囲んでください。
- 名前空間 URI は，半角英数字で指定してください。半角英数字以外を指定した場合の動作は保証されません。

プロトコル

- -servicename オプションに記述する名前空間 URI として有効なのは，http://または urn:で始まるドメイン名だけです。https://や file://などで始まる名前空間 URI は，不正と見なされます。
- -servicename オプションに記述する名前空間 URI を相対パスで指定することはできません。

指定できない情報

-servicename オプションに記述する名前空間 URI には，クエリストリング，アンカー，ポート番号，ユーザ名，およびパスワードは記述できません。

記述できる文字列

区切り文字のスラッシュ（/）またはピリオド（.）で区切られたセグメントには，次の表に示すすべての条件を満たす文字列を記述できます。

表 14-6　名前空間に記述できる文字列の条件（-servicename オプション指定時）

項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z）だけを使用した文字列	http://鈴木.com/ http://133.145.224.19/ http://[1080:2C14;D30:BA04:275:806:270C:418A]/	動作は保証されません（エラーメッセージは出力されません）。
2	Java の予約語以外の文字列	http://xxx.com/abstract	動作は保証されません。
3	先頭が数字でない文字列	http://1xxx.com	

サービス名

- サービス名は，半角英数字およびアンダースコアで指定してください。半角英数字またはアンダースコア以外の文字を指定した場合の動作は保証されません。
- サービス名は，Java で規定された識別子の命名規則に従って指定することをお勧めします。指定したサービス名が Java で規定された識別子の命名規則に従っていない場合，cjwsimport コマンドを実行して Web サービスクライアントを開発するときにコンパイルエラーが発生します。

-portname オプション指定時の注意事項

-portname オプションは，QName 形式で記述します。-portname オプションの記述例を次に示します。

```
{名前空間URI}ポート名
```

名前空間

- 名前空間 URI は，括弧（{ }）で囲んでください。
- 名前空間 URI は，WSDL ファイルの service 要素と同じ名前空間 URI を指定してください。
- 名前空間 URI は，半角英数字で指定してください。半角英数字以外を指定した場合の動作は保証されません。

プロトコル

- -portname オプションに記述する名前空間 URI として有効なのは，http://または urn:で始まるドメイン名だけです。https://や file://などで始まる名前空間 URI は，不正と見なされます。
- -portname オプションに記述する名前空間 URI を相対パスで指定することはできません。

指定できない情報

-portname オプションに記述する名前空間 URI には，クエリストリング，アンカー，ポート番号，ユーザ名，およびパスワードは記述できません。

記述できる文字列

区切り文字のスラッシュ（/）またはピリオド（.）で区切られたセグメントには，次の表に示すすべての条件を満たす文字列を記述できます。

表 14-7　名前空間に記述できる文字列の条件（-portname オプション指定時）

項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z）だけを使用した文字列	http://鈴木.com/ http://133.145.224.19/ http://[1080:2C14;D30:BA04:275:806:270C:418A]/	動作は保証されません（エラーメッセージは出力されません）。
2	Java の予約語以外の文字列	http://xxx.com/abstract	動作は保証されません。
3	先頭が数字でない文字列	http://1xxx.com	

ポート名

ポート名は，半角英数字およびアンダースコアで指定してください。半角英数字またはアンダースコア以外の文字を指定した場合の動作は保証されません。

-classpath／-cp オプション指定時の注意事項

-classpath オプションおよび-cp オプションの指定値についての注意事項を示します。

- オプションを省略した場合は，環境変数 CLASSPATH がクラスパスとして使用されます。オプションを指定した場合は，環境変数 CLASSPATH は無視されます。
- 環境変数 CLASSPATH で指定した値はそのまま適用されるため，空白を含む場合でも，引用符（"）で囲む必要はありません。引用符で囲む形式で指定した場合の動作は保証されません。
- オプションを省略し，環境変数 CLASSPATH も指定していない場合は，カレントディレクトリがクラスパスとして使用されます。
- クラスパスは，相対パス，絶対パスのどちらでも指定できます。
- クラスパスとして JAR ファイルを指定することもできます。
- 複数のクラスパスを指定する場合は，クラスパスの間にセミコロンを記述してください。
- 指定するクラスパスには，次の文字は使用しないでください。使用した場合の動作は保証されません。

  # % & ^

-help／-version オプション指定時の注意事項

hwsgen コマンドのヘルプ情報およびバージョン情報に出力される"wsgen"は，"hwsgen"に読み替えてください。また，ヘルプ情報に出力されるオプションのうち，-inlineSchemas オプションと-Xnocompile オプションは非サポートです。

-d／-s／-keep オプションの組み合わせと出力先ディレクトリ／作業ディレクトリ

コンパイル済みクラスファイル（*.class）は，-d オプションが指定されている場合は-d オプションで指定したディレクトリに，-d オプションが指定されていない場合はカレントディレクトリに出力されます。

ソースファイル（.java）が出力されるかどうか，また，出力される場合の出力先ディレクトリ，および hwsgen コマンドが利用する作業ディレクトリは，オプションの組み合わせによって次の表のとおり異なります。なお，リソースファイルだけを利用したい場合は，ソースファイルを出力しなくてもかまいません。

表 14–8　ソースファイルの出力有無と出力先ディレクトリ

オプションの指定有無			ソースファイルの出力有無と出力先ディレクトリ／作業ディレクトリ
-d	-s	-keep	
○	○	○	-s オプションで指定したディレクトリが作業ディレクトリとして利用され，ソースファイルが出力されます。
○	○	×	
○	×	○	-d オプションで指定したディレクトリが作業ディレクトリとして利用され，ソースファイルが出力されます。
○	×	×	-d オプションで指定したディレクトリが作業ディレクトリとして利用されます。ソースファイルは出力されません。
×	○	○	--s オプションで指定したディレクトリが作業ディレクトリとして利用され，ソースファイルが出力されます。
×	○	×	
×	×	○	カレントディレクトリが作業ディレクトリとして利用され，ソースファイルが出力されます。
×	×	×	カレントディレクトリが作業ディレクトリとして利用されます。ソースファイルは出力されません。

（凡例）

○：オプションが指定されていることを示します。

×：オプションが指定されていないことを示します。

-d／-r／-servicename／-portname／-wsdl オプションの組み合わせと出力先ディレクトリ

リソースファイル（*.wsdl および*.xsd）が出力されるかどうか，また，出力される場合の出力先ディレクトリは，オプションの組み合わせによって次の表のとおり異なります。

表 14–9　リソースファイルの出力有無と出力先ディレクトリ

オプションと指定有無				出力有無	出力先
-d オプション	-r オプション	-servicename, -portname, オプション	-wsdl オプション		
指定あり	指定あり	指定あり	指定あり	○	-r オプションで指定したディレクトリ
			指定なし		

オプションと指定有無				出力有無	出力先
-d オプション	-r オプション	-servicename, -portname, オプション	-wsdl オプション		
		指定なし	指定あり		
			指定なし		
	指定なし	指定あり	指定あり	○	-d オプションで指定したディレクトリ
			指定なし		
		指定なし	指定あり	○	-d オプションで指定したディレクトリ
			指定なし	×	－
指定なし	指定あり	指定あり	指定あり	○	-r オプションで指定したディレクトリ
			指定なし		
		指定なし	指定あり		
			指定なし		
	指定なし	指定あり	指定あり	○	カレントディレクトリ
			指定なし		
		指定なし	指定あり	○	カレントディレクトリ
			指定なし	×	－

（凡例）

－：リソースファイルの出力がないため，該当しないことを示します。

○：リソースファイルが出力されることを示します。

×：リソースファイルが出力されないことを示します。

(3) 生成されるファイル

hwsgen コマンドの実行時に生成されるファイルを次の表に示します。

表 14–10 hwsgen コマンドの生成ファイル一覧

項番	生成ファイル	内容
1	リクエスト bean クラス	要求メッセージの JavaBeans クラスです。生成されるサービス実装クラスが wrapper 形式の場合に出力されます。
2	レスポンス bean クラス	応答メッセージの JavaBeans クラスです。生成されるサービス実装クラスが wrapper 形式の場合に出力されます。
3	フォルト bean クラス	フォルトに対応する JavaBeans クラスです。指定された Java コードにラッパ例外クラスが定義されていて，フォルト bean が存在しない場合に出力されます。

項番	生成ファイル	内容
4	WSDL	WSDL ファイルです。
5	XSD	XML スキーマ定義ファイルです。

ファイル生成時のディレクトリの作成

hwsgen コマンドを実行すると，指定した出力先ディレクトリに，生成されるファイルのパッケージ名に対応するディレクトリが作成され，そのディレクトリにファイルが出力されます。

指定例および出力先を次に示します。

- コマンド指定例

  hwsgen -d ./output -s ./output -keep -cp . com.example.UserInfoImpl

- 出力先

  hwsgen コマンドに指定したサービス実装クラスのクラスファイルに JavaBean クラスがある場合は，JavaBean クラスのソースファイルおよびコンパイル済みクラスファイルが次の jaxws サブパッケージに出力されます（パッケージ名をアノテーションでカスタマイズしている場合を除く）。

  ./output/com/example/jaxws/

また，リソースファイルの生成物は，hwsgen コマンドの引数で指定したディレクトリに出力されます。指定例および出力先を次に示します。

- コマンド指定例

  hwsgen -r ./output -cp . com.example.UserInfoImpl

- 出力先

  ./output/

ファイル生成時の注意事項

生成されるファイルの出力先ディレクトリに同名のファイルがある場合，ファイルが上書きされます。

Javadoc のヘッダ情報の出力

生成されるファイルでは，Application Server に関する情報が，Javadoc としてヘッダ情報に出力されます。

(4) 入力サービス実装クラスと出力リソースファイルの関係

入力サービス実装クラスと出力リソースファイルの関係を次の表に示します。

表 14-11　入力サービス実装クラスと出力リソースファイルの関係

入力サービス実装クラス	出力リソース			
	WSDL		XSD	
	ファイル数	ファイル名	ファイル数※1	ファイル名
SEI なし	1	wsdl:service 要素の name 属性値	1～N※2	wsdl:service 要素の name 属性値＋接尾辞（_schemaN）※2
SEI あり（サービス実装クラスと同じ名前空間）	1	wsdl:service 要素の name 属性値	1～N※2	wsdl:service 要素の name 属性値＋接尾辞（_schemaN）※2
SEI あり（サービス実装クラスと異なる名前空間）	2	• 抽象 WSDL ファイルの場合※3 wsdl:portType 要素の name 属性値 • 実装 WSDL ファイルの場合※4 wsdl:service 要素の name 属性値	1～N※2	wsdl:portType 要素の name 属性値＋接尾辞（_schemaN）※2

注※1

スキーマの名前空間が異なると，そのたびにファイルが生成されます。

注※2

N はスキーマの名前空間の数です。生成できるファイルの上限数は，OS に依存します。

注※3

抽象 WSDL とは，「wsdl:types 要素, wsdl:message 要素, wsdl:portType 要素」の WSDL を指します。

注※4

実装 WSDL とは，「wsdl:binding 要素，wsdl:service 要素」の WSDL を指します。

(5)　処理中の動作

hwsgen コマンドを実行すると，JavaBeans の生成と削除，WSDL および XSD の生成が実行されます。ただし，オプションの指定内容によっては，生成した JavaBeans のソースファイルが削除されない場合もあります。オプションについては，「14.1.2(2)　オプション一覧」を参照してください。

(6)　終了コード

hwsgen コマンドの終了コードを示します。

0：正常終了

1：異常終了

- 処理の途中で続行できる軽微なエラーが検出された場合は，警告メッセージが出力され，処理が続行されます。
- 複数のエラーが検出された場合は，最初に検出されたエラーが標準出力とログに終了したことを示すメッセージが表示され，処理が終了されます。
- エラーが検出される前に生成されたディレクトリやファイルは，コマンドが異常終了しても削除されずに残ります。

(7) 異常終了時の対処

hwsgen コマンドの実行時に異常終了した場合，エラーメッセージが出力され，処理が終了されます。この場合，出力されたエラーの要因を取り除き，hwsgen コマンドを再実行します。

複数のエラーがある場合でも，最初に検出されたエラーが表示されます。この場合，hwsgen コマンドを繰り返し実行して，表示されたエラーの要因を一つずつ取り除いてください。

なお，サービス実装クラスのクラスファイルの不正が原因で異常終了した場合は，クラスファイルの生成元である Java ソースを修正し，コンパイルし直す必要があります。

14.2 UAC が有効な Windows でコマンドラインインタフェースを使用する場合の注意事項

OS が Windows で UAC（ユーザアカウント制御）が有効な場合に，cjwsimport コマンドおよび hwsgen コマンドを実行するときの注意事項を説明します。

14.2.1 管理者がコマンドラインインタフェースを使用する場合

管理者がコマンドラインインタフェースを使用する場合，インストール時の注意事項はありません。

インストール後にコマンドラインインタフェースを実行するときには，管理者としてコマンドプロンプトを起動する必要があります。権限を管理者に昇格させてコマンドプロンプトを起動する方法については，OS のドキュメントを参照してください。

14.2.2 管理者以外がコマンドラインインタフェースを使用する場合

管理者以外がコマンドラインインタフェースを使用する場合の注意事項を説明します。

(1) インストール時の注意事項

管理者以外がコマンドラインインタフェースを使用する場合は，Application Server をデフォルトのインストールディレクトリにインストールしてください。デフォルトのインストールディレクトリ以外のディレクトリにインストールした場合，コマンドラインインタフェースのすべてのログ出力先ディレクトリに対して，コマンドラインインタフェースを実行する管理者以外のユーザも書き込みできるようにアクセス権を設定する必要があります。アクセス権の設定は，管理者が実行します。設定方法については，OS のドキュメントを参照してください。

(2) コマンド実行時の注意事項

管理者以外のユーザがコマンドラインインタフェースを実行する場合の注意事項を示します。

- カレントディレクトリは，UAC の保護対象外のディレクトリを指定します。
- オプション設定で生成するファイルの出力先には，UAC の保護対象外のディレクトリを指定します。
- Application Server がデフォルトのインストールディレクトリにインストールされている場合，コマンドラインインタフェースのログは，次のディレクトリにリダイレクトされます。
  %LoadlAppData%¥VirtualStore¥Program Files ディレクトリ以下の対応するディレクトリ
  リダイレクトについては，OS のドキュメントを参照してください。

15 WSDL から Java へのマッピング

cjwsimport コマンドを実行すると，JAX-WS 2.2 仕様に従って WSDL から Java ソースへマッピングされます。

この章では，WSDL から Java へのデフォルトマッピング，およびマッピングのカスタマイズについて説明します。

15.1　WSDL から Java へのデフォルトマッピング

cjwsimport コマンドの実行時に，WSDL から Java ソースへマッピングされます。このときの対応関係を次の表に示します。

表 15–1　WSDL から Java ソースへのマッピング一覧

項番	WSDL	Java ソース	参照先
1	名前空間	パッケージ名	15.1.1
2	ポートタイプ	SEI 名	15.1.2
3	オペレーション	メソッド名	15.1.3
4	パート	パラメタおよび戻り値	15.1.4，15.1.5
5	型	パラメタおよび戻り値	15.1.6
6	フォルト	例外クラス	15.1.7
7	バインディング	javax.jws.soap.SOAPBinding アノテーション	15.1.8
8	サービス	javax.jws.WebService アノテーションの serviceName 属性	15.1.9

15.1.1　名前空間からパッケージ名へのマッピング

WSDL の名前空間（wsdl:definitions 要素の targetNamespace 属性）からパッケージ名へのマッピングについて説明します。

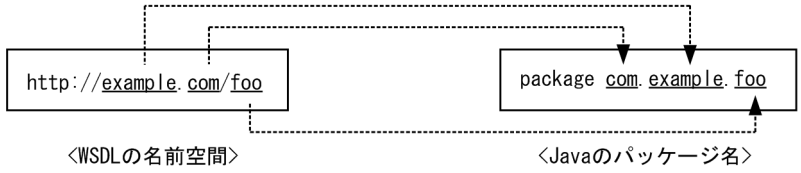
(1)　マッピング

WSDL の名前空間とパッケージ名は，JAX-WS 2.2 仕様に従ってマッピングされます。詳細は，JAX-WS 2.2 仕様を参照してください。

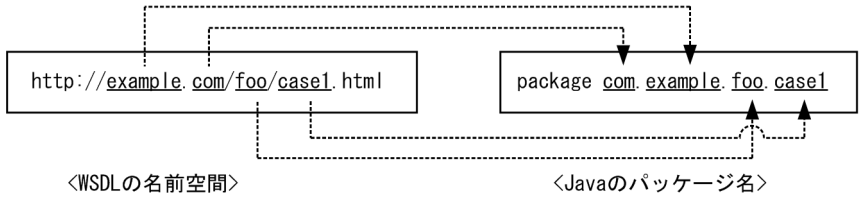
マッピング例を次の図に示します。

図 15–1　名前空間とパッケージ名のマッピング例

●名前空間をサーバ名とディレクトリ名で記述した場合



●名前空間をサーバ名，ディレクトリ名，およびファイル名で記述した場合



(2)　名前空間の条件

WSDL に記述する名前空間の条件について説明します。

- プロトコル

  名前空間は，http://または urn:のプロトコルで記述してください。http://または urn:のプロトコル以外（https://，file://など）を記述した場合，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51002-E）。

  また，相対パスで記述した場合，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51003-E）。

- 名前空間の記述形式

  名前空間には次に示す形式は記述できません。次に示す形式で記述した場合，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51004-E）。

  - 空文字列
  - クエリストリング（例）http://example.com/?a=b
  - アンカー（例）http://example.com/index.html#anchor
  - ポート番号（例）http://example.com:8080/
  - ユーザ名／パスワード（例）http://user:password@example.com

- 記述できる文字列

  区切り文字のスラッシュ（/）またはピリオド（.）で区切られたセグメントには，次の表に示すすべての条件を満たす文字列を記述できます。ただし，バインディング宣言でカスタマイズする場合は，RFC 2396 仕様の規則に従った文字列を記述できます。

表 15-2　名前空間に記述できる文字列の条件

項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z）だけを使用した文字列	http://鈴木.com	動作は保証されません（エラーメッセージは出力されません）。
2	Java の予約語以外の文字列	http://xxx.com/abstract	Java のパッケージ名にマッピングするときに，Java 予約語の先頭にアンダースコア（_）が付きます。 （例）com.xxx._abstract
3	先頭が数字でない文字列	http://1xxx.com	Java のパッケージ名にマッピングするときに，先頭が数字である文字列の先頭にアンダースコア（_）が付きます。 （例）com._1xxx

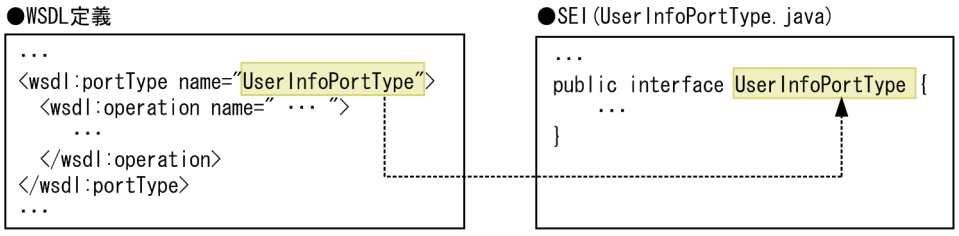
15.1.2　ポートタイプから SEI 名へのマッピング

WSDL のポートタイプ名（wsdl:portType 要素の name 属性）から SEI 名へのマッピングについて説明します。

(1)　マッピング

WSDL のポートタイプと SEI 名は，JAX-WS 2.2 仕様に従ってマッピングされます。マッピング例を次の図に示します。

図 15-2　ポートタイプと SEI 名のマッピング例



マッピングするときに，WSDL のポートタイプ名の先頭文字は大文字に変換されます。変換例を次に示します。

（変換前）portTypeName　　（変換後）PortTypeName

(2)　ポートタイプ名の条件

WSDL のポートタイプ名と名前空間を指定する場合は，パッケージ名も含めて SEI 名が"javax.xml.ws.Provider"にならないようにする必要があります。そのため，ポートタイプ名には"Provider"または"provider"を指定しないでください。また，名前空間には"http://ws.xml.javax"を指定しないでください。

ポートタイプには，次の表に示すすべての条件を満たす文字列を記述できます。ただし，バインディング宣言でカスタマイズする場合は，XML Schema 仕様の xsd:NCName 型として使用できる文字列を記述できます。

表 15–3　ポートタイプに記述できる文字列の条件

項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z），およびアンダースコア（_）だけを使用した文字列	鈴木_portType	動作は保証されません（エラーメッセージは出力されません）。
2	先頭が数字でない文字列	1User_portType	標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51029-E）。

(3)　ポートタイプの記述個数

WSDL に記述できるポートタイプは，1～255 個です。ポートタイプの記述数と動作の対応を次の表に示します。

表 15–4　ポートタイプの記述数と動作の対応

項番	要素	記述数	不正な文字列を指定した場合の動作
1	wsdl:portType	0 個	標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51008-E）。
2		1～255 個	正常終了します。
3		256 個以上	標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51008-E）。

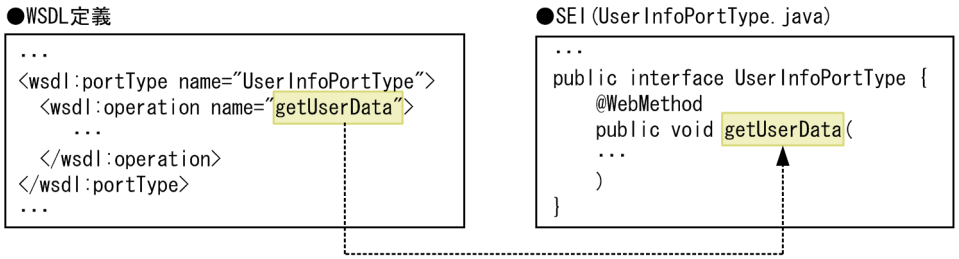
15.1.3　オペレーションからメソッド名へのマッピング

WSDL のオペレーション（wsdl:operation 要素の name 属性）から Java のメソッド名へのマッピングについて説明します。

(1)　マッピング

WSDL のオペレーションと Java のメソッド名は，JAX-WS 2.2 仕様に従ってマッピングされます。マッピング例を次の図に示します。

図 15-3　オペレーションとメソッド名のマッピング例



マッピングするときに，WSDL のオペレーション名の先頭文字は小文字に変換されます。get や set のプレフィクスは付加されません。変換例を次に示します。

（変換前）OperationName　　（変換後）operationName

(2)　オペレーション名の条件

オペレーション名には，次の表に示すすべての条件を満たす文字列を記述できます。ただし，バインディング宣言でカスタマイズする場合は，XML Schema 仕様の xsd:NCName 型として使用できる文字列を記述できます。

表 15-5　オペレーション名に記述できる文字列の条件

項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z），およびアンダースコア（_）だけを使用した文字列	鈴木_operation	動作は保証されません（エラーメッセージは出力されません）。
2	Java の予約語以外の文字列※	abstract	標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51007-E）。
3	先頭が数字以外の文字列	1User_operation	標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51029-E）。

注※

「Abstract」のように，Java 予約語の先頭文字を大文字にした文字列も記述できません（マッピングによって先頭文字が小文字に変換されるため）。

(3)　オペレーションとその子要素の記述個数

WSDL に記述できるオペレーションは，ポートタイプ 1 個につき 1～255 個です。また，オペレーションの子要素には，wsdl:input 要素を 1 個，wsdl:output 要素を 0 個または 1 個，および wsdl:fault 要素を 0～255 個記述できます。

オペレーションの記述数と，動作の対応を次の表に示します。

表 15–6　オペレーションの記述数と動作の対応

項番	要素	記述数	不正な文字列を指定した場合の動作
1	wsdl:operation	0 個	標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51029-E）。
2		1～255 個	正常終了します。
3		256 個以上	標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51029-E）。

オペレーションの子要素の記述数と，動作の対応を次の表に示します。

表 15–7　オペレーションの子要素の記述数と動作の対応

項番	要素	記述数	不正な文字列を指定した場合の動作
1	wsdl:input	0 個	標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51029-E）。
2		1 個	正常終了します。
3		2 個以上	標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51029-E）。
4	wsdl:output	0 個	wsdl:fault 要素の記述数が 0 個の場合 one-way オペレーションにマッピングして，正常終了します。 wsdl:fault 要素の記述数が 1 個以上の場合 標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51029-E）。
5		1 個	request-response オペレーションにマッピングして，正常終了します。
6		2 個以上	標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51029-E）。
7	wsdl:fault	0～255 個	正常終了します。
8		256 個以上	標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51029-E）。

15.1.4　メッセージのパートからパラメタおよび戻り値へのマッピング（wrapper スタイルの場合）

WSDL のメッセージのパート（wsdl:message 要素の wsdl:part 子要素）から Java のメソッドのパラメタおよび戻り値へのマッピングについて説明します。

ここでは，wrapper スタイルの場合について説明します。

- wrapper スタイルの条件について

wrapper スタイルは次に示す条件をすべて満たす場合に，wrapper スタイルとして扱われます。条件を満たさない場合は，non-wrapper スタイルとして扱われます。

- WSDL のオペレーションの soap:body 要素から参照する input メッセージは，パートを 1 個だけ含んでいる。

  パートを 2 個以上含んでいる場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51019-E）。
- WSDL のオペレーションから参照する output メッセージ（存在する場合）は，パートを 1 個だけ含んでいる。

  パートを 2 個以上含んでいる場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51020-E）。
- input メッセージのパートは，ローカル名がオペレーション名と等しいグローバル要素を参照している。
- output メッセージ（存在する場合）のパートは，グローバル要素を参照している。
- input メッセージと output メッセージ（存在する場合）のパートから参照されている要素の型は，xsd:sequence で定義した xsd:complexType である。
- wrapper 要素は子要素を含むだけであり，xsd:any 要素，xsd:anyAttribute 属性，xsd:choise 要素，substitutionGroup 属性，または attribute 要素のようなほかの構成要素を含まない。
- wrapper 要素は nillable でない。

(1) マッピング

- request-response オペレーションの WSDL からのマッピングの場合

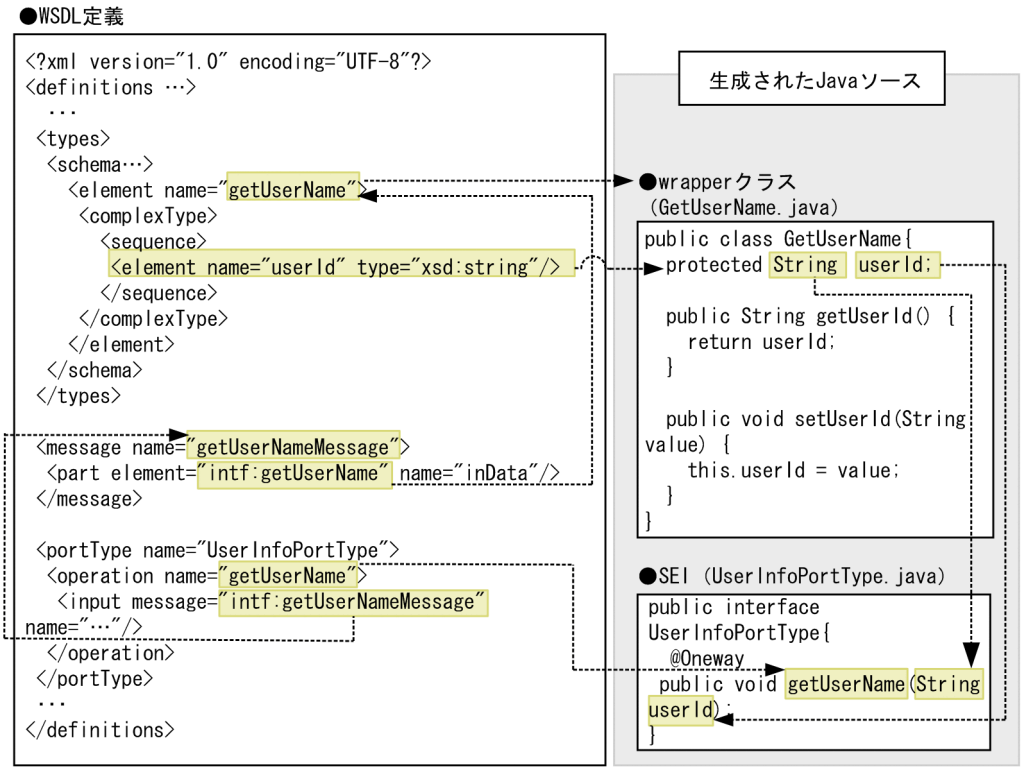
  request-response オペレーションの WSDL（wsdl:input 要素，および wsdl:output 要素を 1 個と，wsdl:fault 要素を 0 個以上定義する）のメッセージのパートから参照する wrapper 子要素と，Java メソッドのパラメタおよび戻り値がマッピングされます。マッピング例を次の図に示します。

図 15-4　メッセージのパートとパラメタおよび戻り値のマッピング例（request-response オペレーション）

- one-way オペレーションの WSDL からのマッピングの場合

  one-way オペレーションの WSDL（wsdl:input 要素を 1 個だけ定義）のメッセージのパートから参照する wrapper 子要素と，Java メソッドのパラメタがマッピングされます。マッピング例を次の図に示します。

図 15-5　メッセージのパートとパラメタのマッピング例（one-way オペレーション）



マッピングするときに，request-response オペレーションまたは one-way オペレーションのどちらの場合でも，WSDL の wrapper 子要素名の先頭文字は小文字に変換されます。

（変換前）WrapperName　　（変換後）wrapperName

- **パートの種類と Java ソースへのマッピングの関係**

  パートの種類（in，inout，out）と，Java ソースへのマッピングの関係を次の表に示します。

表 15-8　パートの種類と Java ソースへのマッピングの関係（wrapper スタイル）

項番	WSDL パートの種類	Java へのマッピング	
		マッピング先	マッピング方法
1	in	パラメタ	javax.xml.ws.Holder<T>クラスでマッピングされません。 java.lang.String などのクラスでマッピングされます。
2	inout	パラメタ	javax.xml.ws.Holder<T>クラスでマッピングされます。※
3	out	パラメタ	javax.xml.ws.Holder<T>クラスでマッピングされます。※
4		戻り値	javax.xml.ws.Holder<T>クラスでマッピングされません。 java.lang.String などのクラスでマッピングされます。

注※

wrapper 子要素の型が xsd:int など，JAXB 仕様で Java のプリミティブ型にマッピングされる型の場合，プリミティブに対応するため，送信時の Holder のインスタンスに null でない値を設定してください。

(2) wrapper 子要素名の条件

wrapper 子要素名には，次の表に示すすべての条件を満たす文字列を記述できます。ただし，バインディング宣言でカスタマイズする場合は，XML Schema 仕様の xsd:NCName 型として使用できる文字列を記述できます。

表 15–9 wrapper 子要素名に記述できる文字列の条件（wrapper スタイル）

項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z），およびアンダースコア（_）だけを使用した文字列	鈴木_wrapper	動作は保証されません（エラーメッセージは出力されません）。
2	Java の予約語以外の文字列※	abstract	標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51018-E）。
3	先頭が数字以外の文字列	1User_wrapper	標準エラー出力とログにエラーメッセージが出力され，処理が終了されます。

注※

「Abstract」のように，Java 予約語の先頭文字を大文字にした文字列も記述できません（マッピングによって先頭文字が小文字に変換されるため）。

(3) 複数の wrapper 子要素が同一の wrapper 子要素となる条件

input メッセージまたは output メッセージに出現する wrapper 子要素を WSDL 内に複数記述した場合，wrapper 子要素のローカル名と XML Schema の型が同じか異なるかによって，wrapper 子要素の扱いは次の表のとおり異なります。

表 15–10 wrapper 子要素のローカル名と XML Schema の型によって異なる wrapper 子要素の扱い

項番	wrapper 子要素のローカル名	wrapper 子要素の XML Schema 型	wrapper 子要素の扱い
1	ローカル名が同じ場合	XML Schema 型が同じ場合	同じ wrapper 子要素として扱われます。それぞれの wrapper 子要素が，xsd:element 要素の ref 属性で間接的に同じグローバル要素を参照している場合も，同じ wrapper 子要素として扱われます。
2		XML Schema 型が異なる場合	別の wrapper 子要素として扱われます。
3	ローカル名が異なる場合	XML Schema 型が同じ場合	別の wrapper 子要素として扱われます。
4		XML Schema 型が異なる場合	

(4) 複数の wrapper 子要素を記述した場合の注意事項

複数の同じ wrapper 子要素と，異なる wrapper 子要素を複合型の子要素として同時に WSDL ファイルに定義している場合，その WSDL ファイルを指定して cjwsimport コマンドに実行したときに，SEI が non-wrapper スタイルでマッピングされます。

SEI が non-wrapper スタイルでマッピングされる WSDL ファイルの例を次に示します。

```
<wsdl:definitions name="TestJaxWsService"
                  xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:tns="http://example.com/example"
                  targetNamespace="http://example.com/example">

   <xsd:element name="getUserData" type="tns:getUserData"/>
    ...
   <xsd:complexType name="getUserData">
     <xsd:sequence>
       <xsd:element name="in0"  type="xsd:string"/>
       <xsd:element name="in0"  type="xsd:string"/>
       <xsd:element name="hoge" type="xsd:string"/>
     </xsd:sequence>
   </xsd:complexType>
   ...
  </xsd:schema>
 </wsdl:types>
...
 <wsdl:message name="getUserDataRequest">
  <wsdl:part name="inputParameters" element="tns:getUserData"/>
 </wsdl:message>
 ...
</wsdl:definitions>
```

(5) パラメタへのマッピングの注意事項

- wrapper 子要素を Java へマッピングするときに，メソッドのパラメタの型は異なっていてもパラメタ名が同じになると，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます。
- in および inout の wrapper 子要素からマッピングされたパラメタは，wrapper 要素内の対応する wrapper 子要素の出現順序でマッピングされます。out の wrapper 子要素からマッピングされたパラメタは，wrapper 要素内の対応する wrapper 子要素の出現順序でマッピングされます。
- in，inout，out の wrapper 子要素が混在している場合，in と inout の wrapper 子要素が，wrapper 要素内の対応する wrapper 子要素の出現順序でマッピングされます。そのあとに out の wrapper 子要素が，wrapper 要素内の対応する wrapper 子要素の出現順序でマッピングされます。
- Java プリミティブ型，Java 配列型，ユーザ定義型である out（戻り値にマッピングされるものは除く），および inout のパラメタは，Java ソースでは Holder 型（javax.xml.ws.Holder<T>）にマッピングされます。その例を次に示します。

（例）

out および inout のパートのデータ型：java.lang.String

Java へのマッピング後のデータ型：javax.xml.ws.Holder<java.lang.String>

- Java へマッピング後のパラメタの数は，0～254 個で指定してください。255 個以上指定した場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51016-E）。

(6) 戻り値へのマッピングの注意事項

out の wrapper 子要素が 1 個の場合，または out の wrapper 子要素のローカル名が"return"の場合，その値がメソッドの戻り値へマッピングされます。ただし，型は異なっていてもローカル名が"return"である wrapper 子要素を複数記述した場合，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます。

15.1.5 メッセージのパートからパラメタおよび戻り値へのマッピング（non-wrapper スタイルの場合）

WSDL のメッセージのパート（wsdl:message 要素の wsdl:part 子要素）から Java のメソッドのパラメタおよび戻り値へのマッピングについて説明します。

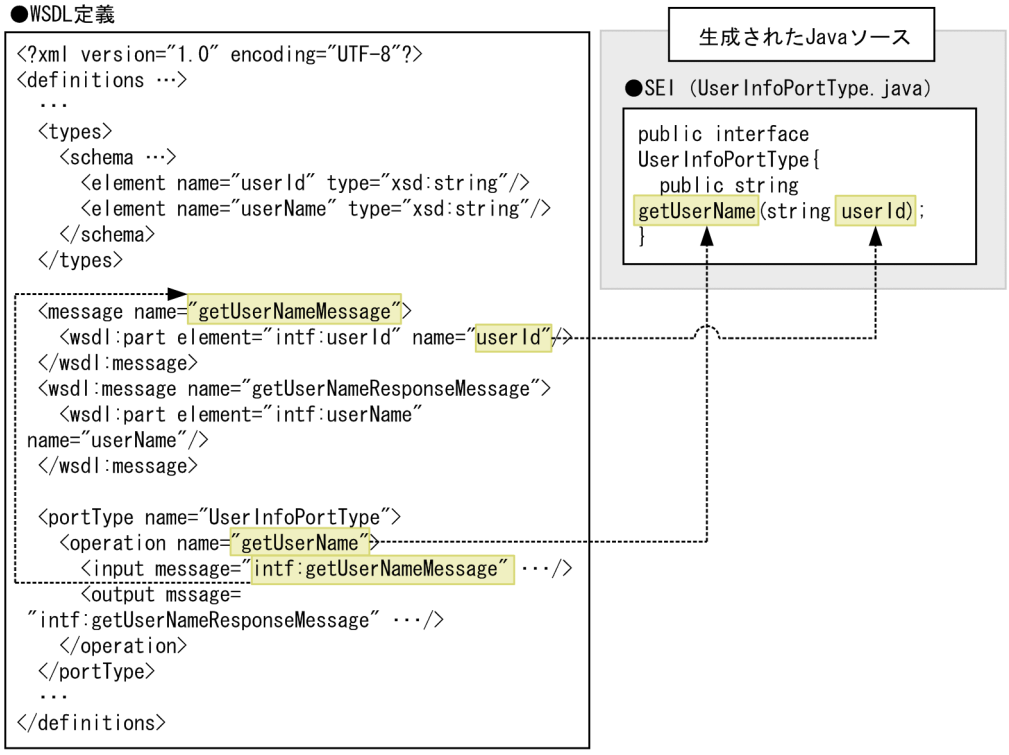
ここでは，non-wrapper スタイルの場合について説明します。

(1) マッピング

- request-response オペレーションの WSDL からのマッピングの場合

  request-response オペレーションの WSDL（wsdl:input 要素，および wsdl:output 要素を 1 個と，wsdl:fault 要素を 0 個以上定義）のメッセージのパートと，Java メソッドのパラメタおよび戻り値のマッピング例を次の図に示します。

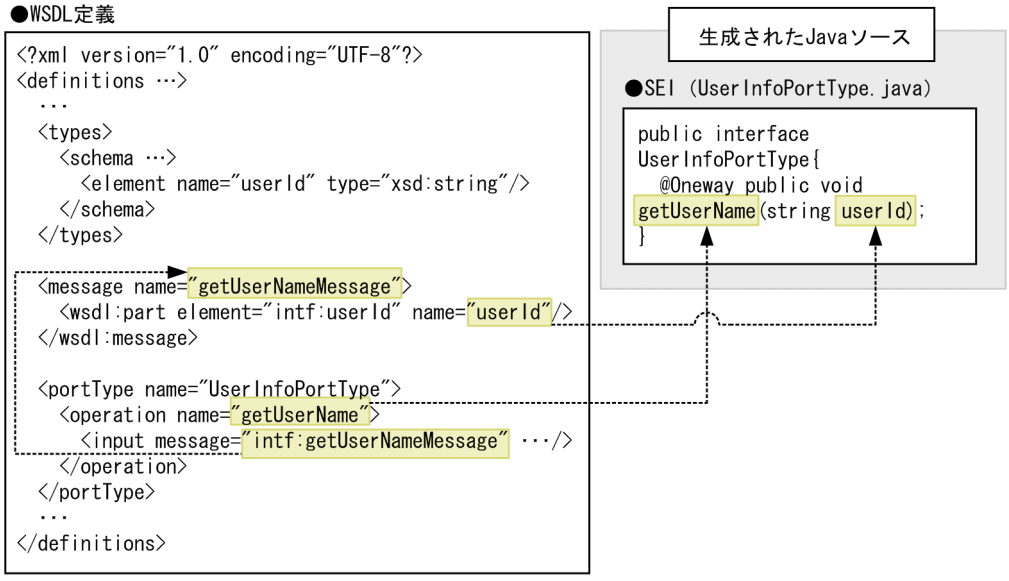
図 15-6　メッセージのパートとパラメタおよび戻り値のマッピング例（request-response オペレーション）



- one-way オペレーションの WSDL からのマッピングの場合

  one-way オペレーションの WSDL（wsdl:input 要素を 1 個だけ定義）のメッセージのパートと，Java メソッドのパラメタのマッピング例を次の図に示します。なお，マッピングした Java メソッドには，javax.jws.Oneway アノテーションを付与します。

図 15-7　メッセージのパートとパラメタのマッピング例（one-way オペレーション）



マッピングするときに，request-response オペレーションまたは one-way オペレーションのどちらの場合でも，メッセージのパート名（wsdl:part 要素の name 属性）の先頭文字は小文字に変換されます。

（変換前）PartName　　（変換後）partName

- **パートの種類と Java ソースへのマッピングの関係**

  パートの種類（in／inout／out）と，Java ソースへのマッピングの関係を次の表に示します。

表 15–11　パートの種類と Java ソースへのマッピングの関係（non-wrapper スタイル）

項番	WSDL パートの種類	Java へのマッピング	
		マッピング先	マッピング方法
1	in	パラメタ	javax.xml.ws.Holder<T>クラスでマッピングされません。 java.lang.String などのクラスでマッピングされます。
2	inout	パラメタ	javax.xml.ws.Holder<T>クラスでマッピングされます。
3	out	パラメタ	javax.xml.ws.Holder<T>クラスでマッピングされます。
4		戻り値	javax.xml.ws.Holder<T>クラスでマッピングされません。 java.lang.String などのクラスでマッピングされます。

(2)　パート名の条件

パート名には，次の表に示すすべての条件を満たす文字列を記述できます。ただし，バインディング宣言でカスタマイズする場合は，XML Schema 仕様の xsd:NCName 型として使用できる文字列を記述できます。

表 15–12　パート名に記述できる文字列の条件（non-wrapper スタイル）

項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z），およびアンダースコア（_）だけを使用した文字列	鈴木_part	動作は保証されません（エラーメッセージは出力されません）。
2	Java の予約語以外の文字列※	abstract	標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51017-E）。
3	先頭が数字以外の文字列	1User_part	標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51029-E）。

注※

「Abstract」のように，Java 予約語の先頭文字を大文字にした文字列も記述できません（マッピングによって先頭文字が小文字に変換されるため）。

(3)　複数のパートから同じグローバル要素を参照している場合の扱い

WSDL 内で input メッセージまたは output メッセージに出現するパートを複数記述した場合，パート名が同じで，かつ参照先のグローバル要素が同じときにだけ，同じパートとして扱われます。

パート名，参照先のグローバル要素のどちらかが異なる場合は，別のパートとして扱われます。

(4) パラメタへのマッピングの注意事項

- in および inout のパートは，input メッセージ内の対応するパートの出現順序でマッピングされます。out のパートは，output メッセージ内の対応するパートの出現順序でマッピングされます。
- in，inout，out のパートが混在している場合，in と inout のパートは input メッセージ内の対応するパートの出現順序でマッピングされ，そのあとに，out のパートが output メッセージ内の対応するパートの出現順序でマッピングされます。
- Java プリミティブ型や Java 配列型，ユーザ定義型である out（戻り値にマッピングされるものは除く）および inout のパートは，Java ソースでは Holder 型（javax.xml.ws.Holder<T>）にマッピングされます。その例を次に示します。

  （例）

  out および inout のパートのデータ型：java.lang.String

  Java へマッピング後のデータ型：javax.xml.ws.Holder<java.lang.String>

(5) 戻り値へのマッピングの注意事項

output メッセージで out のパートが 1 個だけの場合，メソッドの戻り値へマッピングされます。out のパートが 2 個以上の場合，戻り値にマッピングされません。

15.1.6 スキーマ型から Java 型へのマッピング

WSDL のスキーマ（wsdl:types 要素の xsd:schema 子要素）で定義した型から Java 型へのマッピングについて説明します。

(1) マッピング

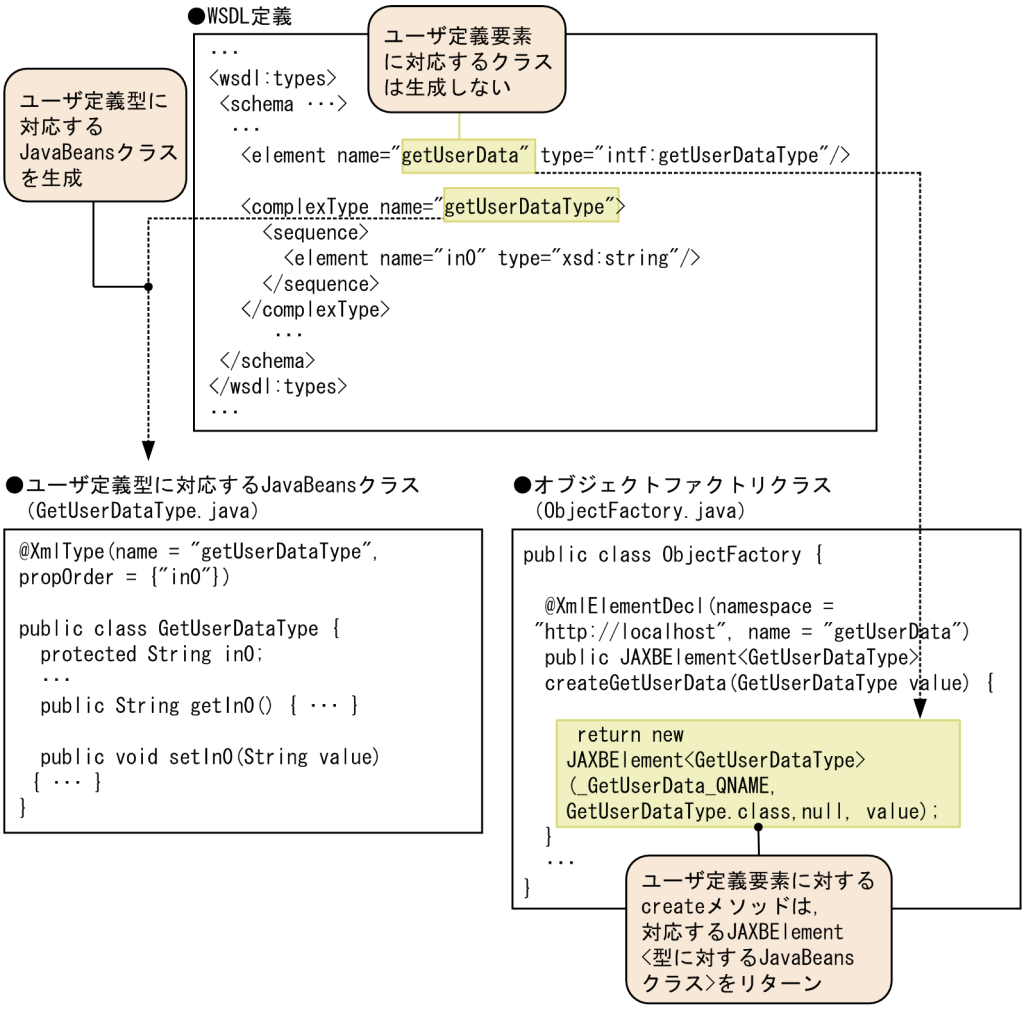
WSDL のスキーマの型と Java 型は，JAXB 2.2 仕様に従ってマッピングされます。

スキーマの型と Java 型のマッピングでは，クラスベースのマッピングが行われます。クラスベースのマッピングの動作を次に示します。

- ユーザ定義型に対応する JavaBeans クラスが生成されます。
- ユーザ定義要素に対応するクラスは生成されません。
- ObjectFactory クラスが出力されます。また，ユーザ定義要素に対する create メソッドでは，対応する JAXBElement<型に対する JavaBeans クラス>が返されます。

マッピング例を次の図に示します。

図 15-8　スキーマ型と Java 型のマッピング例



参考

クラスベースのマッピングとは

JAXB の globalBindings 要素の generateValueClass 属性が true で，かつ generateElementClass 属性が false の状態（指定なしのデフォルトの状態）と同等のマッピングを指します。

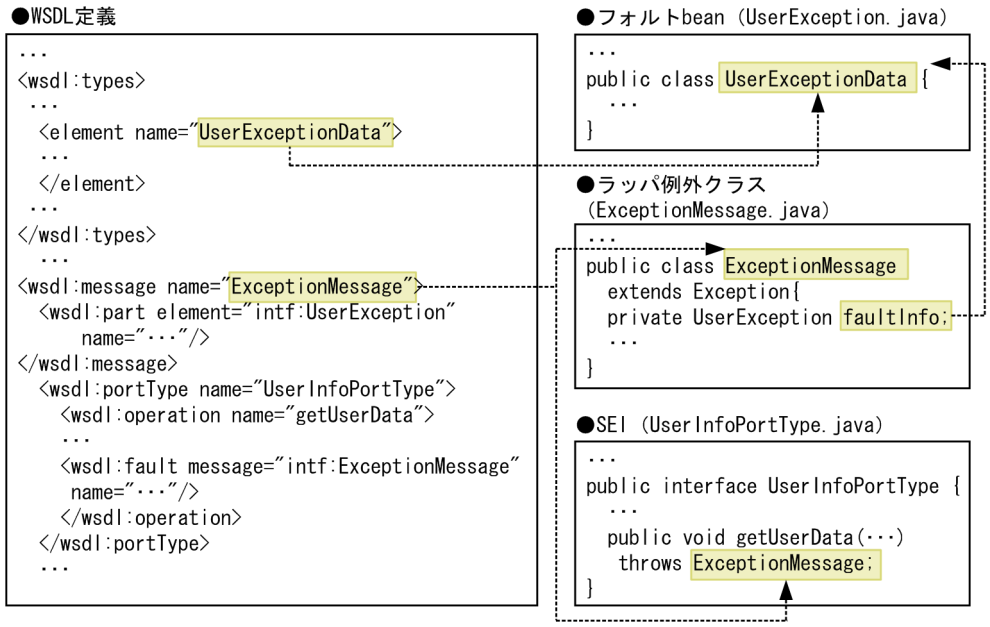
15.1.7　フォルトから例外クラスへのマッピング

WSDL のフォルト（wsdl:fault 要素から参照される wsdl:message 要素の name 属性）から例外クラスへのマッピングについて説明します。

(1) マッピング

cjwsimport コマンドを実行すると，WSDL のフォルトは，JAX-WS 2.2 仕様に従って Java 型にマッピングされます。マッピング例を次の図に示します。

図 15-9　フォルトと例外クラスのマッピング例



- フォルト bean へのマッピング

  JAX-WS 2.2 仕様に従ったフォルト bean が生成されます。フォルトのパートから参照されるグローバル要素宣言が，フォルト bean にマッピングされます。

- 生成されるラッパ例外クラス

  JAX-WS 2.2 仕様に従ったラッパ例外クラスが生成されます。生成されたラッパ例外クラスは，java.lang.Exception クラスを継承し，javax.xml.ws.WebFault アノテーションを持ちます。また，生成されたラッパ例外クラスは，次のメソッドを持ちます。

  - FaultMessageName(String message, FaultBean faultInfo)※のコンストラクタ

    引数として，メッセージ文字列とフォルト bean クラスを持ちます。また，このコンストラクタ中で親クラス javax.xml.ws.WebFault のコンストラクタが呼び出されます。

  - FaultMessageName(String message, FaultBean faultInfo, Throwable cause)※のコンストラクタ

    引数として，メッセージ文字列とフォルト bean クラス，およびプロトコル固有の例外情報を持ちます。また，このコンストラクタ中で親クラス javax.xml.ws.WebFault のコンストラクタが呼び出されます。

  - getFaultInfo()メソッド

    引数はありません。戻り値はフォルト bean クラスです。

注※

"FaultMessageName"は，フォルトから参照されるメッセージ名（wsdl:message 要素の name 属性）を表します。また，引数の"FaultBean"は，フォルト bean クラスの名前を表します。

(2) フォルト名の条件

フォルト名には，次の表に示すすべての条件を満たす文字列を記述できます。ただし，バインディング宣言でカスタマイズする場合は，XML Schema 仕様の xsd:NCName 型として使用できる文字列を記述できます。

表 15–13 フォルト名に記述できる文字列の条件

項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z），およびアンダースコア（_）だけを使用した文字列	鈴木_fault	動作は保証されません（エラーメッセージは出力されません）。
2	先頭が数字以外の文字列	1User_fault	標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51029-E）。

(3) フォルトから参照されるメッセージのパートの個数

フォルトは，パートが 1 個だけ記述されているメッセージを参照できます。フォルトから参照されるメッセージのパートの個数と動作を次の表に示します。

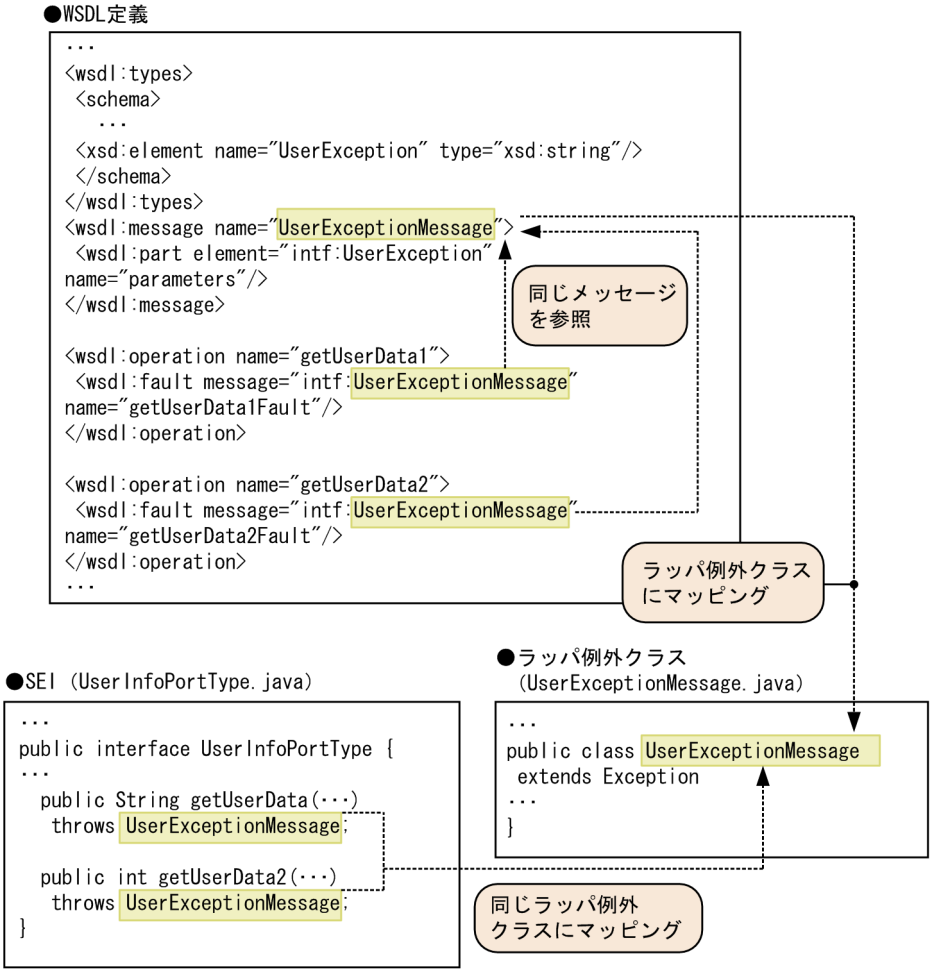
表 15–14 フォルトから参照されるメッセージのパートの個数と動作

項番	記述数	動作
1	0 個	標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51025-E）。
2	1 個	正常終了します。
3	2 個以上	標準エラー出力とログにエラーメッセージが出力され，処理が続行されます（KDJW51025-E）。

(4) 同じオペレーションのフォルトから同じメッセージを参照している場合の扱い

異なる複数のオペレーションのフォルトから同じメッセージを参照している場合，すべて同じフォルトとして扱われます。したがって，Java へマッピングするときに，共通のラッパ例外クラスとしてマッピングされます。その例を次の図に示します。

図 15-10　同じオペレーションのフォルトから同じメッセージを参照している場合のマッピング例



フォルトを記述しているオペレーションと，フォルトから参照されるメッセージの関係を次の表に示します。

表 15-15　フォルトを記述しているオペレーションと参照先のメッセージの関係

項番	フォルトを記述しているオペレーション	フォルトから参照されるメッセージ	フォルトの扱い
1	異なる	同じ	同じフォルトとして扱われます。
2		異なる	別のフォルトとして扱われます。
3	同じ	同じ	標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51026-E）。
4		異なる	別のフォルトとして扱われます。

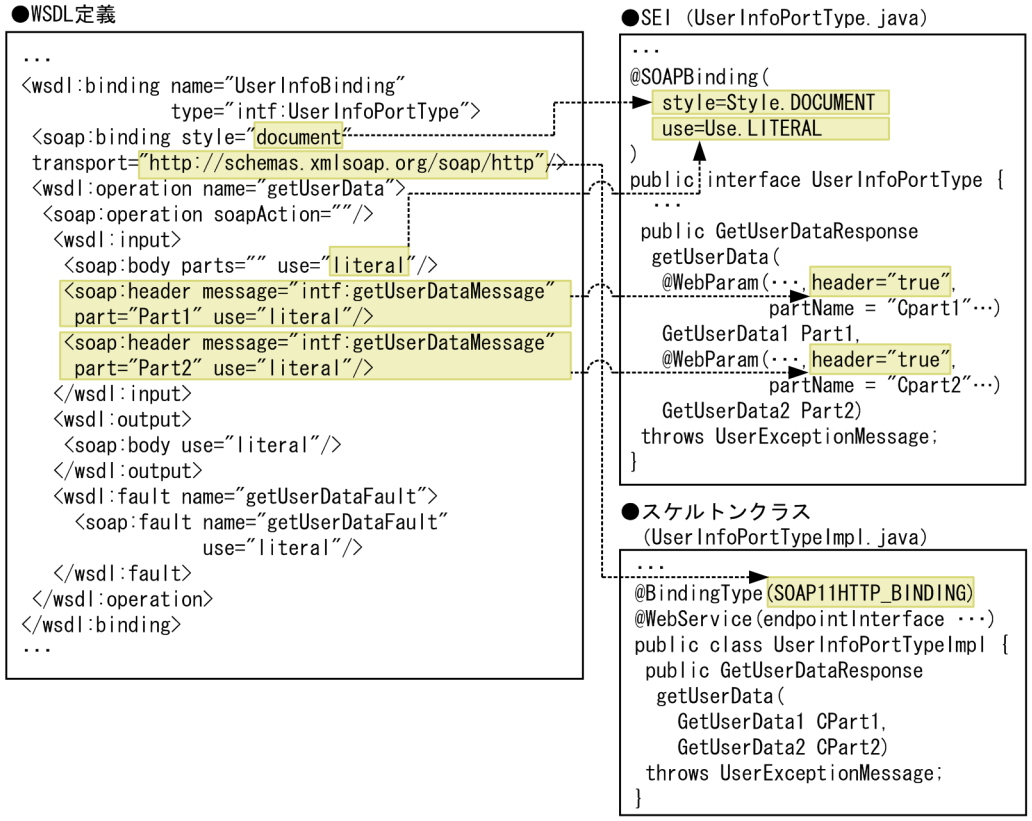
15.1.8　バインディング拡張要素からパラメタへのマッピング

WSDL のバインディングのバインディング拡張要素（wsdl:binding 要素）からメソッドのパラメタへのマッピングについて説明します。

(1) マッピング

WSDL のバインディングのバインディング拡張要素と Java メソッドのパラメタは，JAX-WS 2.2 仕様に従ってマッピングされます。マッピング例を次の図に示します。

図 15–11　バインディング拡張要素とパラメタのマッピング例



- SOAP バインディング

  バインディング拡張要素に，SOAP バインディングを記述できます。

  注意事項

  soap:header 要素を複数記述している場合，それぞれの soap:header 要素に対応するメッセージから参照されるグローバル要素のローカル名は，すべてユニークにしてください。ローカル名が同じ場合，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51205-E）。

  soap:binding または soap12:binding 要素の transport 属性値から javax.xml.ws.BindingType アノテーションへのマッピング

  WSDL の wsdl:binding 要素の子要素である soap:binding 要素または soap12:binding 要素の transport 属性値から，javax.xml.ws.BindingType アノテーションへのマッピングを次の表に示します。

表 15–16　transport 属性値から，javax.xml.ws.BindingType アノテーションへのマッピング

項番	SOAP バージョン	transport 属性値	BindingType アノテーションの値
1	SOAP 1.1	http://schemas.xmlsoap.org/soap/http	http://schemas.xmlsoap.org/soap/http※1
2		http://schemas.xmlsoap.org/wsdl/soap/http?mtom=true	－※2
3	SOAP 1.2	http://schemas.xmlsoap.org/soap/http※3	http://www.w3.org/2003/05/soap/bindings/HTTP/
4		http://www.w3.org/2003/05/soap/bindings/HTTP/	http://www.w3.org/2003/05/soap/bindings/HTTP/
5		http://www.w3.org/2003/05/soap/bindings/HTTP/?mtom=true	－※2

（凡例）

－：なし。

注※1

デフォルト値と同じ値なので，実際には javax.xml.ws.BindingType アノテーションが省略されます。

注※2

cjwsimport コマンド実行時にエラーになります（KDJW51147-E）。

注※3

transport 属性値については標準仕様であいまいなため，JAX-WS ではこの URL を使用できます。

- MIME バインディング

  MIME バインディングはサポートしていません。

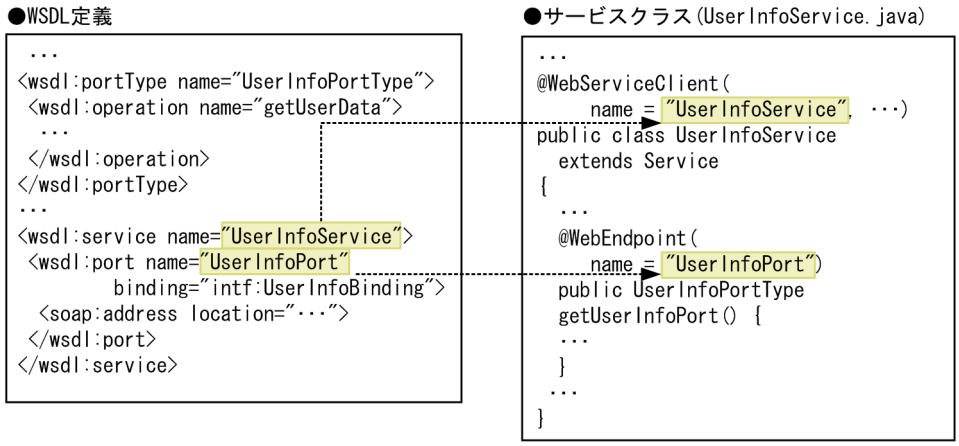
15.1.9　サービスおよびポートからサービスクラスへのマッピング

WSDL のサービス（wsdl:service 要素の name 属性）およびポート（wsdl:port 要素の name 属性）から，サービスクラスへのマッピングについて説明します。

(1)　マッピング

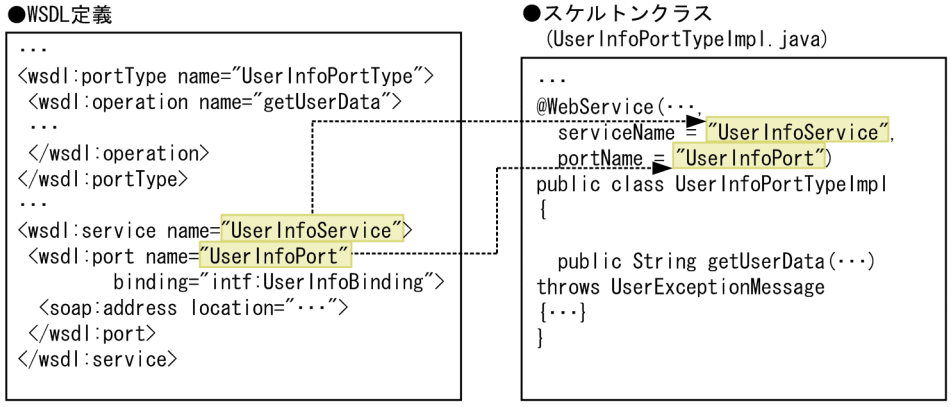
WSDL のサービスおよびポートと，サービスクラスは，JAX-WS 2.2 仕様に従ってマッピングされます。マッピング例を次の図に示します。

図 15-12　サービスおよびポートとサービスクラスのマッピング例



また，WSDL のサービスおよびポートは，スケルトンクラスにもマッピングされます。マッピング例を次の図に示します。

図 15-13　サービスおよびポートとスケルトンクラスのマッピング例



- 生成されるサービスクラス

  生成されるサービスクラスは，javax.xml.ws.Service クラスを継承し，javax.xml.ws.WebServiceClient アノテーションを持ちます。また，生成されるサービスクラスが持つメソッドを次に示します。

表 15-17　サービスクラスが持つメソッド

項番	戻り値の型	メソッド名／説明		サポート
1	−	ServiceName()※1		○
		説明	このコンストラクタ中で親クラス javax.xml.ws.Service※2 (java.net.URL wsdlDocumentLocation, javax.xml.namespace.QName serviceName)のコンストラクタが呼び出されます。 wsdlDocumentLocation および serviceName は，WSDL からマッピングされた定義を使用します。 カタログ機能が有効な場合は，WSDL からマッピングした定義を，カタログファイルで指定した別の URI にマッピングし，	

項番	戻り値の型	メソッド名／説明		サポート
			wsdlDocumentLocation として使用します。カタログ機能については「27. カタログ機能」を参照してください。	
		例外	javax.xml.ws.WebServiceException	
2	−	ServiceName(javax.xml.ws.WebServiceFeature... features)※1		×
3	−	ServiceName(java.net.URL wsdlLocation)※1		○
		説明	このコンストラクタ中で親クラス javax.xml.ws.Service※2 (java.net.URL wsdlDocumentLocation, javax.xml.namespace.QName serviceName)のコンストラクタが呼び出されます。 serviceName は，WSDL からマッピングされた定義を使用します。 カタログ機能が有効な場合は，この引数に指定した WSDL ロケーションを示す URL を，別の WSDL ロケーションを示す URI にマッピングします。カタログ機能については「27. カタログ機能」を参照してください。	
		引数	wsdlLocation： WSDL ロケーションを示す URL です。	
		例外	javax.xml.ws.WebServiceException	
4	−	ServiceName(java.net.URL wsdlLocation, javax.xml.ws.WebServiceFeature... features)※1		×
5	−	ServiceName(java.net.URL wsdlLocation, javax.xml.namespace.QName serviceName)※1		○
		説明	このコンストラクタ中で親クラス javax.xml.ws.Service※2 (java.net.URL wsdlDocumentLocation, javax.xml.namespace.QName serviceName)のコンストラクタが呼び出されます。	
		引数	wsdlLocation： WSDL ロケーションを示す URL です。カタログ機能が有効な場合は，この引数に指定した WSDL ロケーションを示す URL を，別の WSDL ロケーションを示す URI にマッピングします。カタログ機能については「27. カタログ機能」を参照してください。 serviceName： サービスの QName です。	
		例外	javax.xml.ws.WebServiceException	
6	−	ServiceName(java.net.URL wsdlLocation, javax.xml.namespace.QName serviceName, javax.xml.ws.WebServiceFeature... features)		×
7	SEI を実装するプロキシ	getPortName()※3		○

項番	戻り値の型	メソッド名／説明		サポート
		説明	親クラス javax.xml.ws.Service の getPort※2 (javax.xml.namespace.QName portName, java.lang.Class<T> serviceEndpointInterface)メソッドが呼び出されます。 portName および serviceEndpointInterface は，WSDL からマッピングされた定義を使用します。 このメソッドには javax.xml.ws.WebEndpoint アノテーションが付与されます。	
		例外	javax.xml.ws.WebServiceException	
8	SEI を実装するプロキシ	getPortName(javax.xml.ws.WebServiceFeature... features)※3		○
		説明	親クラス javax.xml.ws.Service の getPort※2 (javax.xml.namespace.QName portName, java.lang.Class<T> serviceEndpointInterface, WebServiceFeature... features)メソッドが呼び出されます。 portName および serviceEndpointInterface は，WSDL からマッピングされた定義を使用します。 このメソッドには javax.xml.ws.WebEndpoint アノテーションが付与されます。	
		引数	features： 　可変長の javax.xml.ws.WebServiceFeature 型です。	
		例外	javax.xml.ws.WebServiceException	

（凡例）

－：戻り値の型がないことを示します。

○：Application Server の JAX-WS 機能でサポートしています。

×：Application Server の JAX-WS 機能でサポートしていません。

注※1

"ServiceName"は，サービスクラス名（wsdl:service 要素の name 属性）を表します。

注※2

親クラス javax.xml.ws.Service については，「19.2.2(4)　javax.xml.ws.Service クラス」を参照してください。

注※3

"PortName"は，ポート名（wsdl:port 要素の name 属性）の先頭文字を大文字にした名前を表します。

(2)　サービス名およびポート名の条件

サービス名およびポート名には，次の表に示すすべての条件を満たす文字列を記述できます。ただし，バインディング宣言でカスタマイズする場合は，XML Schema 仕様の xsd:NCName 型として使用できる文字列を記述できます。

表 15-18　サービス名およびポート名に記述できる文字列の条件

項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z），およびアンダースコア（_）だけを使用した文字列	鈴木_service 鈴木_port	動作は保証されません（エラーメッセージは出力されません）。
2	先頭が数字以外の文字列	1User_service 1User_port	標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51029-E）。

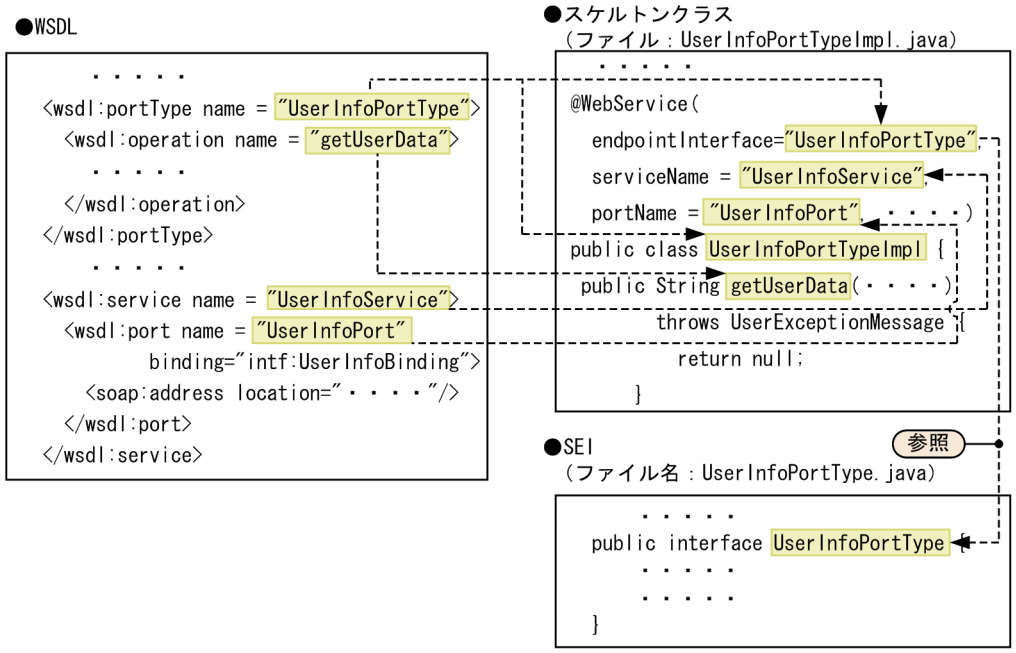
15.1.10　スケルトンクラスへのマッピング

SEI を実装したスケルトンクラスについて説明します。

(1)　マッピング

cjwsimport コマンドを使用すると，WSDL のサービスとポートからスケルトンクラスにマッピングできます。WSDL のサービスとポートからスケルトンクラスへのマッピング例を次の図に示します。

図 15-14　WSDL のサービスとポートからスケルトンクラスへのマッピング例



スケルトンクラスにマッピングする場合は，WSDL のポートタイプ名（wsdl:portType 要素の name 属性）に"Impl"サフィックスを付与します。

15.1.11　WSDL から Java へのマッピングに関する注意事項

WSDL から Java へのマッピングでの注意事項について説明します。

(1) Java メソッドのオーバーロード

1 個のポートタイプに複数のオペレーションを記述する場合，オペレーション名はすべてユニークでなければなりません。したがって，WSDL から Java へのマッピングでは，Java メソッドのオーバーロードはできません。オペレーション名が重複している場合はカスタマイズして，それぞれユニークな名称にしてください。

(2) 名前衝突時のマッピング

cjwsimport コマンドを実行するときに，SEI 名，クラス名，メソッド名，およびパラメタ名で名前衝突が発生することがあります。ここでは，名前衝突時のマッピングについて説明します。

(a) SEI 名およびクラス名の衝突時のマッピング

WSDL から Java ソースへのマッピングで，SEI 名およびクラス名（非例外 Java クラス名，例外クラス名，サービスクラス名，スケルトンクラス名）で名前が衝突した場合，優先順位に従って名前衝突が解決されます。

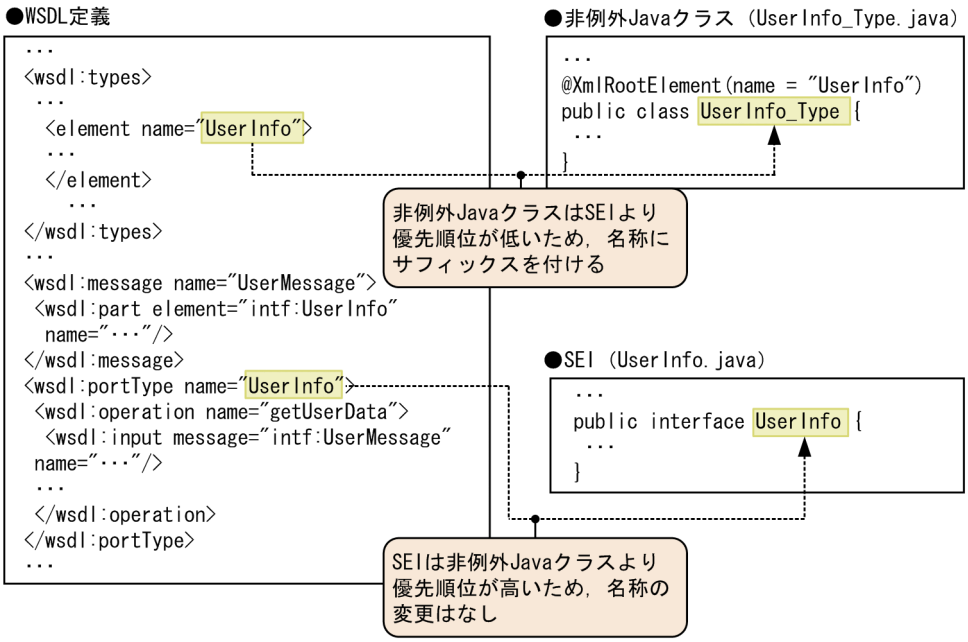
名前衝突時の優先順位と解決方法を次の表に示します。項番は優先順位を表します（項番 1 がいちばん高い）。

表 15-19 名前衝突の優先順位と解決方法

項番	種別	名前衝突時の解決方法
1	SEI 名	優先順位がいちばん高いため，名前は変更されません。
2	非例外 Java クラス名	クラス名に"_Type"サフィックスが付加されます。
3	例外クラス名	クラス名に"_Exception"サフィックスが付加されます。
4	サービスクラス名	クラス名に"_Service"サフィックスが付加されます。
5	スケルトンクラス名	クラス名に"_Impl"サフィックスが付加されます。

SEI 名と非例外 Java クラス名が衝突した場合の名前解決の例を示します。

図 15–15　SEI 名と非例外 Java クラス名の衝突時の名前解決例

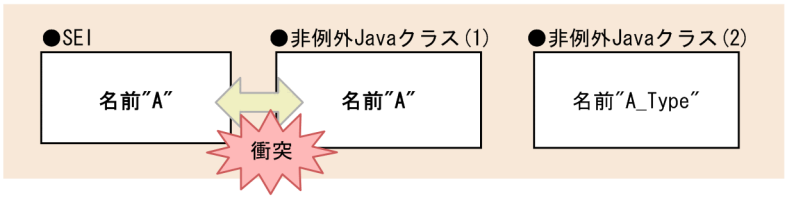


名前解決によってサフィックスが付加されたクラス名と，同名のクラス名が定義されていた場合は，再度，名前衝突が発生します。この場合，定義済みのクラス名からアンダースコアを削除することで解決します。

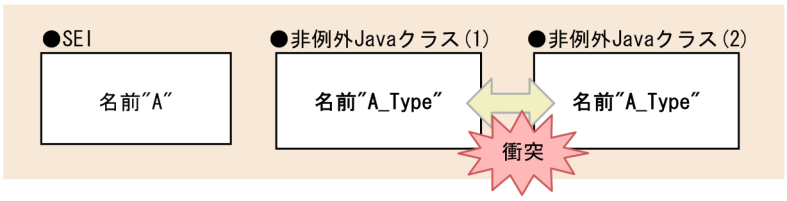
サフィックスが付加されたあとに，名前衝突する場合の名前解決例を示します。

図 15–16　サフィックス付加後に名前衝突する場合の名前解決例

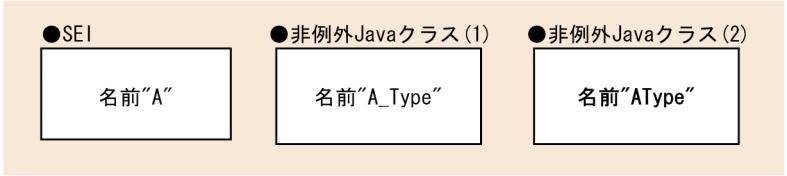
1. SEI，非例外Javaクラス(1)，非例外Javaクラス(2)を定義したが，SEIと非例外Javaクラス(1)の名前が衝突。



2. 優先順位が低い非例外Javaクラス(1)の名前にサフィックスを付けて名前解決しようとしたが，すでに定義されている非例外Javaクラス(2)の名前が衝突。



3. すでに定義されている非例外Javaクラス(2)のアンダースコア(_)を外して名前衝突を解決。



さらに，アンダースコアの削除によって変更されたクラス名と，同名のクラス名が定義されていた場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51030-E※）。

注※

非例外 Java クラスの場合は，異なるメッセージが出力されます。

(b) メソッド名およびパラメタ名の衝突時のマッピング

WSDL から Java ソースへのマッピングで，メソッド同士およびメソッドのパラメタ同士で名前が衝突した場合は，名前衝突は解決されないでエラーになります。

(3) JAXB アノテーションのサポートについて

cjwsimport コマンドは，JAX-WS 2.2 仕様の Comformance 2.17 に対応しています。cjwsimport コマンド実行時に，必要に応じて次に示す JAXB アノテーションが SEI に付与されます。

- javax.xml.bind.annotation.XmlAttachmentRef
- javax.xml.bind.annotation.XmlList
- javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter
- javax.xml.bind.annotation.XmlMimeType

なお，Application Server の JAX-WS 機能では MIME バインディングはサポートしていないため，javax.xml.bind.annotation.XmlMimeType アノテーションは付与されません。MIME バインディングを指定した場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51188-E）。

15.2　WSDL から Java へのマッピングのカスタマイズ

バインディング宣言を使用することで，WSDL から Java ソースへのマッピングをカスタマイズできます。バインディング宣言によるカスタマイズ方法には，次の方法があります。

- 埋め込みによるバインディング宣言でのカスタマイズ
- 外部バインディングファイルによるカスタマイズ

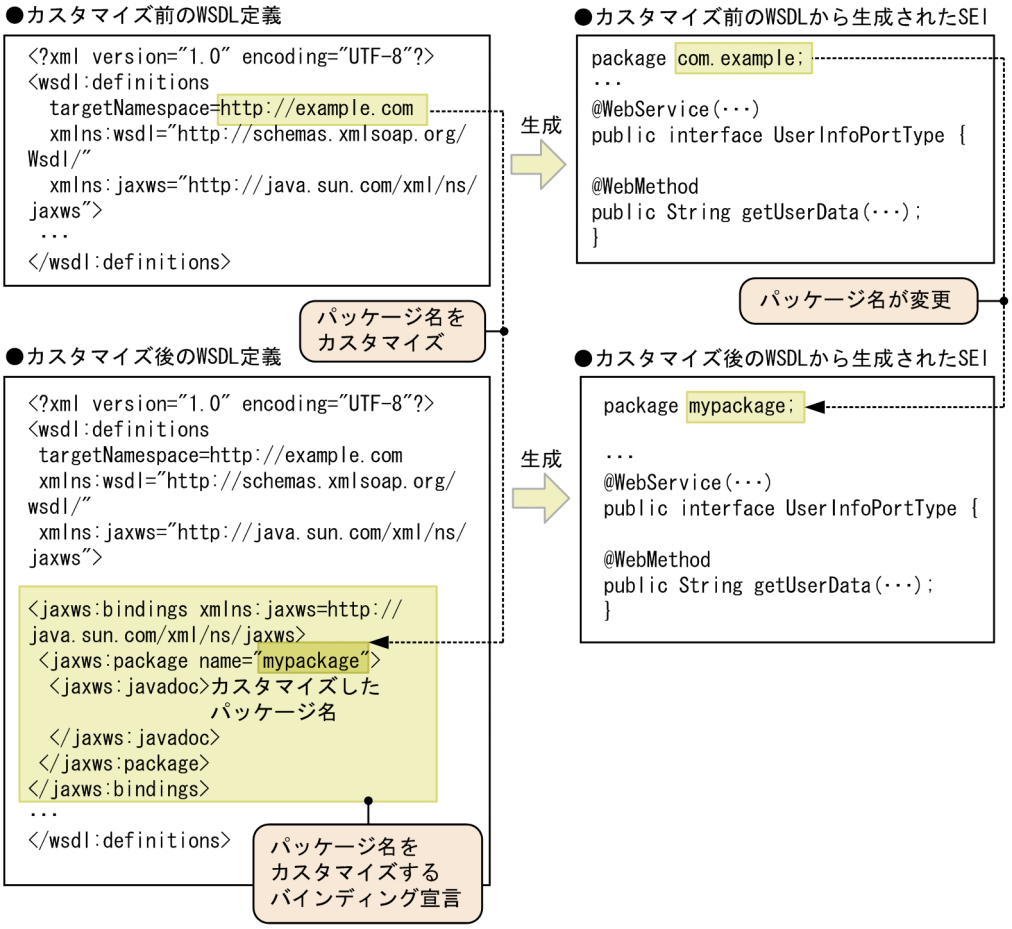
それぞれの方法について，カスタマイズ方法と注意事項について説明します。

15.2.1　埋め込みによるバインディング宣言でのカスタマイズ

埋め込みによるバインディング宣言でのカスタマイズは，jaxws:bindings 要素を使用して，直接バインディング宣言を WSDL 文書に記述し，カスタマイズします。

埋め込みによるバインディング宣言を使用して，パッケージ名をカスタマイズする例を次の図に示します。

図 15-17　パッケージ名のカスタマイズ例（埋め込みによるバインディング宣言）



埋め込みによるバインディング宣言でカスタマイズする場合の留意点について説明します。

(1) jaxws:bindings 要素の指定

jaxws:bindings 要素は，埋め込みによるバインディング宣言のコンテナとして使用します。

ただし，jaxws:bindings 要素は jaxws:bindings 要素の子要素に記述できません。jaxws:bindings 要素の子要素に記述した場合，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51034-E）。

jaxws:bindings 要素の属性と，属性の指定有無による動作を次の表に示します。

表 15–20 jaxws:bindings 要素の属性と指定有無による動作の対応（埋め込みによるバインディング宣言）

項番	要素	属性の指定	動作
1	wsdlLocation	あり	属性は指定できません。指定しても無視されます。
2		なし	正常終了します。
3	node	あり	属性は指定できません。指定しても無視されます。
4		なし	正常終了します。
5	version	あり	属性には"2.0"だけ指定できます。"2.0"以外を指定した場合は値が無視され，"2.0"が指定されていると見なされます。
6		なし	正常終了します。

(2) 使用できるバインディング宣言

埋め込みによるバインディング宣言を使用する場合に，Application Server の JAX-WS 機能で使用できるバインディング宣言の一覧を次の表に示します。各バインディング宣言については，JAX-WS 2.2 仕様を参照してください。

表 15–21 使用できるバインディング宣言（埋め込みによるバインディング宣言）

要素名	属性名	説明
wsdl:definitions/jaxws:bindings 要素	version	version 属性には，WSDL のカスタマイズのバージョンを記述します。
jaxws:bindings 要素の子要素	−	jaxws:bindings 要素の子要素です。
┣ jaxws:package	name	name 属性には，wsdl:definitions 要素の targetNamespace 属性に対応する Java パッケージ名を記述します。
┃ ┗ jaxws:javadoc	−	Java パッケージに付加する Javadoc 文字列です。

要素名	属性名	説明
├ jaxws:enableWrapperStyle	−	WSDL のすべてのオペレーションに対する wrapper スタイルの有効／無効を表します。
└ jaxws:enableAsyncMapping	−	WSDL のすべてのオペレーションに対する非同期マッピングの有効／無効を表します。
wsdl:definitions/wsdl:portType /jaxws:bindings 要素の子要素	−	wsdl:definitions/wsdl:portType 要素に含まれる jaxws:bindings 要素の子要素です。
├ jaxws:class	name	name 属性には，wsdl:portType 要素に対応する SEI の完全修飾名を記述します。
└ jaxws:javadoc	−	SEI に付加する Javadoc 文字列を記述します。
├ jaxws:enableWrapperStyle	−	wsdl:portType 要素に対する wrapper スタイルの有効／無効を表します。
└ jaxws:enableAsyncMapping	−	wsdl:portType 要素に対する非同期マッピングの有効／無効を表します。
wsdl:definitions/wsdl:portType/wsdl:operation /jaxws:bindings 要素の子要素	−	wsdl:definitions/wsdl:portType/ wsdl:operation 要素に含まれる jaxws:bindings 要素の子要素です。
├ jaxws:method	name	name 属性には，wsdl:operation 要素に対応する Java メソッド名を記述します。
└ jaxws:javadoc	−	メソッドに付加する Javadoc 文字列です。
├ jaxws:enableWrapperStyle	−	wsdl:operation 要素に対する wrapper スタイルの有効／無効を表します。
├ jaxws:enableAsyncMapping	−	wsdl:operation 要素に対する非同期マッピングの有効／無効を表します。
└ jaxws:parameter	part	part 属性には，wsdl:message 要素の wsdl:part 子要素を識別する XPath 表現を記述します。
	childElementName	childElementName 属性には，wsdl:part 要素によって参照されるグローバル型定義またはグローバル要素宣言の子要素名を記述します。

要素名	属性名	説明
	name	name 属性には，part 属性および childElementName 属性によって識別される要素のパラメタ名を記述します。
wsdl:definitions/wsdl:portType/wsdl:operation/wsdl:fault /jaxws:bindings 要素の子要素	−	wsdl:definitions/wsdl:portType/wsdl:operation/wsdl:fault 要素に含まれる jaxws:bindings 要素の子要素です。
└ jaxws:class	name	name 属性には，wsdl:fault 要素に対応する例外クラスの完全修飾名を記述します。
└ jaxws:javadoc	−	例外クラスに付加する Javadoc 文字列です。
wsdl:definitions/wsdl:binding/wsdl:operation 要素/jaxws:bindings 要素の子要素	−	wsdl:definitions/wsdl:binding/wsdl:operation 要素に含まれる jaxws:bindings 要素の子要素です。
└ jaxws:parameter	part	part 属性には，wsdl:message 要素の wsdl:part 子要素を識別する XPath 表現を記述します。
	childElementName	childElementName 属性には，wsdl:part 要素によって参照されるグローバル型定義またはグローバル要素宣言の子要素名を記述します。
	name	name 属性には，part 属性および childElementName 属性によって識別される要素のパラメタ名を記述します。
wsdl:definitions/wsdl:service /jaxws:bindings 要素の子要素	−	wsdl:definitions/wsdl:service/要素に含まれる jaxws:bindings 要素の子要素です。
└ jaxws:class	name	name 属性には，wsdl:service 要素に対応するサービスクラスの完全修飾名を記述します。
└ jaxws:javadoc	−	サービスクラス名に付加する Javadoc 文字列です。
wsdl:definitions/wsdl:service/wsdl:port /jaxws:bindings 要素の子要素	−	wsdl:definitions/wsdl:service/wsdl:port 要素に含まれる jaxws:bindings 要素の子要素です。
└ jaxws:method	name	name 属性には，wsdl:port 要素に対応する getter メソッド名を記述します。
├ jaxws:javadoc	−	getter メソッドに付加する Javadoc 文字列です。
└ jaxws:provider	−	"true"を指定した場合，SEI は生成されません。生成されたサービスインタフェースでポートの getter メソッドが省略されます。

要素名			属性名	説明
				jaxws:provider 要素については，「15.2.7 jaxws:provider 要素を記述した場合の動作」を参照してください。

（凡例）

－：バインディング宣言に使用できる属性がないことを示します。

この表に示すように，WSDL で jaxws:bindings 要素およびその子要素を記述できる位置は，JAX-WS 2.2 仕様で規定されています。規定されていない位置にそれらの要素を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51029-E）。

各 jaxws:bindings 要素の子要素として記述できない要素を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51040-E）。

また，jaxws:bindings 要素の属性およびその子要素に記述できない属性を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51029-E）。

(3)　要素および属性の重複

jaxws:bindings 要素およびその子要素内で属性が重複して記述されている場合，標準エラー出力とログに XML Processor のエラーメッセージが出力され，処理が終了されます。

埋め込みによるバインディング宣言を使用して，同じカスタマイズ対象へのカスタマイズを重複して指定できません。また，jaxws:bindings 要素の子要素も重複して指定できません。指定した場合の動作は保証されません。

(4)　jaxws:enableWrapperStyle 要素の優先順位

jaxws:enableWrapperStyle 要素は，WSDL の次の個所に記述できます。複数の個所に同時に記述した場合の優先順位を示します。

1. wsdl:portType/wsdl:operation/jaxws:bindings/jaxws:enableWrapperStyle
2. wsdl:portType/jaxws:bindings/jaxws:enableWrapperStyle
3. wsdl:definitions/jaxws:bindings/jaxws:enableWrapperStyle

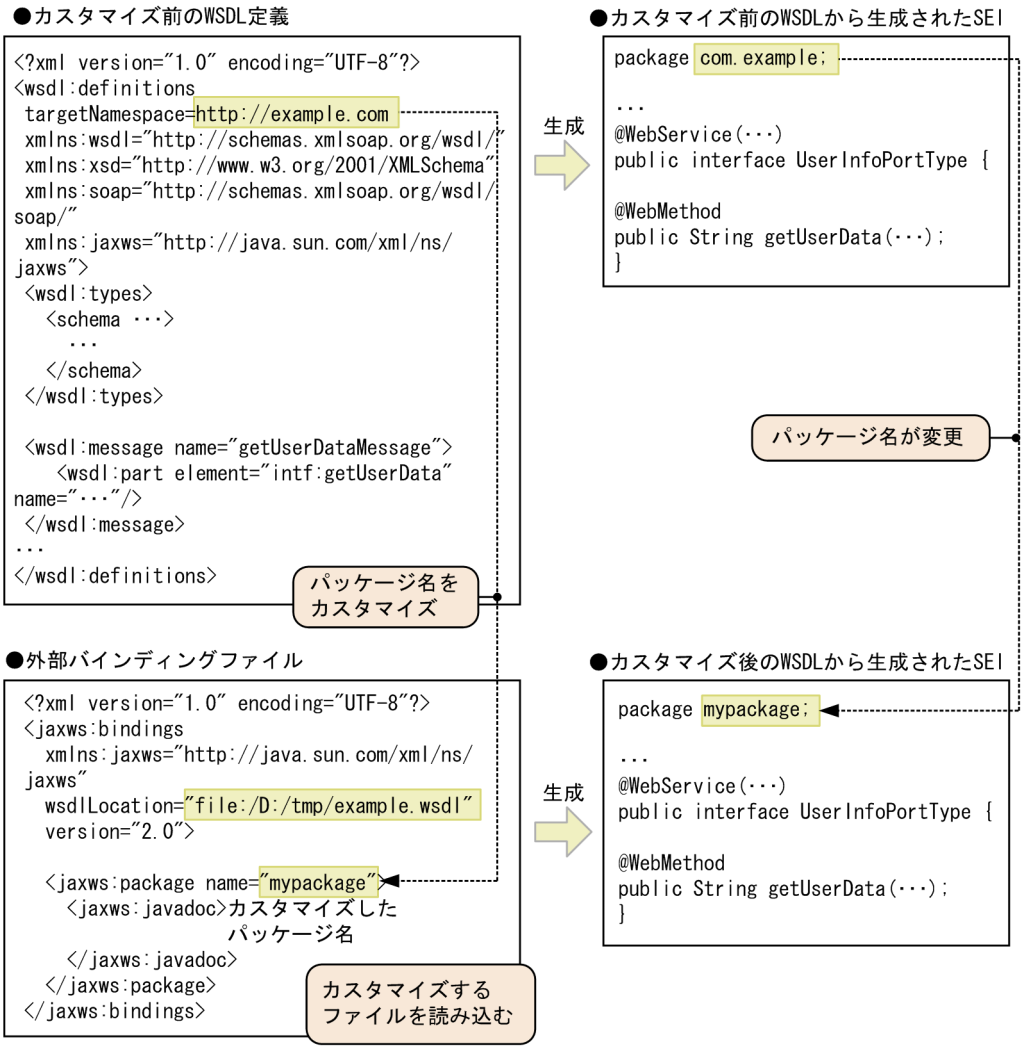
番号は優先順位を表します。項番 1 がいちばん高く，優先順位の高い要素値が有効になります。

15.2.2　外部バインディングファイルによるカスタマイズ

外部バインディングファイルによるカスタマイズは，WSDL とは別にバインディング宣言をまとめて記述したファイルを用意し，そのファイルを WSDL 文書と同時に読み込み，カスタマイズする方法です。外部バインディングファイルは，複数読み込むことができます。

外部バインディングファイルを使用して，パッケージ名をカスタマイズする例を次の図に示します。

図 15-18　パッケージ名のカスタマイズ例（外部バインディングファイル）



外部バインディングファイルでカスタマイズする場合の留意点について説明します。

(1)　jaxws:bindings 要素の指定

外部バインディングファイルの場合も埋め込みによるバインディング宣言と同様に，jaxws:bindings 要素をコンテナとして使用します。

ただし，埋め込みによるバインディング宣言とは異なり，jaxws:bindings 要素を jaxws:bindings 要素の子要素として記述できます。

jaxws:bindings 要素の属性と，属性の指定有無による動作を次の表に示します。

表 15–22　jaxws:bindings 要素の属性と指定有無による動作の対応（外部バインディングファイル）

項番	記述位置	要素	属性の指定	動作
1	ルートの jaxws:bindings※1	wsdlLocation	あり	属性は必ず指定してください。指定したロケーションの WSDL をカスタマイズ対象の WSDL ファイルとします。指定方法については，「15.2.2(1)(a)　wsdlLocation 属性の記述形式」を参照してください。
2			なし	属性を指定していない場合，外部バインディングファイルは無視されます（カスタマイズされないで正常終了します）。
3		node	あり	XPath 1.0 形式で属性を指定できます。指定した要素をカスタマイズ対象とします。指定方法については，「15.2.2(1)(b)　node 属性の記述形式」を参照してください。
4			なし	カスタマイズ対象の要素を WSDL のルート（wsdl:definitions 要素）とします。
5		version	あり	"2.0"を指定できます。指定方法については，「15.2.2(1)(c)　version 属性の記述形式」を参照してください。
6			なし	"2.0"が指定されていると見なされます。
7	非ルートの jaxws:bindings※2	wsdlLocation	あり	属性は指定できません。指定しても無視されます。
8			なし	正常終了します。
9		node	あり	属性は必ず指定してください。指定した要素をカスタマイズ対象とします。指定方法については，「15.2.2(1)(b)　node 属性の記述形式」を参照してください。
10			なし	カスタマイズ対象がないため，カスタマイズされないで正常終了します。
11		version	あり	属性は指定できません。指定しても無視されます。
12			なし	正常終了します。

注※1

「ルートの jaxws:bindings」とは，外部バインディングファイルの最上位に記述された jaxws:bindings 要素を表します。

注※2

「非ルートの jaxws:bindings」とは，ルートの jaxws:bindings の子要素以下に記述された jaxws:bindings 要素を表します。

(a) wsdlLocation 属性の記述形式

jaxws:bindings 要素の wsdlLocation 属性に指定する値は，URL で指定します。URL によって指定するファイルは，リモートファイルでもローカルファイルでも，どちらでもかまいません。ローカルファイルの場合は，相対パスで指定することもできます。

誤った形式で記述した場合，またはファイルがない場合，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51043-E)。

URL は，RFC 2396 仕様の規則に従った文字列を使用してください。RFC 2396 仕様の規則に従っていない文字列を使用する場合は，RFC 2396 仕様の規則に従って UTF-8 でパーセントエンコーディングする必要があります。ただし，"&"はパーセントエンコーディングをしても使用できません。RFC 2396 仕様の規則に従わないで，エンコードもしていない文字や文字列を指定した場合の動作は保証されません。また，wsdlLocation 属性に，WSDL ファイル以外のファイルを指定した場合の動作は保証されません。

wsdlLocation 属性の正しい記述例を示します。

```
<jaxws:bindings xmlns:jaxws=http://java.sun.com/xml/ns/jaxws
  wsdlLocation="file:///D:/tmp/example.wsdl" version="2.0">
 ...
</jaxws:bindings>
```

(b) node 属性の記述形式

jaxws:bindings 要素の node 属性に指定する値は，XPath 1.0 形式で指定します。

誤った形式で記述した場合，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます(KDJW51038-E)。

node 属性の記述例を次に示します。この例では，wsdl:definitions/wsdl:portType 要素の name 属性が，AddNumbersImpl の要素に対するバインディング宣言であることを示しています。

```
<jaxws:bindings node="wsdl:definitions/wsdl:portType[@name='AddNumbersImpl']">
 ...
</jaxws:bindings>
```

非ルートの jaxws:bindings 要素の node 属性では，ルートの jaxws:bindings 要素の node 属性で指定しているカスタマイズ対象要素からの相対パス（XPath 1.0 形式）を指定できます。この場合の記述例を示します。

```
<jaxws:bindings node="wsdl:definitions/wsdl:portType[@name='UserInfoPortType']"
 ...
  <jaxws:bindings node="../wsdl:service[@name='UserInfoService']">
    ...
  </jaxws:bindings>
</jaxws:bindings>
```

(c) version 属性の記述形式

jaxws:bindings 要素の version 属性の値は，"2.0"を指定します。

"2.0"以外の値を記述した場合，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51039-E）。

version 属性の正しい記述例を示します。

```
<jaxws:bindings xmlns:jaxws=http://java.sun.com/xml/ns/jaxws
  wsdlLocation="file:///D:/tmp/example.wsdl" version="2.0">
  ...
</jaxws:bindings>
```

(2) 使用できるバインディング宣言

外部バインディングファイルを使用する場合に，Application Server の JAX-WS 機能で使用できるバインディング宣言の一覧を次の表に示します。各バインディング宣言については，JAX-WS 2.2 仕様を参照してください。

表 15-23 使用できるバインディング宣言（外部バインディングファイル）

要素名	属性名	説明
jaxws:bindings 要素	wsdlLocation	wsdlLocation 属性には，カスタマイズ対象の WSDL ファイルのファイルパス（URL）を記述します。
	node	node 属性には，WSDL 内のカスタマイズ対象の要素を記述します。
	version	version 属性には，WSDL のカスタマイズのバージョンを記述します。
jaxws:bindings 要素の子要素	−	jaxws:bindings 要素の子要素です。
├ jaxws:package └ jaxws:javadoc	name	name 属性には，wsdl:definitions 要素の targetNamespace 属性に対応する Java パッケージ名を記述します。
	−	Java パッケージに付加する Javadoc 文字列です。
├ jaxws:enableWrapperStyle	−	各要素に対する wrapper スタイルの有効／無効を表します。
├ jaxws:enableAsyncMapping	−	各要素に対する非同期マッピングの有効／無効を表します。
├ jaxws:class └ jaxws:javadoc	name	name 属性には，各要素に対応するクラス名を記述します。
	−	クラスに付加する Javadoc 文字列です。

要素名		属性名	説明
｜			
├ ｜ ｜ ｜	jaxws:method ｜	name	name 属性には，各要素に対応する Java メソッド名を記述します。
｜ ｜ ｜	└ jaxws:javadoc	－	メソッドに付加する Javadoc 文字列です。
├ ｜ ｜ ｜ ｜ ｜ ｜ ｜ ｜ ｜ ｜	jaxws:parameter	part	part 属性には，wsdl:message 要素の wsdl:part 子要素を識別する XPath 表現を記述します。
		childElementName	childElementName 属性には，wsdl:part 要素によって参照するグローバル型定義，またはグローバル要素宣言の子要素名を記述します。
		name	name 属性には，part 属性および childElementName 属性によって識別される要素のパラメタ名を記述します。
└	jaxws:provider	－	"true"を指定した場合，SEI は生成されません。生成されたサービスインタフェースでポートの getter メソッドが省略されます。 jaxws:provider 要素については，「15.2.7 jaxws:provider 要素を記述した場合の動作」を参照してください。

（凡例）

－：バインディング宣言に使用できる属性がないことを示します。

この表に示すように，WSDL で jaxws:bindings 要素およびその子要素を記述できる位置は，JAX-WS 2.2 仕様で規定されています。規定されていない位置にそれらの要素を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51029-E）。

各 jaxws:bindings 要素の子要素として記述できない要素を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51040-E）。

また，jaxws:bindings 要素の属性およびその子要素に記述できない属性を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51029-E）。

なお，JAXB 仕様のバインディング宣言は記述できません。記述した場合の動作は保証されません。

(3) 要素および属性の重複

jaxws:bindings 要素およびその子要素内で属性が重複して記述されている場合，標準エラー出力とログに XML Processor のエラーメッセージが出力され，処理が終了されます。

外部バインディングファイルを使用して，同じカスタマイズ対象へのカスタマイズを重複して指定することはできません。指定した場合の動作は保証されません。

(4) wsdl:import 要素で読み込む WSDL に対するカスタマイズ

wsdl:import 要素でインポートする WSDL に対して外部バインディングファイルでカスタマイズする場合，jaxws:bindings 要素の wsdlLocation 属性で wsdl:import 要素でインポートする WSDL を指定してください。

wsdl:import 要素でインポートする WSDL をカスタマイズするときに，誤って wsdl:import 元の WSDL を指定すると，jaxws:bindings 要素の node 属性で示したカスタマイズ対象が見つかりません。このときに，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51187-E）。

15.2.3 埋め込みによるバインディング宣言と外部バインディングファイルの同時指定

埋め込みによるバインディング宣言と外部バインディングファイルは，カスタマイズ対象が異なる場合は，それぞれのカスタマイズ内容が有効になります。

埋め込みによるバインディング宣言と外部バインディングファイルのカスタマイズ対象が同じ場合，埋め込みによるバインディング宣言は無効となります。

15.2.4 jaxws:bindings 要素に指定できる値

jaxws:bindings 要素に指定できる要素および属性の一覧を次の表に示します。

表 15–24 jaxws:bindings 要素の属性の指定可否

要素名		属性名	指定可否
wsdl:definitions/jaxws:bindings 要素の子要素			
├	jaxws:package	name	○
│ └	jaxws:javadoc	−	○
├	jaxws:enableWrapperStyle	−	○
├	jaxws:enableAsyncMapping	−	○
└	jaxws:enableMIMEContent	−	×
wsdl:portType/jaxws:bindings 要素の子要素			
├	jaxws:class	name	○
│ └	jaxws:javadoc	−	○

要素名		属性名	指定可否
├	jaxws:enableWrapperStyle	−	○
└	jaxws:enableAsyncMapping	−	○
wsdl:portType/wsdl:operation/jaxws:bindings 要素の子要素			
├	jaxws:method	name	○
│ └	jaxws:javadoc	−	○
├	jaxws:enableWrapperStyle	−	○
├	jaxws:enableAsyncMapping	−	○
└	jaxws:parameter	part	○
		childElementName	○
		name	○
wsdl:portType/wsdl:operation/wsdl:fault/jaxws:bindings 要素の子要素			
└	jaxws:class	name※	○
	└ jaxws:javadoc	−	○
wsdl:binding/jaxws:bindings 要素の子要素			
└	jaxws:enableMIMEContent	−	×
wsdl:binding/wsdl:operation/jaxws:bindings 要素の子要素			
├	jaxws:enableMIMEContent	−	×
└	jaxws:parameter	part	○
		childElementName	○
		name	○
wsdl:service/jaxws:bindings 要素の子要素			
└	jaxws:class	name	○
	└ jaxws:javadoc	−	○
wsdl:service/wsdl:port/jaxws:bindings 要素の子要素			
├	jaxws:method	name	○
│ └	jaxws:javadoc	−	○
└	jaxws:provider	−	○

（凡例）

−：バインディング宣言に使用できる属性がないことを示します。

○：要素および属性が指定できることを示します。

×：要素および属性が指定できないことを示します（非サポート）。

注※

フォルト名をカスタマイズする場合，カスタマイズ後のフォルト名がほかのフォルト名と重複しないようにしてください。重複した場合，動作は保証されません。

指定できないバインディング要素を指定した場合，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51188-E）。

要素および属性に指定できる値について説明します。

(1) name 属性に指定できる値

name 属性には，次に示す値だけ指定できます。これ以外の値を指定した場合の動作は保証されません。

- Java 識別子として使用できる値
- 半角英数字（0～9，A～Z，a～z）
- アンダースコア（_），ドルマーク（$）
- ピリオド（.）※

  注※

  wsdl:portType 要素，wsdl:fault 要素，または wsdl:service 要素をカスタマイズする場合にだけ使用できます。

クラス名やメソッド名を name 属性でカスタマイズする場合，カスタマイズした名称がほかの名称と重複するとエラーとなるため，重複しないように注意してください。

wsdl:portType 要素に対応する SEI 名，wsdl:fault 要素に対応する例外クラス名，または wsdl:service 要素に対応するサービス名を jaxws:class 要素でカスタマイズする場合，パッケージ名を含むクラス名を指定すると，指定したパッケージ名の Java クラスになります。パッケージ名を含まないクラス名を指定すると，WSDL の名前空間（wsdl:definitions 要素の targetNamespace 属性）からマッピングされたパッケージ名を付与した Java クラスになります。

(2) jaxws:javadoc 要素に指定できる値

jaxws:javadoc 要素には，次に示す値だけ指定できます。これ以外の値を指定した場合の動作は保証されません。

- 半角英数字（0～9，A～Z，a～z）
- 半角記号（"/*"，"//"，"¥"，"¥n¥r"）※1
- 半角カタカナ
- 全角ひらがな，全角カタカナ，全角英数字
- 第一水準の全角漢字
- Java 予約語

- 空白および空文字※2

注※1

半角記号（"*/"）は，Javadoc の終端として扱われるため使用できません。

注※2

空白および空文字の行は，ソースコード生成時に削除されます。

(3) jaxws:enableWrapperStyle 要素に指定できる値

jaxws:enableWrapperStyle 要素には，boolean 値（"true"または"false"）を指定できます。これ以外の値を指定した場合の動作は保証されません。

なお，WSDL を wrapper スタイルで記述し，かつこの要素の値を"true"に指定した場合にだけ，wrapper スタイルが有効になります。

(4) jaxws:enableAsyncMapping 要素に指定できる値

jaxws:enableAsyncMapping 要素には，boolean 値（"false"）だけ指定できます。これ以外の値を指定した場合の動作は保証されません。

(5) part 属性に指定できる値

part 属性には，次に示す値だけ指定できます。これ以外の値を指定した場合の動作は保証されません。

- 存在する wsdl:message 要素の wsdl:part 子要素の XPath 表現
- 存在する soap:header から参照する wsdl:message の wsdl:part 子要素の XPath 表現
- wsdl:operation 要素内で使用されていない wsdl:message 要素の wsdl:part 子要素の XPath 表現

(6) childElementName 属性に指定できる値

存在する型定義の修飾名の QName だけ指定できます。存在しない型定義の修飾名の QName を指定した場合，エラーにならないで正常終了します。このとき，カスタマイズ対象は無視されます。

(7) jaxws:provider 要素に指定できる値

boolean 値（"true"または"false"）を指定できます。これ以外の値を指定した場合の動作は保証されません。

(8) 非サポートの要素

Application Server の JAX-WS 機能で非サポートのバインディング要素は指定できません。指定した場合の動作は保証されません。

15.2.5　カスタマイズ対象となった要素の値

埋め込みによるバインディング宣言または外部バインディングファイルによってカスタマイズ対象となった WSDL 文書の要素の値は，Java にマッピングされません。したがって，WSDL 1.1 仕様としてカスタマイズ対象に記述できる文字であればどのような文字を記述してもかまいません。

15.2.6　名前衝突時の対応

カスタマイズすることで SEI 名，クラス名，メソッド名，およびパラメタ名で名前衝突が発生するおそれがあります。名前衝突の解決方法（優先順位など）は，デフォルトマッピングの場合と同様です。名前衝突の解決方法については，「15.1.11(2)　名前衝突時のマッピング」を参照してください。

ただし，SEI 名およびクラス名に関しては，優先順位が低い方をカスタマイズした場合は名前衝突したときに名前解決されますが，優先順位が高い方をカスタマイズした場合は名前衝突したときに名前衝突は解決されません。このとき，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます(KDJW51030-E※)。

注※

非例外 Java クラスの場合は，異なるメッセージが出力されます。

15.2.7　jaxws:provider 要素を記述した場合の動作

wsdl:port 要素に jaxws:provider 要素を記述して cjwsimport コマンドを実行した場合，wsdl:port 要素用の SEI は生成されません。警告メッセージが出力され，処理が続行されます（KDJW51206-W)。この場合，javax.xml.ws.Provider インタフェースを使用してプロバイダ実装クラスを生成してください。

また，生成したサービスクラスで wsdl:port 要素の getter メソッドが省略されます。

cjwsimport コマンドに-generateService オプションを指定し，wsdl:port 要素に jaxws:provider 要素を記述した場合，スケルトンクラスは生成されません。警告メッセージが出力され，処理が続行されます(KDJW51207-W)。

15.2.8　SEI 名をカスタマイズする場合の注意事項

次の条件でカスタマイズする場合，SEI 名には"Provider"以外を指定し，パッケージ名には"http://ws.xml.javax"以外を指定して，SEI 名がパッケージ名を含めて"javax.xml.ws.Provider"にならないようにしてください。

- パッケージ名：jaxws:package 要素
- SEI 名：jaxws:class 要素

"javax.xml.ws.Provider"は，wsdl:port 要素に jaxws:provider 要素を記述した場合の予約語として利用しており，サービス実装クラスで javax.xml.ws.Provider インタフェースを利用するときに指定します。

15.2.9 jaxws:parameter 要素で inout パラメタ名をカスタマイズする場合の注意事項

jaxws:parameter 要素で inout パラメタ名をカスタマイズする場合は，input メッセージと output メッセージの両方のパラメタ名をカスタマイズしてください。どちらか片方だけをカスタマイズした場合の動作は保証されません。

15.2.10 SEI 名を jaxws:class 要素でカスタマイズした場合のスケルトンクラス名

SEI 名を jaxws:class 要素でカスタマイズした場合，スケルトンクラス名は，カスタマイズした SEI 名に"Impl"サフィックスを付与してマッピングされます。

16 Java から WSDL へのマッピング

hwsgen コマンドを実行すると，JAX-WS 2.2 仕様に従って Java ソースから WSDL へマッピングされます。

この章では，Java から WSDL へのデフォルトマッピング，およびマッピングのカスタマイズについて説明します。

16.1　Java から WSDL へのデフォルトマッピング

Java ソースから WSDL へマッピングする場合の対応関係を次の表に示します。

表 16–1　Java ソースから WSDL へのマッピング一覧

項番	Java ソース	WSDL	参照先
1	パッケージ名	WSDL の名前空間	16.1.1
2	SEI 名	ポートタイプ	16.1.3
3	SEI のメソッド名	オペレーション	16.1.4
4	SEI のメソッドのパラメタおよび戻り値	パート	16.1.5，16.1.6
5	SEI のラッパ例外クラス	フォルト	16.1.7
6	SEI および Web サービス実装クラス	バインディング	16.1.8
7	Web サービス実装クラス	サービスとポート	16.1.9

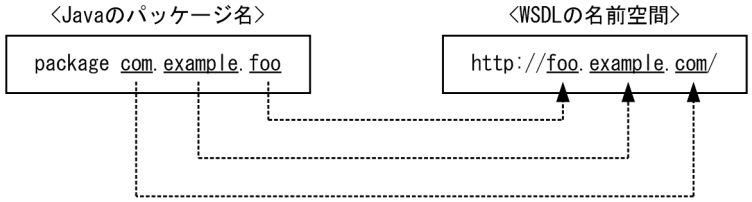
16.1.1　パッケージ名から名前空間へのマッピング

Java のパッケージ名から WSDL の名前空間（wsdl:definitions 要素の targetNamespace 属性）へのマッピングについて説明します。

(1)　マッピング

SEI および Web サービス実装クラスのパッケージ名と，WSDL の名前空間は，JAX-WS 2.2 仕様に従ってマッピングされます。マッピング例を次の図に示します。

図 16–1　Java のパッケージ名と名前空間のマッピング例



(2)　パッケージ名の条件

Java のパッケージ名のピリオド（.）で区切られたセグメントには，次の表に示すすべての条件を満たす文字列を記述できます。ただし，javax.jws.WebService アノテーションの targetNamespace 要素を使用する場合，Java 言語仕様で定められている Java 識別子の命名規則に従った文字列を記述できます。

表 16–2　Java のパッケージ名の各セグメントに記述できる文字列の条件

項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z）だけを使用した文字列	package com.鈴木	動作は保証されません（エラーメッセージは出力されません）。
2	Java 言語仕様で定められている Java 識別子の命名規則に従った文字列	package com.abstract;	javac コマンド実行時にコンパイルエラーとなり，終了します。

(3)　javax.jws.WebService アノテーションの targetNamespace 要素の利用

入力となる SEI または Web サービス実装クラスがデフォルトパッケージの場合，javax.jws.WebService アノテーションの targetNamespace 要素で名前空間名を記述してください。

javax.jws.WebService アノテーションの targetNamespace 要素については，「16.2.9 javax.jws.WebService アノテーション」を参照してください。

16.1.2　Web サービス実装クラスから SEI へのマッピング

Web サービス実装クラスから SEI へのマッピングに当たり，前提となる条件および留意点について説明します。

(1)　Web サービス実装クラスの条件

Web サービス実装クラスの条件を示します。

- SEI のすべてのメソッドを実装する必要があります。
- Web サービスのオペレーションとして，Object クラスの finalize メソッドをオーバーライドするような finalize メソッドは定義できません。
- public なデフォルトコンストラクタを定義する必要があります。
- javax.jws.WebService アノテーションを記述する必要があります。javax.jws.WebService アノテーションを記述していない場合は，Web サービス実装クラスではないと判断されます。
- javax.jws.WebService アノテーションを static なインナークラスで定義できます。
- Web サービス実装クラスのアクセス修飾子は public にしてください。final や abstract は指定できません。

(2) javax.jws.WebService アノテーションの endpointInterface 要素の利用

Web サービス実装クラスから SEI へのマッピングでは，javax.jws.WebService アノテーションの endpointInterface 要素を利用して，Web サービス実装クラスと SEI を関連づけることができます。

Web サービス実装クラスだけ定義する場合，endpointInterface 要素は使用しません。この場合，Web サービス実装クラスが持つ情報から，SEI に定義する抽象的な情報が抽出され，仮想の SEI があるものと見なされます（暗黙の SEI)。

javax.jws.WebService アノテーションの endpointInterface 要素については，「16.2.9 javax.jws.WebService アノテーション」を参照してください。

16.1.3 SEI 名からポートタイプへのマッピング

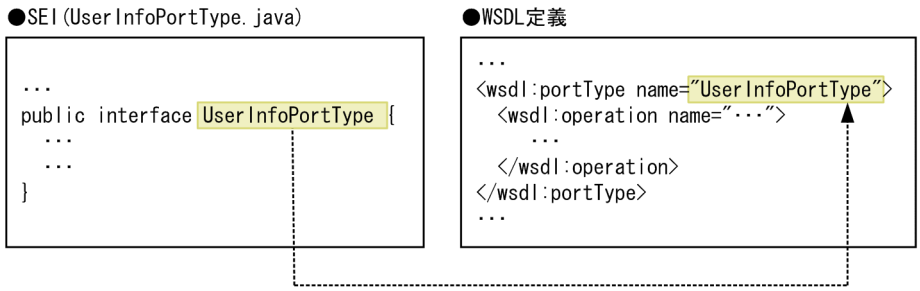
Java の SEI 名から WSDL のポートタイプ名（wsdl:portType 要素の name 属性）へのマッピングについて説明します。

(1) マッピング

Java の SEI 名と WSDL のポートタイプは，JAX-WS 2.2 仕様に従ってマッピングされます。

サービス実装クラスの javax.jws.WebService アノテーションで endpointInterface 要素を使用していない場合，Web サービス実装クラス名と同じ名称の暗黙の SEI が存在するものと見なされ，WSDL のポートタイプにマッピングされます。マッピング例を次の図に示します。

図 16-2 SEI 名とポートタイプのマッピング例



(2) SEI の条件

Web サービス実装クラスの条件を示します。

- javax.jws.WebService アノテーションを記述する必要があります。
- java.rmi.Remote インタフェースを継承してもかまいません。

(3) SEI 名の条件

SEI 名には，次の表に示すすべての条件を満たす文字列を記述できます。ただし，javax.jws.WebService アノテーションの name 要素を使用する場合，Java 言語仕様で定められている Java 識別子の命名規則に従った文字列を記述できます。

表 16-3 SEI 名に記述できる文字列の条件

項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z），およびアンダースコア（_）だけを使用した文字列	鈴木_sei	動作は保証されません（エラーメッセージは出力されません）。
2	Java 言語仕様で定められている Java 識別子の命名規則に従った文字列	abstract	javac コマンド実行時にコンパイルエラーとなり，終了します。

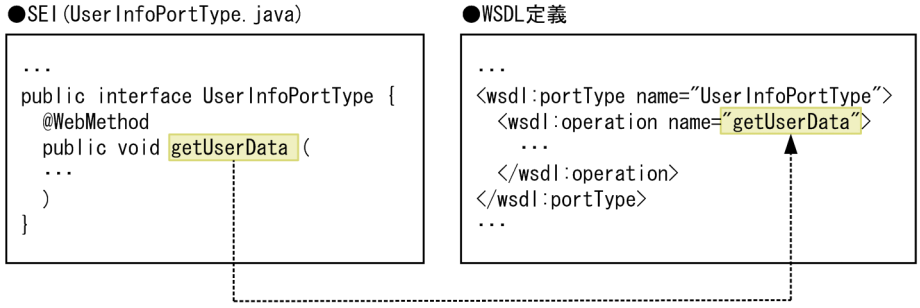
16.1.4 SEI のメソッド名からオペレーションへのマッピング

SEI のメソッド名から WSDL のオペレーション（wsdl:operation 要素の name 属性）へのマッピングについて説明します。

(1) マッピング

SEI のメソッド名と WSDL のオペレーションは，JAX-WS 2.2 仕様に従ってマッピングされます。マッピング例を次の図に示します。

図 16-3 SEI のメソッド名とオペレーションのマッピング例



SEI のメソッドを公開対象とするには，次に示す条件を満たす必要があります。

- アクセス修飾子が public である
- static 修飾子または final 修飾子が適用されていない
- javax.jws.WebMethod アノテーションでアノテートされている場合，javax.jws.WebMethod アノテーションの exclude 要素が true ではない

SEI のメソッド名からオペレーションへのマッピング規則を次に示します。

- javax.jws.WebMethod アノテーションの有無に関係なく，SEI のすべての公開対象メソッドが WSDL のオペレーションにマッピングされます。
- javax.jws.WebService アノテーションの endpointInterface 要素を使用していない場合，Web サービス実装クラスの公開対象メソッドを暗黙の SEI が持っているものと見なされ，WSDL のオペレーションにマッピングされます。暗黙の SEI にマッピングされるメソッドについては，「16.2.6 javax.jws.WebMethod アノテーション」を参照してください。
- Web サービス実装クラスが別の Web サービス実装クラスを継承している場合，次の条件に合うすべてのメソッドが WSDL のオペレーションにマッピングされます。

  （条件）

  Web サービス実装クラスおよび親の Web サービス実装クラスが持つ javax.jws.WebMethod アノテーションの exclude 要素が，true ではない公開対象メソッド。
- Web サービス実装クラスが別の Web サービス実装クラスを継承し，親クラスのメソッドをオーバーライドしている場合，Web サービス実装クラスでオーバーライドした公開対象メソッドが WSDL のオペレーションにマッピングされます。親クラスでオーバーライドされたメソッドはマッピングされません。
- SEI に定義できる公開対象メソッドおよび Web サービス実装クラスの公開対象メソッドは，255 個まで定義できます。
- SEI のメソッド名からマッピングされる wsdl:operation 要素の name 属性の値は，WSDL 内でユニークである必要があります。

(2) メソッド名の条件

メソッド名には，次の表に示すすべての条件を満たす文字列を記述できます。

表 16-4 メソッド名に記述できる文字列の条件

項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z），およびアンダースコア（_）だけを使用した文字列	鈴木_sei	動作は保証されません（エラーメッセージは出力されません）。
2	Java 言語仕様で定められている Java 識別子の命名規則に従った文字列	abstract	javac コマンド実行時にコンパイルエラーとなり，終了します。

ただし，次に示すアノテーションをすべて使用する場合，Java 言語仕様で定められている Java 識別子の命名規則に従った文字列を記述できます。

- javax.jws.WebMethod アノテーションの operationName 要素
- javax.xml.ws.RequestWrapper アノテーションの localName 要素および className 要素（wrapper スタイルの場合）
- javax.xml.ws.ResponseWrapper アノテーションの localName 要素および className 要素（wrapper スタイルの場合）

- javax.jws.WebParam アノテーションの name 要素（non-wrapper スタイルの場合）
- javax.jws.WebResult アノテーションの name 要素（non-wrapper スタイルの場合）

(3) オーバーロードによる名前衝突

メソッドのオーバーロードを使用している場合，デフォルトマッピングでは名前衝突が発生するため，名前がユニークになるようにアノテーションでカスタマイズする必要があります。

オーバーロードによる名前衝突が発生する個所と，参照先を次の表に示します。

表 16-5 オーバーロードによる名前衝突の発生個所と参照先

項番	名前衝突の発生個所	アノテーションの参照先	
		wrapper スタイル	non-wrapper スタイル
1	オペレーション名	16.2.6(2)	
2	input のグローバル要素※	16.2.16(1)，16.2.16(2)	16.2.7(2)，16.2.7(5)
3	output のグローバル要素※	16.2.17(1)，16.2.17(2)	16.2.8(2)，16.2.8(4)
4	リクエスト bean クラス名	16.2.16(3)	
5	レスポンス bean クラス名	16.2.17(3)	

注※

ローカル名または名前空間のどちらかをカスタマイズすれば，名前衝突は発生しません。

16.1.5 パラメタおよび戻り値からメッセージのパートへのマッピング（wrapper スタイルの場合）

SEI のメソッドのパラメタから WSDL（wsdl:part 要素の name 属性）へのマッピングについて説明します。

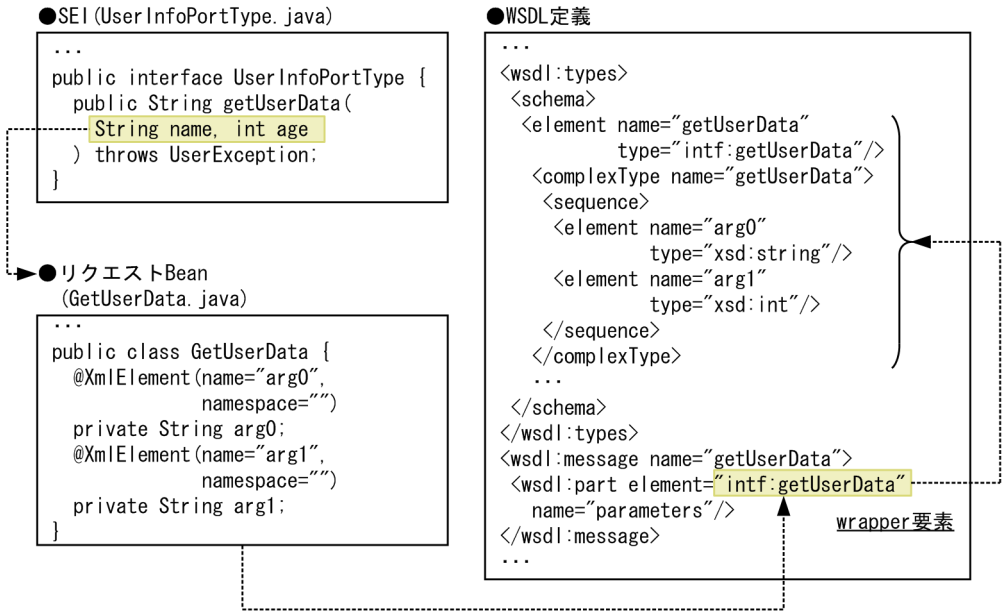
ここでは，wrapper スタイルの場合について説明します。

(1) マッピング

wrapper スタイルの場合，SEI のメソッド名および SEI のメソッド名と同じ名称のリクエスト bean が生成されます。また，接尾辞"Response"を付加したレスポンス bean が生成されます。リクエスト bean およびレスポンス bean の生成には，hwsgen コマンドで自動生成する方法と，Web サービスの開始時に動的に生成する方法があります。動的に生成する方法の場合は，Web サービスの開始時にエラーが発生しないように，コンパイルした Web サービス実装クラスに対して hwsgen コマンドを実行することで，事前にエラーチェックができます。詳細については，「10.23.1 hwsgen コマンドによるエラーチェックについて」を参照してください。

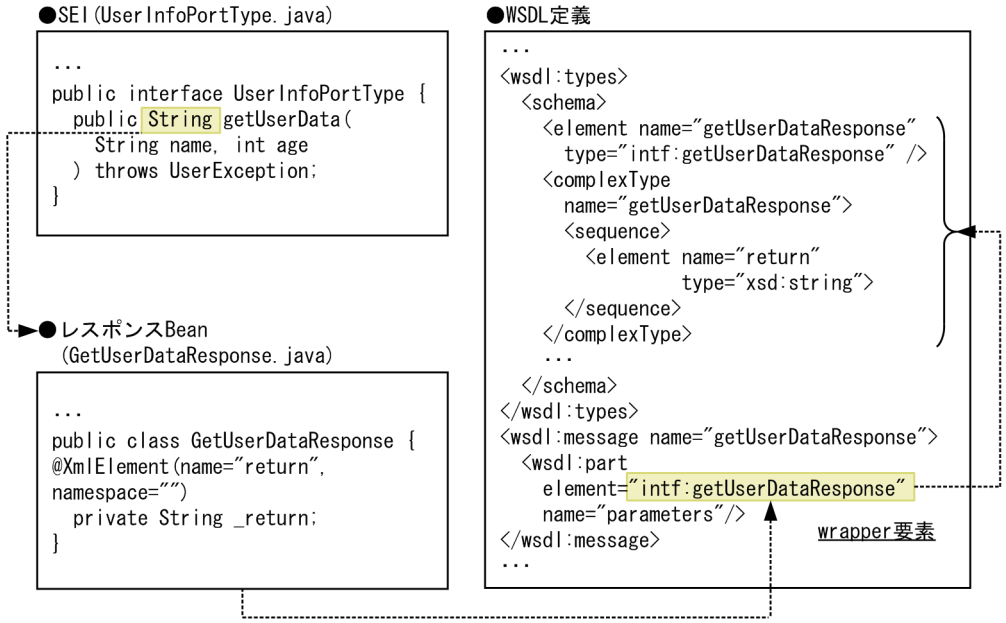
メソッドのパラメタとメッセージのパートのマッピング例を次の図に示します。

図 16-4　メソッドのパラメタとメッセージのパートのマッピング例（wrapper スタイル）



メソッドの戻り値とメッセージのパートのマッピング例を次の図に示します。

図 16-5　メソッドの戻り値とメッセージのパートのマッピング例（wrapper スタイル）



メソッドのパラメタおよび戻り値からメッセージのパートへのマッピング規則を次に示します。

- パラメタおよび戻り値は，wrapper 要素の子要素として空の名前空間（""）でマッピングされます。wrapper 要素は，SEI と同じ名前空間でマッピングされます。
- in パラメタおよび inout パラメタは，argN※という名称でリクエスト bean のプロパティとしてマッピングされます。
- out パラメタおよび inout パラメタは，argN※という名称でレスポンス bean のプロパティとしてマッピングされます。

また，戻り値は，return という名称で，レスポンス bean のプロパティとしてマッピングされます。このとき，予約語にならないように，フィールド名の接頭辞にはアンダースコア（_）が付加されます。

- マッピングされたリクエスト bean およびレスポンス bean のプロパティには，javax.xml.bind.annotation.XmlElement アノテーションがアノテートされます。
- アノテートされた javax.xml.bind.annotation.XmlElement アノテーションには，リクエスト bean のプロパティでは name 要素に argN※という名称が，namespace 要素に空の名前空間("")が設定されます。レスポンス bean のプロパティでは name 要素に return という名称が，namespace 要素に空の名前空間("")が設定されます。
- パラメタから WSDL のパートへのマッピングでは，"parameters"という固定値で input メッセージのパート名にマッピングされます。
- 戻り値から WSDL のパートへのマッピングでは，"parameters"という固定値で output メッセージのパート名にマッピングされます。

注※

argN の N は，パラメタの順番に依存した 0 以上の整数を表します。

javax.xml.bind.annotation.XmlElement アノテーションについては，「16.2.10 javax.xml.bind.annotation.XmlElement アノテーション」を参照してください。

(2) パラメタに指定できる Java 型

Holder（javax.xml.ws.Holder）型以外の Java 型と，Holder 型を指定するときの条件および注意事項について説明します。

(a) Holder 型以外の Java 型

Holder 型以外の Java 型は，JAXB 2.2 仕様に従って WSDL のスキーマの型にマッピングされます。
Holder 型以外の Java 型を指定するときの注意事項について説明します。

- Java プリミティブ型は，out および inout パラメタに指定できません。
- Java プリミティブ型は，javax.xml.ws.Holder クラスの型パラメタとしても指定できません。指定した場合，javac コマンドの実行時にエラーとなり終了します。
- Java 型を out および inout パラメタに指定する場合，javax.xml.ws.Holder クラスの型パラメタとして指定できます。

(b) javax.xml.ws.Holder 型

javax.xml.ws.Holder 型を指定するときの注意事項について説明します。

- javax.xml.ws.Holder クラスの型パラメタに，javax.jws.WebParam アノテーションを指定する場合，mode 要素で Mode.OUT または Mode.INOUT を指定する必要があります。
- 次の場合の動作は保証されません。

- javax.xml.ws.Holder クラスをメソッドの引数以外に指定した場合
- javax.xml.ws.Holder クラスの配列を使用した場合
- javax.xml.ws.Holder クラスの型パラメタに型を指定していない場合
- javax.xml.ws.Holder クラスの型パラメタに，javax.xml.ws.Holder クラスやそれを継承したクラスを指定した場合
- Web サービス開始時にリクエスト bean とレスポンス bean を動的に生成する際，javax.xml.ws.Holder クラスの型パラメタに byte 型以外の多次元配列と byte 型の 3 次元以上の配列を指定した場合

(c) Java 型のマッピング

Java 型をマッピングするときの注意事項について説明します。

- javax.jws.WebParam アノテーションの mode 要素でカスタマイズしていない場合，javax.xml.ws.Holder クラスの型パラメタ以外の引数は，in パラメタとしてマッピングされ，javax.xml.ws.Holder クラスの型パラメタは，inout パラメタとしてマッピングされます。out パラメタとしてマッピングする方法については，「16.2.7(4) mode 要素（javax.jws.WebParam）」を参照してください。
- input メッセージ名は，オペレーション名でマッピングされます。output メッセージ名は，オペレーション名に接尾辞"Response"を付加した値でマッピングされます。
- SEI のメソッドのパラメタは，Java の仕様に従って 254 個まで定義できます。255 個以上定義した場合は，javac コマンドの実行時にコンパイルエラーとなり終了します。

(3) Java メソッドのパラメタの条件

Java メソッドのパラメタ名は，WSDL にマッピングされないため，Java 言語仕様で定める Java 識別子の命名規則に従って記述してください。

(4) パラメタと戻り値の組み合わせ

in パラメタ，inout パラメタ，out パラメタ，および戻り値は，自由に組み合わせて記述できます。

(5) 名前衝突時の動作

wrapper bean クラス名とグローバル要素の名前についての規則，および名前衝突したときの動作について説明します。

- wrapper bean クラス名

  生成される wrapper bean クラス名は，パッケージ内でユニークな名前である必要があります。ただし，大文字／小文字の違いは無視されます。

  すでに存在するクラスと名前が重複していた場合は，上書きされます。ただし，そのクラスが javac コマンドの引数に含まれていた場合は，javac コマンド実行時にエラーチェックされます。

- **グローバル要素（ローカル名および名前空間）**

  グローバル要素（ローカル名および名前空間）は，WSDL 内でユニークである必要があります。ユニークでない場合の動作は保証されません。

(6) java.util.Map クラスの使用

SEI の引数または戻り値に java.util.Map クラスを使用する場合，SEI の java.util.Map 型の引数または戻り値に対して，次の作業をする必要があります。

1. **value type を作成します。**

   JAXB 2.2 仕様に従い，java.util.Map（bound type）に対応する value type（マーシャル／アンマーシャルできる JavaBean クラス）を作成します。

2. **アダプタを作成します。**

   javax.xml.bind.annotation.adapters.XmlAdapter を継承する java.util.Map(bound type)および value type を相互変換するアダプタを作成し，unmarshal メソッドおよび marshal メソッドを実装します。

3. **javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter アノテーションでアノテートします。**

   java.util.Map 型の引数または戻り値を，2.のアダプタを値に持つ XmlJavaTypeAdapter アノテーションでアノテートします。

4. **javac コマンドを実行します。**

   アノテート済みの SEI を javac コマンドでコンパイルします。

   value type，アダプタ，xmlJavaTypeAdapter アノテーションが適用されたリクエスト bean クラス／レスポンス bean クラスの関係と実装例を次に示します。

図 16-6　java.util.Map の使用例

●リクエストBean(GetUserData.java)

```
 ...
public class GetUserData {
 @XmlJavaTypeAdapter(MapAdapter.class)
 private Map<String,String> arg0;
  ...
}
```

●Mapに対応するJavaBean
(BindMapBean.java)

```
...
@XmlAccessorType(XmlAccessType.FIELD)
public class BindMapBean {
  @XmlElement
  private List<String> key;
  @XmlElement
  private List<String> value;

  public List<String> getKey() {
    if(key == null){
      key = new ArrayList<String>();
    }
    return key;
  }
  public List<String> getValue() {
    if(value == null){
      value = new ArrayList<String>();
    }
    return value;
  }
}
```

●アダプタ (MapAdapter.java)

```
import java.util.Iterator;
import java.util.Map;
import java.util.HashMap;
import
javax.xml.bind.annotation.adapters.XmlAdapter;

public class MapAdapter extends XmlAdapter
<BindMapBean,Map<String, String>>{
  public Map<String, String>
  unmarshal(BindMapBean bean) {
   Map<String, String> unmarshaled;
   unmarshaled = new HashMap<String, String>();
   Iterator<String> keyIte =
bean.getKey().iterator();
   Iterator<String> valueIte =
bean.getValue().iterator();
    while (keyIte.hasNext()) {
      unmarshaled.put(keyIte.next(),
valueIte.next());
    }
    return (unmarshaled);
  }

  public BindMapBean marshal(Map<String, String>
value) {
    BindMapBean marshaled = new BindMapBean();
    Iterator<String> iterator =
value.keySet().iterator();
    while (iterator.hasNext()) {
       String key = iterator.next();
       marshaled.getKey().add( key );
       marshaled.getValue().add( value.get( key )
);
       }
       return (marshaled);
  }
}
```

16.1.6　パラメタおよび戻り値からメッセージのパートへのマッピング（non-wrapper スタイルの場合）

SEI のメソッドのパラメタから WSDL（wsdl:part 要素の name 属性）へのマッピングについて説明します。

ここでは，non-wrapper スタイルの場合について説明します。

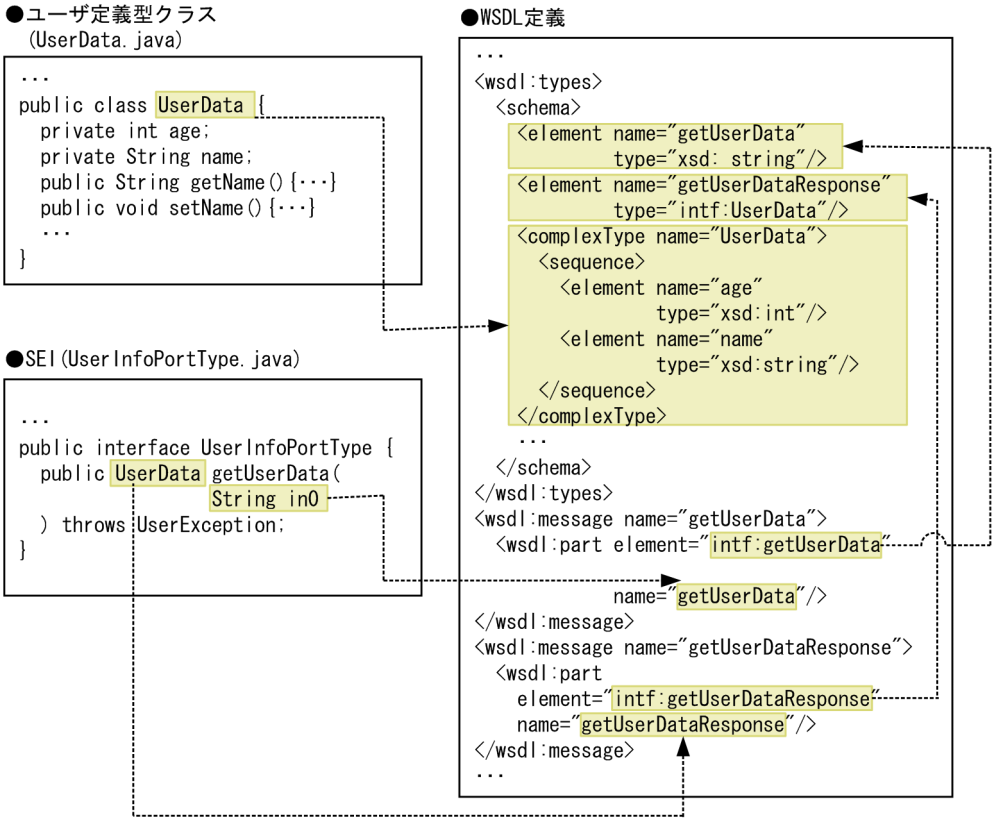
(1)　マッピング

non-wrapper スタイルの場合，SEI のメソッドパラメタはオペレーション名と同じ値の名称で，WSDL のパートおよびグローバル要素にマッピングされます。戻り値は，オペレーション名に接尾辞"Response" を付加した名称で，WSDL のパートおよびグローバル要素にマッピングされます。なお，non-wrapper スタイルでは，リクエスト bean およびレスポンス bean は必要ありません。このため，hwsgen コマンドによるリクエスト bean およびレスポンス bean の自動生成はしませんが，コンパイルした Web サー

ビス実装クラスに対して hwsgen コマンドを実行することで，事前にエラーチェックができます。詳細については，「10.23.1　hwsgen コマンドによるエラーチェックについて」を参照してください。

マッピング例を次の図に示します。

図 16–7　メソッドのパラメタおよび戻り値とメッセージのパートのマッピング例（non-wrapper スタイル）



(2)　パラメタに指定できる Java 型

パラメタに指定できる Java 型については，wrapper 型の場合と同様です。「16.1.5(2)　パラメタに指定できる Java 型」を参照してください。

(3)　パラメタ名の条件

Java メソッドのパラメタ名は，WSDL にマッピングされないため，Java 言語仕様で定める Java 識別子の命名規則に従って記述してください。

(4)　パラメタと戻り値の組み合わせ

SOAP ヘッダであるメソッドのパラメタは，幾つでも指定できます。ただし，SOAP ボディであるメソッドのパラメタは，戻り値との関係で指定可否および指定個数が決まります。メソッドの戻り値の条件と，パラメタの指定方法を次の表に示します。

表 16-6　メソッドの戻り値の条件とパラメタの指定方法

項番	戻り値の条件		パラメタの指定方法
	戻り値の有無	指定位置	
1	なし	－	• in パラメタおよび inout パラメタはどちらか 1 個指定できます。 • out パラメタおよび inout パラメタはどちらか 1 個指定できます。
2	あり	SOAP ヘッダ	• in パラメタおよび inout パラメタはどちらか 1 個指定できます。 • out パラメタおよび inout パラメタはどちらか 1 個指定できます。
3	あり	SOAP ボディ	• in パラメタを 1 個指定できます。※1 • out パラメタおよび inout パラメタは指定できません。

(5)　アノテーションの指定とパート名へのマッピングの関係

メソッドのパラメタから WSDL のパート（wsdl:part 要素の name 属性）へのマッピングは，javax.jws.WebParam アノテーションおよび javax.jws.WebMethod アノテーションの要素値の指定によって異なります。

アノテーションの指定内容と，マッピング方法の関係を次の表に示します。

表 16-7　アノテーションの指定と WSDL のパート名へのマッピングの関係

項番	アノテーションの要素指定の有無			WSDL のパート名へのマッピング方法
	javax.jws.WebParam @partName	javax.jws.WebParam @name	javax.jws.WebMethod @operationName	
1	指定あり	－	－	指定ありのアノテーションの要素値が wsdl:part 要素の name 属性にマッピングされます。
2	指定なし	指定あり	－	
3		指定なし	指定あり	Java のメソッド名が wsdl:part 要素の name 属性にマッピングされます。
4			指定なし	

（凡例）

－：要素の指定有無がマッピングに影響しないことを示します（指定しても指定しなくてもマッピング方法は同じです）。

(6)　名前衝突時の動作

パート名およびグローバル要素の名前衝突時の動作について説明します。

- パート名の衝突

  non-wrapper スタイルでは，Java ソースからマッピングされる WSDL の wsdl:part 要素の名前を WSDL 内でユニークにする必要があります。ユニークではない場合の動作は保証されません。

- グローバル要素（ローカル名および名前空間）の衝突

グローバル要素（ローカル名および名前空間）は WSDL 内でユニークにする必要があります。ユニークではない場合の動作は保証されません。

(7)　java.util.Map クラスの使用

java.util.Map クラスを使用する方法については，「16.1.5(6)　java.util.Map クラスの使用」を参照してください。

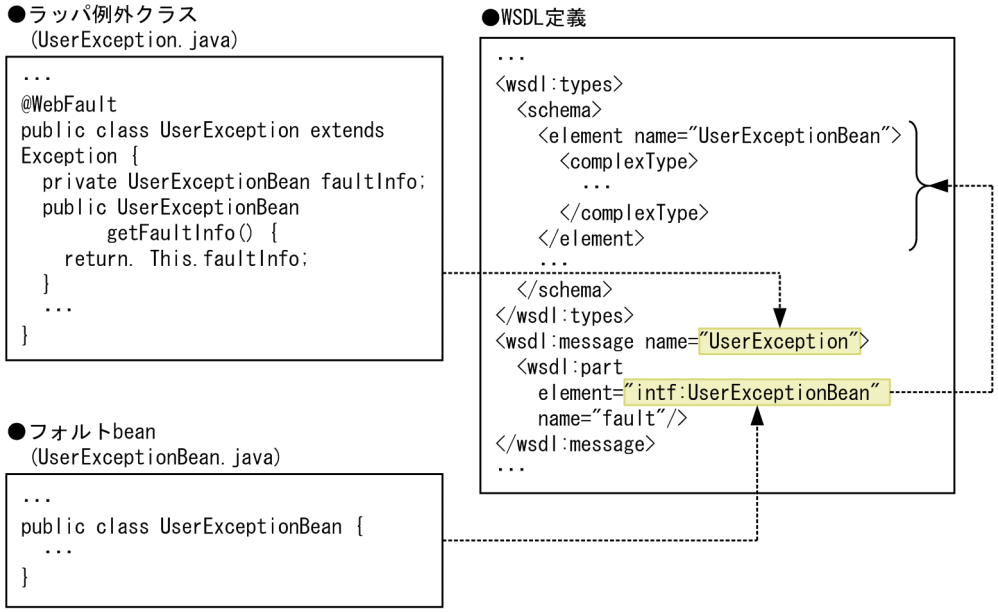
16.1.7　Java のラッパ例外クラスからフォルトへのマッピング

Java のラッパ例外クラスから WSDL のフォルト（wsdl:fault 要素，一つの wsdl:part 子要素を持つ wsdl:message 要素，および XML Schema のグローバル要素宣言）へのマッピングについて説明します。

(1)　マッピング

Java のラッパ例外クラスとフォルトは，JAX-WS 2.2 仕様に従ってマッピングされます。マッピング例を次の図に示します。

図 16-8　例外クラスとフォルトのマッピング例



ラッパ例外クラスからフォルトへのマッピング規則を次に示します。

- ラッパ例外クラスが javax.xml.ws.WebFault アノテーションを持つ場合で，かつフォルト bean を返す getFaultInfo メソッドを持つ場合，すでにフォルト bean があるため，コマンド実行時にフォルト bean は生成されません。
- ラッパ例外クラスが javax.xml.ws.WebFault アノテーションも getFaultInfo メソッドも持たない場合，ラッパ例外クラスの名前の接尾辞に"Bean"を付加した名前のフォルト bean が生成されます。

- 生成されるフォルト bean は，ラッパ例外クラスとその親クラスが持つ，Throwable から継承される getMessage という getter と同じ型／名称のプロパティを持ちます。
- 生成したフォルト bean には，javax.xml.bind.annotation.XmlType アノテーションがアノテートされます。

  アノテートされた javax.xml.bind.annotation.XmlType アノテーションには，name 要素に例外クラス名が，namespace 要素に SEI の名前空間が，propOrder 要素にラッパ例外クラスが持つすべてのプロパティ名が設定されます。propOrder 要素のプロパティ名は，各文字の Unicode 値に従って昇順にソートされた String 型配列で設定されます。

  javax.xml.bind.annotation.XmlType アノテーションについては，「16.2.12 javax.xml.bind.annotation.XmlType アノテーション」を参照してください。
- フォルトメッセージ名には，ラッパ例外クラス名と同じ値でマッピングされます。また，フォルトメッセージのパート名には，fault という固定値でマッピングされます。
- 一つのメソッドでスローする例外は，255 個まで定義できます。
- メソッドがスローする例外クラスが見つからない場合は，コンパイルエラーが発生します。

(2) ラッパ例外クラスの条件

ラッパ例外クラスの条件を示します。

- ラッパ例外クラスは，java.lang.Exception，java.lang.RuntimeException，java.rmi.RemoteException の例外クラスを継承してもかまいません。ただし，java.lang.RuntimeException と java.rmi.RemoteException，およびそのサブクラスはラッパ例外クラスとして扱われません。
- 同じ SEI 内の複数のメソッドで，同じラッパ例外クラスをスローしてもかまいません。

(3) ラッパ例外クラス名の条件

ラッパ例外クラス名には，次の表に示すすべての条件を満たす文字列を記述できます。ラッパ例外クラス名は，アノテーションを指定しても WSDL 内で使用されるため，次の表の条件に従う必要があります。

表 16-8 ラッパ例外クラス名に記述できる文字列の条件

項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z），およびアンダースコア（_）だけを使用した文字列	鈴木_exception	動作は保証されません（エラーメッセージは出力されません）。
2	Java 言語仕様で定められている Java 識別子の命名規則に従った文字列	abstract	javac コマンド実行時にコンパイルエラーとなり，終了します。

(4)　名前衝突時の動作

フォルト bean の名前は，パッケージ内でユニークにする必要があります。ただし，大文字／小文字の違いは無視されます。

フォルト bean からマッピングするグローバル要素（ローカル名および名前空間）は，WSDL 内でユニークにする必要があります。ユニークでない場合の動作は保証されません。

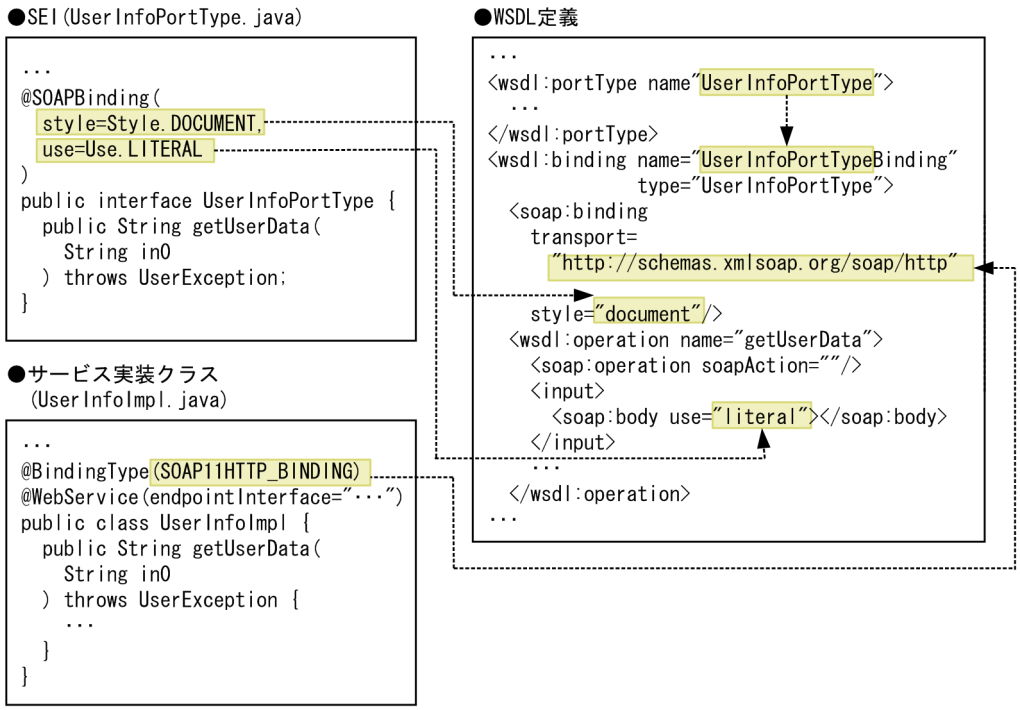
16.1.8　SEI からバインディングへのマッピング

Java の SEI から WSDL のバインディング（wsdl:binding 要素の name 属性）へのマッピングについて説明します。

(1)　マッピング

Java の SEI および Web サービス実装クラスと，WSDL のバインディングは，JAX-WS 2.2 仕様に従ってマッピングされます。マッピング例を次の図に示します。

図 16-9　SEI とバインディングのマッピング例



SEI からバインディングへのマッピング規則を次に示します。

- Java の SEI および Web サービス実装クラスは，WSDL の一つの wsdl:binding 要素と，0 個以上の wsdl:port 拡張要素にマッピングされます。
- WSDL のバインディング名は，ポートタイプ名の接頭辞に"Binding"を付加した名前になります。ポートタイプ名の形式については，「16.1.3　SEI 名からポートタイプへのマッピング」を参照してください。

(2) SOAP トランスポートと転送バインディング

javax.xml.ws.BindingType アノテーションでカスタマイズしていない場合，デフォルトマッピングでは，SOAP 1.1overHTTP でバインディングされます。

これは，wsdl:binding 要素の子要素である soap:binding 要素の transport 属性に，http://schemas.xmlsoap.org/soap/http が指定されることを意味します。

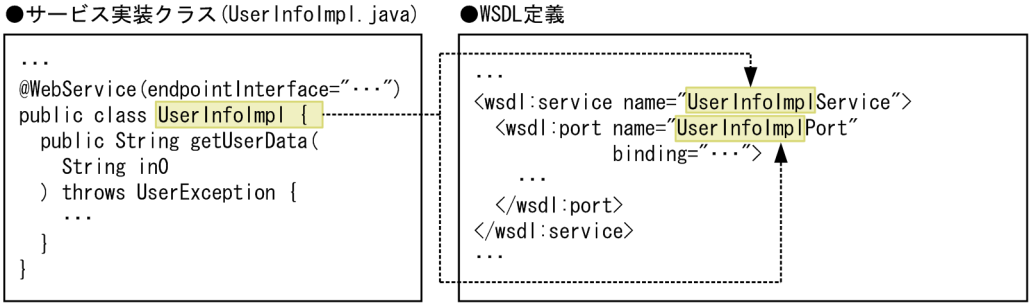
16.1.9 Web サービス実装クラスからサービスおよびポートへのマッピング

Web サービス実装クラスから WSDL のサービス（wsdl:service 要素の name 属性）およびポート（wsdl:port 要素の name 属性）へのマッピングについて説明します。

(1) マッピング

Web サービス実装クラスと，WSDL のサービスおよびポートは，JAX-WS 2.2 仕様に従ってマッピングされます。マッピング例を次の図に示します。

図 16-10 Web サービス実装クラスと，サービスおよびポートのマッピング例



Web サービス実装クラスからサービスおよびポートへのマッピング規則を次に示します。

- WSDL のサービス名を javax.jws.WebService アノテーションの serviceName 要素でカスタマイズしていない場合，Web サービス実装クラスの名前の接尾辞に，"Service"を付加したものを wsdl:service 要素の name 属性の値とします。
- WSDL のポート名を javax.jws.WebService アノテーションの portName 要素でカスタマイズしていない場合，javax.jws.WebService アノテーションの name 要素値の接尾辞に，"Port"を付加したものを wsdl:port 要素の name 属性の値とします。
- javax.jws.WebService アノテーションの portName 属性および name 属性でカスタマイズしていない場合，Web サービス実装クラスの名前の接尾辞に，"Port"を付加したものを wsdl:port 要素の name 属性の値とします。

(2) Web サービス実装クラス名の条件

Web サービス実装クラス名には，次の表に示すすべての条件を満たす文字列を記述できます。

表 16–9　Web サービス実装クラス名に記述できる文字列の条件

項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z），およびアンダースコア（_）だけを使用した文字列	鈴木_service	動作は保証されません（エラーメッセージは出力されません）。
2	Java 言語仕様で定められている Java 識別子の命名規則に従った文字列	abstract	javac コマンド実行時にコンパイルエラーとなり，終了します。

16.1.10　Java から WSDL へのマッピングに関する注意事項

Java から WSDL へのマッピングでの注意事項について説明します。

(1)　ジェネリクスの型削除

JavaBean（リクエスト bean，レスポンス bean，フォルト bean）生成時には，ジェネリクスの型が削除されます。ジェネリクスの型削除の例を次の表に示します。

表 16–10　ジェネリクスの型削除の例

型削除前	型削除後
T※	NumbersData
List<E>	List<java.lang.Object>
List<? extends NumbersData>	List<NumbersData>
List<? super NumbersData>	List< java.lang.Object >
Map<K, V>	Map< java.lang.Object, java.lang.Object >
Map<? extends NumbersKey, ? extends NumbersData>	Map<NumbersKey, NumbersData>
Map<? super NumbersKey, ? super NumbersData>	Map< java.lang.Object, java.lang.Object >
Iterator<E>	Iterator< java.lang.Object >
Iterator<? extends NumbersData>	Iterator<NumbersData>
Iterator<? super NumbersData>	Iterator< java.lang.Object >
List<List<? extends NumbersData>>	List<List<NumbersData>

注※

T の型定義は，<T extends NumbersData>の場合を示します。

メソッド引数や戻り値を javax.jws.WebParam や javax.jws.WebResult アノテーションでカスタマイズしている場合も，ジェネリクスの型は削除されます（カスタマイズも有効）。また，メソッド引数や戻り値が wrapper スタイルであっても，ジェネリクスの型は削除されます。なお，メソッド引数や戻り値が non-wrapper スタイルの場合にはジェネリクスの型は削除されません。

(2) JAXB アノテーションのサポートについて

Application Server の JAX-WS 機能は，JAX-WS 2.2 仕様の Comformance 3.14 に対応しています。コマンド実行時には，必要に応じて次に示す JAXB アノテーションが解釈されます。

- javax.xml.bind.annotation.XmlAttachmentRef
- javax.xml.bind.annotation.XmlList
- javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter
- javax.xml.bind.annotation.XmlMimeType

Application Server の JAX-WS 機能では MIME バインディングはサポートされません。

Application Server の JAX-WS 機能が提供するアノテーションプロセッサでは，javax.xml.bind.annotation.XmlList アノテーションは wrapper スタイルの場合には解釈されますが，non-wrapper スタイルの場合には解釈されません。

これらの JAXB アノテーションで，SEI およびサービス実装クラス以外の引数や戻り値をアノテートした場合の動作は保証されません。

Application Server の JAX-WS 機能が提供するアノテーションプロセッサでは，javax.xml.bind.annotation.XmlJavaTypeAdapter アノテーションおよび javax.xml.bind.annotation.XmlMimeType アノテーションは，SEI またはサービス実装クラスの引数や戻り値，または JavaBean のフィールドをアノテートした場合に解釈されます。パッケージや，インタフェース，またはクラスをアノテートした場合の動作は保証されません。

開発した Web サービスを呼び出すときに，Web サービスの引数や戻り値のサブクラスを使用するとエラーが発生します。正常に呼び出すには，SEI および Web サービス実装クラスを定義するときに，javax.xml.bind.annotation.XmlSeeAlso アノテーションによってサブクラスを関連づける必要があります。

(3) ジェネリクス型の制限

wrapper スタイルの場合にはメソッド引数や戻り値にジェネリクス型を使用できますが，non-wrapper スタイルの場合にはジェネリクス型を使用できません。

(4) ジェネリクスを使用したクラスまたはインタフェースの使用について

サービス実装クラスは，型変数をパラメタ化したクラスまたはインタフェースを継承できません。継承する場合，サービス実装クラスで再定義したメソッドに exclude 要素の要素値が"true"の javax.jws.WebMethod アノテーションをアノテートしてサービスメソッドから除外する必要があります。

型変数をパラメタ化したクラスまたはインタフェースを継承したサービス実装クラスをコンパイルすると，サービス実装クラス内で再定義したメソッド名と同じ名前の合成ブリッジ・メソッド（synthetic bridge methods）を JDK のコンパイラが暗黙的に生成します。コンパイルしたサービス実装クラスから生成したメタデータ（WSDL）にはユニークでなければならないオペレーション名を 2 つマッピングすることにな

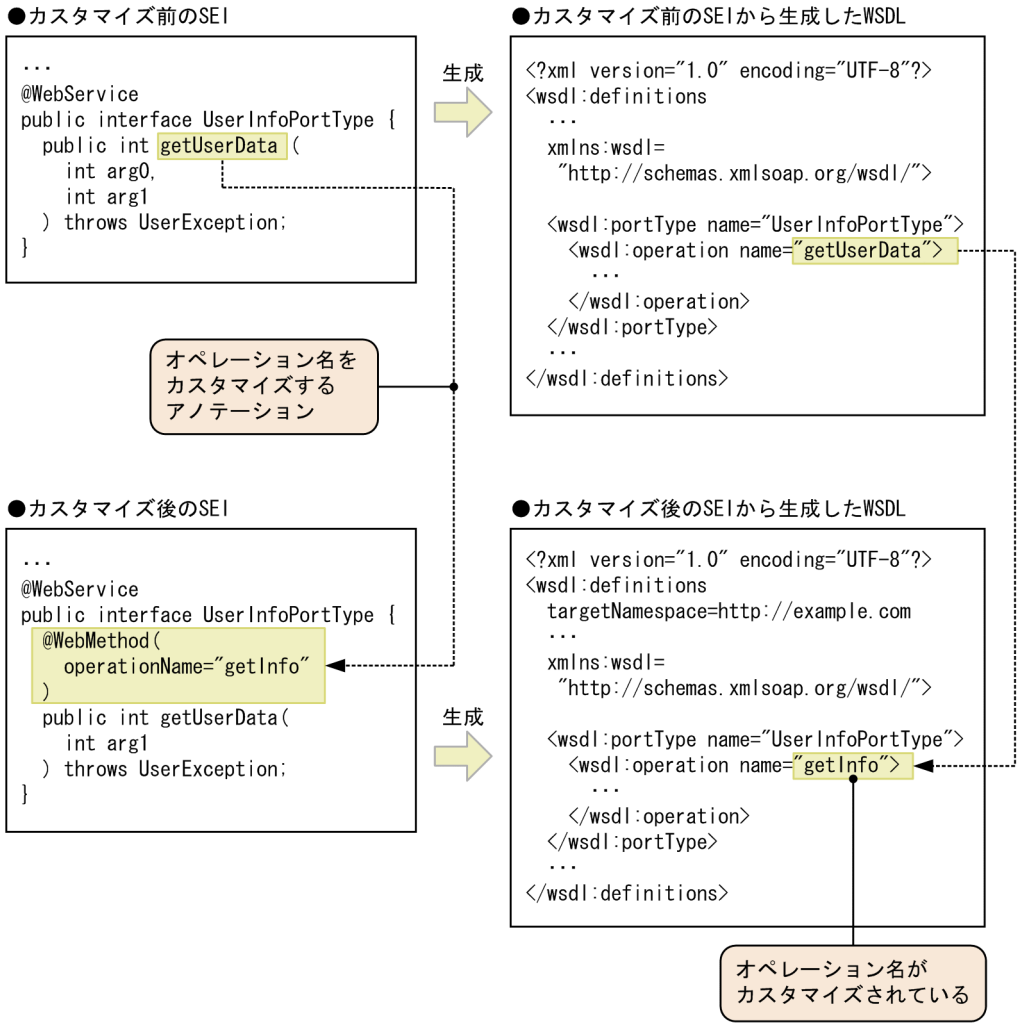
るため，メタデータ（WSDL）の解析に失敗します。オペレーション名のマッピングについては，「16.1.4 SEI のメソッド名からオペレーションへのマッピング」を参照してください。

16.2　Java から WSDL へのマッピングのカスタマイズ

アノテーションを使用することで，Java から WSDL へのマッピングをカスタマイズできます。

アノテーションを使用したカスタマイズ例を次の図に示します。

図 16-11　アノテーションを使用したカスタマイズ例



アノテーションには，SEI に定義できるもの，Web サービス実装クラスに定義できるもの，および両方に定義できるものがあります。ただし，javax.jws.WebService アノテーションの endpointInterface 要素を使用しない場合は，Web サービス実装クラスの情報から抽象的な情報が抽出されて暗黙の SEI があるものと見なされます。このときにかぎり，SEI に定義するアノテーションを，Web サービス実装クラスに定義することが許容されます。

アノテーションの要素を明示的にデフォルト値と同じ値でカスタマイズしても，要素値が指定されていない場合と同じように処理されます。

16.2.1　アノテーション一覧

カスタマイズ時に使用できるアノテーション，および自動生成されるアノテーションを次の表に示します。各アノテーションについては，JAX-WS 2.2 仕様を参照してください。

表 16–11　カスタマイズ時に使用する JAX-WS のアノテーションの一覧

項番	アノテーション		説明	定義の可否
	アノテーション名	要素名		
1	com.sun.xml.ws.developer.StreamingAttachment	dir	受信した添付ファイルを含む SOAP メッセージに含まれる MIME ボディを一時ファイルとして出力する際に，一時ファイルを生成するディレクトリ名です。	○
		parseEagerly	受信した添付ファイルを含む SOAP メッセージを詳細に解析するかどうかを設定します。	○
		memoryThreshold	受信した添付ファイルを含む SOAP メッセージに含まれる MIME ボディをメモリ上に展開するかどうかを判定するためのしきい値です。	○
2	javax.jws.HandlerChain	file	ハンドラチェインファイルの場所を指す URL および相対パスです。	○
3	javax.jws.Oneway	−	Web サービスで one-way オペレーションを使用する場合に指定します。	○
4	javax.jws.soap.SOAPBinding	parameterStyle	メソッドのパラメタのスタイル（Document wrapped または Document Bare スタイル）です。	○
		style	メッセージのエンコードスタイルです。Application Server の JAX-WS 機能では document だけ指定できます。	○
		use	メッセージのフォーマットスタイルです。Application Server の JAX-WS 機能では literal だけ指定できます。	○
5	javax.jws.WebMethod	action	SOAP アクションを決定する文字列です。	○
		exclude	Web メソッドとして，公開の可否を指定します。	○
		operationName	メソッドに一致する wsdl:operation 要素名です。	○
6	javax.jws.WebParam	header	メッセージのヘッダから，パラメタを取得するかどうかを表します。	○
		mode	パラメタの流れの方向（in，inout，および out）です。	○
		name	パラメタを表す XML 要素のローカル名です。	○

項番	アノテーション		説明	定義の可否
	アノテーション名	要素名		
		partName	パラメタを表す wsdl:part 要素名です。	○
		targetNamespace	XML の名前空間名です。	○
7	javax.jws.WebResult	header	メッセージのヘッダから，戻り値を取得するかどうかを表します。	○
		name	戻り値を表す XML 要素のローカル名です。	○
		partName	戻り値を表す wsdl:part 要素名です。	○
		targetNamespace	XML の名前空間名です。	○
8	javax.jws.WebService	endpointInterface	Web サービスの抽象 Web サービス規約を定義する，SEI の完全修飾名です。	○
		name	Web サービス名を表す wsdl:portType 要素名です。	○
		portName	Web サービスのポート名を表す wsdl:port 要素名です。	○
		serviceName	Web サービスのサービス名を表す wsdl:service 要素名です。	○
		targetNamespace	Web サービスの名前空間名です。	○
		wsdlLocation	Web サービスの WSDL を示す URL です。	○※1，※3
9	javax.xml.ws.Action	fault	アドレッシング・ヘッダの wsa:Action 要素の値です。Web サービスがフォルトメッセージを送信する場合に必要です。	○
		input	アドレッシング・ヘッダの wsa:Action 要素の値です。Web サービスがリクエストメッセージを受信する場合に必要です。	○
		output	アドレッシング・ヘッダの wsa:Action 要素の値です。Web サービスがレスポンスメッセージを送信する場合に必要です。	○
10	javax.xml.ws.BindingType	value	SEI を公開するときに使用するバインディングです。	○
11	javax.xml.ws.FaultAction	className	例外クラス名です。	○
		value	アドレッシング・ヘッダの wsa:Action 要素の値です。className に指定した例外クラス名に該当するフォルトメッセージを，Web サービスから送信する場合に必要です。	○
12	javax.xml.ws.RequestWrapper	className	リクエスト bean クラス名です。	○
		localName	対象要素のローカル名です。	○

項番	アノテーション		説明	定義の可否
	アノテーション名	要素名		
		targetNamespace	対象要素の名前空間名です。	○
		partName	リクエスト wrapper 要素を参照する input メッセージのパート名です。	○
13	javax.xml.ws.ResponseWrapper	className	レスポンス bean クラス名です。	○
		localName	対象要素のローカル名です。	○
		targetNamespace	対象要素の名前空間名です。	○
		partName	レスポンス wrapper 要素を参照する output メッセージのパート名です。	○
14	javax.xml.ws.ServiceMode	value	サービスモードです。	○
15	javax.xml.ws.soap.Addressing	enabled	アドレッシング機能を有効にするかどうかを設定します。	○
		required	Web サービスを呼び出す場合にアドレッシング・ヘッダを必須とするかどうかを設定します。	○
		response	エンドポイントが匿名のレスポンスまたは非匿名のレスポンスを必須とするかどうかを設定します。	○
16	javax.xml.ws.soap.MTOM	enabled	MTOM/XOP 仕様形式の添付ファイルを使用するかどうかを設定します。	○
		threshold	バイナリデータを MTOM/XOP 仕様形式の添付ファイルとして送信するためのしきい値を設定します。	○
17	javax.xml.ws.WebFault	faultBean	フォルト bean クラス名です。	○
		name	対象要素のローカル名です。	○
		targetNamespace	対象要素の名前空間名です。	○
		messageName	フォルトメッセージ名です。	○
18	javax.xml.ws.WebServiceProvider	targetNamespace	Web サービスの名前空間名です。	○
		portName	Web サービスのポート名です。	○
		serviceName	Web サービスのサービス名です。	○
		wsdlLocation	Web サービスの WSDL を示す URL です。	○※3
19	javax.xml.ws.RespectBinding※1	enabled	wsdl:binding 要素の内容が有効かどうかを表します。	○※1
20	javax.xml.ws.WebEndpoint※2	name	ポートのローカル名です。	×※2

項番	アノテーション		説明	定義の可否
	アノテーション名	要素名		
21	javax.xml.ws.WebServiceClient ※2	name	Web サービスのローカル名です。	×※2
		targetNamespace	Web サービスの名前空間名です。	×※2
		wsdlLocation	Web サービスの WSDL を示す URL です。	×※2, ※3

（凡例）

－：要素がないことを示します。

○：アノテーションおよび要素が指定できることを示します。

×：アノテーションおよび要素が指定できないことを示します（非サポート）。

注※1

指定した値は無視されます（警告メッセージは表示されません）。

注※2

WSDL から生成されたクラスに自動的に付与されるアノテーションになるため，指定できません。

注※3

カタログ機能でのマッピングはサポートしていません。

Application Server の Web サービスで定義できる JAXB のアノテーションの一覧を次に示します。なお，各アノテーションについては，JAXB の標準仕様を参照してください。

- javax.xml.bind.annotation.XmlAttachmentRef
- javax.xml.bind.annotation.XmlList
- javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter
- javax.xml.bind.annotation.XmlMimeType
- javax.xml.bind.annotation.XmlSeeAlso
- javax.xml.bind.annotation.XmlElement
- javax.xml.bind.annotation.XmlType

JSR-181 仕様および JAX-WS 2.2 仕様のアノテーションのうち，次の表に示すアノテーションは指定できません。指定した場合，動作は保証されません。

表 16–12　サポート外のアノテーションの一覧

項番	アノテーション	備考
1	javax.xml.ws.WebServiceRefs	JSR109 仕様に関連する機能はサポートしていません。
2	javax.xml.ws.spi.WebServiceFeatureAnnotation	このアノテーションを使用するための API（フィーチャをパラメタに取るメソッド）はサポートしていません。
3	javax.jws.soap.InitParam	JSR181 仕様で推奨していません（廃止対象）。
4	javax.jws.soap.SOAPMessageHandler	JSR181 仕様で推奨していません（廃止対象）。

項番	アノテーション	備考
5	javax.jws.soap.SOAPMessageHandlers	JSR181 仕様で推奨していません（廃止対象）。

16.2.2 com.sun.xml.ws.developer.StreamingAttachment アノテーション

com.sun.xml.ws.developer.StreamingAttachment アノテーションについて，次に説明します。

Web サービス側

ストリーミングを使用する Web サービスに指定するアノテーションです。Web サービス実装クラスだけに指定できます。SEI に指定した場合は無視されます。また，プロバイダ実装クラスに指定した場合，動作は保証されません。

Web サービスクライアント側

ポートをインジェクトするフィールドまたは setter メソッドに指定するアノテーションです。詳細については，「10.21.1(4)　フィーチャの有効化」を参照してください。それ以外のフィールドやメソッドに指定した場合は無視されます。

cjwsimport コマンドが生成する Web サービス実装クラスのスケルトンクラスを使用して，ストリーミングを使用する Web サービスを作成する場合，Web サービス実装クラスのスケルトンクラスには com.sun.xml.ws.developer.StreamingAttachment アノテーションをマッピングしないので，com.sun.xml.ws.developer.StreamingAttachment アノテーションを指定する必要があります。なお，Web サービス実装クラスに com.sun.xml.ws.developer.StreamingAttachment アノテーションを指定しても，サービス側 JAX-WS エンジンが発行する WSDL ファイルや hwsgen コマンドが生成する WSDL ファイルには，ストリーミングが使用されていることを示す要素や属性は出現しません。

com.sun.xml.ws.developer.StreamingAttachment アノテーションは Web サービス開始時に参照する値であるため，hwsgen コマンドの実行時には解釈しません。

com.sun.xml.ws.developer.StreamingAttachment アノテーションを使用した例を次の図に示します。

図 16-12　com.sun.xml.ws.developer.StreamingAttachment アノテーションを使用した例（Web サービス側）

●Webサービス実装クラス

```
・・・・・
@StreamingAttachment(dir="C:/TMP", parseEagerly=true,
memoryThreshold=50000L)
@WebService(endpointInterface =
"jaxwstp.example.service.ExamplePortType", targetNamespace = "http://
service.example.jaxwstp/", serviceName = "ExampleService", portName =
"ExamplePort")
public class ExampleBinding implements ExamplePortType {
```

(1) dir 要素（com.sun.xml.ws.developer.StreamingAttachment)

dir 要素は，Web サービスでストリーミングを使用する場合に，受信した添付ファイルを含む MIME Multipart/Related 構造の SOAP メッセージに含まれる MIME ボディを一時ファイルとして出力するときに使用するディレクトリを指定します。デフォルト値は空文字（"")です。

dir 要素の要素値に空文字("")または null を指定した場合，一時ファイルの出力先は Java のデフォルトの一時ファイルディレクトリ（システムプロパティにある"java.io.tmpdir"キーに対応する値）となります。また，存在しないディレクトリ名，アクセス権のないディレクトリ名，または存在するファイル名を指定した場合，Web サービス開始時にメッセージを出力し，受信した添付ファイルを含む MIME Multipart/Related 構造の SOAP メッセージに含まれる MIME ボディをメモリ上に展開します（KDJW10026-W)。

(2) parseEagerly 要素 (com.sun.xml.ws.developer.StreamingAttachment)

parseEagerly 要素は，Web サービスでストリーミングを使用する際に，受信した添付ファイルを含む MIME Multipart/Related 構造の SOAP メッセージを詳細に解析するかどうかを指定します。デフォルト値は false です。

(3) memoryThreshold 要素 (com.sun.xml.ws.developer.StreamingAttachment)

memoryThreshold 要素は，Web サービスでストリーミングを使用する際に，受信した添付ファイルを含む MIME Multipart/Related 構造の SOAP メッセージに含まれる MIME ボディを一時ファイルとして出力するかどうかを判定するためのしきい値(単位：バイト)を指定します。memoryThreshold 要素には，16KB（16384L）より大きい値または-1 を指定します。これら以外の値を指定した場合，動作は保証されません。-1 を指定した場合，常に MIME ボディをメモリ上に展開します。デフォルト値は「1048576L」です。

受信した MIME Multipart/Related 構造の SOAP メッセージに含まれる MIME ボディをメモリ上に展開するかどうかの判定については，「32.3　一時ファイル（ストリーミング)」を参照してください。

16.2.3　javax.jws.HandlerChain アノテーション

javax.jws.HandlerChain アノテーションは，クラスまたはインタフェース宣言に定義した場合に有効になります。

SEI と Web サービス実装クラスに同時に javax.jws.HandlerChain アノテーションを指定した場合，Web サービス実装クラスに指定したアノテーションが優先されます。

(1) file 要素（javax.jws.HandlerChain）

file 要素では，ハンドラチェイン設定ファイルを指定します。javax.jws.HandlerChain アノテーションでアノテートしたクラス，またはインタフェースからの相対パスで指定します。Application Server の JAX-WS 機能では，URL 形式での指定はサポートしていません。

参照できなかったり，開けなかったりするパスを指定した場合は，Web サービスの初期化時に，標準エラー出力とログにエラーメッセージが出力されます（KDJW00010-E）。

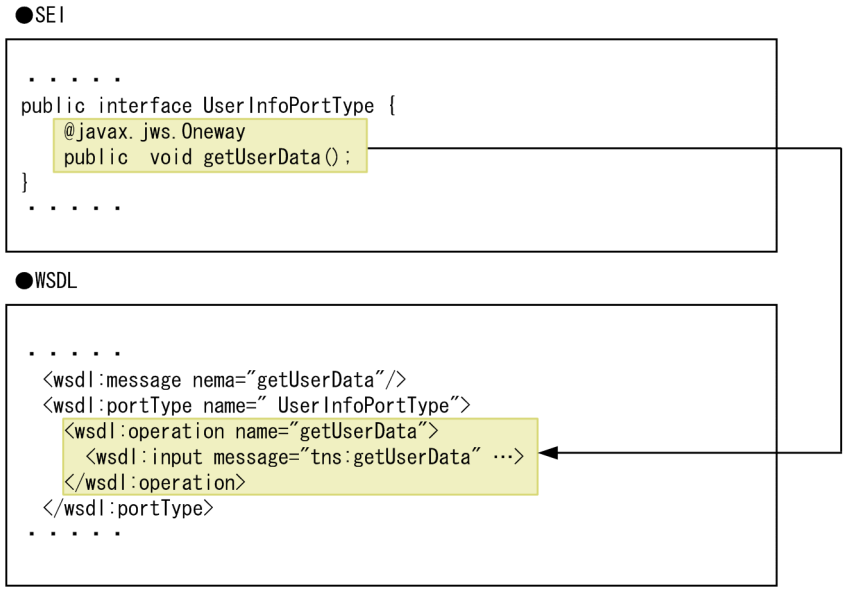
なお，file 要素は，Web サービスの呼び出しで参照される値になるため，hwsgen コマンドの実行時には解釈されません。file 要素の使用例については，「36.9.1　Web サービス側のハンドラチェインの設定」を参照してください。

16.2.4　javax.jws.Oneway アノテーション

javax.jws.Oneway アノテーションは，Web サービスで one-way オペレーションを使用する場合に指定します。javax.jws.Oneway アノテーションをアノテートした Web サービス実装クラスのメソッドは，入力メッセージだけを持ち，出力メッセージは持ちません。

javax.jws.Oneway アノテーションを使用したマッピング例を次の図に示します。

図 16–13　javax.jws.Oneway アノテーションを使用したマッピング例



javax.jws.Oneway アノテーションをアノテートした Web サービス実装クラスのメソッドは，次に示す条件を満たす必要があります。

- 戻り値の型は void を指定してください。
- BARE スタイルの Web サービスの場合，SOAP ヘッダ以外の in パラメタを一つだけ指定してください。

- javax.jws.WebParam アノテーションの Mode 要素を使用する場合，javax.jws.WebParam.Mode.IN を使用してください。
- Web サービス実装クラスのメソッドで，java.lang.RuntimeException と java.rmi.RemoteException およびそのサブクラス以外の例外が，一つも宣言されていないことを確認してください。
- 引数の型を Holder 型で指定しないでください。

javax.jws.Oneway アノテーションを使用するときの注意事項について説明します。

- Web サービス実装クラスで，javax.jws.Oneway アノテーションと javax.xml.ws.soap.Addressing アノテーションを併用しないでください。one-way オペレーションはレスポンスメッセージが存在しませんが，アドレッシング機能はレスポンスメッセージの送信先を指定します。このため，同時に使用した場合の動作は保証されません。
- one-way オペレーションは応答の SOAP メッセージが存在しないため，Web サービス実装クラスの javax.jws.Oneway アノテーションをアノテートしたメソッドの実装では，例外を明示的にスローしないようにしてください。

16.2.5 javax.jws.soap.SOAPBinding アノテーション

javax.jws.soap.SOAPBinding アノテーションは，SOAP メッセージのプロトコルのマッピングをカスタマイズするときに使用できます。

(1) style 要素（javax.jws.soap.SOAPBinding）

Application Server の JAX-WS 機能が提供するアノテーションプロセッサでは，DOCUMENT/LITERAL スタイルだけ使用できます。

javax.jws.soap.SOAPBinding アノテーションを，クラスまたはインタフェース宣言とメソッド宣言で同時に指定した場合，メソッド宣言で指定した値が優先されます。

(2) use 要素（javax.jws.soap.SOAPBinding）

Application Server の JAX-WS 機能が提供するアノテーションプロセッサでは，DOCUMENT/LITERAL スタイルだけ使用できます。

(3) parameterStyle 要素（javax.jws.soap.SOAPBinding）

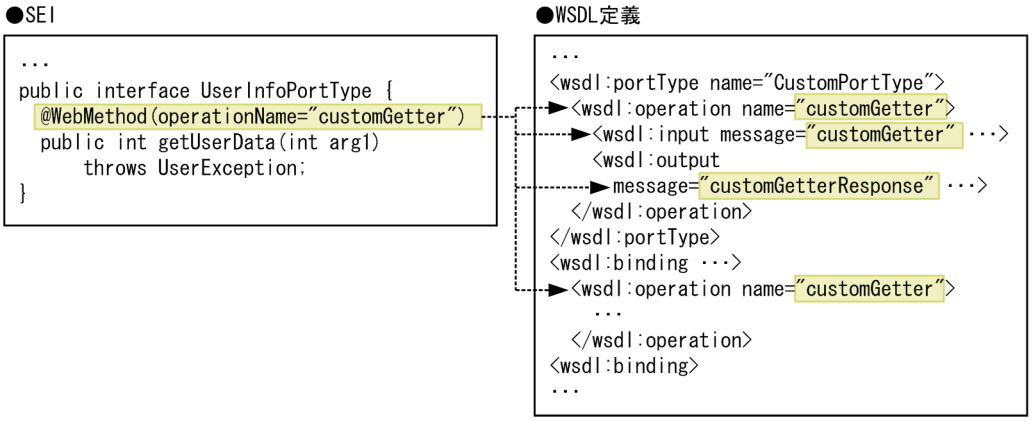
wrapper スタイルの場合，parameterStyle 要素に SOAPBinding.ParameterStyle.WRAPPED を指定します。また，non-wrapper スタイルの場合，parameterStyle 要素に SOAPBinding.ParameterStyle.BARE を指定します。

16.2.6　javax.jws.WebMethod アノテーション

javax.jws.WebMethod アノテーションは，オペレーションのマッピングをカスタマイズするときに使用できます。

javax.jws.WebMethod アノテーションを使用したカスタマイズ例を次の図に示します。

図 16-14　javax.jws.WebMethod アノテーションを使用したカスタマイズ例



(1)　exclude 要素（javax.jws.WebMethod）

Web サービス実装クラスが持つ public メソッドのうち，オペレーションとして公開したくない public メソッドは，exclude 要素の要素値が"true"の javax.jws.WebMethod アノテーションでアノテートすることで，マッピングされるオペレーションから除外できます。継承した親クラスが持つメソッドの場合，そのメソッドをオーバーライドし，exclude 要素の要素値が"true"の javax.jws.WebMethod アノテーションでアノテートすることで，マッピングされるオペレーションから除外できます。

exclude 要素を指定するときの注意事項について説明します。

- Web サービス実装クラスでこの要素の要素値に"true"を指定した場合，Web サービス実装クラスの javax.jws.WebMethod アノテーションに，ほかの要素を指定できません。
- SEI でこの要素を指定する場合，要素値には"false"を指定してください。
- Web サービス実装クラスで親クラスのメソッドをオーバーライドしている場合で，親クラスと Web サービス実装クラス（子クラス）の両方にこの要素を指定した場合，Web サービス実装クラス（子クラス）に指定した要素が優先されます。
- Web サービス実装クラスで javax.jws.WebService アノテーションの endpointInterface 要素を指定していない場合，Web サービス実装クラスが持つ static 修飾子または final 修飾子を適用していないすべての public メソッドが暗黙の SEI にマッピングされます。また，Web サービス実装クラスが持つ static 修飾子または final 修飾子を適用していない public メソッドに，exclude 要素が"true"ではない javax.jws.WebMethod アノテーションを一つでも指定しているとき，次の表の条件を満たす Web サービス実装クラスのメソッドが，暗黙の SEI にマッピングされます。

表 16–13　アノテーションの指定と暗黙の SEI へのマッピング

項番	Web サービス実装クラスのメソッド（WebMethod アノテーション）	暗黙の SEI が持つメソッド
1	なし	マッピングされます。
2	あり（exclude 要素なし）	マッピングされます。
3	exclude=false	マッピングされます。
4	exclude=true	マッピングされません。

(2)　operationName 要素（javax.jws.WebMethod）

operationName 要素は，オペレーション名のマッピングをカスタマイズするときに指定します。operationName 要素に要素値を指定すると，オペレーション名をマッピングのデフォルト値に持つメッセージ名がカスタマイズできます。non-wrapper スタイルの場合は，グローバル要素名およびパート名もカスタマイズできます。

operationName 要素を指定するときの注意事項について説明します。

- operationName 要素は半角英数字とアンダースコア（_）で指定します。それ以外の文字を指定した場合の動作は保証されません（エラーメッセージは出力されません）。
- Java 言語仕様で規定される Java 識別子の命名規則に従った値を指定してください。Java 識別子の命名規則に従っていない場合，Web サービスクライアントの開発で，cjwsimport コマンドを実行するときにコンパイルエラーとなります。
- wrapper スタイルの場合，javax.xml.ws.RequestWrapper アノテーションの localName 要素は「オペレーション名」と同じである必要があります。ただし，javax.xml.ws.RequestWrapper アノテーションが存在しない場合，または javax.xml.ws.RequestWrapper アノテーションの localName 要素に空文字（""）が指定されている場合，同一かどうかの比較処理は行われません。

(3)　action 要素（javax.jws.WebMethod）

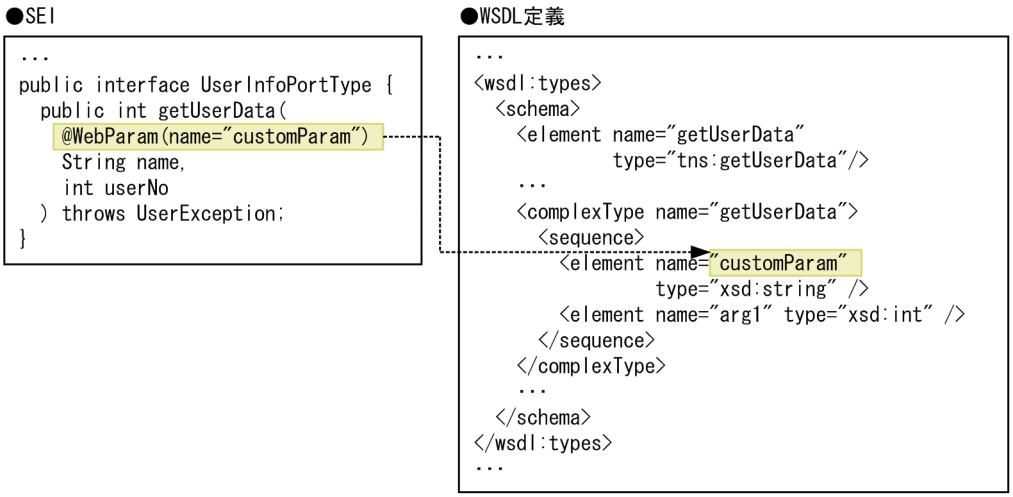
action 要素は，SOAP オペレーションのアクションにマッピングされます。action 要素には，RFC 2396 で規定された xsd:anyURI を満たす文字を指定できますが，指定した値はランタイムでは無視されます。

16.2.7　javax.jws.WebParam アノテーション

javax.jws.WebParam アノテーションは，引数のマッピングをカスタマイズするときに使用できます。

javax.jws.WebParam アノテーションを使用したカスタマイズ例を次の図に示します。

図 16–15　javax.jws.WebParam アノテーションを使用したカスタマイズ例



(1)　header 要素（javax.jws.WebParam）

引数をヘッダパラメタとしてマッピングする場合，header 要素の要素値に"true"を指定します。

header 要素は，non-wrapper スタイルで指定できます。

(2)　name 要素（javax.jws.WebParam）

name 要素は，wrapper スタイルの場合，引数からマッピングする wrapper 要素の子要素の名前をカスタマイズするときに使用します。non-wrapper スタイルの場合，引数からマッピングするグローバル要素のローカル名をカスタマイズするときに使用します。non-wrapper スタイルで partName 要素を指定していない場合，name 要素の要素値を指定することで，パート名もカスタマイズできます。

name 要素を指定するときの注意事項について説明します。

- name 要素は半角英数字とアンダースコア（_）で指定します。それ以外の文字を指定した場合の動作は保証されません（エラーメッセージは出力されません）。
- wrapper スタイルの場合，Java 言語仕様で規定される Java 識別子の命名規則に従った値を指定してください。Java 識別子の命名規則に従っていない場合は，Web サービスクライアントの開発で cjwsimport コマンドを実行するときにコンパイルエラーとなります。

(3)　partName 要素（javax.jws.WebParam）

partName 要素は，パート名のマッピングをカスタマイズするときに指定します。

partName 要素を指定するときの注意事項について説明します。

- partName 要素は，non-wrapper スタイルで有効です。wrapper スタイルで partName 要素を指定した場合は無視されます。
- partName 要素と name 要素を同時に指定した場合，有効となる要素を次に示します。

  - wrapper スタイルの場合：name 要素の値が有効になります。
  - non-wrapper スタイルの場合：partName 要素の値が有効になります。
- partName 要素は半角英数字とアンダースコア（_）で指定します。それ以外の文字を指定した場合の動作は保証されません（エラーメッセージは出力されません）。
- non-wrapper スタイルの場合，Java 言語仕様で規定される Java 識別子の命名規則に従った値を指定してください。Java 識別子の命名規則に従っていない場合は，Web サービスクライアントの開発で cjwsimport コマンドを実行するときにコンパイルエラーとなります。

(4) mode 要素（javax.jws.WebParam）

mode 要素では，パラメタの流れる方向を表す値を指定します。指定できる値を次に示します。

- WebParam.Mode.IN
- WebParam.Mode.OUT
- WebParam.Mode.INOUT

(5) targetNamespace 要素（javax.jws.WebParam）

targetNamespace 要素は，引数からマッピングするグローバル要素の名前空間をカスタマイズするときに使用します。

targetNamespace 要素には，http://または urn:のプロトコルを名前空間として指定します。指定できる名前空間の形式および文字列を示します。

- **プロトコル**
  名前空間のプロトコルは，http://または urn:のプロトコルで記述してください。
- **名前空間の記述形式**
  名前空間には次に示す形式は記述できません。
  - クエリストリング（例）http://example.com/?a=b
  - アンカー（例）http://example.com/index.html#anchor
  - ポート番号（例）http://example.com:8080/
  - ユーザ名／パスワード（例）http://user:password@example.com
- **記述できる文字列**
  区切り文字のスラッシュ（/）またはピリオド（.）で区切られたセグメントには，次の表に示すすべての条件を満たす文字列を記述できます。

表 16-14　名前空間に記述できる文字列の条件（javax.jws.WebParam）

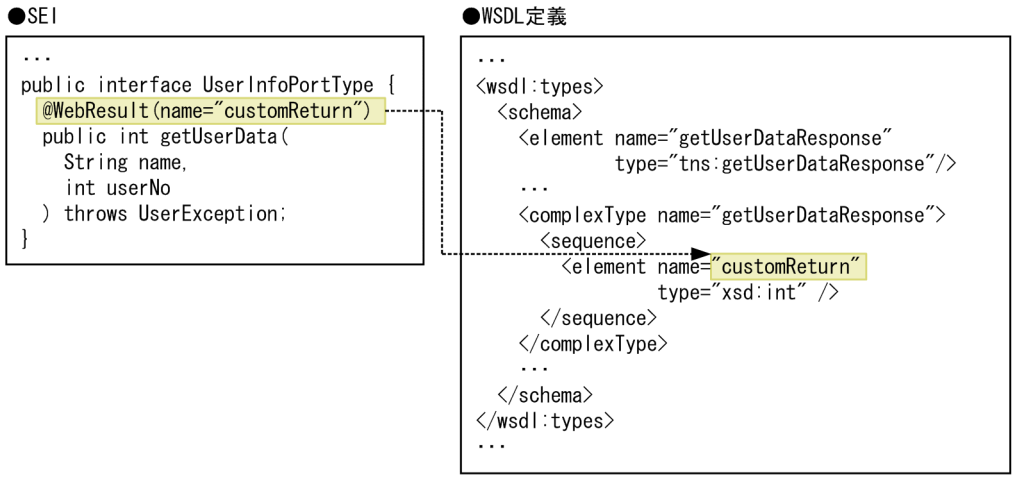
項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z）だけを使用した文字列	http://鈴木.com http://133.145.224.19/ http://[1080:2C14;D30:BA04:275:806:270C:418A]/	動作は保証されません（エラーメッセージは出力されません）。
2	Java の予約語以外の文字列	http://xxx.com/abstract	動作は保証されません。
3	先頭が数字でない文字列	http://1xxx.com	

16.2.8　javax.jws.WebResult アノテーション

javax.jws.WebResult アノテーションは，戻り値のマッピングをカスタマイズするときに使用できます。

javax.jws.WebResult アノテーションを使用したカスタマイズ例を次の図に示します。

図 16-16　javax.jws.WebResult アノテーションを使用したカスタマイズ例



(1)　header 要素（javax.jws.WebResult）

戻り値をヘッダパラメタとしてマッピングする場合，header 要素の要素値に"true"を指定します。

header 要素は，non-wrapper スタイルで指定できます。

(2)　name 要素（javax.jws.WebResult）

name 要素は，wrapper スタイルの場合，戻り値からマッピングする wrapper 要素の子要素の名前をカスタマイズするときに使用します。non-wrapper スタイルの場合，引数からマッピングするグローバル要

素のローカル名をカスタマイズするときに使用します。non-wrapper スタイルで partName 要素を指定していない場合，name 要素の要素値を指定することで，パート名もカスタマイズできます。

name 要素を指定するときの注意事項について説明します。

- name 要素は半角英数字とアンダースコア（_）で指定します。それ以外の文字を指定した場合の動作は保証されません（エラーメッセージは出力されません）。
- wrapper スタイルの場合，Java 言語仕様で規定される Java 識別子の命名規則に従った値を指定してください。Java 識別子の命名規則に従っていない場合は，Web サービスクライアントの開発で cjwsimport コマンドを実行するときにコンパイルエラーとなることがあります。
- 引数が 254 個ある wrapper スタイルのメソッドで，name 要素を"return"以外の値にカスタマイズした場合，cjwsimport コマンドの実行時に引数の数が 255 個になり，コンパイルエラーとなるおそれがあります。

(3) partName 要素（javax.jws.WebResult）

partName 要素は，パート名のマッピングをカスタマイズするときに指定します。

partName 要素を指定するときの注意事項について説明します。

- partName 要素は，non-wrapper スタイルで有効です。wrapper スタイルで partName 要素を指定した場合は無視されます。
- partName 要素と name 要素を同時に指定した場合，有効となる要素を次に示します。
  - wrapper スタイルの場合：name 要素の値が有効になります。
  - non-wrapper スタイルの場合：partName 要素の値が有効になります。
- partName 要素は半角英数字とアンダースコア（_）で指定します。それ以外の文字を指定した場合の動作は保証されません（エラーメッセージは出力されません）。
- non-wrapper スタイルの場合，Java 言語仕様で規定される Java 識別子の命名規則に従った値を指定してください。Java 識別子の命名規則に従っていない場合は，Web サービスクライアントの開発で cjwsimport コマンドを実行するときにコンパイルエラーとなります。
- 引数が 254 個ある wrapper スタイルのメソッドで，name 要素を"return"以外の値にカスタマイズした場合，cjwsimport コマンドの実行時に引数の数が 255 個になり，コンパイルエラーとなるおそれがあります。

(4) targetNamespace 要素（javax.jws.WebResult）

targetNamespace 要素は，戻り値からマッピングするグローバル要素の名前空間をカスタマイズするときに使用します。

targetNamespace 要素には，http://または urn:のプロトコルを名前空間として指定します。指定できる名前空間の形式および文字列を示します。

- **プロトコル**

  名前空間のプロトコルは，http://または urn:のプロトコルで記述してください。

- **名前空間の記述形式**

  名前空間には次に示す形式は記述できません。

  - クエリストリング（例）http://example.com/?a=b
  - アンカー（例）http://example.com/index.html#anchor
  - ポート番号（例）http://example.com:8080/
  - ユーザ名／パスワード（例）http://user:password@example.com

- **記述できる文字列**

  区切り文字のスラッシュ（/）またはピリオド（.）で区切られたセグメントには，次の表に示すすべての条件を満たす文字列を記述できます。

表 16-15　名前空間に記述できる文字列の条件（javax.jws.WebResult）

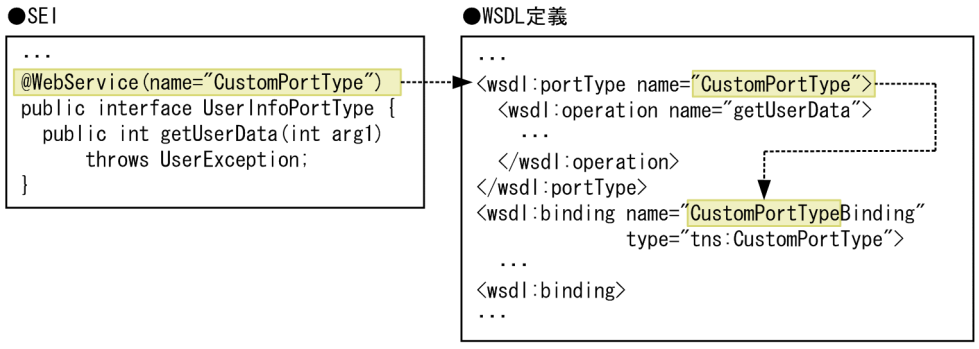
項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z）だけを使用した文字列	http://鈴木.com http://133.145.224.19/ http://[1080:2C14;D30:BA04:275:806:270C:418A]/	動作は保証されません（エラーメッセージは出力されません）。
2	Java の予約語以外の文字列	http://xxx.com/abstract	動作は保証されません。
3	先頭が数字でない文字列	http://1xxx.com	

16.2.9　javax.jws.WebService アノテーション

javax.jws.WebService アノテーションは，SEI および Web サービス実装クラスに必須です。

javax.jws.WebService アノテーションを使用したカスタマイズ例を次の図に示します。

図 16-17　javax.jws.WebService アノテーションを使用したカスタマイズ例



なお，javax.xml.ws.WebServiceProvider アノテーションと javax.jws.WebService アノテーションは，どちらか一方しか指定できません。

(1) targetNamespace 要素（javax.jws.WebService）

SEI で targetNamespace 要素を指定した場合，指定した名前空間は，types 要素，message 要素，および portType 要素に対して有効です。

Web サービス実装クラスでこの要素を指定した場合，指定した名前空間は，binding 要素および service 要素に対して有効です。Web サービス実装クラスで，javax.jws.WebService アノテーションの endpointInterface 要素を使用しない場合，暗黙の SEI と Web サービス実装クラスの両方に同じ名前空間を指定したものと見なされます。

targetNamespace 要素には，http://または urn:のプロトコルを名前空間として指定します。指定できる名前空間の形式および文字列を示します。

- **プロトコル**
  名前空間のプロトコルは，http://または urn:のプロトコルで記述してください。
- **名前空間の記述形式**
  名前空間には次に示す形式は記述できません。
  - クエリストリング（例）http://example.com/?a=b
  - アンカー（例）http://example.com/index.html#anchor
  - ポート番号（例）http://example.com:8080/
  - ユーザ名／パスワード（例）http://user:password@example.com
- **記述できる文字列**
  区切り文字のスラッシュ（/）またはピリオド（.）で区切られたセグメントには，次の表に示すすべての条件を満たす文字列を記述できます。

表 16–16　名前空間に記述できる文字列の条件（javax.jws.WebService）

項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z）だけを使用した文字列	http://鈴木.com http://133.145.224.19/ http://[1080:2C14;D30:BA04:275:806:270C:418A]/	動作は保証されません（エラーメッセージは出力されません）。
2	Java の予約語以外の文字列	http://xxx.com/abstract	動作は保証されません。
3	先頭が数字でない文字列	http://1xxx.com	

(2) endpointInterface 要素（javax.jws.WebService）

endpointInterface 要素を指定するときの注意事項について説明します。

- endpointInterface 要素は，Web サービス実装クラスの javax.jws.WebService アノテーションで，javax.jws.WebService アノテーションを持つ SEI を指定します。
- Web サービス実装クラスの javax.jws.WebService アノテーションに endpointInterface 要素を指定している場合，javax.jws.WebService アノテーションの name 要素を同時に指定できません。
- SEI には，javax.jws.WebService アノテーションの endpointInterface 要素を指定できません。
- Web サービス実装クラスの javax.jws.WebService アノテーションに endpointInterface 要素を指定している場合，Web サービス実装クラス内に次に示すアノテーションは指定できません。
  - javax.jws.WebMethod
  - javax.jws.WebParam
  - javax.jws.WebResult
  - javax.jws.Oneway
  - javax.jws.SOAPBinding

(3) name 要素（javax.jws.WebService）

name 要素は，ポートタイプ名のマッピングをカスタマイズするときに指定します。name 要素に要素値を指定すると，ポートタイプ名をマッピングのデフォルト値に持つすべての要素もカスタマイズできます。

name 要素を指定するときの注意事項について説明します。

- SEI の javax.jws.WebService アノテーションで使用する場合，name 要素は半角英数字とアンダースコア（_）で指定します。それ以外の文字を指定した場合の動作は保証されません（エラーメッセージは出力されません）。
- Web サービス実装クラスの javax.jws.WebService アノテーションに endpointInterface 要素を指定している場合，Web サービス実装クラスで javax.jws.WebService アノテーションの name 要素を同時に指定できません。

(4) serviceName 要素（javax.jws.WebService）

serviceName 要素は，サービス名のマッピングをカスタマイズするときに指定します。

serviceName 要素を指定するときの注意事項について説明します。

- SEI には，javax.jws.WebService アノテーションの serviceName 要素は指定できません。
- Web サービス実装クラスの javax.jws.WebService アノテーションで使用する場合，serviceName 要素は半角英数字とアンダースコア（_）で指定します。それ以外の文字を指定した場合，動作は保証されません。

- Java 言語仕様で規定される Java 識別子の命名規則に従った値を指定してください。Java 識別子の命名規則に従っていない場合は，Web サービスクライアントの開発で cjwsimport コマンドを実行するときにコンパイルエラーとなります。

(5) portName 要素（javax.jws.WebService）

portName 要素は，ポート名のマッピングをカスタマイズするときに指定します。

- SEI には，javax.jws.WebService アノテーションの portName 要素は指定できません。
- Web サービス実装クラスの javax.jws.WebService アノテーションで使用する場合，portName 要素は半角英数字とアンダースコア（_）で指定します。それ以外の文字を指定した場合の動作は保証されません。

(6) wsdlLocation 要素（javax.jws.WebService）

wsdlLocation 要素は，Web サービスの呼び出しで参照される値になるため，hwsgen コマンドの実行時には解釈されません。

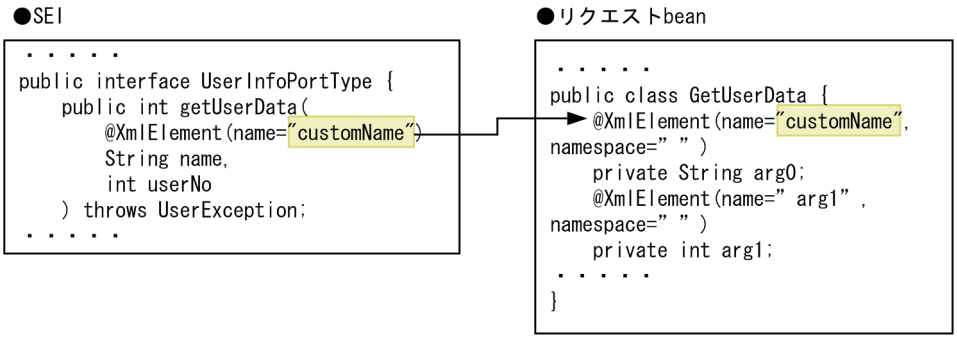
16.2.10 javax.xml.bind.annotation.XmlElement アノテーション

javax.xml.bind.annotation.XmlElement アノテーションは，SEI のサービスメソッドの引数や戻り値に指定します。指定すると，生成されるリクエスト bean，またはレスポンス bean のプロパティの javax.xml.bind.annotation.XmlElement アノテーションにマッピングされます。

javax.xml.bind.annotation.XmlElement アノテーションは，wrapper スタイルで指定します。non-wrapper スタイルで指定した場合の動作の保証はされません。

javax.xml.bind.annotation.XmlElement アノテーションを使用したマッピングの例を次の図に示します。

図 16-18 javax.xml.bind.annotation.XmlElement アノテーションを使用したマッピングの例



(1) name 要素（javax.xml.bind.annotation.XmlElement）

name 要素は，生成されるリクエスト bean またはレスポンス bean のプロパティをアノテートする javax.xml.bind.annotation.XmlElement アノテーションの name 要素にマッピングされます。

javax.xml.bind.annotation.XmlElement アノテーションが，javax.jws.WebParam アノテーションまたは javax.jws.WebResult アノテーションと同時に存在する場合の注意事項を次に示します。

- javax.jws.WebParam アノテーションと同時に存在する場合

  javax.xml.bind.annotation.XmlElement アノテーションと javax.jws.WebParam アノテーションの name 要素を同じ値にする必要があります。ただし，name 要素に"##default"を指定したとき，または javax.jws.WebParam アノテーションの name 要素に空文字("")を指定したとき，同一かどうかの比較処理は行われません。

- javax.jws.WebResult アノテーションと同時に存在する場合

  javax.xml.bind.annotation.XmlElement アノテーションと javax.jws. WebResult アノテーションの name 要素を同じ値にする必要があります。ただし，name 要素に"##default"を指定したとき，または javax.jws.WebResult アノテーションの name 要素に空文字("")を指定したとき，同一かどうかの比較処理は行われません。

(2) namespace 要素（javax.xml.bind.annotation.XmlElement)

namespace 要素は，生成されるリクエスト bean またはレスポンス bean のプロパティをアノテートする javax.xml.bind.annotation.XmlElement アノテーションの namespace 要素にマッピングされます。

javax.xml.bind.annotation.XmlElement アノテーションが，javax.jws.WebParam アノテーションまたは javax.jws.WebResult アノテーションと同時に存在する場合の注意事項を次に示します。

- javax.jws.WebParam アノテーションと同時に存在する場合

  この要素と javax.jws.WebParam アノテーションの targetNamespace 要素を同じ値にする必要があります。ただし，namespace 要素に"##default"を指定したとき，または javax.jws.WebParam アノテーションの targetNamespace 要素に空文字("")を指定したとき，同一かどうかの比較処理は行われません。

- javax.jws.WebResult アノテーションと同時に存在する場合

  この要素と javax.jws. WebResult アノテーションの targetNamespace 要素を同じ値にする必要があります。ただし，namespace 要素に"##default"を指定したとき，または javax.jws.WebResult アノテーションの targetNamespace 要素に空文字("")を指定したとき，同一かどうかの比較処理は行われません。

(3) nillable 要素（javax.xml.bind.annotation.XmlElement)

nillable 要素は，生成されるリクエスト bean またはレスポンス bean のプロパティをアノテートする javax.xml.bind.annotation.XmlElement アノテーションの nillable 要素にマッピングされます。

nillable 要素には，true または false の値を指定できます。false の値を指定した場合，生成されるリクエスト bean またはレスポンス bean のプロパティの javax.xml.bind.annotation.XmlElement アノテーションに nillable 要素はマッピングされません。

(4) required 要素（javax.xml.bind.annotation.XmlElement）

required 要素は，生成されるリクエスト bean またはレスポンス bean のプロパティをアノテートする javax.xml.bind.annotation.XmlElement アノテーションの required 要素にマッピングされます。

required 要素には，true または false の値を指定できます。false の値を指定した場合，生成されるリクエスト bean またはレスポンス bean のプロパティの javax.xml.bind.annotation.XmlElement アノテーションに required 要素はマッピングされません。

16.2.11 javax.xml.bind.annotation.XmlMimeType アノテーション

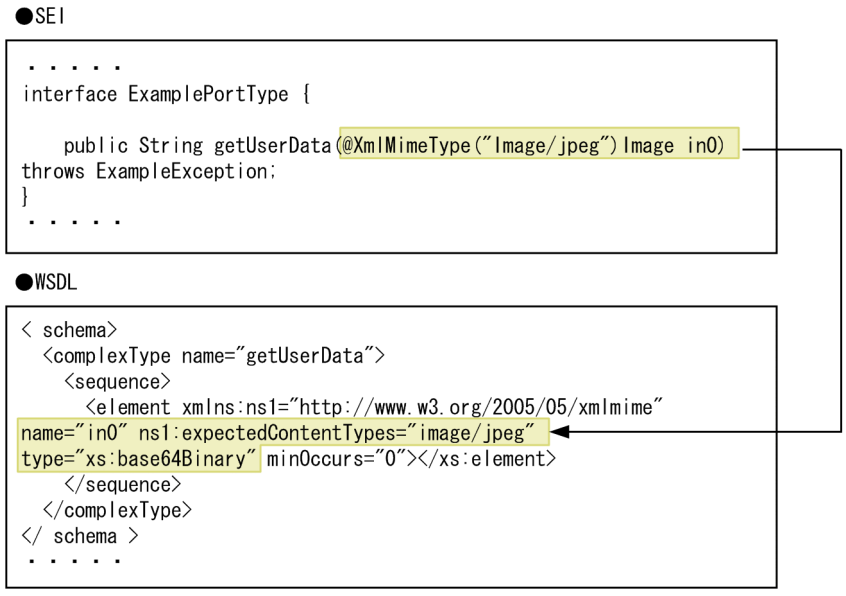
javax.xml.bind.annotation.XmlMimeType アノテーションは，Java 型と MIME タイプの関連づけに使用される JAXB のアノテーションで，javax.xml.ws.soap.MTOM アノテーションを使用する場合で，Java 型と MIME タイプを関連づけるときに使用します。

javax.xml.bind.annotation.XmlMimeType アノテーションは，SEI や暗黙の SEI がある Web サービス実装クラスが持つサービスメソッドの引数，戻り値，またはユーザ定義型の getter メソッドに指定できます。それ以外の個所（暗黙の SEI がない Web サービス実装クラスが持つサービスメソッドの引数，戻り値，ユーザ定義例外のフィールドなど）に指定した場合は動作が保証されません。

javax.xml.bind.annotation.XmlMimeType アノテーションを SEI などにアノテートしている場合，Web サービス側の JAX-WS エンジンが発行する WSDL ファイルや hwsgen コマンドが生成する WSDL ファイルは，javax.xml.bind.annotation.XmlMimeType アノテーションをアノテートした Java 型に対応するスキーマの要素に，アノテーションの value 要素に指定した値を持つ xmime:expectedContentTypes 属性が付与されます。

javax.xml.bind.annotation.XmlMimeType アノテーションを使用した例を次の図に示します。

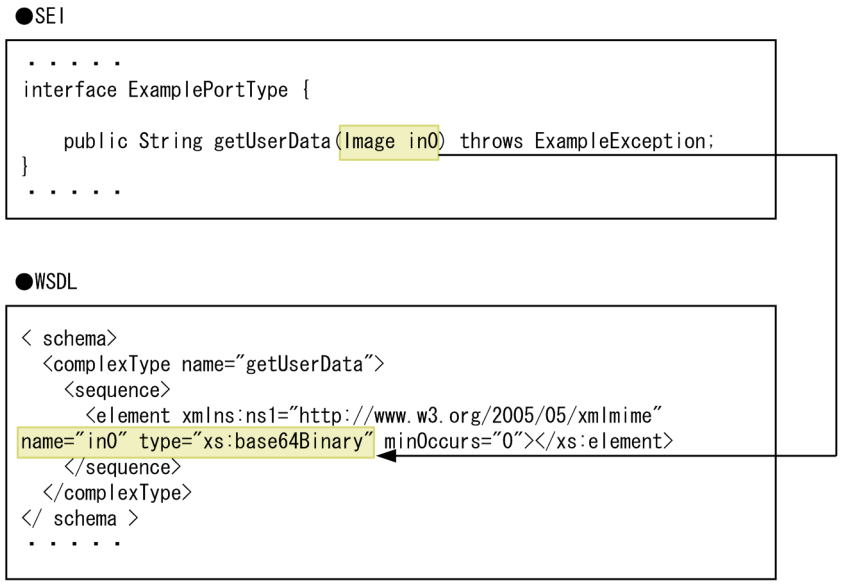
図 16-19 javax.xml.bind.annotation.XmlMimeType アノテーションを使用した例



Java 型と MIME タイプを関連づけない場合，javax.xml.bind.annotation.XmlMimeType アノテーションをアノテートする必要はありません。そのとき，MTOM/XOP 仕様形式の添付ファイルで送信されるメッセージの添付ファイルパートにある Content-Type フィールドの値は，Java 型に対応する初期値が使用されます。

javax.xml.bind.annotation.XmlMimeType アノテーションを使用しない例を次の図に示します。

図 16-20　javax.xml.bind.annotation.XmlMimeType アノテーションを使用しない例



javax.xml.bind.annotation.XmlMimeType アノテーションをアノテートし，MIME タイプを関連づけることができる Java 型とその指定個所を次の表に示します。なお，それ以外の型に javax.xml.bind.annotation.XmlMimeType アノテーションをアノテートした場合の動作は保証されません。

表 16-17　MIME タイプを関連づけることができる Java 型とその指定個所

項番	Java 型	関連づけの可否	指定個所			
			メソッド引数	メソッド戻り値	ユーザ定義型のフィールド	ユーザ定義例外のフィールド
1	java.awt.Image	○※1	○	○	△※2	×※3
2	javax.xml.transform.Source	○	○	○	△※2	×※3
3	javax.activation.DataHandler	○	○	○	△※2	×※3
4	java.awt.Image の配列型	△※4	○	○	△※2	×※3
5	javax.activation.DataHandler の配列型	△※4	○	○	△※2	×※3
6	java.util.List<Image>	○	○	○	△※2	×※3
7	java.util.List<DataHandler>	○	○	○	△※2	×※3

項番	Java 型	関連づけの可否	指定個所			
			メソッド引数	メソッド戻り値	ユーザ定義型のフィールド	ユーザ定義例外のフィールド
8	javax.xml.ws.Holder<Image>	○	○	×※5	×※5	×※5
9	javax.xml.ws.Holder<Source>	○	○	×※5	×※5	×※5
10	javax.xml.ws.Holder<DataHandler>	○	○	×※5	×※5	×※5

（凡例）

○：指定できます。

△：条件付きで指定できます。

×：指定できません。

注※1

JAXB 仕様に従います。java.awt.Image クラスは Java SE 仕様でのグラフィカルイメージを表現する抽象クラスで，データ形式の規定はありません。この関連づけを使用して画像データをインスタンス化した場合，具象クラスのインスタンスには復号化した情報だけが保持される可能性があります。そのため，JPEG 形式のように符号化するときに情報が削減される可能性のある画像を添付ファイルとして送信する場合，受信側のインスタンスが送信側のインスタンスや元のデータと異なることがあります。

画像を元の形式のまま扱いたい場合は，javax.activation.DataHandler にマッピングされる MIME タイプ（application/octet-stream など）を使用してください。

注※2

Java プロパティに MIME タイプを関連づける場合，javax.xml.bind.annotation.XmlMimeType アノテーションを getter メソッドにアノテートしてください。フィールドおよび setter メソッドにアノテートした場合，動作は保証されません。Java プロパティに MIME タイプを関連づける例を次に示します。

```
package com.sample;

import java.awt.Image;

public class UserData {

    private Image image;

    public void setImage(Image image) {
        this.image = image;
    }

    @javax.xml.bind.annotation.XmlMimeType("image/png")
    public Image getImage() {
        return image;
    }
}
```

注※3

ユーザ定義例外のフィールドに javax.xml.bind.annotation.XmlMimeType アノテーションを指定した場合，動作は保証されません。

注※4

1 次元配列だけ使用でき，多次元配列は使用できません。多次元配列を使用した場合，動作は保証されません。

注※5

Holder 型は引数にだけ指定できます。戻り値には指定できません。

(1) value 要素（javax.xml.bind.annotation.XmlMimeType）

value 要素は，javax.xml.bind.annotation.XmlMimeType アノテーションをアノテートした Java 型に関連づける MIME タイプのテキスト表現を指定します。

指定する MIME タイプは，アノテーションをアノテートした Java 型に対して適切な MIME タイプでなければなりません。指定する MIME タイプが適切ではない場合や複数の MIME タイプをコンマ区切りで記述した場合，動作は保証されません。また，指定する MIME タイプには，"text/xml"および"application/xml"の charset パラメタを除いて，パラメタを記述しないでください。"text/xml"および"application/xml"の charset パラメタ以外のパラメタを記述した場合，動作は保証されません。

Java 型に指定できる MIME タイプを次の表に示します。

表 16-18　Java 型に指定できる MIME タイプ

項番	Java 型	value 要素に指定できる MIME タイプ
1	java.awt.Image，java.awt.Image の配列型， java.util.List<Image>，または javax.xml.ws.Holder<Image>	image/png※1
2		image/jpeg※1
3		image/*※2
4	javax.xml.transform.Source または javax.xml.ws.Holder<Source>	text/xml※3
5		application/xml
6	javax.activation.DataHandler， javax.activation.DataHandler の配列型， java.util.List<DataHandler>，または javax.xml.ws.Holder<DataHandler>	上記以外※4

注※1

表中にない image タイプを送信する場合は，javax.activation.DataHandler クラスを使用して送信してください。

注※2

MIME タイプに"image/*"を指定した場合，送信する SOAP メッセージの添付ファイルパートにある MIME ヘッダの Content-Type フィールドの値は，java.awt.Image 型を使用したときの初期値（"image/png"）です。

注※3

MIME タイプに"text/xml"を指定した場合，送信する SOAP メッセージの添付ファイルパートにある MIME ヘッダの Content-Type フィールドの値は，javax.xml.transform.Source 型を使用したときの初期値（"application/xml"）です。

注※4

MIME タイプに"application/*"や未知の MIME タイプを指定した場合，送信する SOAP メッセージの添付ファイルパートにある MIME ヘッダの Content-Type フィールドの値は，javax.activation.DataHandler 型を使用したときの初期値（javax.activation.DataHandler オブジェクトの MIME タイプ）です。

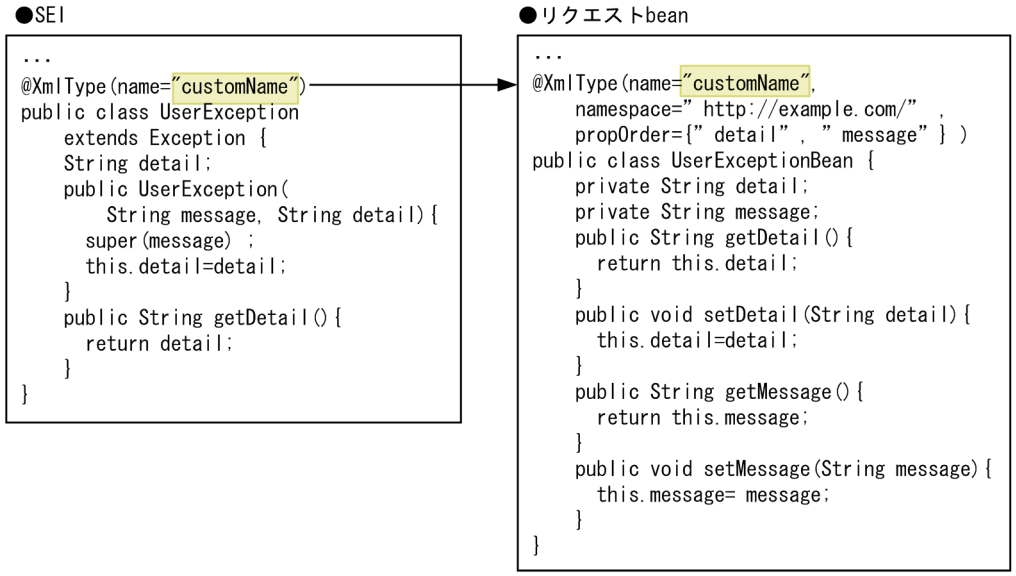
16.2.12　javax.xml.bind.annotation.XmlType アノテーション

javax.xml.bind.annotation.XmlType アノテーションは例外クラスに指定します。指定すると，生成されるフォルト bean の javax.xml.bind.annotation.XmlType アノテーションにマッピングされます。

javax.xml.bind.annotation.XmlType アノテーションを例外クラス以外で指定した場合の動作の保証はされません。

javax.xml.bind.annotation.XmlType アノテーションを使用したマッピング例を次の図に示します。

図 16-21　javax.xml.bind.annotation.XmlType アノテーションを使用したマッピング例



(1)　name 要素（javax.xml.bind.annotation.XmlType）

name 要素は，生成されるフォルト bean の javax.xml.bind.annotation.XmlType アノテーションの name 要素にマッピングされます。

(2)　namespace 要素（javax.xml.bind.annotation.XmlType）

namespace 要素は，生成されるフォルト bean の javax.xml.bind.annotation.XmlType アノテーションの namespace 要素にマッピングされます。

(3) propOrder 要素（javax.xml.bind.annotation.XmlType）

propOrder 要素は，生成されるフォルト bean のプロパティ順序，およびフォルト bean の javax.xml.bind.annotation.XmlType アノテーションの propOrder 要素にマッピングされます。

propOrder 要素には，例外クラスに存在するプロパティのプロパティ名の String 型配列だけで指定できます。

propOrder 要素を使用する場合の注意事項を次に示します。

- プロパティの数が 2 個未満のとき，生成されるフォルト bean の javax.xml.bind.annotation.XmlType アノテーションの propOrder 要素はマッピングされません。
- 空文字（""）をプロパティ名として設定したとき，動作は保証されません。

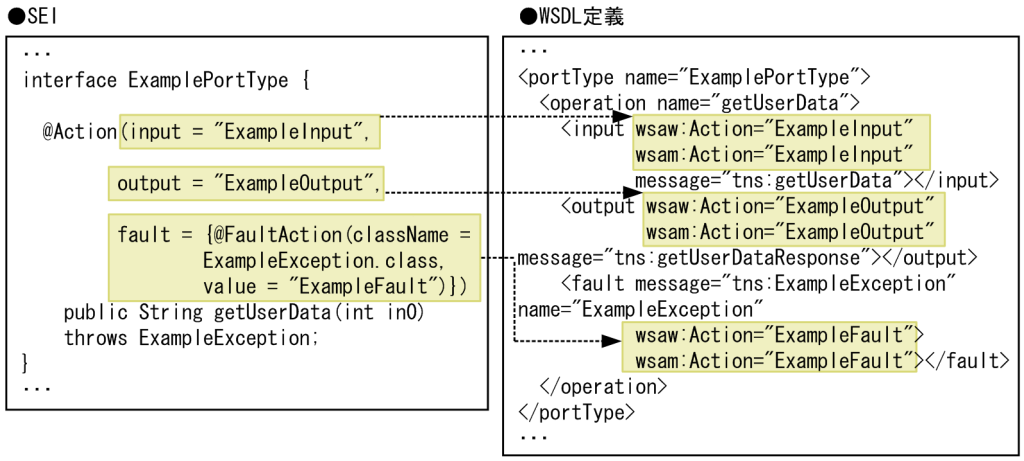
16.2.13 javax.xml.ws.Action アノテーション

javax.xml.ws.Action アノテーションでは，オペレーションの input，output，および fault の各メッセージで Web サービスが使用するアドレッシング・ヘッダの wsa:Action 要素の値を指定できます。

javax.xml.ws.Action アノテーションは，SEI でだけ指定できます。

javax.xml.ws.Action アノテーションを使用したカスタマイズ例を次の図に示します。

図 16-22 javax.xml.ws.Action アノテーションを使用したカスタマイズ例



(1) fault 要素（javax.xml.ws.Action）

fault 要素には，Web サービスがフォルトメッセージを送信する場合に使用する，アドレッシング・ヘッダの wsa:Action 要素の値（javax.xml.ws.FaultAction アノテーション）を指定します。null や{null}を指定した場合の動作は保証されません。

fault 要素は Web サービス開始時に参照されるだけです。hwsgen コマンドの実行時には解釈されません。

(2) input 要素（javax.xml.ws.Action）

input 要素には，Web サービスがリクエストメッセージを受信する場合に使用するアドレッシング・ヘッダの，wsa:Action 要素の値を指定します。input 要素には，RFC 2396 で規定された xsd:anyURI を満たす文字を指定してください。それ以外の文字を指定した場合の動作は保証されません。

input 要素に空白を指定した場合，空白がそのままアドレッシング・ヘッダの wsa:Action 要素の値となります。空文字を指定した場合は，その指定は無視され，input 要素を記述していないものと見なされます。

input 要素は Web サービス開始時に参照されるだけです。hwsgen コマンドの実行時には解釈されません。

(3) output 要素（javax.xml.ws.Action）

output 要素には，Web サービスがレスポンスメッセージを送信する場合に使用する，アドレッシング・ヘッダの wsa:Action 要素の値を指定します。output 要素には，RFC 2396 で規定された xsd:anyURI を満たす文字を指定してください。それ以外の文字を指定した場合の動作は保証されません。

output 要素に空白を指定した場合，空白がそのままアドレッシング・ヘッダの wsa:Action 要素の値となります。空文字を指定した場合は，その指定は無視され，output 要素を記述していないものと見なされます。

output 要素は Web サービス開始時に参照されるだけです。hwsgen コマンドの実行時には解釈されません。

16.2.14 javax.xml.ws.BindingType アノテーション

javax.xml.ws.BindingType アノテーションは，Web サービス実装クラスで指定するアノテーションです。SEI に指定した場合は無視されます。

(1) value 要素（javax.xml.ws.BindingType）

value 要素には，javax.xml.ws.soap.SOAPBinding インタフェースの次に示すフィールド値が指定できます。

- "SOAP11HTTP_BINDING"（SOAP1.1 over HTTP）
- "SOAP12HTTP_BINDING"（SOAP1.2 over HTTP）
- "SOAP11HTTP_MTOM_BINDING"（デフォルトで MTOM/XOP 仕様形式の添付ファイルが有効である SOAP1.1 over HTTP）
- "SOAP12HTTP_MTOM_BINDING"（デフォルトで MTOM/XOP 仕様形式の添付ファイルが有効である SOAP1.2 over HTTP）

javax.xml.ws.BindingType アノテーションから，WSDL の wsdl:binding 要素の子要素である soap:binding 要素または soap12:binding 要素の transport 属性値へのマッピングを次の表に示します。

表 16–19　BindingType アノテーションから transport 属性値へのマッピング

項番	SOAP バージョン	BindingType アノテーションの値	transport 属性値
1	SOAP 1.1	http://schemas.xmlsoap.org/soap/http または @SOAPBinding.SOAP11HTTP_BINDING	http://schemas.xmlsoap.org/soap/http
2		http://schemas.xmlsoap.org/wsdl/soap/http?mtom=true または @SOAPBinding.SOAP11HTTP_MTOM_BINDING	
3	SOAP 1.2	http://www.w3.org/2003/05/soap/bindings/HTTP/ または @SOAPBinding.SOAP12HTTP_BINDING	com.cosminexus.jaxws.publish_wsdl.soap12binding プロパティの指定値によって，WSDL の transport 属性値が異なります。
4		http://www.w3.org/2003/05/soap/bindings/HTTP/?mtom=true または @SOAPBinding.SOAP12HTTP_MTOM_BINDING	

com.cosminexus.jaxws.publish_wsdl.soap12binding プロパティの指定値と WSDL の transport 属性値の関係を次の表に示します。

表 16–20　プロパティの指定値と WSDL の transport 属性値の関係

項番	プロパティ指定有無	プロパティ指定値	transport 属性値
1	指定なし	なし	http://www.w3.org/2003/05/soap/bindings/HTTP/
2	指定あり	DEFAULT	
3		WSI_BP20_TRANSPORT	http://schemas.xmlsoap.org/soap/http※

注※

transport 属性値については標準仕様であいまいなため，JAX-WS ではこの URL を使用できます。

16.2.15　javax.xml.ws.FaultAction アノテーション

javax.xml.ws.FaultAction アノテーションでは，Web サービスがフォルトメッセージを送信する場合に使用するアドレッシング・ヘッダの wsa:Action 要素の値を指定します。

javax.xml.ws.FaultAction アノテーションが指定できるのは，javax.xml.ws.Action アノテーションの fault 要素内だけです。SEI やサービス実装クラスのメソッドに指定しても，無効となります。

(1) className 要素（javax.xml.ws.FaultAction）

className 要素は，javax.xml.ws.FaultAction アノテーションの必須要素です。Web サービスが送信する例外クラスのクラス名を指定してください。className 要素を指定しないと，Web サービス側の JAX-WS エンジンで例外が発生し，Web サービスを開始できません（KDJW40013-E）。

なお，className 要素に Web サービスが送信する例外クラス以外を指定した場合は，動作は保証されません。

className 要素は Web サービス開始時に参照されるだけです。hwsgen コマンドの実行時には解釈されません。

(2) value 要素（javax.xml.ws.FaultAction）

value 要素には，className 要素で指定した例外クラスのフォルトメッセージを Web サービスが送信する場合に使用するアドレッシング・ヘッダの wsa:Action 要素の値を指定します。value 要素には，RFC 2396 で規定された xsd:anyURI を満たす文字を指定してください。それ以外の文字を指定した場合の動作は保証されません。

value 要素に空白を指定した場合，空白がそのままアドレッシング・ヘッダの wsa:Action 要素の値となります。空文字を指定した場合は，その指定は無視され，value 要素を記述していないものと見なされます。

value 要素は Web サービス開始時に参照されるだけです。hwsgen コマンドの実行時には解釈されません。

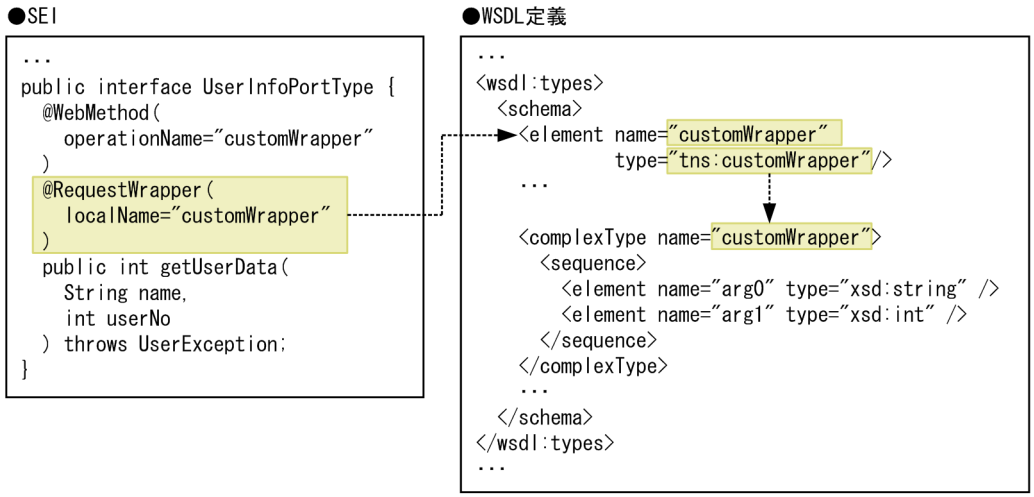
16.2.16 javax.xml.ws.RequestWrapper アノテーション

javax.xml.ws.RequestWrapper アノテーションは，wrapper スタイルで指定できます。

javax.xml.ws.RequestWrapper アノテーションは SEI で指定するアノテーションです。

javax.xml.ws.RequestWrapper アノテーションを使用したカスタマイズ例を次の図に示します。

図 16-23　javax.xml.ws.RequestWrapper アノテーションを使用したカスタマイズ例



(1)　localName 要素（javax.xml.ws.RequestWrapper）

localName 要素は，リクエスト wrapper 要素のローカル名のマッピングをカスタマイズするときに指定します。localName 要素に要素値を指定した場合，wrapper 要素の型名もカスタマイズできます。

localName 要素を指定するときの注意事項について説明します。

- localName 要素は半角英数字とアンダースコア（_）で指定します。それ以外の文字を指定した場合の動作は保証されません（エラーメッセージは出力されません）。
- localName 要素とオペレーション名は同じ名前にしてください。ただし，javax.xml.ws.RequestWrapper アノテーションが存在しない場合，または javax.xml.ws.RequestWrapper アノテーションの localName 要素に空文字（""）が指定されている場合，同一かどうかの比較処理は行われません。

(2)　targetNamespace 要素（javax.xml.ws.RequestWrapper）

targetNamespace 要素は，リクエスト wrapper 要素の名前空間のマッピングをカスタマイズするときに指定します。

targetNamespace 要素には，http://または urn:のプロトコルを名前空間として指定します。指定できる名前空間の形式および文字列を示します。

- **プロトコル**

  名前空間のプロトコルは，http://または urn:のプロトコルで記述してください。
- **名前空間の記述形式**

  名前空間には次に示す形式は記述できません。
  - クエリストリング（例）http://example.com/?a=b
  - アンカー（例）http://example.com/index.html#anchor

- ポート番号（例）http://example.com:8080/
- ユーザ名／パスワード（例）http://user:password@example.com

- **記述できる文字列**

  区切り文字のスラッシュ（/）またはピリオド（.）で区切られたセグメントには，次の表に示すすべての条件を満たす文字列を記述できます。ただし，バインディング宣言でカスタマイズする場合は，XML Schema 仕様の xsd:NCName 型として使用できる文字列を記述できます。

表 16–21　名前空間に記述できる文字列の条件（javax.xml.ws.RequestWrapper）

項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z）だけを使用した文字列	http://鈴木.com http://133.145.224.19/ http://[1080:2C14;D30:BA04:275:806:270C:418A]/	動作は保証されません（エラーメッセージは出力されません）。
2	Java の予約語以外の文字列	http://xxx.com/abstract	動作は保証されません。
3	先頭が数字でない文字列	http://1xxx.com	

(3)　className 要素（javax.xml.ws.RequestWrapper）

className 要素は，生成するリクエスト bean のクラス名を完全修飾名で指定します。

className 要素を指定するときの注意事項について説明します。

- className 要素は半角英数字，アンダースコア（_），およびドルマーク（$）で指定します。それ以外の文字を指定した場合の動作は保証されません（エラーメッセージは出力されません）。
- Java 言語仕様で規定される Java 識別子の命名規則に従った値を指定してください。

(4)　partName 要素（javax.xml.ws.RequestWrapper）

partName 要素は，リクエスト wrapper 要素を参照する input メッセージのパート名のマッピングをカスタマイズするときに指定します。

partName 要素は半角英数字とアンダースコア（_）で指定します。それ以外の文字を指定した場合の動作は保証されません（エラーメッセージは出力されません）。

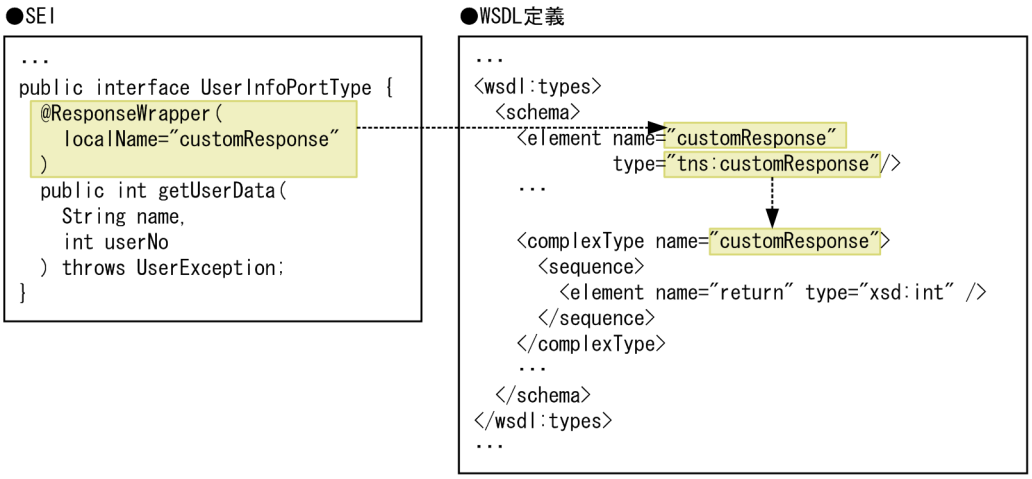
16.2.17　javax.xml.ws.ResponseWrapper アノテーション

javax.xml.ws.ResponseWrapper アノテーションは，wrapper スタイルで指定できます。

javax.xml.ws.ResponseWrapper アノテーションは SEI で指定するアノテーションです。Web サービス実装クラスに指定した場合は無視されます。

javax.xml.ws.ResponseWrapper アノテーションを使用したカスタマイズ例を次の図に示します。

図 16-24　javax.xml.ws.ResponseWrapper アノテーションを使用したカスタマイズ例



(1)　localName 要素（javax.xml.ws.ResponseWrapper）

localName 要素は，レスポンス wrapper 要素のローカル名のマッピングをカスタマイズするときに指定します。localName 要素に要素値を指定した場合，wrapper 要素の型名もカスタマイズできます。

localName 要素は半角英数字とアンダースコア（_）で指定します。それ以外の文字を指定した場合の動作は保証されません（エラーメッセージは出力されません）。

(2)　targetNamespace 要素（javax.xml.ws.ResponseWrapper）

targetNamespace 要素は，レスポンス wrapper 要素の名前空間のマッピングをカスタマイズするときに指定します。

targetNamespace 要素には，http://または urn:のプロトコルを名前空間として指定します。指定できる名前空間の形式および文字列を示します。

- **プロトコル**

  名前空間のプロトコルは，http://または urn:のプロトコルで記述してください。
- **名前空間の記述形式**

  名前空間には次に示す形式は記述できません。
  - クエリストリング（例）http://example.com/?a=b
  - アンカー（例）http://example.com/index.html#anchor
  - ポート番号（例）http://example.com:8080/
  - ユーザ名／パスワード（例）http://user:password@example.com
- **記述できる文字列**

区切り文字のスラッシュ（/）またはピリオド（.）で区切られたセグメントには，次の表に示すすべての条件を満たす文字列を記述できます。

表 16-22　名前空間に記述できる文字列の条件（javax.xml.ws.ResponseWrapper）

項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z）だけを使用した文字列	http://鈴木.com http://133.145.224.19/ http://[1080:2C14;D30:BA04:275:806:270C:418A]/	動作は保証されません（エラーメッセージは出力されません）。
2	Java の予約語以外の文字列	http://xxx.com/abstract	動作は保証されません。
3	先頭が数字でない文字列	http://1xxx.com	

(3)　className 要素（javax.xml.ws.ResponseWrapper）

className 要素は，生成するレスポンス bean のクラス名を完全修飾名で指定します。

className 要素を指定するときの注意事項について説明します。

- className 要素はパッケージの区切り文字であるピリオド（.），半角英数字，アンダースコア（_），およびドルマーク（$）で指定します。それ以外の文字を指定した場合の動作は保証されません（エラーメッセージは出力されません）。
- Java 言語仕様で規定される Java 識別子の命名規則に従った値を指定してください。

(4)　partName 要素（javax.xml.ws.ResponseWrapper）

partName 要素は，レスポンス wrapper 要素を参照する output メッセージのパート名をカスタマイズするときに指定します。

partName 要素は半角英数字とアンダースコア（_）で指定します。それ以外の文字を指定した場合の動作は保証されません（エラーメッセージは出力されません）。

16.2.18　javax.xml.ws.ServiceMode アノテーション

javax.xml.ws.ServiceMode アノテーションでは，プロバイダがアクセスする対象がプロトコルメッセージのペイロード（SOAP ボディ）だけなのか，プロトコルメッセージ全体（SOAP エンベロープ）なのかを指定します。

javax.xml.ws.ServiceMode アノテーションは，Web サービス開始時に参照されるだけです。hwsgen コマンドの実行時には解釈されません。

(1) value 要素（javax.xml.ws.ServiceMode）

value 要素には，javax.xml.ws.Service.Mode.MESSAGE または
javax.xml.ws.Service.Mode.PAYLOAD を指定します。デフォルト値は
javax.xml.ws.Service.Mode.PAYLOAD です。

javax.xml.ws.Service.Mode.MESSAGE を指定するとプロトコルメッセージのすべてが，
javax.xml.ws.Service.Mode.PAYLOAD を指定するとプロトコルメッセージのペイロードだけが，プロバイダのインスタンスに渡されます。

16.2.19 javax.xml.ws.soap.Addressing アノテーション

javax.xml.ws.soap.Addressing アノテーションは，アドレッシング機能を使用する場合に必要です。

javax.xml.ws.soap.Addressing アノテーションについて，次に説明します。

Web サービス側

サービス実装クラスでだけ指定できます。

Web サービスクライアント側

ポートをインジェクトするフィールドまたは setter メソッドに指定できます。詳細については，「10.21.1(4) フィーチャの有効化」を参照してください。それ以外のフィールドやメソッドに指定した場合は無視されます。

javax.xml.ws.soap.Addressing アノテーションを使用したカスタマイズ例を次の図に示します。

図 16-25　javax.xml.ws.soap.Addressing アノテーションを使用したカスタマイズ例（Web サービス側）

(1)　enabled 要素（javax.xml.ws.soap.Addressing）

enabled 要素では，Web サービスでアドレッシング機能を有効にするかどうかを指定します。"true"を指定した場合はアドレッシング機能が有効に，"false"を指定した場合はアドレッシング機能は無効となります。デフォルト値は"true"です。

enabled 要素は Web サービス開始時に参照されるだけです。hwsgen コマンドの実行時には解釈されません。

(2)　required 要素（javax.xml.ws.soap.Addressing）

required 要素では，Web サービスを呼び出す場合のリクエストメッセージに，アドレッシング・ヘッダを必須とするかどうかを指定します。"true"を指定した場合はアドレッシング・ヘッダが必須に，"false"を指定した場合はアドレッシング・ヘッダは任意となります。デフォルト値は"false"です。

required 要素は Web サービス開始時に参照されるだけです。hwsgen コマンドの実行時には解釈されません。

(3) responses 要素（javax.xml.ws.soap.Addressing）

responses 要素は，WS-Addressing が有効の場合に，エンドポイントが要求する応答エンドポイントの種別を指定します。responses 要素に指定できる値を次の表に示します。

表 16–23 responses 要素に指定できる値

項番	responses 要素の値	説明
1	javax.xml.ws.soap.AddressingFeature.Responses.ALL	デフォルト値です。 すべての URI を指定できます。
2	javax.xml.ws.soap.AddressingFeature.Responses.ANONYMOUS	匿名の URI だけを指定できます。
3	javax.xml.ws.soap.AddressingFeature.Responses.NON_ANONYMOUS	非匿名の URI だけを指定できます。

responses 要素の要素値に javax.xml.ws.soap.AddressingFeatures.Responses.ANONYMOUS を指定した場合，エンドポイントに送信するメッセージのアドレッシング・ヘッダの wsa:ReplyTo 要素および wsa:FaultTo 要素に匿名 URI を指定する必要があります。非匿名 URI を指定すると，javax.xml.ws.WebServiceException を返します。

responses 要素の要素値に javax.xml.ws.soap.AddressingFeatures.Responses.NON_ANONYMOUS を指定した場合，非匿名 URI を指定する必要があります。匿名 URI を指定すると，javax.xml.ws.WebServiceException を返します。

デフォルトでは javax.xml.ws.soap.AddressingFeatures.Responses.ALL を指定した場合と同様で，すべての URI を指定できます。

responses 要素は，Web サービス開始時に参照されるだけです。hwsgen コマンドの実行時には解釈されません。

16.2.20 javax.xml.ws.soap.MTOM アノテーション

javax.xml.ws.soap.MTOM アノテーションは，MTOM/XOP 仕様形式の添付ファイルを使用する Web サービスに指定します。

javax.xml.ws.soap.MTOM アノテーションについて，次に説明します。

Web サービス側

Web サービス実装クラスだけで指定できます。SEI に指定すると無視されます。また，プロバイダ実装クラス（javax.xml.ws.provider インタフェースを実装するクラス）に指定した場合は動作が保証されません。

Web サービスクライアント側

ポートをインジェクトするフィールドまたは setter メソッドに指定できます。詳細については，「10.21.1(4)　フィーチャの有効化」を参照してください。それ以外のフィールドやメソッドに指定した場合は無視されます。

cjwsimport コマンドが生成する Web サービス実装クラスのスケルトンクラスを使用して MTOM/XOP 仕様形式の添付ファイルを使用する Web サービスを作成する場合，Web サービス実装クラスのスケルトンクラスには javax.xml.ws.soap.MTOM アノテーションをマッピングしないので，javax.xml.ws.soap.MTOM アノテーションを指定する必要があります。なお，Web サービス実装クラスに javax.xml.ws.soap.MTOM アノテーションを指定しても，Web サービス側の JAX-WS エンジンが発行する WSDL ファイルや hwsgen コマンドが生成する WSDL ファイルには，MTOM/XOP 仕様形式の添付ファイルが使用されていることを示す要素や属性は出現しません。

javax.xml.ws.soap.MTOM アノテーションは，Web サービス開始時に参照されるだけです。hwsgen コマンドの実行時には解釈されません。

Web サービス側で javax.xml.ws.soap.MTOM アノテーションを使用した例を次に示します。

```
. . . . .
@MTOM
@WebService(endpointInterface = "jaxwstp.example.service.ExamplePortType", targetNamespace =
 "http://service.example.jaxwstp/", serviceName = "ExampleService", portName = "ExamplePort"
)
public class ExampleBinding implements ExamplePortType {
```

(1)　enabled 要素（javax.xml.ws.soap.MTOM）

enabled 要素は，Web サービスで MTOM/XOP 仕様形式の添付ファイルを使用するかどうかを指定します。"true"を指定した場合は MTOM/XOP 仕様形式の添付ファイルが使用でき，"false"を指定した場合は MTOM/XOP 仕様形式の添付ファイルが使用できません。デフォルト値は"true"です。

(2)　threshold 要素（javax.xml.ws.soap.MTOM）

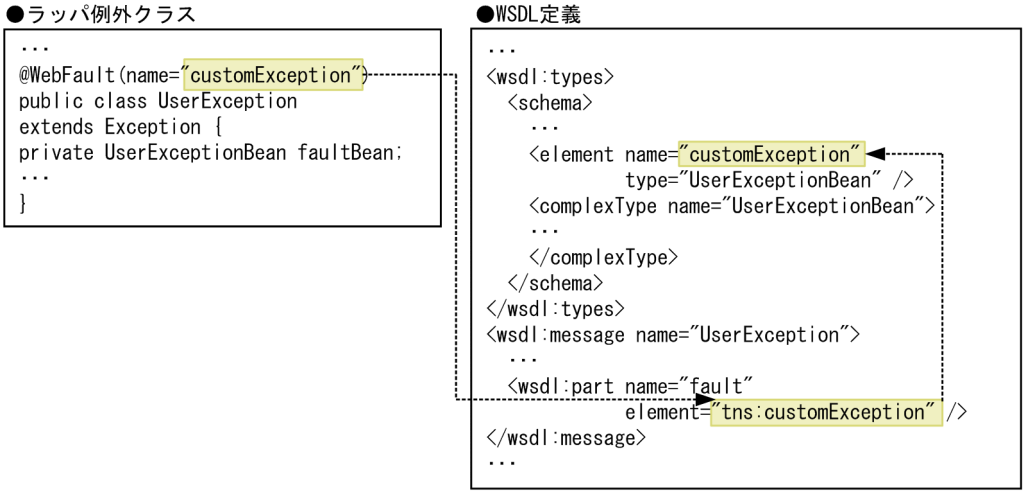
threshold 要素は，Web サービスで MTOM/XOP 仕様形式の添付ファイルが使用できるときに，バイナリデータを MTOM/XOP 仕様形式の添付ファイルとして送信するためのしきい値です。javax.activation.DataHandler クラス以外で，さらに指定された値を超えるバイナリデータ（threshold 要素値≦送信するデータのサイズ）の場合，MTOM/XOP 仕様形式の添付ファイルとして送信されます。デフォルト値は"0"です。

16.2.21　javax.xml.ws.WebFault アノテーション

javax.xml.ws.WebFault アノテーションは，戻り値のマッピングをカスタマイズするときに使用できます。

javax.xml.ws.WebFault アノテーションを使用したカスタマイズ例を次の図に示します。

図 16-26　javax.xml.ws.WebFault アノテーションを使用したカスタマイズ例



(1)　name 要素（javax.xml.ws.WebFault）

name 要素は，フォルト bean からマッピングするグローバル要素のローカル名をカスタマイズするときに使用します。

name 要素は半角英数字とアンダースコア（_）で指定します。それ以外の文字を指定した場合の動作は保証されません（エラーメッセージは出力されません）。

(2)　targetNamespace 要素（javax.xml.ws.WebFault）

targetNamespace 要素は，フォルト bean からマッピングするグローバル要素の名前空間をカスタマイズするときに指定します。

targetNamespace 要素には，http://または urn:のプロトコルを名前空間として指定します。指定できる名前空間の形式および文字列を示します。

- **プロトコル**

  名前空間のプロトコルは，http://または urn:のプロトコルで記述してください。

- **名前空間の記述形式**

  名前空間には次に示す形式は記述できません。

  - クエリストリング（例）http://example.com/?a=b
  - アンカー（例）http://example.com/index.html#anchor
  - ポート番号（例）http://example.com:8080/
  - ユーザ名／パスワード（例）http://user:password@example.com

- **記述できる文字列**

  区切り文字のスラッシュ（/）またはピリオド（.）で区切られたセグメントには，次の表に示すすべての条件を満たす文字列を記述できます。ただし，バインディング宣言でカスタマイズする場合は，XML Schema 仕様の xsd:NCName 型として使用できる文字列を記述できます。

表 16-24　名前空間に記述できる文字列の条件（javax.xml.ws.WebFault）

項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z）だけを使用した文字列	http://鈴木.com http://133.145.224.19/ http://[1080:2C14;D30:BA04:275:806:270C:418A]/	動作は保証されません（エラーメッセージは出力されません）。
2	Java の予約語以外の文字列	http://xxx.com/abstract	動作は保証されません。
3	先頭が数字でない文字列	http://1xxx.com	

(3)　faultBean 要素（javax.xml.ws.WebFault）

faultBean 要素は，生成するフォルト bean のクラス名を完全修飾名で指定します。ラッパ例外クラスが，javax.xml.ws.WebFault アノテーションおよびフォルト bean を返す getFaultInfo メソッドを持つ場合，faultBean 要素を指定しても，フォルト bean は生成されません。

faultBean 要素を指定するときの注意事項について説明します。

- faultBean 要素は半角英数字とアンダースコア（_）で指定します。それ以外の文字を指定した場合の動作は保証されません（エラーメッセージは出力されません）。
- Java 言語仕様で規定される Java 識別子の命名規則に従った値を指定してください。

(4)　messageName 要素（javax.xml.ws.WebFault）

messageName 要素は，ラッパ例外クラスに対応するフォルトメッセージ名（wsdl:fault 要素から参照される wsdl:message 要素の name 属性）をカスタマイズするときに指定します。

messageName 要素は半角英数字とアンダースコア（_）で指定します。それ以外の文字を指定した場合の動作は保証されません（エラーメッセージは出力されません）。

16.2.22　javax.xml.ws.WebServiceProvider アノテーション

javax.xml.ws.WebServiceProvider アノテーションは，javax.xml.ws.provider インタフェースを実装するクラスに指定し，プロバイダの要件を満たすクラスが Web サービスのエンドポイントを定義していることを宣言します。

javax.xml.ws.WebServiceProvider アノテーションと javax.jws.WebService アノテーションは，どちらか一方しか指定できません。

javax.xml.ws.WebServiceProvider アノテーションは，Web サービス開始時に参照されるだけです。hwsgen コマンドの実行時には解釈されません。

(1) targetNamespace 要素（javax.xml.ws.WebServiceProvider）

targetNamespace 要素には，http://または urn:のプロトコルを名前空間として指定します。指定できる名前空間の形式および文字列を示します。

- **プロトコル**
  名前空間のプロトコルは，http://または urn:のプロトコルで記述してください。
- **名前空間の記述形式**
  名前空間には次に示す形式は記述できません。
  - クエリストリング（例）http://example.com/?a=b
  - アンカー（例）http://example.com/index.html#anchor
  - ポート番号（例）http://example.com:8080/
  - ユーザ名／パスワード（例）http://user:password@example.com
- **記述できる文字列**
  区切り文字のスラッシュ（/）またはピリオド（.）で区切られたセグメントには，次の表に示すすべての条件を満たす文字列を記述できます。

表 16-25　名前空間に記述できる文字列の条件（javax.xml.ws.WebServiceProvider）

項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0～9，A～Z，a～z）だけを使用した文字列	http://鈴木.com/ http://133.145.224.19/ http://[1080:2C14;D30:BA04:275:806:270C:418A]/	動作は保証されません（エラーメッセージは出力されません）。
2	Java の予約語でない文字列	http://xxx.com/abstract	動作は保証されません。
3	先頭が数字でない文字列	http://1xxx.com/	

(2) serviceName 要素（javax.xml.ws.WebServiceProvider）

serviceName 要素は半角英数字とアンダースコア（_）で指定します。それ以外の文字を指定した場合の動作は保証されません（エラーメッセージは出力されません）。

(3) portName 要素（javax.xml.ws.WebServiceProvider）

portName 要素は半角英数字とアンダースコア（_）で指定します。それ以外の文字を指定した場合，動作は保証されません（エラーメッセージは出力されません）。

(4) wsdlLocation 要素（javax.xml.ws.WebServiceProvider）

wsdlLocation 要素については，「10.6　メタデータの発行」を参照してください。

17 Web リソースとプロバイダ

この章では，RESTful Web サービス（Web リソース）のリソースクラスとプロバイダのサポート範囲について説明します。

17.1　リソースクラス

リソースクラスのリソースメソッド，サブリソースメソッド，およびサブリソースロケータの違い，ならびにルートリソースクラスとサブリソースクラスの違いについて説明します。

リソースクラスのリソースメソッド，サブリソースメソッド，およびサブリソースロケータは，Path アノテーション，および要求メソッド識別子の有無によって定義されます。それぞれの定義を次に示します。

表 17-1　リソースメソッド，サブリソースメソッド，およびサブリソースロケータの定義

項番	メソッドまたはロケータ	Path アノテーション	要求メソッド識別子
1	リソースメソッド	×	○
2	サブリソースメソッド	○	○
3	サブリソースロケータ	○	×

(凡例)

○：使用できることを示します。

×：使用できないことを示します。

ルートリソースクラスのインスタンスは，JAX-RS エンジンによって生成されます。このとき，コンストラクタのパラメタ，フィールド，および bean プロパティには JAX-RS 仕様に従ってインジェクトされます。

一方，サブリソースクラスのインスタンスは，JAX-RS エンジンによって生成されません。サブリソースクラスは，対応するサブリソースロケータでインスタンス化する必要があります。このため，コンストラクタのパラメタ，フィールド，および bean プロパティの初期化は，サブリソースロケータ，またはサブリソースクラスで行う必要があります。

17.1.1　ルートリソースクラス

ルートリソースクラスは，リソースメソッド，サブリソースメソッド，またはサブリソースロケータのどれかを一つ以上持ち，クラスレベルで Path アノテーションでアノテートされた Java の public のクラスです。

リソースメソッドとサブリソースロケータを持つルートリソースクラスの例を次に示します。

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Path;

//ルートリソースクラス
@Path(”/root”)
public class Resource2 {

  //コンテキストルート+ ”/root/sub1”への要求に対応するサブリソースロケータ
  @Path(”/sub1”)
```

```
  public SubResource1 subResourceLocator1() {
    //処理するサブリソースクラスのインスタンスを戻す
    return new SubResource1();
  }

  //コンテキストルート+ "/root/sub2"への要求に対応するサブリソースロケータ
  @Path("/sub2")
  public SubResource2 subResourceLocator2() {
    //処理するサブリソースクラスのインスタンスを戻す
    return new SubResource2();
}
  //リソースメソッド
  @GET
  public String getValue() {
    String returnValue = "";
    //戻り値を構築する
    return returnValue;
  }

}
```

com.sample.resources.Resource2 はルートリソースクラスです。クラスレベルで Path アノテーションでアノテートされていることに注意してください。このルートリソースクラスは，二つのサブリソースロケータ subResourceLocator1()と subResourceLocator2()および HTTP GET リクエストを処理するリソースメソッド getValue()を持ちます。サブリソースロケータが Path アノテーションでアノテートされ，リソースメソッドが要求メソッド識別子でアノテートされていることに注意してください。二つのクラス SubResource1 と SubResource2 はサブリソースクラスです。詳細は，次の項目を参照してください。

- 「17.1.1(4)　リソースメソッド」
- 「17.1.1(5)　サブリソースメソッド」
- 「17.1.1(6)　サブリソースロケータ」
- 「17.1.7　サブリソースクラス」

サブリソースメソッドを持つルートリソースクラスの例を次に示します。

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Path;

// ルートリソースクラス
@Path("/root")
public class Resource3 {

  // コンテキストルート+ "root/sub1"への要求に対応するサブリソースメソッド
  @Path("/sub1")
  @GET
  public String subResourceMethod1() {
    String value = "";
    // 処理を実行して，実行結果を返す
    return value;
  }
```

```
  // コンテキストルート+ ”root/sub2”への要求に対応するサブリソースメソッド
  @Path(”/sub2”)
  @GET
  public String subResourceMethod2() {
    String value = ””;
    // 処理を実行して，実行結果を返す
    return value;
  }
}
```

com.sample.resources.Resource3 はルートリソースクラスです。このルートリソースクラスは，二つのサブリソースメソッド subResourceMethod1()と subResourceMethod2()を持ちます。サブリソースメソッドは，Path アノテーションと要求メソッドの両方でアノテートされます。

Path アノテーションの URL にアスタリスク（*）を指定されている場合は，リソースメソッドだけを呼び出すことができます。サブリソースメソッド，およびサブリソースロケータを呼び出した場合は，例外マッピングプロバイダで処理できる java.lang.StringIndexOutOfBoundsException がスローされます。

(1) ライフサイクル

JAX-RS エンジンは，Web リソースに対する要求ごとにルートリソースクラスをインスタンス化します。ルートリソースクラスのライフサイクルは次のとおりです。

1. コンストラクタが呼び出される
2. 必要なインジェクションが行われる
3. 適切なメソッドが呼び出される
4. GC（ガーベージコレクション）の対象になる

(2) コンストラクタ

ルートリソースクラスは，public デフォルトコンストラクタ（明示的に宣言されないコンストラクタ）も含めて，少なくとも一つ以上の public コンストラクタを持つ必要があります。

パラメタを持つ public コンストラクタの例を次に示します。

```
package com.sample.resources;

import javax.ws.rs.DefaultValue;
import javax.ws.rs.Encoded;
import javax.ws.rs.QueryParam;
import javax.ws.rs.Path;
import javax.ws.rs.GET;

//ルートリソースクラス
@Path(”/root”)
public class Resource1 {
  private String query1;
```

```
  //パラメタを持つpublicコンストラクタ
  public Resource1(@Encoded @DefaultValue("abc") @QueryParam("query") String query){
    this.query1 = query;
  }

  //リソースメソッド
  @GET
  public String getValue() {
    return "Your requested query parameter ¥"query¥" is: " + this.query1;
  }
}
```

この例では，ルートリソースクラス com.sample.resources.Resource1 は，QueryParam アノテーションでアノテートされたパラメタ query を持つ public コンストラクタ Resource1()によってインスタンス化されます。

パラメタ query を自動で URL デコードされるのを無効化するために Encoded アノテーションが使用されています。また，クライアントから送信された要求にパラメタ query にインジェクトする値がない場合のデフォルト値を指定するために DefaultValue アノテーションが組み合わされて使用されています。

パラメタを持たない public コンストラクタの例を次に示します。

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Path;

//ルートリソースクラス
@Path("/root")
public class Resource2 {
  //パラメタを持たないpublicコンストラクタ
  public Resource2() {
    // 処理を実行する
  }

 //リソースメソッド
  @GET
  public String getValue(){
    return "Your request was accepted.";
  }
}
```

この例では，ルートリソースクラス com.sample.resources.Resource2 は，パラメタを持たない public コンストラクタ Resource2()によってインスタンス化されます。

パラメタを持つ public コンストラクタが一つ以上宣言されている例を次に示します。

```
package com.sample.resources;

import javax.ws.rs.Encoded;
import javax.ws.rs.MatrixParam;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
```

```
import javax.ws.rs.QueryParam;

//ルートリソースクラス
@Path("/root/")
public class Resource3 {
  private String matrix1;
  private String query1;

  //パラメタを持たないpublicコンストラクタ
  public Resource3() {
    // 処理を実行する
  }

  //パラメタを持つpublicコンストラクタ（1番目）
  @Encoded
  public Resource3(@MatrixParam("matrix1") String matrix1) {
    this.matrix1 = matrix1;
  }

  //パラメタを持つpublicコンストラクタ（2番目）
  @Encoded
  public Resource3(@FormParam("form1") String form1,
      @QueryParam("query1") String query1) {
    this.form1 = form1;
    this.query1 = query1;
  }

  //リソースメソッド
  @GET
  public String getValue() {
    return "Your requested matrix parameter ¥"matrix1¥" is: " + this.matrix1 +  "¥n" +
      "Your requested query parameter ¥"query1¥" is: " + this.query1;
  }
}
```

この例では，ルートリソースクラス com.sample.resources.Resource3 は，それぞれ MatrixParam アノテーションおよび QueryParam アノテーションでアノテートされたパラメタ matrix1 およびパラメタ query1 を持つ，2 番目の public コンストラクタによってインスタンス化されます。

パラメタ matrix1 とパラメタ query1 を自動で URL デコードされるのを無効化するために，Encoded アノテーションがコンストラクタのレベルで使用されています。

デフォルトコンストラクタの例を次に示します。

```
package com.sample.resources;

import javax.ws.rs.FormParam;
import javax.ws.rs.POST;
import javax.ws.rs.Path;

//ルートリソースクラス
@Path("/root")
public class Resource4 {

  //リソースメソッド
```

```
  @POST
  public String getValue(@FormParam("form1") String form1) {
    return "Your requested form parameter ¥"form1¥" is: " + form1;
  }
}
```

この例では，ルートリソースクラス com.sample.resources.Resource4 は，暗黙的に宣言されているデフォルトコンストラクタ Resource4()によってインスタンス化されます。

デフォルトコンストラクタで java.io.IOException が発生した場合，エラーとなり（KDJJ10039-E），ルートリソースクラスはインスタンス化されません。HTTP ステータスコードには 500 が返されます。

public コンストラクタが一つも宣言されていない場合，エラーとなり（KDJJ10006-E），ルートリソースクラスはインスタンス化されません。HTTP ステータスコードには 500 が返されます。

なお，次に示すコンストラクタは public コンストラクタではありません。

- private コンストラクタ
- protected コンストラクタ
- アクセス識別子を持たないコンストラクタ

コンストラクタのパラメタで使用できるインジェクション用アノテーションと，オプションのアノテーションの組み合わせを次の表に示します。

表 17-2　コンストラクタのパラメタで使用できるアノテーション

項番	インジェクション用アノテーション	オプションのアノテーション	
		Encoded	DefaultValue
1	MatrixParam	○	○
2	QueryParam	○	○
3	PathParam	○	×
4	CookieParam	×	○
5	HeaderParam	×	○
6	FormParam	×	○
7	Context	×	×

（凡例）

○：インジェクション用アノテーションと組み合わせて使用できることを示します。

×：インジェクション用アノテーションが使用できないことを示します。

オプションの Encoded アノテーションを使用すると，インジェクション対象のコンストラクタのパラメタが自動で URL デコードされるのを無効化できます。

オプションの DefaultValue アノテーションを使用すると，インジェクション対象のコンストラクタのパラメタにインジェクトする値がない場合に仮定されるデフォルト値を指定できます。

ルートリソースクラスがパラメタを持つ public コンストラクタを一つ以上持つ場合，JAX-RS エンジンは，最もパラメタ数が多いコンストラクタを使用してルートリソースクラスをインスタンス化します。最もパラメタ数が多いコンストラクタを二つ以上持つ場合，最初に定義されているコンストラクタを使用してルートリソースクラスをインスタンス化します。このとき，警告メッセージがログに出力されます（KDJJ20010-W）。

次のどちらかの場合，HTTP リクエストは処理されません。HTTP ステータスコードには 500 が返されます。なお，ログを確認するときは，JAX-RS 機能のログファイルではなく J2EE サーバのログファイルを確認してください。

- ルートリソースクラスのコンストラクタのパラメタのうち，インジェクション用アノテーションと DefaultValue アノテーションが使用されているパラメタで，JAX-RS エンジンがインジェクトするときにランタイム例外が発生した場合
- ルートリソースクラスのデフォルトコンストラクタで java.lang.InstantiationException，java.lang.IllegalAccessException，java.lang.reflect.InvocationTargetException，または java.io.IOException のどれでもないチェック済み例外（ランタイム例外でない例外）が発生した場合

(3) フィールドおよび bean プロパティ

ルートリソースクラスのフィールドおよび bean プロパティで使用できるインジェクション用アノテーションと，オプションのアノテーションの組み合わせを次の表に示します。JAX-RS エンジンは，ルートリソースクラスをインスタンス化する際に，アノテーションに基づいてアノテートされたフィールドおよび bean プロパティに値をインジェクトします。なお，Bean プロパティは，読み取り専用プロパティでなくても構いません。

表 17-3 フィールドおよび bean プロパティで使用できるアノテーション

項番	インジェクション用アノテーション	オプションのアノテーション	
		Encoded	DefaultValue
1	MatrixParam	○	○
2	QueryParam	○	○
3	PathParam	○	×
4	CookieParam	×	○
5	HeaderParam	×	○
6	FormParam	×	○
7	Context	×	×

（凡例）

○：インジェクション用のアノテーションと組み合わせて使用できることを示します。

×：インジェクション用のアノテーションが使用できないことを示します。

オプションの Encoded アノテーションを使用すると，インジェクション対象のフィールドまたは bean プロパティが自動で URL デコードされるのを無効化できます。

オプションの DefaultValue アノテーションを使用すると，インジェクション対象のフィールドまたは bean プロパティにインジェクトする値がない場合に仮定されるデフォルト値を指定できます。

ルートリソースクラスのフィールドで DefaultValue アノテーションを使用する例を次に示します。

```
private @DefaultValue(”value1”) @QueryParam(”id”) String id;
```

この例では，クエリパラメタ id が URL に指定されていない場合，フィールド id は"value1"になります。

ルートリソースクラスの bean プロパティで DefaultValue アノテーションを使用する例を次に示します。

```
private String property1;

@DefaultValue(”10”) @QueryParam(”prop1”)
public void setProperty1(String property1) {
  this.property1 = property1;
}
```

この例では，クエリパラメタ prop1 が URI に指定されていない場合，bean プロパティ property1 の値は"10"になります。

ルートリソースクラスのフィールドで Encoded アノテーションを使用する例を次に示します。

```
private @Encoded @QueryParam(”id”) String id;
```

この例では，フィールド id の値は，自動で URL デコードされません。

ルートリソースクラスの bean プロパティで Encoded アノテーションを使用する例を次に示します。

```
private String property1;

@Encoded @QueryParam(”prop1”)
public void setProperty1(String property1) {
  this.property1 = property1;
}
```

この例では，bean プロパティ property1 の値は，自動で URL デコードされません。

ルートリソースクラスのフィールド，または bean プロパティのうち，インジェクション用アノテーションと DefaultValue アノテーションが使用されているパラメタで，JAX-RS エンジンがインジェクトするときにランタイム例外が発生した場合，HTTP リクエストは処理されません。HTTP ステータスコードには 500 が返されます。なお，ログを確認するときは，JAX-RS 機能のログファイルではなく J2EE サーバのログファイルを確認してください。

(4) リソースメソッド

リソースメソッドは，JAX-RS 仕様で定義された要求メソッド識別子のどれかでアノテートされた，ルートリソースクラスのメソッドです。ルートリソースクラスは，一つ以上のリソースメソッドを持つことができます。

JAX-RS 仕様で定義された要求メソッド識別子を次に示します。

- GET アノテーション
- POST アノテーション
- PUT アノテーション
- DELETE アノテーション
- HEAD アノテーション
- OPTIONS アノテーション

一つのリソースメソッドに対し，二つ以上の要求メソッド識別子を使用している場合，エラーとなり(KDJJ10006-E)，ルートリソースクラスはインスタンス化されません。HTTP ステータスコードには 500 が返されます。

二つ以上のリソースメソッドに対し，同じ要求メソッド識別子を使用している場合，エラーとなり(KDJJ10006-E)，ルートリソースクラスはインスタンス化されません。HTTP ステータスコードには 500 が返されます。

HTTP リクエストに対し，ディスパッチするリソースメソッドが一つもない場合，HTTP ステータスコードに 405 が設定された，例外マッピングプロバイダで処理できる javax.ws.rs.WebApplicationException がスローされます。

リソースメソッドで要求メソッド識別子を使用している例を次に示します。

```
@GET
@Encoded
@Produces(”text/plain”)
public String echo(@QueryParam(”id”) String id){
  return ”ID is: ” + id;
}
```

この例では，echo()メソッドが GET 要求メソッド識別子でアノテートされています。また，echo()メソッドは，Content-Type が"text/plain"である HTTP レスポンスを返すために，"text/plain"を値に持つ Produces アノテーションでアノテートされています。なお，パラメタ id は，クエリパラメタ id を受け取る QueryParam アノテーションでアノテートされ，さらにクエリパラメタが自動で URL デコードされるのを無効化するために Encoded アノテーションでアノテートされています。

(a) 可視性

リソースメソッドは，要求メソッド識別子が適用された public メソッドである必要があります。次に示すメソッドは，要求メソッド識別子でアノテートされていてもリソースメソッドではありません。

- Private メソッド
- Protected メソッド
- アクセス識別子を持たないメソッド

要求メソッド識別子を上記のメソッドのどれかに適用した場合，警告メッセージまたはエラーメッセージがログに出力されます（KDJJ20003-W または KDJJ10006-E）。KDJJ20003-W および KDJJ10006-E については，「13.7.1 Web リソース初期化時の構文チェック（KDJJ20003-W と KDJJ10006-E）」を参照してください。

(b) パラメタのアノテーション

リソースメソッドのパラメタで使用できるインジェクション用アノテーションと，オプションのアノテーションの組み合わせを次の表に示します。

表 17-4 リソースメソッドのパラメタで使用できるアノテーション

項番	インジェクション用アノテーション	オプションのアノテーション	
		Encoded	DefaultValue
1	MatrixParam	○	○
2	QueryParam	○	○
3	PathParam	○	×
4	CookieParam	×	○
5	HeaderParam	×	○
6	FormParam	×	○
7	Context	×	×

（凡例）
○：インジェクション用のアノテーションと組み合わせて使用できることを示します。
×：インジェクション用のアノテーションが使用できないことを示します。

オプションの Encoded アノテーションを使用すると，インジェクション対象のリソースメソッドのパラメタが自動で URL デコードされるのを無効化できます。

オプションの DefaultValue アノテーションを使用すると，インジェクション対象のリソースメソッドのパラメタにアノテートされるアノテーションのデフォルト値を指定できます。

リソースメソッドのパラメタでアノテーションを使用する例を次に示します。

```
@GET
@Produces("text/plain")
public String echo(@Encoded @DefaultValue("10") @QueryParam("id") String id, @Encoded @Matri
xParam("matrix1") String matrix1){
  return "ID is: " + id + "¥nMatrix1 is: " + matrix1;
}
```

この例では，リソースメソッド echo()は，QueryParam アノテーションでアノテートされたパラメタ id および MatrixParam アノテーションでアノテートされたパラメタ matrix1 を含みます。パラメタ id は，自動で URL デコードされるのを無効化し，デフォルト値を指定するために，さらに Encoded アノテーションおよび DefaultValue アノテーションでアノテートされています。matrix パラメタは，さらに自動で URL デコードされるのを無効化するために Encoded アノテーションでアノテートされています。

ルートリソースクラスのリソースメソッド，またはサブリソースメソッドのパラメタのうち，インジェクション用アノテーションと DefaultValue アノテーションが使用されているパラメタで，JAX-RS エンジンがインジェクトするときにランタイム例外が発生した場合，エラーとなり（KDJJ10009-E，KDJJ10006-E），HTTP リクエストは処理されません。HTTP ステータスコードには 500 が返されます。

(c)　エンティティパラメタ

エンティティパラメタの説明については，「17.1.2　エンティティパラメタ」を参照してください。

(d)　戻り値

戻り値の説明については，「17.1.3　戻り値」を参照してください。

(5)　サブリソースメソッド

Path アノテーションでアノテートされたリソースメソッドを，特に，サブリソースメソッドと呼びます。サブリソースメソッドとリソースメソッドの違いは，Path アノテーションを使用しているかどうかだけです。

サブリソースメソッドの例を次に示します。

```
package com.sample.resources;

import javax.ws.rs.POST;
import javax.ws.rs.Path;

//ルートリソースクラス
@Path("/root/")
public class Resource1 {

  //サブリソースメソッド
  @Path("sub1")
  @POST
  public String doSomething(String entityBody) {
    return "By Sub-Resource Method.";
  }
```

```
  //リソースメソッド
  @POST
  public String doOthers(String entityBody){
    return ”By Resource Method.”;
  }
}
```

この例では，doSomething()メソッドがサブリソースメソッドです。ルートリソースクラス com.sample.resources.Resource1 を含む Web アプリケーション（WAR ファイル）のコンテキストルートが，"example"で，Web アプリケーションが"sample.com"というホストで公開されているとします。その場合，URL"http://sample.com/example/root/sub1"に対する HTTP POST 要求は，サブリソースメソッド doSomething()にディスパッチされます。

一方，URL"http://sample.com/example/root"に対する HTTP POST 要求は，リソースメソッド doOthers()にディスパッチされます。

(6) サブリソースロケータ

要求メソッド識別子が適用されていない，Path アノテーションだけでアノテートされたルートリソースクラスのメソッドをサブリソースロケータと呼びます。

サブリソースロケータは，HTTP リクエストに対する残りの処理を行うサブリソースクラスを返します。サブリソースクラスについては，「17.1.7　サブリソースクラス」を参照してください。

ルートリソースクラスのサブリソースロケータの例を次に示します。

```
package com.sample.resources;

import javax.ws.rs.Encoded;
import javax.ws.rs.QueryParam;
import javax.ws.rs.PathParam;
import javax.ws.rs.GET;
import javax.ws.rs.Path;

//ルートリソースクラス
@Path(”/root/”)
public class Resource {

  //コンテキストルート+ ”/root/sub1”への要求に対応するサブリソースロケータ
  @Path(”sub1”)
  public SubResource getRequestHandler(@PathParam(”id”) String id,
      @Encoded @QueryParam(”query1”) String query1) {
    return new SubResource(id, query1);
  }
}
```

対応するサブリソースクラスの例を次に示します。

```
//サブリソースクラス
public class SubResource {
  private String id;
```

```
  private String query;

  public SubResource(String id, String query){
    this.id = id;
    this.query = query;
  }
  @GET
  public String getRequestParameter(){
    return "ID is: " + this.id + "¥nQuery is: " + this.query;
  }
}
```

この例では，ルートリソースクラス com.sample.resources.Resource は，HTTP リクエストを直接処理しません。サブリソースロケータ getRequestHandler が返すサブリソースクラスは，com.sample.resources.SubResource が処理します。

リソースクラス com.sample.resources.Resource を含む Web アプリケーション（WAR ファイル）のコンテキストルートが"example"で，Web アプリケーションが"sample.com"というホストで公開されているとします。その場合，URL"http://sample.com/example/root/sub1?query1=10"に対する HTTP GET リクエストは，ルートリソースクラス com.sample.resources.Resource のメソッド getRequestHandler()にディスパッチされます。

HTTP GET リクエストに対する残りの処理は，サブリソースクラス com.sample.resources.SubResource のリソースメソッド getRequestParameter()によって処理されます。

サブリソースロケータのパラメタとしてエンティティパラメタを使用した場合，エラーとなり（KDJJ10006-E），クライアントからの HTTP リクエストは処理されません。HTTP ステータスコードには 500 が返されます。

サブリソースロケータが public でない場合の動作は，リソースメソッドと同じです。

サブリソースロケータの戻り値の型が void の場合，エラーとなり（KDJJ10006-E），クライアントからの HTTP リクエストは処理されません。HTTP ステータスコードには 500 が返されます。

サブリソースロケータのパラメタのうち，インジェクション用アノテーションと DefaultValue アノテーションが使用されているパラメタで，JAX-RS エンジンがインジェクトするときにランタイム例外が発生した場合，HTTP リクエストは処理されません。HTTP ステータスコードには 500 が返されます。なお，ログを確認するときは，JAX-RS 機能のログファイルではなく J2EE サーバのログファイルを確認してください。

17.1.2 エンティティパラメタ

リソースメソッドのパラメタのうち，アノテーションでアノテートされていないパラメタをエンティティパラメタと呼びます。エンティティパラメタの値は，HTTP エンティティボディです。

リソースメソッドでエンティティパラメタを使用する例を次に示します。

```
@POST
public String getRequestParameters(@MatrixParam(”matrix”) String matrix,
    String entity) {
  return ”Matrix is:” + matrix + ”¥mEntity Body is: ” + entity;
}
```

この例では，リソースメソッド getRequestParameters()は，MatrixParam アノテーションでアノテートされたパラメタ matrix と，アノテートされていないパラメタ（エンティティパラメタ）entity を持っています。エンティティボディの内容が"Entity Content"である HTTP リクエストを受信した場合，エンティティパラメタ entity の値は"Entity Content"になります。

(1) エンティティパラメタに使用できる Java の型と MIME メディアタイプの組み合わせ

エンティティパラメタに使用できる Java の型と MIME メディアタイプの組み合わせを次の表に示します。なお，POJO に JAXB 仕様のアノテーションは使用しないでください。使用した場合，説明とは異なる動作になるおそれがあります。

表 17-5　エンティティパラメタに使用できる Java の型と MIME メディアタイプの組み合わせ

項番	Java の型	charset※1	MIME メディアタイプ
1	byte[]	×	任意(*/*)
2	java.lang.String	○	任意(*/*)
3	java.io.InputStream	×	任意(*/*)
4	java.io.Reader	○	任意(*/*)
5	java.io.File※2	○	任意(*/*)
6	javax.activation.DataSource	○	任意(*/*)
7	javax.xml.transform.Source※3	×	text/xml, application/xml, application/*+xml
8	javax.xml.bind.JAXBElement<String>※4	×	text/xml, application/xml, application/*+xml
9	XmlRootElement アノテーションおよび/または XmlType アノテーションでアノテートされた JAXB クラス※4	×	text/xml, application/xml, application/*+xml
10	javax.ws.rs.core.MultivaluedMap<String,String>	○	application/x-www-form-urlencoded
11	org.w3c.dom.Document	×	text/xml, application/xml,

項番	Java の型	charset※1	MIME メディアタイプ
			application/*+xml
12	java.util.List<T>※5	×	text/xml, application/xml, application/*+xml
13	java.awt.image.RenderedImage	×	application/octet-stream, image/jpeg
14	com.cosminexus.jersey.core.provider.EntityHolder<T>※6	△	T に指定した型と同じ MIME メディアタイプです。
15	POJO※7	○	application/json

（凡例）

任意(*/*)：すべての MIME メディアタイプをサポートしていることを示します。

注※1

Content-Type HTTP ヘッダに charset パラメタが含まれる場合，エンティティパラメタにインジェクトされるときにその情報が考慮されるかどうかを示します。

○：考慮されます。Content-Type HTTP ヘッダに charset パラメタが含まれない場合は，UTF-8 が仮定されます。

△：T に指定した型に依存します。

×：考慮されません。

なお，Consumes アノテーションの値に charset パラメタが含まれていても無視されます。

注※2

JAX-RS エンジンは，J2EE サーバの環境に temp ディレクトリを作成し，一時ファイルを保存します。J2EE サーバの環境は cjsetup コマンドを使用して構築します。cjsetup コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjsetup（J2EE サーバのセットアップとアンセットアップ)」を参照してください。

注※3

次に示す実装クラスを使用できます。

・javax.xml.transform.stream.StreamSource

・javax.xml.transform.sax.SAXSource

・javax.xml.transform.dom.DOMSource

注※4

MIME メディアタイプが application/fastinfoset または application/json の場合，エラーにならないで正常終了します。

注※5

T には XmlRootElement アノテーションおよび/または XmlType アノテーションでアノテートされた JAXB クラスを指定できます。

注※6

T にはこの表の項番 2 から項番 13，および項番 15 の型を指定できます。

注※7

JSON POJO マッピングを有効にしてください。JSON POJO マッピングが有効でない場合の動作は，サポートされない Java 型がエンティティパラメタに指定された場合の動作と同じです。JSON POJO マッピングを有効にする方法については「18. JSON POJO マッピング」を参照してください。

(2) エンティティパラメタに関する注意事項

エンティティパラメタに関するそのほかの注意事項は次のとおりです。

エンティティパラメタへの変換で例外が発生した場合の動作について

エンティティパラメタへの変換で例外が発生した場合，エラーとなります。例外の処理については「17.1.8 例外ハンドリング」を参照してください。

エンティティパラメタの型がサポートされない Java の型の場合，または HTTP リクエストのエンティティボディが使用できない MIME メディアタイプの場合の動作について

次に示す Java の型のエンティティパラメタでは，HTTP リクエストのエンティティボディが使用できない MIME メディアタイプの場合，エラーとなり（KDJJ10024-E），HTTP ステータスコードに 415 が設定された，例外マッピングプロバイダで処理できる javax.ws.rs.WebApplicationException がスローされます。

1. javax.xml.bind.JAXBElement<String>
2. XmlRootElement アノテーションおよび/または XmlType アノテーションでアノテートされた JAXB クラス
3. javax.ws.rs.core.MultivaluedMap<String,String>
4. java.util.List<T>
5. POJO

ただし，1.または 2.で HTTP リクエストのエンティティボディの MIME メディアタイプが application/fastinfoset または application/json のときは，エラーにならないで正常終了します。

java.awt.image.RenderedImage のエンティティパラメタでは，HTTP リクエストのエンティティボディの MIME メディアタイプが image/*の場合，例外マッピングプロバイダで処理できる java.io.IOException がスローされます。

- com.cosminexus.jersey.core.provider.EntityHolder<T>のエンティティパラメタでは，HTTP リクエストにエンティティボディがある場合，次に示すどちらかのときにエラーとなり（KDJJ10003-E），HTTP ステータスコードに 500 が設定された，例外マッピングプロバイダで処理できる javax.ws.rs.WebApplicationException がスローされます。

  ・T がサポートされない型（表の項番 1 および項番 14）のとき

  ・T がサポートされる型（表の項番 2 から項番 13 まで）だが，HTTP リクエストのエンティティボディが使用できない MIME メディアタイプのとき

- エンティティパラメタの型がサポートされない Java の型を使用した場合，エラーとなり（KDJJ10024-E），HTTP ステータスコードに次の値が設定された，例外マッピングプロバイダで処理できる javax.ws.rs.WebApplicationException がスローされます。

  400：エンティティパラメタの型が java.lang.Object で，HTTP リクエストのエンティティボディの MIME メディアタイプが application/xml，text/xml，または application/*+xml のとき

  415：エンティティパラメタの型が java.lang.Object 以外の場合，またはエンティティパラメタの型が java.lang.Object で，HTTP リクエストのエンティティボディの MIME メディアタイプが application/xml，text/xml，application/*+xml のどれでもない場合

ただし，エンティティパラメタの型が javax.mail.internet.MimeMultipart で，HTTP リクエストのエンティティボディの MIME メディアタイプが multipart/*のときは，エラーにならないで正常終了します。

リソースメソッドで使用できるエンティティパラメタの数について

リソースメソッドで使用できるエンティティパラメタは一つだけです。リソースメソッドが複数のエンティティパラメタを持つ場合，警告メッセージがログに出力されます（KDJJ20012-W）。HTTP リクエストのエンティティボディは最初のエンティティパラメタだけにインジェクトされ，二つ目以降のエンティティパラメタの状態は保証されません。

GET 要求メソッド識別子を持つリソースメソッドについて

GET 要求メソッド識別子を持つリソースメソッドがエンティティパラメタを持つ場合，警告メッセージまたはエラーメッセージがログに出力されます（KDJJ20003-W または KDJJ10006-E）。
KDJJ20003-W および KDJJ10006-E については，「13.7.1　Web リソース初期化時の構文チェック（KDJJ20003-W と KDJJ10006-E）」を参照してください。

注意

Encoded アノテーションおよび DefaultValue アノテーションがエンティティパラメタにアノテートされている場合，無視されます。

エンティティパラメタの型パラメタが解決できない場合のメッセージの出力について

エンティティパラメタの型パラメタが解決できない場合は，警告メッセージまたはエラーメッセージがログに出力されます（KDJJ10006-E または KDJJ20003-W）。KDJJ20003-W と KDJJ10006-E については，「13.7.1　Web リソース初期化時の構文チェック（KDJJ20003-W と KDJJ10006-E）」を参照してください。

エンティティパラメタに特定の型が指定された場合の注意

エンティティパラメタの型が javax.ws.rs.core.MultivaluedMap<String,String>，または com.cosminexus.jersey.core.provider.EntityHolder<javax.ws.rs.core.MultivaluedMap<String,String>>の場合，次の注意事項があります。

- エンティティボディが JAX-RS 機能のサーブレットやサーブレットフィルタ以外からすでにアクセスされている場合，警告メッセージがログに出力されます（KDJJ20007-W）。

  このとき，エンティティパラメタの状態は不定です。エンティティボディに含まれるフォームパラメタは，FormParam アノテーションでアノテートされたパラメタから参照してください。

- エンティティボディに含まれるフォームパラメタ数の上限値は，デフォルトでは 10,000 です。

  リクエストのパラメタ数が指定した値を超えた場合，エラーとなり（KDJJ10042-E），HTTP ステータスコードに 413 を設定した，例外マッピングプロバイダで処理できる javax.ws.rs.WebApplicationException がスローされます。必要に応じて J2EE サーバ用ユーザプロパティファイル（usrconf.properties）の webserver.connector.limit.max_parameter_count プロパティで変更してください。

  J2EE サーバ用ユーザプロパティファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「2.2.3　usrconf.properties（J2EE サーバ用ユーザプロパティファイル）」を参照してください。

エンティティパラメタの型が POJO の場合は，次の注意事項があります。

- JSON POJO マッピングが有効になっていない場合，エラーとなり（KDJJ10024-E），HTTP ステータスコードに 415 を設定した，例外マッピングプロバイダで処理できる javax.ws.rs.WebApplicationException がスローされます。JSON POJO マッピングを有効にする方法については「18.　JSON POJO マッピング」を参照してください。

17.1.3　戻り値

リソースメソッドの戻り値に使用できる Java 型と MIME メディアタイプの組み合わせを次の表に示します。戻り値は HTTP レスポンスのエンティティボディに変換されます。なお，POJO に JAXB 仕様のアノテーションは使用しないでください。使用した場合，説明とは異なる動作になるおそれがあります。

表 17-6　リソースメソッドの戻り値に使用できる Java の型と MIME メディアタイプの組み合わせ

項番	Java の型	charset※1	MIME メディアタイプ
1	byte[]	×	任意(*/*)
2	java.lang.String	○	任意(*/*)
3	java.io.InputStream	×	任意(*/*)
4	java.io.Reader	○	任意(*/*)
5	java.io.File	×	任意(*/*)
6	javax.activation.DataSource	×	任意(*/*)
7	javax.xml.transform.Source※2	×	text/xml, application/xml, application/*+xml
8	javax.xml.bind.JAXBElement<String>※3	○	text/xml, application/xml, application/*+xml
9	XmlRootElement アノテーションでアノテートされた JAXB クラス※3	○	text/xml, application/xml, application/*+xml
10	javax.ws.rs.core.MultivaluedMap<String, String>	○	application/x-www-form-urlencoded
11	javax.ws.rs.core.StreamingOutput	×	任意(*/*)
12	org.w3c.dom.Document	×	text/xml, application/xml, application/*+xml
13	java.util.List<T>※4	○	text/xml, application/xml,

項番	Java の型	charset※1	MIME メディアタイプ
			application/*+xml
14	java.awt.image.RenderedImage	×	image/jpeg
15	void	−	任意(*/*)
16	javax.ws.rs.core.Response	○	任意(*/*)
17	javax.ws.rs.core.GenericEntity<T>※5	△	T に指定した型と同じ MIME メディアタイプです。
18	POJO※6	×※7	application/json

（凡例）

－：該当しないことを示します。

任意(*/*)：すべての MIME メディアタイプをサポートしていることを示します。

注※1

Produces アノテーションまたは戻り値に charset パラメタが含まれる場合，HTTP レスポンスに変換するときに，その情報が Content-Type HTTP ヘッダの charset パラメタに反映されるかどうかを示します。

○：反映されます。Produces アノテーションおよび戻り値に charset パラメタが含まれない場合は UTF-8 が仮定されます。

△：T に指定した型に依存します。

×：反映されません。

注※2

次に示す実装クラスを使用できます。

・javax.xml.transform.stream.StreamSource

・javax.xml.transform.sax.SAXSource

・javax.xml.transform.dom.DOMSource

注※3

MIME メディアタイプが application/fastinfoset または application/json の場合，エラーにならないで正常終了します。

注※4

T には XmlRootElement アノテーションでアノテートされた JAXB クラスを指定できます。

注※5

T にはこの表の項番 17 以外の型を指定できます。

注※6

JSON POJO マッピングを有効にしてください。JSON POJO マッピングが有効でない場合の動作は，サポートされない Java 型がエンティティパラメタに指定された場合の動作と同じです。JSON POJO マッピングを有効にする方法については「18. JSON POJO マッピング」を参照してください。

注※7

Content-Type HTTP ヘッダに charset パラメタを追加しないでください。

戻り値と HTTP レスポンスの対応を次の表に示します。

表 17–7　戻り値と HTTP レスポンスの対応

項番	戻り値		HTTP レスポンス	
	型	値	HTTP ステータスコード	エンティティボディ
1	void	−	204	空のエンティティボディ
2	Response	null ではないインスタンス	200	Response のエンティティプロパティ
3		null	204	空のエンティティボディ
4	String	null ではないインスタンス	200	String の値
5		null	204	空のエンティティボディ
6	void，Response，String 以外のサポートしている Java の型	null ではないインスタンス	200	戻り値のクラスに応じて変換されたエンティティボディ
7		null	204	空のエンティティボディ

（凡例）

−：戻り値の値がないことを示します。

戻り値の型がサポートされない Java の型の場合，エラーとなり（KDJJ10026-E），HTTP ステータスコードに 500 が設定された，例外マッピングプロバイダで処理できる javax.ws.rs.WebApplicationException がスローされます。ただし，戻り値の型が javax.mail.internet.MimeMultipart で，MIME メディアタイプが multipart/*のときは，エラーにならないで正常終了します。

HTTP レスポンスのエンティティボディへの変換で例外が発生した場合，エラーとなります。例外の処理については「17.1.8　例外ハンドリング」を参照してください。

戻り値の型が次に示す Java の型で，HTTP レスポンスのエンティティボディが使用できない MIME メディアタイプの場合，エラーとなり（KDJJ10026-E），HTTP ステータスコードに 500 が設定された，例外マッピングプロバイダで処理できる javax.ws.rs.WebApplicationException がスローされます。

1. javax.xml.bind.JAXBElement<String>
2. XmlRootElement アノテーションでアノテートされた JAXB クラス
3. javax.ws.rs.core.MultivaluedMap<String, String>
4. java.util.List<T>
5. java.awt.image.RenderedImage

ただし，1.または 2.で HTTP リクエストのエンティティボディの MIME メディアタイプが application/fastinfoset または application/json のときは，エラーにならないで正常終了します。

次に示す戻り値の場合，警告メッセージまたはエラーメッセージがログに出力されます（KDJJ20003-W または KDJJ10006-E）。KDJJ20003-W と KDJJ10006-E については，「13.7.1　Web リソース初期化時の構文チェック（KDJJ20003-W と KDJJ10006-E）」を参照してください。

- GET 要求メソッド識別子を持つリソースメソッドの戻り値の型が void の場合
- 戻り値の型パラメタが解決できない場合

17.1.4　パラメタ型

インジェクション用アノテーションを指定できるパラメタ型の一覧，および DefaultValue アノテーションを組み合わせて使用できるかどうかを次の表に示します。

表 17-8　各アノテーションをサポートしているパラメタの型

項番	データ型		アノテーション						
			PathParam	QueryParam	MatrixParam	CookieParam	HeaderParam	FormParam	Context
1	プリミティブ	int	△	○※1	○※1	○※2	○※1	○※1	×
2		short	△	○※1	○※1	○※2	○※1	○※1	×
3		long	△	○※1	○※1	○※2	○※1	○※1	×
4		float	△	○※1	○※1	○※2	○※1	○※1	×
5		double	△	○※1	○※1	○※2	○※1	○※1	×
6		char	△	○※1	○※1	○※2	○※1	○※1	×
7		byte	△	○※1	○※1	○※2	○※1	○※1	×
8		boolean	△	○※1	○※1	○※2	○※1	○※1	×
9	ラッパクラス	Integer	△	○※1	○※1	○※2	○※1	○※1	×
10		Short	△	○※1	○※1	○※2	○※1	○※1	×
11		Long	△	○※1	○※1	○※2	○※1	○※1	×
12		Float	△	○※1	○※1	○※2	○※1	○※1	×
13		Double	△	○※1	○※1	○※2	○※1	○※1	×
14		Character	×	×	×	×	×	×	×
15		Byte	△	○※1	○※1	○※2	○※1	○※1	×
16		Boolean	△	○※1	○※1	○※2	○※1	○※1	×
17	String 型の引数を一つ持つコンストラクタを持つ型		△※7	○※1, ※7	○※1, ※7	○※2, ※7	○※1, ※7	○※1, ※7	×
18	String 型の引数を一つ持ち，その型のインスタンスを返す static な valueOf メソッドを持つ型		△※7	○※1, ※7	○※1, ※7	○※2, ※7	○※1, ※7	○※1, ※7	×

項番	データ型		アノテーション						
			PathParam	QueryParam	MatrixParam	CookieParam	HeaderParam	FormParam	Context
19	String 型の引数を一つ持ち，その型のインスタンスを返す static な fromString メソッドを持つ型		△※7	○※1, ※7	○※1, ※7	○※2, ※7	○※1, ※7	○※1, ※7	×
20	String 型の引数を一つ持ち，その型のインスタンスを返す static な fromString メソッドを持つ enum 型		△※3, ※7	○※1, ※3, ※7	○※1, ※3, ※7	○※2, ※3, ※7	○※1, ※3, ※7	○※1, ※3, ※7	×
21	String 型の引数を一つ持ち，その型のインスタンスを返す static な valueOf と fromString メソッドの両方を持つ enum 以外の型		△※4, ※7	○※1, ※4, ※7	○※1, ※4, ※7	○※2, ※4, ※7	○※1, ※4, ※7	○※1, ※4, ※7	×
22	上記以外		×	×	×	×	×	×	×
23	List<T>	T が Integer の場合	△	○	○	×	○	○	×
24		T が Short の場合	△	○	○	×	○	○	×
25		T が Long の場合	△	○	○	×	○	○	×
26		T が Float の場合	△	○	○	×	○	○	×
27		T が Double の場合	△	○	○	×	○	○	×
28		T が Character の場合	×	×	×	×	×	×	×
29		T が Byte の場合	△	○	○	×	○	○	×
30		T が Boolean の場合	△	○	○	×	○	○	×
31		T が String 型の引数を一つ持つコンストラクタを持つ型の場合	△※7	○※7	○※7	×	○※7	○※7	×
32		T が String 型の引数を一つ持ち，その型のインスタンスを返す static な valueOf メソッドを持つ場合	△※7	○※7	○※7	×	○※7	○※7	×
33		T が String 型の引数を一つ持ち，その型のインスタンスを返す static な fromString メソッドを持つ場合	△※7	○※7	○※7	×	○※7	○※7	×
34		T が String 型の引数を一つ持ち，その型のインスタンスを返す static な fromString メソッドを持つ enum 型の場合	△※3, ※7	○※3, ※7	○※3, ※7	×	○※3, ※7	○※3, ※7	×
35		T が String 型の引数を一つ持ち，その型のインスタンス	△※4, ※7	○※4, ※7	○※4, ※7	×	○※4, ※7	○※4, ※7	×

項番	データ型		アノテーション						
			PathParam	QueryParam	MatrixParam	CookieParam	HeaderParam	FormParam	Context
		を返す static な valueOf メソッドと fromString メソッドの両方を持つ enum 以外の型の場合							
36		T が上記以外の場合	×	×	×	×	×	×	×
37	Set<T>※5	T が Integer の場合	△	○	○	×	○	○	×
38		T が Short の場合	△	○	○	×	○	○	×
39		T が Long の場合	△	○	○	×	○	○	×
40		T が Float の場合	△	○	○	×	○	○	×
41		T が Double の場合	△	○	○	×	○	○	×
42		T が Character の場合	×	×	×	×	×	×	×
43		T が Byte の場合	△	○	○	×	○	○	×
44		T が Boolean の場合	△	○	○	×	○	○	×
45		T が String 型の引数を一つ持つコンストラクタを持つ型の場合	△※7	○※7	○※7	×	○※7	○※7	×
46		T が String 型の引数を一つ持ち，その型のインスタンスを返す static な valueOf メソッドを持つ場合	△※7	○※7	○※7	×	○※7	○※7	×
47		T が String 型の引数を一つ持ち，その型のインスタンスを返す static な fromString メソッドを持つ場合	△※7	○※7	○※7	×	○※7	○※7	×
48		T が String 型の引数を一つ持ち，その型のインスタンスを返す static な valueOf メソッドと fromString メソッドの両方を持つ enum 型の場合	△※3, ※7	○※3, ※7	○※3, ※7	×	○※3, ※7	○※3, ※7	×
49		T が String 型の引数を一つ持ち，その型のインスタンスを返す valueOf メソッドと fromString メソッドの両方を持つ enum 以外の型の場合	△※4, ※7	○※4, ※7	○※4, ※7	×	○※4, ※7	○※4, ※7	×
50		T が上記以外の場合	×	×	×	×	×	×	×

項番	データ型		アノテーション						
			PathParam	QueryParam	MatrixParam	CookieParam	HeaderParam	FormParam	Context
51	Sorted Set<T>※5, ※6	T が Integer の場合	×	×	×	×	×	×	×
52		T が Short の場合	×	×	×	×	×	×	×
53		T が Long の場合	×	×	×	×	×	×	×
54		T が Float の場合	×	×	×	×	×	×	×
55		T が Double の場合	×	×	×	×	×	×	×
56		T が Character の場合	×	×	×	×	×	×	×
57		T が Byte の場合	×	×	×	×	×	×	×
58		T が Boolean の場合	×	×	×	×	×	×	×
59		T が String 型の引数を一つ持つコンストラクタを持つ型の場合	×※7, ※8	×※7, ※9	×※7, ※9	×	×※7, ※9	×※7, ※9	×
60		T が String 型の引数を一つ持ち，その型のインスタンスを返す static な valueOf メソッドを持つ場合	×※7, ※8	×※7, ※9	×※7, ※9	×	×※7, ※9	×※7, ※9	×
61		T が String 型の引数を一つ持ち，その型のインスタンスを返す static な fromString メソッドを持つ場合	×	×	×	×	×	×	×
62		T が String 型の引数を一つ持ち，その型のインスタンスを返す static な valueOf メソッドと fromString メソッドの両方を持つ enum 型の場合	×	×	×	×	×	×	×
63		T が String 型の引数を一つ持ち，その型のインスタンスを返す static な valueOf メソッドと fromString メソッドの両方を持つ enum 以外の型の場合	×	×	×	×	×	×	×
64		T が上記以外の場合	×	×	×	×	×	×	×
65	PathSegment		△	×	×	×	×	×	×
66	コンテキスト型	UriInfo	×	×	×	×	×	×	△
67		HttpHeaders	×	×	×	×	×	×	△

項番	データ型		アノテーション						
			PathParam	QueryParam	MatrixParam	CookieParam	HeaderParam	FormParam	Context
68		Request	×	×	×	×	×	×	△
69		SecurityContext	×	×	×	×	×	×	△
70		Providers	×	×	×	×	×	×	△
71		ServletConfig	×	×	×	×	×	×	△
72		ServletContext	×	×	×	×	×	×	△
73		HttpServletRequest	×	×	×	×	×	×	△
74		HttpServletResponse	×	×	×	×	×	×	△

（凡例）

○：インジェクション用アノテーションを使用できることを示します。

×：インジェクション用アノテーションを使用できないことを示します。

△：インジェクション用アノテーションを使用できることを示します。ただし，DefaultValue アノテーションを組み合わせて使用しないでください。

注※1

同じ名称のパラメタを複数受け取った場合，JAX-RS エンジンは最初のパラメタの値だけを使用します。

注※2

同じ名称のパラメタを複数受け取った場合，JAX-RS エンジンは最後のパラメタの値をインジェクトします。

注※3

String 型の引数を一つ持ち，その型のインスタンスを返す static な valueOf メソッドと static な fromString メソッドの両方を持つ型の場合，JAX-RS エンジンは static な fromString メソッドを使用します。

注※4

String 型の引数を一つ持ち，その型のインスタンスを返す static な valueOf メソッドと static な fromString メソッドの両方を持つ型の場合，JAX-RS エンジンは static な valueOf メソッドを使用します。

注※5

enum 型以外の場合，T に指定するクラス，またはその親以上のクラスでは，Java 言語の規約に従って java.lang.Object の equals()メソッドと hashCode()メソッドを適切に実装している必要があります。

注※6

enum 型以外の場合，T に指定するクラス，またはその親以上のクラスでは，java.lang.Comparable インタフェースを実装している必要があります。

注※7

valueOf メソッド，fromString メソッド，およびコンストラクタは，インジェクションの検証のため，初期化時に複数回呼び出されることがあります。

注※8

T が String 型の場合は使用できます。ただし，DefaultValue アノテーションを組み合わせて使用しないでください。

注※9

T が String 型の場合は使用できます。

上記の表の項番 22，項番 36，項番 50，項番 64，および項番 65 の型に各アノテーションが使用された場合，エラーが発生します（KDJJ10006-E）。ルートリソースクラスでは HTTP ステータスコード 500 の HTTP レスポンスが返されます。サブリソースクラスでは，例外マッピングプロバイダで処理できる java.lang.RuntimeException がスローされます。

また，上記の表の項番 14，項番 28，項番 42，および項番 56 の型に各アノテーションが使用された場合，例外マッピングプロバイダで処理できる java.lang.RuntimeException がスローされます。

17.1.5　例外のマッピング

ルートリソースクラスのリソースメソッドのパラメタ，コンストラクタのパラメタ，フィールド，bean プロパティ，およびサブリソースクラスのリソースメソッドのパラメタへのインジェクションで例外がスローされた場合，JAX-RS エンジンは次の表に示すように処理します。サポートされる Java 型とアノテーションについては，「17.1.4　パラメタ型」を参照してください。

表 17-9　インジェクションで発生した例外に対する JAX-RS エンジンの対応

項番	アノテーション	インジェクションで発生した例外	
		WebApplicationException	その他
1	MatrixParam	「17.1.8　例外ハンドリング」で説明するように WebApplicationException を処理します。	スローされた例外を，エンティティがなく，HTTP ステータスコード 404 である WebApplicationException でラップします。さらに，「17.1.8　例外ハンドリング」で説明するように処理します。
2	QueryParam		
3	PathParam		
4	CookieParam		スローされた例外を，エンティティがなく，HTTP ステータスコード 400 である WebApplicationException でラップします。さらに，「17.1.8　例外ハンドリング」で説明するように処理します。
5	FormParam		
6	HeaderParam		

注意事項

WebApplicationException は，「17.1.4　パラメタ型」の「各アノテーションをサポートしているパラメタの型」を説明している表の各パラメタ型のうち，項番 17～21，項番 31～36，項番 45～50，項番 59～64 だけで発生します。

17.1.6　URI テンプレート

Path アノテーションは，ルートリソースクラス，サブリソースメソッド，またはサブリソースロケータがどの URL に対する HTTP リクエストを処理するのかを指定するために使用します。Path アノテーションの値は URI テンプレートと呼ばれます。

ルートリソースにクラスレベルで指定される場合は，Web リソースを含む Web アプリケーション（WAR ファイル）のコンテキストルートに対して相対的な URI を示します。また，サブリソースメソッドまたはサブリソースロケータの場合は，ルートリソースクラスの URI テンプレートに対して相対的な URI を示します。

アノテーションの値は，自動的にエンコードされます。例えば，次に示すアノテーションの意味は同じです。

- @Path (”widget list/{id}”)
- @Path (”widget%20list/{id}”)

二つ以上のルートリソースクラスの Path アノテーションが同じ URI テンプレートを持つ場合，または同じ正規表現に解決される URI テンプレートを持つ場合，エラーとなり（KDJJ10006-E），ルートリソースクラスはインスタンス化されません。HTTP ステータスコードには 500 が返されます。

二つ以上のサブリソースメソッドの Path アノテーションが同じ URI テンプレートを持つ場合，または同じ正規表現に解決される URI テンプレートを持つ場合，メディアタイプ宣言や要求メソッド識別子などのほかの情報も一致するようなとき，エラーとなります（KDJJ10006-E）。ルートリソースクラスでは，HTTP ステータスコード 500 の HTTP レスポンスが返されます。サブリソースクラスでは，例外マッピングプロバイダで処理できる java.lang.RuntimeException がスローされます。

二つ以上のサブリソースロケータの Path アノテーションが同じ URI テンプレートを持つ場合，または同じ正規表現に解決される URI テンプレートを持つ場合，エラーとなります（KDJJ10006-E）。ルートリソースクラスでは HTTP ステータスコード 500 の HTTP レスポンスが返されます。サブリソースクラスでは，例外マッピングプロバイダで処理できる java.lang.RuntimeException がスローされます。

Path アノテーションがインタフェースや抽象クラスにアノテートされている場合，エラーとなり（KDJJ10006-E），クライアントからの要求は処理されません。HTTP ステータスコードには 500 が返されます。

(1) テンプレートパラメタ

URI テンプレートは，テンプレートパラメタと呼ばれる埋め込みのパラメタを 0 個以上含めることができます。テンプレートパラメタは，開始括弧（{）で記述し始め，スラッシュ（/）ではない一文字以上の英数字および記号を続けて記述し，最後に閉じ括弧（}）を記述します。テンプレートパラメタの実際の値は，PathParam アノテーションでアノテートされたパラメタ，フィールド，または bean プロパティにインジェクトすると，取得できます。テンプレートパラメタの記述方法については，標準仕様を確認してください。

テンプレートパラメタの例を次に示します。

```
package com.someshop;

import javax.ws.rs.PathParam;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
```

```
//ルートリソースクラス
@Path(”/customers”)
public class CustomerResource {
  //サブリソースメソッド
  @GET
  @Path(”{id}”)
  public String getCustomer(@PathParam(”id”) int id) {
    //割り当てた値を返すために実行する
  }
}
```

この例では，Path アノテーションに含まれる表現{id}はテンプレートパラメタです。ルートリソースクラス com.someshop.CustomerResource を含む Web アプリケーション（WAR ファイル）のコンテキストルートが"resource"で，Web アプリケーションが"someshop.com"というホストで公開されているとします。その場合，URL"http://someshop.com/resource/customers/333"に対する HTTP GET リクエストは，サブリソースメソッド getCustomer()にディスパッチされ，テンプレートパラメタ id の実際の値は PathParam アノテーションでアノテートされたパラメタ id にインジェクトされます。

一方，URL"http://someshop.com/resource/customers/333/444"に対する HTTP GET リクエストは，どのメソッドにもディスパッチされません。HTTP ステータスコードには 404 が返されます。

テンプレートパラメタは，Path アノテーションの値（URI テンプレート）のどこにでも埋め込むことができます。テンプレートパラメタを複数使用している例を次に示します。

```
package com.someshop;

import javax.ws.rs.PathParam;
import javax.ws.rs.GET;
import javax.ws.rs.Path;

//ルートリソースクラス
@Path(”/”)
public class CustomerResource {
  //サブリソースクラス
  @GET
  @Path(”customers/{firstname}-{lastname}”)
  public String getCustomer(@PathParam(”firstname”) String firstname,
      @PathParam(”lastname”) String lastname) {
    //割り当てた値を返すために実行する
  }
}
```

この例では，Path アノテーションに含まれる表現{firstname}および{lastname}は，ハイフンによって区切られた二つのテンプレートパラメタです。

URL"http://someshop.com/resource/customers/John-Smith"に対する HTTP GET リクエストは，サブリソースメソッド getCustomer()にディスパッチされ，テンプレートパラメタ firstname およびテンプレートパラメタ lastname の実際の値は，それぞれ PathParam アノテーションでアノテートされたパラメタ firstname およびパラメタ lastname にインジェクトされます。

(a) 正規表現

Path アノテーションでは，ワイルドカード以外の正規表現を使用できます。テンプレートパラメタでの正規表現の使用例を次に示します。

```
package com.someshop;

import javax.ws.rs.PathParam;
import javax.ws.rs.GET;
import javax.ws.rs.Path;

//ルートリソースクラス
@Path(”/customers”)
public class CustomerResource {
  //サブリソースメソッド1
  @GET
  @Path(”{id : ¥¥d+}”)
  public String getCustomer(@PathParam(”id”) int id) {
    // Implementation to return appropriate value
  }
//サブリソースメソッド2
@GET
  @Path(”{path : .+}”)
  public String getCustomerIdAndName(@PathParam(”path”) String path) {
    //割り当てた値を返すために実行する
  }
}
```

この例では，Path アノテーションに含まれる表現{id：¥¥d+}は，正規表現を使用しているテンプレートパラメタです。ここでは，識別子 id と正規表現"¥¥d+"を組み合わせて使用しています。識別子と正規表現は，コロン（:）で区切ります。

正規表現"¥¥d+"は，一桁以上の数字と一致します。URL"http://someshop.com/resource/customers/333"に対する HTTP GET リクエストは，サブリソースメソッド getCustomer()にディスパッチされます。

正規表現".+"は，どんな文字とでも一致します。URL"http://someshop.com/resource/customers/33/John/Smith"に対する HTTP GET リクエストは，サブリソースメソッド getCustomerIdAndName()にディスパッチされます。

(b) テンプレートパラメタを使用する場合の注意事項

次に示す場合，エラーが発生します（KDJJ10006-E）。

- テンプレートパラメタに不正な文字が使用されている。
- テンプレートパラメタに不正な構文の正規表現が記述されている。

ルートリソースクラスでは HTTP ステータスコード 500 の HTTP レスポンスが返されます。サブリソースクラスでは例外マッピングプロバイダで処理できる java.lang.RuntimeException がスローされます（KDJJ10006-E）。

17.1.7　サブリソースクラス

サブリソースクラスは，リソースメソッド，サブリソースメソッド，またはサブリソースロケータのどれかを一つ以上持ち，クラスレベルで Path アノテーションでアノテートされていない Java のクラスです。

サブリソースクラスの例を次に示します。

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Path;

public class Resource {
  @Path(”/subresourcemethod1”)
  @GET
  public String subResourceMethod1() {
    return ”from sub resource method1”;
  }
  @GET
  public String resourceMethod() {
    return ”from resource method”;
  }
}
```

サブリソースクラスのインスタンスは，JAX-RS エンジンによって生成されません。サブリソースクラスは，対応するサブリソースロケータでインスタンス化する必要があります。

(1)　メカニズム

サブリソースクラスは，次のように生成され，HTTP リクエストを処理します。

1. HTTP リクエストがサブリソースロケータにディスパッチされます。
2. サブリソースロケータはサブリソースクラスを生成し，HTTP リクエストの処理を生成したサブリソースクラスに委譲します。
3. HTTP リクエストはサブリソースクラスによって直接処理されるか，同じ仕組みによって，さらにサブリソースクラスに委譲されます。

サブリソースロケータについては，「17.1.1(6)　サブリソースロケータ」を参照してください。

JAX-RS エンジンは，サブリソースロケータのメソッドシグネチャで宣言された戻り値型ではなく，実行時にサブリソースロケータが返すインスタンスをサブリソースクラスとして扱い，処理を委譲します。

例えば，M，N，および O の三つのサブリソースクラスがあるとします。N は M を継承し，O は N を継承しています。同様に，戻り型 M を持つ R というサブリソースロケータがあるとします。サブリソースロケータが M のインスタンスを返した場合，サブリソースクラス M が HTTP リクエストを実行します。同様に，サブリソースロケータが N のインスタンスを返す場合は，サブリソースクラス N が HTTP リクエストを実行します。O のインスタンスを返す場合は，サブリソースクラス O が HTTP リクエストを実行します。

(2) ライフサイクル

サブリソースクラスのインスタンスは，JAX-RS エンジンによって生成されません。サブリソースクラスは，対応するサブリソースロケータでインスタンス化する必要があります。このため，サブリソースロケータまたはサブリソースクラスで，コンストラクタのパラメタ，フィールド，および bean プロパティを初期化する必要があります。

(3) コンストラクタ

サブリソースクラスのコンストラクタのパラメタでは，JAX-RS 仕様のアノテーションを使用しないでください。使用されている場合は無視されます。

(4) フィールドおよび bean プロパティ

JAX-RS 仕様のアノテーションをサブリソースクラスのフィールドおよび bean プロパティで使用しないでください。使用されている場合は，無視されます。

(5) リソースメソッド，サブリソースメソッド，およびサブリソースロケータ

サブリソースクラスのリソースメソッド，サブリソースメソッド，およびサブリソースメソッドは，次に説明する相違点を除いてルートリソースクラスの場合と同じです。ルートリソースクラスについては，次の個所を参照してください。

- 「17.1.1(4) リソースメソッド」
- 「17.1.1(5) サブリソースメソッド」
- 「17.1.1(6) サブリソースロケータ」

サブリソースロケータの戻り値の型が void の場合，エラーとなり，クライアントからの HTTP リクエストは処理されません。HTTP ステータスコードには 500 が返されます。なお，ログは JAX-RS 機能のログファイルではなく，J2EE サーバのログファイルを確認してください。

次に示す条件に当てはまる場合は，エラーとなり（KDJJ10006-E），例外マッピングプロバイダで処理できる java.lang.RuntimeException がスローされます。

- 一つのリソースメソッドに対し，二つ以上の要求メソッド識別子を使用している場合
- 二つ以上のリソースメソッドに対し，同じ要求メソッド識別子を使用している場合
- サブリソースロケータがエンティティパラメタを持っている場合

17.1.8 例外ハンドリング

JAX-RS エンジンは，次に示す個所からスローされる例外をここで説明するようにハンドリングします。

- リソースメソッド

- サブリソースメソッド
- サブリソースロケータ
- ルートリソースクラスのコンストラクタおよびサブリソースクラス

(1) WebApplicationException（例外マッピングプロバイダがない場合）

WebApplicationException がスローされる場合で，WebApplicationException またはその親以上の例外に対応する例外マッピングプロバイダがないとき，JAX-RS エンジンは次の表に示すように WebApplicationException をハンドリングします。

表 17-10 WebApplicationException のハンドリング（例外マッピングプロバイダがない場合）

項番	条件		ハンドリング結果	
	response プロパティ	response プロパティの HTTP ステータスコード	HTTP レスポンスの HTTP ステータスコード	メッセージ ID
1	設定されている	• 499 以下 • 列挙型 Response.Status にある値	WebApplicationException の response プロパティが持つ値が使用されます。	KDJJ30021-I
2	設定されている	• 499 以下 • 列挙型 Response.Status にない値	WebApplicationException の response プロパティが持つ値が使用されます。	KDJJ30022-I
3	設定されている	• 500 以上 • 列挙型 Response.Status にある値	WebApplicationException の response プロパティが持つ値が使用されます。	KDJJ10018-E
4	設定されている	• 500 以上 • 列挙型 Response.Status にない値	WebApplicationException の response プロパティが持つ値が使用されます。	KDJJ10019-E
5	設定されていない	－	500	KDJJ10018-E

（凡例）

－：該当しないことを示します。

WebApplicationException を生成し，response プロパティを設定する例を次に示します。

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;

//ルートリソースクラス
@Path("/root")
public class Resource {
```

```
  //サブリソースメソッド
  @Path("/subresourcemethod")
  @GET
  public String subResourceMethod() {
    //ResponseBuilderを使用してResponseインスタンスを生成
    ResponseBuilder rb = Response.status(208).
    entity("entity for WebApplicationException");

    //WebApplicationExceptionへResponseインスタンスを設定
    throw new WebApplicationException(rb.build());
}
}
```

この例では，ルートリソースクラス com.sample.resources.Resource を含む Web アプリケーション（WAR ファイル）のコンテキストルートが"resource"で，Web アプリケーションが"example.com"というホストで公開されているとします。その場合，URL"http://example.com/resource/root/subresourcemethod"に対する HTTP GET リクエストは，メソッド subResourceMethod()にディスパッチされます。また，HTTP レスポンスは，WebApplicationException の response プロパティから変換されます。

(2) その他の例外（例外マッピングプロバイダがない場合）

WebApplicationException 以外の例外がスローされる場合で，その例外またはその親以上の例外に対応する例外マッピングプロバイダがないとき，JAX-RS エンジンは次の表に示すようにその例外をハンドリングします。

表 17-11 その他の例外（例外マッピングプロバイダがない場合）

項番	条件	ハンドリングの結果		
	例外の種類	JAX-RS エンジンの動作	HTTP レスポンスの HTTP ステータスコード	メッセージ ID
1	ランタイム例外	そのランタイム例外をスローし直します。	500	KDJJ10010-E, KDJJ10039-E
2	上記以外	RuntimeException でその例外をラップしてスローします	500	KDJJ10017-E, KDJJ10039-E

(3) 例外マッピングプロバイダがある場合

スローされる例外，またはその親以上の例外に対応する例外マッピングプロバイダがある場合は，例外マッピングプロバイダの動作に依存します。なお，スローされる例外とその親以上の例外に対応する例外マッピングプロバイダが複数存在する場合は，その例外に最も近い例外（スローされた例外を含む）を処理できる例外マッピングプロバイダが動作します。

17.1.9　メディアタイプ宣言

Consumes アノテーションと Produces アノテーションをそれぞれ使用することで，Web リソースでサポートする MIME メディアタイプを指定できます。Consumes アノテーションと Produces アノテーションを使用していない場合は，すべてのメディアタイプがサポートされていると見されます。

Consumes アノテーションと Produces アノテーションは，次に示す場所で使用できます。

- ルートリソースクラス（クラスレベル）
- サブリソースクラス（クラスレベル）
- リソースメソッド（メソッドレベル）
- サブリソースメソッド（メソッドレベル）

メソッドレベルで使用されているアノテーションは，クラスレベルで使用されているアノテーションより優先されます。

二つ以上のリソースメソッドまたはサブリソースメソッドが同じ MIME メディアタイプを処理できる場合で，要求メソッド識別子やパスなどほかの情報も一致するようなとき，エラーが発生します（KDJJ10006-E)。ルートリソースクラスでは，HTTP ステータスコード 500 の HTTP レスポンスが返されます。サブリソースクラスでは，例外マッピングプロバイダで処理できる java.lang.RuntimeException がスローされます。

HTTP リクエストの Content-Type ヘッダが，どの Consumes アノテーションにも一致しない場合，エラーとなり（KDJJ10040-E)，HTTP ステータスコードに 415 が設定された，例外マッピングプロバイダで処理できる javax.ws.rs.WebApplicationException がスローされます。

HTTP レスポンスの Accept HTTP ヘッダが，どの Produces アノテーションにも一致しない場合，エラーとなり（KDJJ10041-E)，HTTP ステータスコードに 406 が設定された，例外マッピングプロバイダで処理できる javax.ws.rs.WebApplicationException がスローされます。

メディアタイプ宣言の例を次に示します。

```
package com.sample.resources;

java.awt.image.RenderedImage
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.Consumes;

@Path(”sample”)
@Produces(”image/jpeg”)
public class ImageBasedResource {

@GET
  public RenderedImage getAsImage() {
  //実装
```

```
  }

  @GET
  @Produces("text/html")
  public String getAsHtml() {
    //実装
  }

  @POST
  @Consumes("image/jpeg")
  public void addWidget(RenderedImage image) {
    //実装
  }
}
```

この例では，MIME メディアタイプ image/jpeg の HTTP レスポンスを要求する HTTP GET リクエストを処理するためにリソースメソッド getAsImage()が呼び出されます。

また，MIME メディアタイプ text/html の HTTP レスポンスを要求する HTTP GET リクエストを処理するためにリソースメソッド getAsHtml が呼び出されます。

さらに，MIME メディアタイプ image/jpeg のエンティティボディを持つ HTTP POST 要求を処理するためにリソースメソッド addImage が呼び出されます。

17.1.10　URL デコードの無効化

デフォルトでは，JAX-RS エンジンは次に示すアノテーションでアノテートされたパラメタ，フィールド，または bean プロパティにインジェクトするとき，インジェクトされる値が URL エンコードされている場合は自動でデコードします。デコードされていない元の値を利用した場合は，Encoded アノテーションを各アノテーションと組み合わせて使用してください。

- MatrixParam アノテーション
- QueryParam アノテーション
- PathParam アノテーション

17.1.11　アノテーションの継承

親クラスやインタフェースのメソッドで使用されている JAX-RS 仕様のアノテーションは，子クラスや実装クラスに継承されます。

アノテーションが継承される条件を次に示します。

- 子クラスのメソッドおよびそのパラメタで JAX-RS 仕様のアノテーションが使用されていない場合
- 実装クラスのメソッドおよびそのパラメタで JAX-RS 仕様のアノテーションが使用されていない場合

親クラスを継承し，かつインタフェースを実装している場合で，アノテーションを継承する条件に両方合致する場合は親クラスのアノテーションを優先します。

複数の親クラスを継承している場合，または複数のインタフェースを実装している場合，それぞれ最初に継承または実装している親クラスまたはインタフェースのアノテーションを優先します。

アノテーションが継承される例を次に示します。

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.QueryParam;

//インタフェース
public interface A {

  //JAX-RS仕様のアノテーションを使用しているメソッド
  @GET
  public String getValue(@QueryParam("query") String query);

}

package com.sample.resources;

import javax.ws.rs.Path;

//インタフェースを実装するルートリソースクラス
@Path("/root/")
public class Resource implements A {

  //メソッドの実装
  public String getValue(String query) {
    //実装

  }
}

package com.sample.resources;

import javax.ws.rs.Path;
import javax.ws.rs.Produces;

//インタフェースを実装するルートリソースクラス
@Path("/root1/")
public class Resource1 implements A {

  //メソッドの実装
  @Produces("text/xml")
  public String getValue(String query) {
    //実装
  }
}
```

この例では，インタフェース com.sample.resources.A，ルートリソースクラス com.sample.resources.Resource，およびルートリソースクラス com.sample.resources.Resource1 を

含む Web アプリケーション（WAR ファイル）のコンテキストルートが"resource"で，"example.com"というホストで公開されているとします。

URL"http://example.com/resource/root?query=10"に対する HTTP GET リクエストは，リソースメソッド getValue()にディスパッチされます。リソースメソッド getValue()はインタフェースの GET アノテーションを継承しているためです。

一方で，URL"http://example.com/resource/root1?query=10"に対する HTTP GET リクエストは，リソースメソッド getValue()にディスパッチされないで，HTTP ステータスコード 405 が返ります。リソースメソッド getValue()はインタフェースの GET アノテーションを継承していないためです。

17.2　プロバイダ

プロバイダは，Provider アノテーションでアノテートされ，標準仕様によって定義されるインタフェースを実装するクラスです。

プロバイダは，Web アプリケーション（WAR ファイル）ごとにインスタンス化されます。プロバイダのライフサイクルは次のとおりです。

1. コンストラクタが呼び出される
2. 必要なインジェクションが行われる
3. 適切なメソッドが呼び出される
4. GC（ガーベージコレクション）の対象になる

17.2.1　エンティティプロバイダ

JAX-RS 機能ではビルトインのエンティティプロバイダを提供しています。

17.2.2　例外マッピングプロバイダ

例外マッピングプロバイダで処理できる例外から HTTP レスポンスへのマッピングをカスタマイズする場合は，例外マッピングプロバイダを実装してください。

例外マッピングプロバイダは，ExceptionMapper<T>インタフェースを実装し，Provider アノテーションでアノテートします。

例外マッピングプロバイダの例を次に示します。

```
package com.sample.providers;

import javax.ws.rs.core.Response;
import javax.ws.rs.ext.ExceptionMapper;
import javax.ws.rs.ext.Provider;

//RuntimeExceptionからHTTPレスポンスへのマッピングをカスタマイズする例外マッピングプロバイダ
@Provider
public class RuntimeExceptionMapper implements
    ExceptionMapper<RuntimeException> {

  public Response toResponse(RuntimeException runtimeException) {
    int httpStatus = 0;
    String entity = "";

    //ResponseBuilderクラスを使用してHTTPレスポンスを作成する
    return Response.status(httpStatus).entity(entity).build();
```

```
    }
}
```

この例では，プロバイダ com.sample.providers.RuntimeExceptionMapper が，例外マッピングプロバイダで，型パラメタに RuntimeException を指定した ExceptionMapper インタフェースを実装しています。ここでは，ResponseBuilder クラスの toResponse メソッドを呼び出して HTTP レスポンスを作成しています。

なお，例外マッピングプロバイダは，一つの例外に対して一つだけ作成できます。一つの例外に対して二つ以上の例外マッピングプロバイダを作成すると，エラーになり（KDJJ10028-E，KDJJ10039-E），例外マッピングプロバイダはインスタンス化されません。HTTP ステータスコードには 500 が返されます。

(1) コンストラクタ

例外マッピングプロバイダは，public デフォルトコンストラクタ（明示的に宣言されないコンストラクタ）も含めて，少なくとも一つ以上の public コンストラクタを持つ必要があります。

public コンストラクタが一つも宣言されていない場合，エラーとなり（KDJJ10002-E，KDJJ10006-E），例外マッピングプロバイダはインスタンス化されません。HTTP ステータスコードには 500 が返されます。

次に示すコンストラクタは public コンストラクタではありません。

- private コンストラクタ
- protected コンストラクタ
- アクセス識別子を持たないコンストラクタ

コンストラクタのパラメタでは，インジェクション用アノテーションとして Context アノテーションを使用できます。Context アノテーション以外のインジェクション用アノテーションが使用されている場合，エラーとなり（KDJJ10006-E），例外マッピングプロバイダはインスタンス化されません。HTTP ステータスコードには 500 が返されます。使用できないインジェクション用アノテーションを次に示します。

- MatrixParam
- QueryParam
- PathParam
- CookieParam
- HeaderParam
- FormParam

これらのパラメタにインジェクション用アノテーション以外を指定した場合，エラーメッセージが出力されます（KDJJ10006-E）。HTTP ステータスコードには 500 が返されます。

例外マッピングプロバイダがパラメタを持つ public コンストラクタを一つ以上持つ場合，JAX-RS エンジンは，最もパラメタ数が多いコンストラクタを使用して例外マッピングプロバイダをインスタンス化しま

す。最もパラメタ数が多いコンストラクタを二つ以上持つ場合，最初に定義されているコンストラクタを使用して例外マッピングプロバイダをインスタンス化します。このとき，警告メッセージがログに出力されます（KDJJ20011-W）。

例外マッピングプロバイダがインスタンス化されるときにランタイム例外が発生した場合，エラーとなります（KDJJ10028-E，KDJJ10039-E）。HTTP ステータスコードには 500 が返されます。

(2) フィールドおよび bean プロパティ

例外マッピングプロバイダのフィールドおよび bean プロパティ上では，インジェクション用アノテーションとして Context アノテーションを使用できます。

Context アノテーション以外のインジェクション用アノテーションが使用されている場合，無視されます。使用できないアノテーションを次に示します。

- MatrixParam
- QueryParam
- PathParam
- CookieParam
- HeaderParam
- FormParam

18 JSON POJO マッピング

この章では，JAX-RS 機能が提供している，JSON と POJO のマッピングを行うビルトインのエンティティプロバイダのマッピングについて説明します。

なお，POJO に JAXB 仕様のアノテーションは使用しないでください。使用した場合，説明とは異なる動作になるおそれがあります。

18.1　JSON POJO マッピングの設定

JSON POJO マッピングでは，POJO から JSON，または JSON から POJO へのデータ変換ができます。

ここでは，サーバ側およびクライアント側で JSON POJO マッピングをする方法について説明します。

18.1.1　サーバ側

JSON POJO マッピングは，次の方法で有効または無効を指定できます。

- 共通定義ファイル（cjrconf.properties）

  com.sun.jersey.api.json.POJOMappingFeature の値に「true（有効）」または「false（無効）」を指定します。値の大文字と小文字は区別されません。
- web.xml のサーブレット初期化パラメタ

  サーブレット初期化パラメタ（com.sun.jersey.api.json.POJOMappingFeature）で次の表に示す値を設定します。値の大文字と小文字は区別されません。

表 18-1　サーブレット初期化パラメタ（com.sun.jersey.api.json.POJOMappingFeature）で指定できる値

値	説明
true	JSON POJO マッピングが有効になります。
false	JSON POJO マッピングが無効になります。
true，false 以外の値	web.xml のサーブレット初期化パラメタで指定した値は無視され，共通定義ファイルで指定した値が採用されます。

なお，共通定義ファイルとサーブレット初期化パラメタの両方が指定された場合は，サーブレット初期化パラメタで指定した値が優先されます。

18.1.2　クライアント側

クライアントでは，共通定義ファイル（cjrconf.properties）の com.sun.jersey.api.json.POJOMappingFeature の値に「true（有効）」または「false（無効）」を指定するか，Client オブジェクトのプロパティマップに com.sun.jersey.api.json.POJOMappingFeature フィーチャを追加してください。com.sun.jersey.api.json.POJOMappingFeature フィーチャを追加する実装例を次に示します。

```
// ClientConfigオブジェクトを生成する
ClientConfig cc = new DefaultClientConfig();
// ClientConfigオブジェクトにJSON POJOマッピングを有効にするための
// フィーチャを追加する
cc.getFeatures().put(JSONConfiguration.FEATURE_POJO_MAPPING, true);
```

```
// 上記で生成したClientConfigオブジェクトを指定して
// Clientオブジェクトを生成する
Client client = Client.create(cc);
```

この例では，JSONConfiguration.FEATURE_POJO_MAPPING フィーチャをあらかじめ追加した ClientConfig を指定して，Client オブジェクトを生成しています。

なお，共通定義ファイルと Client オブジェクトの両方が指定された場合は，Client オブジェクトに指定した値が優先されます。

18.2　POJO から JSON へのマッピング

JSON フォーマットの詳細については RFC 4627 を参照してください。ここでは，POJO から JSON へのマッピング要件と，指定できるデータ型について説明します。

18.2.1　マッピング要件

POJO から JSON へのマッピング要件について次に説明します。

(1)　POJO

- POJO クラスは，public スコープまたはパッケージスコープで定義してください。final 修飾子を指定しても問題ありません。
- 任意のコンストラクタを含めることができます。
- JSON の同じ要素にマッピングできるフィールドまたはプロパティが複数ある場合，JsonMappingException がスローされます。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。
- フィールドおよび Bean プロパティの名称が同じ場合，Bean プロパティが優先されます。

(2)　フィールド

- フィールドは public スコープで定義してください。public スコープ以外で定義された場合はマッピングされません。
- static 修飾子または transient 修飾子をフィールドに指定しないでください。指定されている場合はマッピングされません。
- final 修飾子は指定しても問題ありません。

(3)　Bean プロパティ

- Bean プロパティの getter メソッドは public スコープで定義してください。public スコープ以外で定義された場合はマッピングされません。
- 同じプロパティに対して大文字・小文字の違いだけがある getter メソッドが複数ある場合，JsonMappingException がスローされます。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。
- static を Bean プロパティに指定しないでください。指定されている場合はマッピングされません。
- final 修飾子は指定しても問題ありません。
- Bean プロパティは書き込み可能なプロパティである必要はありません。Bean プロパティの詳細については JavaBeans 仕様を参照してください。

18.2.2　指定できるデータ型

POJO のフィールドおよび Bean プロパティに指定できるデータ型を次に示します。

表 18–2　POJO のフィールドおよび Bean プロパティに指定できるデータ型（POJO から JSON へのマッピング）

項番	データ	
1	プリミティブ	int
2		short
3		long
4		float
5		double
6		char
7		byte
8		boolean
9	ラッパクラス	Integer
10		Short
11		Long
12		Float
13		Double
14		Character
15		Byte
16		Boolean
17	java.lang.String	
18	java.math.BigInteger	
19	java.math.BigDecimal	
20	java.util.Date	
21	java.util.Calendar	
22	java.lang.Enum	
23	POJO※1	
24	java.util.List<T>※2	
25	java.util.Set<T>※2	
26	java.util.Map<T,T>※2	

項番	データ
27	項番 1～26 のどれかの配列

注※1

フィールドまたは Bean プロパティとして再帰的に POJO を持つことができます。サポートされる POJO の条件については，「18.2　POJO から JSON へのマッピング」を参照してください。

注※2

T の型は表の項番 1～26 のどれかになります。

注意事項を次に示します。

- フィールドまたは Bean プロパティが初期化されていない場合，生成された JSON にマッピングされる値は，それぞれのデータ型のデフォルト値（プリミティブ型ではそれぞれのデフォルト値，オブジェクト型では null）になります。
- 表の項番 9～27 の型の値が null の場合，生成される JSON の対応する値には null がマッピングされます。
- 表の項番 24～27 の型の値に null が含まれている場合，生成される JSON の対応する値には null がマッピングされます。
- 表の項番 20 または 21 の型の値が null 以外の場合，生成される JSON の対応する値はミリ秒で表現される等しい値がマッピングされます。例えば，データ型として Date が使用されている場合，Date クラスの getTime()メソッドを呼び出して得られる値がマッピングされます。Calendar 型では，Calendar クラスの getTime()で取得される Date オブジェクトに対して Date クラスの getTime()メソッドを呼び出して得られる値がマッピングされます。
- 表の項番 26 の型の値が null の場合，JsonMappingException がスローされます。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。
- 表の項番 27 が char 配列または byte 配列のとき，生成される JSON の対応する値は，配列ではなくそれぞれ次の値がマッピングされます。
  - char 配列：配列から生成される文字列
  - byte 配列：配列から生成される Base64 エンコードされた文字列

  例を次に示します。
  - 値が{'a','b'}で名称が"bean"である char[]型の Bean プロパティは，{"bean":["a","b"]}ではなく{"bean":["ab"]}にマッピングされます。
  - 値が{1,2}で名称が"bean"である byte[]型の Bean プロパティは{"bean":[1,2]}ではなく{"bean":["AQ=="]}にマッピングされます。

18.2.3　例外ハンドリング

POJO から JSON へのマッピングは，JAX-RS 仕様のエンティティプロバイダの仕組みで実装されています。このため，マッピングで発生する例外はほかのマッピングで例外が発生する場合と同じように処理さ

れます。マッピングで例外がどのように処理されるかについては，サーバ側については「17.1.3　戻り値」を，クライアント側については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。

18.3　JSON から POJO へのマッピング

JSON フォーマットの詳細については RFC 4627 を参照してください。ここでは，JSON から POJO へのマッピング要件と，指定できるデータ型について説明します。

18.3.1　マッピング要件

(1)　POJO

- POJO クラスは，public スコープまたはパッケージスコープで定義してください。final 修飾子を指定しても問題ありません。
- POJO クラスはデフォルトコンストラクタを持つ必要があります。デフォルトコンストラクタは明示的に宣言してもしなくてもどちらでも問題ありません。また，public スコープ，private スコープ，protected スコープ，またはパッケージスコープのどれでも問題ありません。POJO クラスがデフォルトコンストラクタを持たない場合，JsonMappingException がスローされます。Java 言語仕様にあるとおり，パラメタを持つコンストラクタを宣言した場合は明示的にデフォルトコンストラクタを宣言する必要があることに注意してください。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。
- POJO クラスはインナークラスとして宣言された型のフィールドまたは Bean プロパティを持つことができます。この場合，インナークラスはインタフェースまたは非 static なクラスにしないでください。インタフェースまたは非 static なクラスであるインナークラスとして宣言された型のフィールドまたは Bean プロパティがある場合，JsonMappingException がスローされます。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。
- POJO にマッピングできない値が JSON フォーマットにある場合，JsonMappingException がスローされます。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。
- フィールドおよび Bean プロパティの名称が同じ場合，Bean プロパティが優先されます。
- POJO がマッピングの対象となるフィールドも Bean プロパティも持たなくてもエラーにならないのは，JSON フォーマットが空の場合だけです。

(2)　フィールド

- フィールドは public スコープで定義してください。public スコープではない場合は，JsonMappingException がスローされます。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。
- static 修飾子，または transient 修飾子および final 修飾子をフィールドに指定しないでください。指定されている場合はマッピングの対象になりません。
- JSON フォーマットに対応する値がないフィールドは初期化されません（例外はスローされません）。

(3) Bean プロパティ

- Bean プロパティの setter メソッドには public スコープを推奨します。なお，private スコープ，protected スコープ，パッケージスコープも使用できます。
- 大文字・小文字の違いだけがある複数の setter メソッドを宣言しないでください。複数宣言されている場合，JsonMappingException がスローされます。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。
- static を Bean プロパティに指定しないでください。指定されている場合はマッピングの対象になりません。
- final 修飾子を指定しても問題ありません。
- Bean プロパティは読み取り可能なプロパティである必要はありません。Bean プロパティの詳細については JavaBeans 仕様を参照してください。
- JSON フォーマットに対応する値がない Bean プロパティの setter メソッドは呼び出されません（例外はスローされません）。

18.3.2　指定できるデータ型

POJO のフィールドおよび Bean プロパティに指定できるデータ型を次に示します。

表 18–3　POJO のフィールドおよび Bean プロパティに指定できるデータ型（JSON から POJO へのマッピング）

項番	データ	
1	プリミティブ	int
2		short
3		long
4		float
5		double
6		char
7		byte
8		boolean
9	ラッパクラス	Integer
10		Short
11		Long
12		Float
13		Double

項番	データ	
14		Character
15		Byte
16		Boolean
17	java.lang.String	
18	java.math.BigInteger	
19	java.math.BigDecimal	
20	java.util.Date	
21	java.util.Calendar	
22	java.lang.Enum	
23	POJO※1	
24	java.util.List<T>※2	
25	java.util.Set<T>※2	
26	java.util.Map<K,V>※2	
27	項番 1～26 のどれかの配列※2，※3	

注※1

フィールドまたは Bean プロパティとして再帰的に POJO を持つことができます。サポートされる POJO の条件については，「18.3　JSON から POJO へのマッピング」を参照してください。

注※2

T，K，および V の型は表の項番 1～26 のどれかになります。

注※3

対応する JSON フォーマットの値は，配列の構造に準拠している必要があります。

要素を設定するときの注意事項を次に示します。

- JSON フォーマットに次の誤りがある場合，JsonParseException がスローされます。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。
  - 数値，文字列，配列，オブジェクト，true，false または null ではない値が含まれている場合
  - 二つのオブジェクトがコンマで区切られていない場合
  - トークンの間に許可されていない文字が使用されている場合（許可されている文字はスペース，ラインフィード，キャリッジリターン，水平タブ）
  - 数値の形式が不正な場合（12，1.2eE8，0X3F7A など）
- データ型が数値（表の項番 1～5，7，9～13，15）のフィールドまたは Bean プロパティに対応する JSON フォーマットの値が数値でない場合，JsonMappingException がスローされます。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。

(1) int 型（プリミティブ型）

- 対応する JSON フォーマットの値が空の文字列の場合，JsonMappingException がスローされます。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。
- 対応する JSON フォーマットの値が null の場合，int 型のデフォルト値で初期化されます。
- 対応する JSON フォーマットの値は int 型の範囲内の数値である必要があります。値が int 型の範囲外の場合，動作は保証されません。

(2) short 型（プリミティブ型）

- 対応する JSON フォーマットの値が null または空の文字列の場合，short 型のデフォルト値で初期化されます。
- 対応する JSON フォーマットの値は short 型の範囲内の数値である必要があります。値が short 型の範囲外の場合，動作は保証されません。

(3) long 型（プリミティブ型）

- 対応する JSON フォーマットの値が空の文字列の場合，JsonMappingException がスローされます。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。
- 対応する JSON フォーマットの値が null の場合，long 型のデフォルト値で初期化されます。
- 対応する JSON フォーマットの値は long 型の範囲内の数値である必要があります。値が long 型の範囲外の場合，動作は保証されません。

(4) float 型（プリミティブ型）

- 対応する JSON フォーマットの値が空の文字列の場合，JsonMappingException がスローされます。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。
- 対応する JSON フォーマットの値が null の場合，float 型のデフォルト値で初期化されます。
- 対応する JSON フォーマットの値は float 型の範囲内の数値である必要があります。値が float 型の範囲外の場合，動作は保証されません。

(5) double 型（プリミティブ型）

- 対応する JSON フォーマットの値が空の文字列の場合，JsonMappingException がスローされます。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。
- 対応する JSON フォーマットの値が null の場合，double 型のデフォルト値で初期化されます。
- 対応する JSON フォーマットの値は double 型の範囲内の数値である必要があります。値が double 型の範囲外の場合，動作は保証されません。

(6) char 型（プリミティブ型）

- 対応する JSON フォーマットの値が空の文字列の場合，JsonMappingException がスローされます。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。
- 対応する JSON フォーマットの値が null の場合，char 型のデフォルト値で初期化されます。
- 対応する JSON フォーマットの値は char 型の範囲内の数値である必要があります。値が char 型の範囲外の場合，動作は保証されません。

(7) byte 型（プリミティブ型）

- 対応する JSON フォーマットの値が null または空の文字列の場合，byte 型のデフォルト値で初期化されます。
- 対応する JSON フォーマットの値は byte 型の範囲内の数値である必要があります。値が byte 型の範囲外の場合，動作は保証されません。

(8) boolean 型（プリミティブ型）

- 対応する JSON フォーマットの値が null または空の文字列の場合，boolean 型のデフォルト値で初期化されます。
- 対応する JSON フォーマットの値に true，false（大文字と小文字を区別します）以外が設定された場合，JAX-RS エンジンの動作は定義されません。

(9) Integer 型（ラッパクラス型）

- 対応する JSON フォーマットの値が null または空の文字列の場合，null がマッピングされます。
- 対応する JSON フォーマットの値は Integer 型の範囲内の数値である必要があります。値が Integer 型の範囲外の場合，動作は保証されません。

(10) Short 型（ラッパクラス型）

- 対応する JSON フォーマットの値が空の文字列の場合，Short 型のデフォルト値で初期化されます。
- 対応する JSON フォーマットの値が null の場合，null がマッピングされます。
- 対応する JSON フォーマットの値は Short 型の範囲内の数値である必要があります。値が Short 型の範囲外の場合，動作は保証されません。

(11) Long 型（ラッパクラス型）

- 対応する JSON フォーマットの値が null または空の文字列の場合，null がマッピングされます。
- 対応する JSON フォーマットの値は Long 型の範囲内の数値である必要があります。値が Long 型の範囲外の場合，動作は保証されません。

(12) Float 型（ラッパクラス型）

- 対応する JSON フォーマットの値が null または空の文字列の場合，null がマッピングされます。
- 対応する JSON フォーマットの値は Float 型の範囲内の数値である必要があります。値が Float 型の範囲外の場合，動作は保証されません。

(13) Double 型（ラッパクラス型）

- 対応する JSON フォーマットの値が null または空の文字列の場合，null がマッピングされます。
- 対応する JSON フォーマットの値は Double 型の範囲内の数値である必要があります。値が Double 型の範囲外の場合，動作は保証されません。

(14) Character 型（ラッパクラス型）

- 対応する JSON フォーマットの値が空の文字列の場合，JsonMappingException がスローされます。例外ハンドリングについては「18.3.3 例外ハンドリング」を参照してください。
- 対応する JSON フォーマットの値が null の場合，Char 型のデフォルト値で初期化されます。
- 対応する JSON フォーマットの値は Char 型の範囲内の数値である必要があります。値が char 型の範囲外の場合，動作は保証されません。

(15) Byte 型（ラッパクラス型）

- 対応する JSON フォーマットの値が空の文字列の場合，Short 型のデフォルト値で初期化されます。
- 対応する JSON フォーマットの値が null の場合，null がマッピングされます。
- 対応する JSON フォーマットの値は Short 型の範囲内の数値である必要があります。値が Short 型の範囲外の場合，動作は保証されません。

(16) Boolean 型（ラッパクラス型）

- 対応する JSON フォーマットの値が空の文字列の場合，Boolean 型のデフォルト値で初期化されます。
- 対応する JSON フォーマットの値が null の場合，null がマッピングされます。
- 対応する JSON フォーマットの値に true，false（大文字と小文字を区別します）以外が設定された場合，JAX-RS エンジンの動作は定義されません。

(17) java.lang.String 型

- 対応する JSON フォーマットの値が空の文字列の場合，空の文字列がマッピングされます。
- 対応する JSON フォーマットの値が null の場合，null がマッピングされます。
- 対応する JSON フォーマットの値に引用符（"）が含まれている場合，エスケープされている必要があります。エスケープされていない場合は JsonParseException がスローされます。例外ハンドリングについては「18.3.3 例外ハンドリング」を参照してください。

- 対応する JSON フォーマットの値に¥n，¥r，¥t，¥b などの制御文字が含まれている場合，エスケープされている必要があります。エスケープされていない場合は JsonParseException がスローされます。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。
- 対応する JSON フォーマットの値に制御文字の¥u が含まれている場合，エスケープされている必要があります。また，¥u に続く文字列は 16 進数の数値である必要があります。これらの条件に反する場合は JsonParseException がスローされます。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。

(18)　java.math.BigInteger 型，java.math.BigDecimal 型

- 対応する JSON フォーマットの値が null または空の文字列の場合，null がマッピングされます。
- 対応する JSON フォーマットの値が数値以外の場合，JsonMappingException がスローされます。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。

(19)　java.util.Date 型，java.util.Calendar 型

- 対応する JSON フォーマットの値が null または空の文字列の場合，null がマッピングされます。
- サポートされている標準のフォーマットは次のとおりです。
  - yyyy-MM-dd'T'HH:mm:ss.SSSZ
  - yyyy-MM-dd'T'HH:mm:ss.SSS'Z'
  - EEE, dd MMM yyyy HH:mm:ss zzz
  - yyyy-MM-dd

  それぞれのパターン文字の意味と例を次の表に示します。

要素	意味	例
yyyy	年を 4 桁の数値で表します。	2013
MM	月を 2 桁の数値で表します。	04
MMM	月を 3 文字の文字列で表します。	Apr
dd	月における日を 2 桁の数値で表します。	30
EEE	曜日を 3 文字の文字列で表します。	Sun
'T'	固定文字です。	−
HH	1 日における時を 2 桁の数値で表します。	23
mm	分を 2 桁の数値で表します。	30
ss	秒を 2 桁の数値で表します。	10
SSS	ミリ秒を 3 桁の数値で表します。	978
'Z'	固定文字です。	−
Z	RFC 822 で定義されているタイムゾーンを表します。	-0530

要素	意味	例
zzz	標準時間を表します。	IST

（凡例）

－：固定文字のため例はありません。

Date または Calendar のフォーマットにおける各パターン文字，特にロケールに依存する「E」および「M」については，Java SE 仕様の SimpleDateFormat クラスに関する情報を参照してください。

- 対応する JSON フォーマットの値がサポートされないフォーマットの場合，JsonMappingException がスローされます。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。
- 対応する JSON フォーマットの値が long 型の範囲の数値の場合，つまり Date 型，または Calendar 型に対応する値がミリ秒で表現されている場合，それぞれそのミリ秒で初期化された Date 型，または Calendar 型のオブジェクトがマッピングされます。値が long 型の範囲外の場合，動作は保証されません。
- 例えば，対応する JSON フォーマットの値が 1346850421185 の場合，Date 型のフィールドまたは Bean プロパティには，date パラメタに 1346850421185 を指定して Date クラスの Date(long date) コンストラクタを呼び出した結果（Date オブジェクト）がマッピングされます。Calendar 型のフィールドまたは Bean プロパティには，同じように Date オブジェクトを生成し，さらに date パラメタにその Date オブジェクトを指定して Calendar クラスの setTime(Date date)メソッドの呼び出した結果（Calendar オブジェクト）がマッピングされます。

(20)　java.lang.Enum 型

- 対応する JSON フォーマットの値が空の文字列の場合，JsonMappingException がスローされます。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。
- 対応する JSON フォーマットの値が null の場合，null がマッピングされます。
- 対応する JSON フォーマットの値が Enum 型に含まれない場合，JsonMappingException がスローされます。例外ハンドリングついては「18.3.3　例外ハンドリング」を参照してください。

(21)　java.util.List<T>型

- 対応する JSON フォーマットの値が空のリスト（[]）の場合，空のリストがマッピングされます。
- 対応する JSON フォーマットの値が null の場合，null がマッピングされます。
- 対応する JSON フォーマットのリストに null が含まれている場合，マッピングされる List<T>型のフィールド，または Bean プロパティの対応する要素には null が設定されます。
- 対応する JSON フォーマットのリストに含まれる値は，T に設定する型の条件に準拠している必要があります。条件については，「18.3.2　指定できるデータ型」のそれぞれの型の注意事項を参照してください。条件に反する場合の動作はそれぞれの注意事項に記載されているとおりです。

(22)　java.util.Set<T>型

- 対応する JSON フォーマットの値が空のリスト（[]）の場合，空のセットがマッピングされます。
- 対応する JSON フォーマットの値が null の場合，null がマッピングされます。
- 対応する JSON フォーマットのリストに null が含まれている場合，マッピングされる Set<T>型のフィールド，または Bean プロパティの対応する要素には null が設定されます。
- 対応する JSON フォーマットのセットに含まれる値は，T に設定する型の条件に準拠している必要があります。条件については，「18.3.2　指定できるデータ型」のそれぞれの型の注意事項を参照してください。条件に反する場合の動作はそれぞれの注意事項に記載されているとおりです。

(23)　java.util.Map<K,V>型

- 対応する JSON フォーマットの値が空のマップ（{}）の場合，空のマップがマッピングされます。
- 対応する JSON フォーマットの値が null の場合，null がマッピングされます。
- 対応する JSON フォーマットのマップにキーとして null が含まれている場合，JsonParseException がスローされます。例外ハンドリングについては「18.3.3　例外ハンドリング」を参照してください。
- 対応する JSON フォーマットのマップに値として null が含まれている場合，マッピングされる Map<K,V>型のフィールド，または Bean プロパティの対応する値には null が設定されます。
- 対応する JSON フォーマットのマップに含まれるキーおよび値は，それぞれ K，V に設定する型の条件に準拠している必要があります。条件については，「18.3.2　指定できるデータ型」のそれぞれの型の注意事項を参照してください。条件に反する場合の動作はそれぞれの注意事項に記載されているとおりです。

18.3.3　例外ハンドリング

JSON から POJO へのマッピングは，JAX-RS 仕様のエンティティプロバイダの仕組みで実装されています。このため，マッピングで発生する例外はほかのマッピングで例外が発生する場合と同じように処理されます。マッピングで例外がどのように処理されるかについては，サーバ側については「17.1.2　エンティティパラメタ」を，クライアント側については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。

18.4　マッピング中に発生する例外

JSON POJO マッピング中に発生する例外を次の表に示します。

表 18–4　JSON POJO マッピング中に発生する例外

項番	インタフェースまたはクラス
com.cosminexus.org.codehaus.jackson.map パッケージ	
1	JsonMappingException
com.cosminexus.org.codehaus.jackson パッケージ	
2	JsonParseException
3	JsonGenerationException

19 JAX-WS 仕様のサポート範囲

この章では，Web サービスを開発するときに留意が必要な，JAX-WS 仕様のサポート範囲について説明します。

19.1　JAX-WS 2.2 仕様のサポート範囲

JAX-WS 2.2 仕様の機能のサポート範囲，および Conformance への対応について説明します。

19.1.1　JAX-WS 2.2 仕様の機能のサポート範囲

JAX-WS 2.2 仕様の機能のサポート範囲を次の表に示します。

なお，前提条件については，「1.4.2　機能および仕様に関する前提条件」を参照してください。

表 19–1　JAX-WS 2.2 仕様の機能のサポート範囲

分類			サポート	備考
大分類※1	小分類※2			
2	WSDL 1.1 から Java へのマッピング		○	
2	埋め込みによるバインディング宣言，または外部バインディングファイルによるマッピングのカスタマイズ		○	バンディング宣言のサポート範囲については，「15.2　WSDL から Java へのマッピングのカスタマイズ」を参照してください。
2	生成される Java コードへのアノテーションの付与		○	アノテーションのサポート範囲については，「16.2.1　アノテーション一覧」を参照してください。
2.1	名前空間からパッケージへのマッピング		○	マッピングについては，「15.1.1　名前空間からパッケージ名へのマッピング」を参照してください。
2.2	ポートタイプから SEI へのマッピング		○	マッピングについては，「15.1.2　ポートタイプから SEI 名へのマッピング」を参照してください。
2.3	オペレーションから SEI のメソッドへのマッピング		○	マッピングについては，「15.1.3　オペレーションからメソッド名へのマッピング」を参照してください。
2.3	メソッドのオーバーロード		○	メソッドのオーバーロードについては，「15.1.11(1)　Java メソッドのオーバーロード」を参照してください。
2.3	MEP:request-response オペレーションのサポート		○	
2.3	MEP:one-way オペレーションのサポート		○	
2.3.1.1	Non-wrapper スタイルのオペレーションのサポート		○	
2.3.1.2	Wrapper スタイルのオペレーションのサポート		○	
2.3.2	パラメタの順序と戻り値の型	parameterOrder 属性	×	

分類			サポート	備考
大分類※1	小分類※2			
		parameterOrder 属性以外	○	
2.3.3	javax.xml.ws.Holder<T>クラス		○	javax.xml.ws.Holder<T>クラスのサポート範囲については，「19.2.4(12) javax.xml.ws.Holder<T>クラス」を参照してください。
2.3.4	非同期マッピング		×	
2.4.1	W3CEndpointReference クラス		○	
2.5	フォルトからサービス固有例外へのマッピング		○	マッピングについては，「15.1.7 フォルトから例外クラスへのマッピング」を参照してください。
2.6	SOAP バインディング	SOAP 1.1/HTTP	○	
		SOAP 1.2/HTTP	○	
2.6	MIME バインディング		×	
2.6.2.1	soap:header 要素		○	soap:header 要素のサポート範囲（記述できる構文）については，「20.1.22　soap:header 要素」を参照してください。
2.7	サービスとポートからサービスクラスへのマッピング		○	マッピングについては，「15.1.9 サービスおよびポートからサービスクラスへのマッピング」を参照してください。
2.8	XML の名前から Java の識別子へのマッピング		○	
2.8.1	名前が衝突した場合の処理		○	名前衝突時の処理については，「15.1.11(2)　名前衝突時のマッピング」を参照してください。
3	Java から WSDL 1.1 へのマッピング		○	
3	アノテーションによるマッピングのカスタマイズ		○	アノテーションのサポート範囲については，「16.2.1　アノテーション一覧」を参照してください。
3.1	Java の識別子から XML の名前へのマッピング		○	
3.2	パッケージから名前空間へのマッピング		○	
3.3	暗黙の SEI からポートタイプへのマッピング		○	
3.4	SEI からポートタイプへのマッピング		○	マッピングについては，「16.1.3 SEI 名からポートタイプへのマッピング」を参照してください。

分類			サポート	備考
大分類※1	小分類※2			
3.5	SEI のメソッドからオペレーションへのマッピング		○	マッピングについては，「16.1.4 SEI のメソッド名からオペレーションへのマッピング」を参照してください。
3.5.1	one-way のオペレーション		○	
3.6	パラメタと戻り値のマッピング		○	マッピングについては，「16.1.5 パラメタおよび戻り値からメッセージのパートへのマッピング（wrapper スタイルの場合）」および「16.1.6 パラメタおよび戻り値からメッセージのパートへのマッピング（non-wrapper スタイルの場合）」を参照してください。
3.6	soap:header 要素		○	
3.6.2.1	Document Wrapped スタイルのマッピング		○	
3.6.2.2	Document Bare スタイルのマッピング		○	
3.6.2.3	RPC スタイルのマッピング		×	
3.7	サービス固有例外からフォルトへのマッピング		○	マッピングについては，「16.1.7 Java のラッパ例外クラスからフォルトへのマッピング」を参照してください。
3.8	サービスクラスからバインディングへのマッピング		○	マッピングについては，「16.1.8 SEI からバインディングへのマッピング」を参照してください。
3.9	ジェネリクスの処理		○	ジェネリクスの処理については，「16.1.10(1) ジェネリクスの型削除」および「16.1.10(3) ジェネリクス型の制限」を参照してください。
3.10	SOAP/HTTP バインディング	SOAP 1.1/HTTP	○	
		SOAP 1.2/HTTP	○	
3.11	サービスクラスからサービスとポートへのマッピング		○	マッピングについては，「16.1.9 Web サービス実装クラスからサービスおよびポートへのマッピング」を参照してください。
4	クライアント API		○	API のサポート範囲については，「19.2 API のサポート範囲」を参照してください。

分類			サポート	備考
大分類※1	小分類※2			
5	サービス API		○	API のサポート範囲については，「19.2　API のサポート範囲」を参照してください。
6	コア API		○	API のサポート範囲については，「19.2　API のサポート範囲」を参照してください。
7	アノテーション		○	アノテーションのサポート範囲については，「16.2.1　アノテーション一覧」を参照してください。
8	バインディング宣言		○	バンディング宣言のサポート範囲については，「15.2　WSDL から Java へのマッピングのカスタマイズ」を参照してください。
9.1.1	論理ハンドラ		○	論理ハンドラについては，「36.4 ハンドラの型」を参照してください。
9.1.1	プロトコルハンドラ		○	プロトコルハンドラについては，「36.4　ハンドラの型」を参照してください。
9.2.1.1	Web サービスクライアント側でのハンドラの設定	API による動的な設定	○	ハンドラの設定については，「36.9 ハンドラチェインの設定」を参照してください。
		上記以外	×	
9.2.1.2	ハンドラの順番づけ		○	ハンドラの実行順序については，「36.5　ハンドラチェインの編成と実行順序」を参照してください。
9.2.1.3	Web サービス側でのハンドラの設定	javax.jws.HandlerChain アノテーションによる設定	○	ハンドラの設定については，「36.9 ハンドラチェインの設定」を参照してください。
		上記以外	×	
9.2.2	WSEE 仕様（JSR-109）に基づくハンドラのデプロイ		×	
9.3	ハンドラの処理モデル		○	ハンドラの処理については，「36.5 ハンドラチェインの編成と実行順序」を参照してください。
9.4	メッセージコンテキスト		○	メッセージコンテキストのプロパティのサポート範囲については，「19.2.5(1)　メッセージコンテキストのプロパティのサポート範囲」を参照してください。
10.1.1	SOAP バインディングの動的な（プログラム的な）設定		○	

分類		サポート	備考
大分類※1	小分類※2		
10.1.2	SOAP バインディングの静的な（デプロイメントによる）設定	×	
10.2	SOAP バインディングの処理モデル	○	
10.3	SOAP メッセージコンテキスト	○	メッセージコンテキストのサポート範囲については，「19.2.5　メッセージコンテキストの使用」を参照してください。
10.4.1	SOAP 1.1/HTTP バインディング	○	
10.4.1	SOAP 1.2/HTTP バインディング	○	
10.4.1.1	MTOM 仕様	○	
10.4.1.2	one-way のオペレーション	○	
10.4.1.3	HTTP ベーシック認証	○	
10.4.1.4	セッション管理	○	
10.4.1.5	WS-Addressing 仕様	○	
11	XML/HTTP バインディング	×	

（凡例）

○：Application Server の JAX-WS 機能でサポートしています。

×：Application Server の JAX-WS 機能でサポートしていません。

空欄：特に補足する内容がないことを示します。

注※1

JAX-WS 2.2 仕様の該当個所（章節項）を示します。

注※2

JAX-WS 2.2 仕様の該当個所に記載されている内容を示します。

19.1.2　Conformance への対応

Conformance へ対応しているかどうかを次の表に示します。なお，Conformance の番号は，JAX-WS 2.2 仕様の"Appendix A Conformance Requirements"に準じます。

表 19–2　Conformance への対応

分類		対応	備考
番号※1	タイトル※2		
2.1	WSDL 1.1 support	○	

分類			対応	備考
番号※1	タイトル※2			
2.2	Customization required		○	バンディング宣言のサポート範囲については，「15.2　WSDL から Java へのマッピングのカスタマイズ」を参照してください。
2.3	Annotations on generated classes		○	アノテーションのサポート範囲については，「16.2.1　アノテーション一覧」を参照してください。
2.4	Definitions mapping		○	wsdl:definitions 要素の targetNamespace 属性に指定できる値については，「15.1.1(2)　名前空間の条件」を参照してください。
2.5	WSDL and XML Schema import directives		○	wsdl:import 要素については，「26.3 wsdl:import 要素の書式」を参照してください。
2.6	Optional WSDL extensions		○	Conformance 自体には対応していますが，JAX-WS 2.2 仕様で規定されていない WSDL の拡張要素や属性はサポートしていません。
2.7	SEI naming		○	wsdl:portType 要素の name 属性に指定できる値については，「15.1.2(2)　ポートタイプ名の条件」を参照してください。
2.8	javax.jws.WebService required		○	
2.9	javax.xml.bind.XmlSeeAlso required		○	
2.10	Method naming		○	wsdl:operation 要素の name 属性に指定できる値については，「15.1.3(2)　オペレーション名の条件」を参照してください。
2.11	javax.jws.WebMethod required		○	
2.12	Transmission primitive support	one-way	○	
		request-response	○	
2.13	Using javax.jws.OneWay		○	
2.14	Using javax.jws.SOAPBinding		○	
2.15	Using javax.jws.WebParam		○	wsdl:part 要素の name 属性に指定できる値については，「15.1.5(2)　パート名の条件」を参照してください。
2.16	Using javax.jws.WebResult		○	
2.17	Generating @Action		○	
2.18	Generating @Action input		○	
2.19	Generating @Action output		○	

分類		対応	備考
番号※1	タイトル※2		
2.20	Generating @Action fault	○	
2.21	use of JAXB annotations	○	アノテーションのサポート範囲については，「16.2.1　アノテーション一覧」を参照してください。
2.22	Non-wrapped parameter naming	○	wsdl:part 要素の name 属性に指定できる値については，「15.1.5(2)　パート名の条件」を参照してください。
2.23	Default mapping mode	○	
2.24	Disabling wrapper style	○	
2.25	Wrapped parameter naming	○	wrapper 子要素の name 属性に指定できる値については，「15.1.4(2)　wrapper 子要素名の条件」を参照してください。
2.26	Parameter name clash	○	
2.27	Using javax.xml.ws.RequestWrapper	○	
2.28	Using javax.xml.ws.ResponseWrapper	○	
2.29	Use of Holder	○	
2.30	Asynchronous mapping required	×	
2.31	Asynchronous mapping option	×	
2.32	Asynchronous method naming	−	非同期関連の機能はサポートしていません。
2.33	Asynchronous parameter naming	−	非同期関連の機能はサポートしていません。
2.34	Failed method invocation	−	非同期関連の機能はサポートしていません。
2.35	Response bean naming	−	非同期関連の機能はサポートしていません。
2.36	Asynchronous fault reporting	−	非同期関連の機能はサポートしていません。
2.37	Asynchronous fault cause	−	非同期関連の機能はサポートしていません。
2.38	JAXB class mapping	○	
2.39	JAXB customization use	○	
2.40	JAXB customization clash	○	
2.41	javax.xml.ws.wsaddressing.W3CEndpointReference	○	
2.42	javax.xml.ws.WebFault required	○	
2.43	Exception naming	○	wsdl:fault 要素および wsdl:message 要素の name 属性に指定できる値については，「15.1.7(2)　フォルト名の条件」を参照してください。

分類			対応	備考
番号※1	タイトル※2			
2.44	Fault equivalence		○	
2.45	Fault equivalence		○	
2.46	Required WSDL extensions	SOAP	○	
		MIME	×	
2.47	Unbound message parts		○	
2.48	Duplicate headers in binding		○	
2.49	Duplicate headers in message		○	
2.50	Use of MIME type information		−	MIME バインディングはサポートしていません。
2.51	MIME type mismatch		−	MIME バインディングはサポートしていません。
2.52	MIME part identification		−	MIME バインディングはサポートしていません。
2.53	Service superclass required		○	
2.54	Service class naming		○	wsdl:service 要素の name 属性に指定できる値については，「15.1.9(2)　サービス名およびポート名の条件」を参照してください。
2.55	javax.xml.ws.WebServiceClient required		○	
2.56	Generated service default constructor		○	
2.57	Generated service(WebServiceFeature ...) constructor		○	
2.58	Generated service(URL) constructor		○	
2.59	Generated service(URL,WebServiceFeature...) constructor		○	
2.60	Generated service(URL,QName) constructor		○	
2.61	Generated service(URL,QName,WebServiceFeature...) constructor		○	
2.62	Failed getPort Method		○	
2.63	javax.xml.ws.WebEndpoint required		○	
3.1	WSDL 1.1 support		○	WSDL 1.1 仕様のサポート範囲については，「20.1　WSDL 1.1 仕様のサポート範囲」を参照してください。

分類		対応	備考
番号※1	タイトル※2		
3.2	Standard annotations	○	アノテーションのサポート範囲については，「16.2.1　アノテーション一覧」を参照してください。
3.3	Java identifier mapping	○	
3.4	Method name disambiguation	○	
3.5	Package name mapping	○	Java の識別子やアノテーションで指定できる値については，「16.　Java から WSDL へのマッピング」を参照してください。
3.6	WSDL and XML Schema import directives	○	
3.7	Class mapping	○	
3.8	portType naming	○	Java の識別子やアノテーションで指定できる値については，「16.　Java から WSDL へのマッピング」を参照してください。
3.9	Inheritance flattening	○	
3.10	Inherited interface mapping	○	Conformance 自体には対応していますが，Application Server の JAX-WS 機能ではこの Conformance で説明されているようなマッピングはされません。
3.11	Operation naming	○	Java の識別子やアノテーションで指定できる値については，「16.　Java から WSDL へのマッピング」を参照してください。
3.12	Generating wsam:Action	○	
3.13	One-way mapping	○	
3.14	One-way mapping errors	○	
3.15	use of JAXB annotations	○	
3.16	Overriding JAXB types empty namespace	○	
3.17	Parameter classification	○	
3.18	Parameter naming	○	Java の識別子やアノテーションで指定できる値については，「16.　Java から WSDL へのマッピング」を参照してください。
3.19	Result naming	○	Java の識別子やアノテーションで指定できる値については，「16.　Java から WSDL へのマッピング」を参照してください。
3.20	Header mapping of parameters and results	○	
3.21	Dynamically generating wrapper beans	○	

分類		対応	備考
番号※1	タイトル※2		
3.22	Default wrapper bean names	○	Java の識別子やアノテーションで指定できる値については，「16. Java から WSDL へのマッピング」を参照してください。
3.23	Default wrapper bean package	○	Java の識別子やアノテーションで指定できる値については，「16. Java から WSDL へのマッピング」を参照してください。
3.24	Wrapper element names	○	Java の識別子やアノテーションで指定できる値については，「16. Java から WSDL へのマッピング」を参照してください。
3.25	Wrapper bean name clash	○	
3.26	Default Wrapper wsdl:part names	○	
3.27	Customizing Wrapper wsdl:part names	○	
3.28	Wrapper property	○	
3.29	Null Values in rpc/literal	−	rpc/literal スタイルはサポートしていません。
3.30	Exception naming	○	Java の識別子やアノテーションで指定できる値については，「16. Java から WSDL へのマッピング」を参照してください。
3.31	wsdl:message naming	○	
3.32	wsdl:message naming using WebFault	○	
3.33	java.lang.RuntimeExceptions and java.rmi.RemoteExceptions	○	
3.34	Fault bean’s @XmlType	○	
3.35	Fault bean name clash	○	
3.36	Dynamically generating exception beans	○	
3.37	Binding selection	○	
3.38	SOAP binding support	○	
3.39	SOAP binding style required	○	
3.40	Service creation	○	
3.41	Port selection	○	Java の識別子やアノテーションで指定できる値については，「16. Java から WSDL へのマッピング」を参照してください。
3.42	Port binding	○	
3.43	Use of Addressing	○	

分類		対応	備考
番号※1	タイトル※2		
4.1	Service completeness	×	未サポートの API があります。
4.2	Service Creation Failure	○	
4.3	Service creation using features	○	
4.4	Use of Executor	−	非同期関連の機能はサポートしていません。
4.5	Default Executor	−	非同期関連の機能はサポートしていません。
4.6	javax.xml.ws.BindingProvider.getEndpointReference	○	
4.7	BindingProvider’s W3CEndpointReference	○	
4.8	Message context decoupling	○	
4.9	Required BindingProvider properties	○	
4.10	Optional BindingProvider properties	○	
4.11	Additional context properties	○	
4.12	Asynchronous response context	−	非同期関連の機能はサポートしていません。
4.13	Proxy support	○	
4.14	Implementing BindingProvider	○	
4.15	Service.getPort failure	○	
4.16	Proxy's Addressing use	○	
4.17	Remote Exceptions	○	
4.18	Exceptions During Handler Processing	○	
4.19	Other Exceptions	○	
4.20	Dispatch support	○	
4.21	Failed Dispatch.invoke	○	
4.22	Failed Dispatch.invokeAsync	−	非同期関連の機能はサポートしていません。
4.23	Failed Dispatch.invokeOneWay	○	
4.24	Reporting asynchronous errors	−	javax.xml.ws.Response インタフェースはサポートしていません。
4.25	Marshalling failure	○	
4.26	Use of the Catalog	○	
5.1	Provider support required	○	
5.2	Provider default constructor	○	
5.3	Provider implementation	○	

分類		対応	備考
番号※1	タイトル※2		
5.4	WebServiceProvider annotation	○	
5.5	Endpoint publish(String address, Object implementor) Method	−	javax.xml.ws.Endpoint クラスはサポートしていません。
5.6	Default Endpoint Binding	−	javax.xml.ws.Endpoint クラスはサポートしていません。
5.7	Other Bindings	−	javax.xml.ws.Endpoint クラスはサポートしていません。
5.8	Publishing over HTTP	−	javax.xml.ws.Endpoint クラスはサポートしていません。
5.9	WSDL Publishing	−	javax.xml.ws.Endpoint クラスはサポートしていません。
5.10	Checking publishEndpoint Permission	−	javax.xml.ws.Endpoint クラスはサポートしていません。
5.11	Required Metadata Types	−	javax.xml.ws.Endpoint クラスはサポートしていません。
5.12	Unknown Metadata	−	javax.xml.ws.Endpoint クラスはサポートしていません。
5.13	Use of Executor	−	javax.xml.ws.Endpoint クラスはサポートしていません。
5.14	Default Executor	−	javax.xml.ws.Endpoint クラスはサポートしていません。
5.15	Endpoint's W3CEndpointReference	○	
5.16	Building W3CEndpointReference	○	
6.1	Read-only handler chains	○	
6.2	Concrete javax.xml.ws.spi.Provider required	○	
6.3	Provider createAndPublishEndpoint Method	−	javax.xml.ws.Provider インタフェースはサポートしていません。
6.4	Concrete javax.xml.ws.spi.ServiceDelegate required	○	
6.5	Protocol specific fault generation	○	
6.6	Protocol specific fault consumption	○	
6.7	One-way operations	○	
6.8	javax.xml.ws.WebServiceFeatures	−	javax.xml.ws.WebServiceFeature クラスはサポートしていません。
6.9	enabled property	−	javax.xml.ws.WebServiceFeature クラスはサポートしていません。

分類		対応	備考
番号※1	タイトル※2		
6.10	javax.xml.ws.soap.MTOMFeature	○	
6.11	javax.xml.ws.RespectBindingFeature	−	javax.xml.ws.RespectBindingFeature クラスはサポートしていません。
6.12	HTTP SPI in SE platform	×	JAX-WS では，Java SE 単体での開発および運用をサポートしていません。
7.1	Correctness of annotations	○	アノテーションのサポート範囲については，「16.2.1　アノテーション一覧」を参照してください。
7.2	Handling incorrect annotations	○	アノテーションのサポート範囲については，「16.2.1　アノテーション一覧」を参照してください。
7.3	Unsupported WebServiceFeatureAnnotation	○	
7.4	WebServiceProvider and WebService	○	
7.5	JSR-181 conformance	○	アノテーションのサポート範囲については，「16.2.1　アノテーション一覧」を参照してください。
8.1	Standard binding declarations	○	
8.2	Binding language extensibility	○	
8.3	Multiple binding files	○	
9.1	Handler framework support	○	
9.2	Logical handler support	○	
9.3	Other handler support	○	API のサポート範囲については，「19.2 API のサポート範囲」を参照してください。
9.4	Incompatible handlers	○	
9.5	Incompatible handlers	○	
9.6	Handler chain snapshot	○	
9.7	HandlerChain annotation	○	
9.8	Handler resolver for a HandlerChain annotation	○	
9.9	Binding handler manipulation	○	
9.10	Handler initialization	○	
9.11	Handler destruction	○	
9.12	Invoking close	○	
9.13	Order of close invocations	○	

分類		対応	備考
番号※1	タイトル※2		
9.14	Message context property scope	○	
10.1	SOAP required roles	○	
10.2	SOAP required roles	○	
10.3	Default role visibility	○	
10.4	Default role persistence	○	
10.5	None role error	○	
10.6	Incompatible handlers	○	
10.7	Incompatible handlers	○	
10.8	Logical handler access	○	
10.9	SOAP 1.1 HTTP Binding Support	○	
10.10	SOAP 1.2 HTTP Binding Support	○	
10.11	SOAP MTOM Support	○	
10.12	Semantics of MTOM enabled	○	
10.13	MTOM support	○	
10.14	SOAP bindings with MTOM disabled	○	
10.15	SOAP bindings with MTOM enabled	○	
10.16	MTOM on Other SOAP Bindings	−	ほかの javax.xml.ws.soap.SOAPBinding インタフェースの実装をサポートしていないので，この Conformance には該当しません。
10.17	One-way operations	○	
10.18	HTTP basic authentication support	○	
10.19	Authentication properties	○	
10.21	URL rewriting support	×	
10.22	Cookie support	○	
10.22	SSL session support	○	Conformance 自体には対応していますが，Application Server の JAX-WS 機能では SSL セッションをベースとする状態管理をサポートしていません。
10.23	SOAP Addressing Support	○	
11.1	XML/HTTP Binding Support	×	
11.2	Incompatible handlers	×	

分類		対応	備考
番号※1	タイトル※2		
11.3	Incompatible handlers	×	
11.4	Logical handler access	×	
11.5	One-way operations	×	
11.6	HTTP basic authentication support	×	
11.7	Authentication properties	×	
11.8	URL rewriting support	×	
11.9	Cookie support	×	
11.10	SSL session support	×	

(凡例)

○：対応しています。

×：対応していません。

－：該当しません。

空欄：特に補足する内容がないことを示します。

注※1

JAX-WS 2.2 仕様の"Appendix A Conformance Requirements"の番号を示します。

注※2

Conformance に記載されたタイトルです。

19.2 APIのサポート範囲

19.2.1 インタフェースおよびクラスの一覧（JAX-WS）

ここでは，JAX-WS APIのインタフェースおよびクラスの種類について説明します。また，インタフェースおよびクラスのサポート範囲についても示します。

(1) インタフェースおよびクラスの種類

JAX-WS APIのインタフェースおよびクラスは，次に示すAPI群に分類されます。

- クライアントAPI

  ディスパッチベースまたはAPIベースのWebサービスクライアントで使用するAPIです。スタブベースのWebサービスクライアントでは，コマンドで生成されたサービスクラスやスタブを使用してWebサービスにアクセスするため，クライアントAPIは使用しません。

- サービスAPI

  Webサービスに高度な実装を記述する場合に使用するAPIです。プロバイダ実装クラスを使用したWebサービスやハンドラなど，複雑な機能を利用する場合に使用します。

- コアAPI

  WebサービスとWebサービスクライアントのどちらでも使用できるAPIです。inoutおよびoutパラメタを保持するためのHolderクラス，または例外などが含まれます。

(2) インタフェースおよびクラスの一覧表

JAX-WS APIのインタフェースおよびクラスの一覧を次の表に示します。Application ServerのJAX-WS機能でサポートしていないインタフェースおよびクラスを使用してWebサービスを開発した場合の動作は保証されません。

表19–3 JAX-WS APIのインタフェースおよびクラス一覧

項番	インタフェースまたはクラス名	説明	サポート
javax.xml.wsパッケージ			
1	*AsyncHandler<T>*	−	×
2	*Binding*	−	×
3	*BindingProvider*	プロトコルバインディングと関連づけられたコンテキストオブジェクトへのアクセスを提供するインタフェースです。	○
4	*Dispatch<T>*	XMLメッセージを送信するためのインタフェースです。	○

項番	インタフェースまたはクラス名	説明	サポート
5	*LogicalMessage*	プロトコルに捕らわれない XML メッセージを表現し，メッセージのペイロードへアクセスする方法を提供するメソッドを含むインタフェースです。	○
6	*Provider<T>*	XML メッセージを受信するためのインタフェースです。	○
7	*Response<T>*	−	×
8	*WebServiceContext*	Web サービス実装クラスまたはプロバイダ実装クラスに対して現在処理中のリクエストに関連する情報へのアクセスを提供するインタフェースです。 javax.annotation.Resource アノテーションを指定し，現在処理中のリクエストに関連する情報をインジェクトして使用します。	○
9	Endpoint	−	×
10	EndpointReference	Web サービスエンドポイントのリモート参照の WS-Addressing EndpointReference を表す抽象クラスです。	○
11	Holder<T>	型 T の値を保持するクラスです。	○
12	RespectBindingFeature	−	×
13	Service	Web サービスクライアントが使用するための Web サービスを表すクラスです。	○
14	WebServiceFeature	ユーザが直接使用することはありません。	○
15	WebServicePermission	−	×
16	ProtocolException	プロトコルレベルでのフォルト情報を，クライアントに通知するためのクラスです。	○
17	WebServiceException	JAX-WS API の実行時例外を表す例外クラスです。	○
javax.xml.ws.handler パッケージ			
18	*Handler<C extends MessageContext>*	ハンドラの基底インタフェースです。	○
19	*HandlerResolver*	プロキシに設定されたハンドラチェインを制御するために，Web サービスクライアントの実装者が実装するインタフェースです。	○
20	*LogicalHandler<C extends LogicalMessageContext>*	論理ハンドラです。論理ハンドラを実装する場合はこのインタフェースを実装してください。 このインタフェースには，メソッドはありません。	○
21	*LogicalMessageContext*	論理ハンドラ用のメッセージコンテキストです。	○
22	*MessageContext*	プロパティセットを管理するメソッドを提供するインタフェースです。	○

項番	インタフェースまたはクラス名	説明	サポート
23	*PortInfo*	ハンドラリゾルバがハンドラチェインの生成を求められたときに，どのポートのために求められているのかクエリするために使用する情報です。	○
javax.xml.handler.soap パッケージ			
24	*SOAPHandler<T extends SOAPMessageContext>*	SOAP ハンドラです。SOAP ハンドラを実装する場合はこのインタフェースを実装してください。	○
25	*SOAPMessageContext*	SOAP ハンドラ用のメッセージコンテキストです。	○
javax.xml.ws.http パッケージ			
26	*HTTPBinding*	−	×
27	HTTPException	−	×
javax.xml.ws.soap パッケージ			
28	*SOAPBinding*	SOAP バインディング用の抽象クラスです。	○
29	AddressingFeature	WS-Addressing を使用することを表すフィーチャクラスです。	○
30	MTOMFeature	MTOM/XOP 仕様形式の添付ファイルを使用することを表すフィーチャクラスです。	○
31	SOAPFaultException	SOAP フォルトの例外を表すクラスです。	○
javax.xml.ws.spi パッケージ			
32	Provider	ServiceDelegate および Endpoint オブジェクトを作成する抽象クラスです。 ユーザが直接使用することはありません。	○
33	ServiceDelegate	Service オブジェクトによって内部的に使用される抽象クラスです。 ユーザが直接使用することはありません。	○
javax.xml.ws.wsaddressing パッケージ			
34	W3CEndpointReference	EndpointReference 抽象クラスの実装クラスです。	○
35	W3CEndpointReferenceBuilder	W3CEndpointReference クラスを作成するために使用されるビルダクラスです。	○
com.sun.xml.ws.developer パッケージ			
36	StreamingAttachmentFeature	ストリーミングを使用することを表すフィーチャクラスです。	○
37	StreamingDataHandler	ストリーミングを使用した添付ファイルを表す抽象クラスです。	○
org.jvnet.mimepull パッケージ			

項番	インタフェースまたはクラス名	説明	サポート
38	MIMEConfig	MIME メッセージの構文解析と出力に関する設定をするためのクラスです。	○

（凡例）

－：説明がないことを示します（非サポートのため）。

○：Application Server の JAX-WS 機能でサポートしています。

×：Application Server の JAX-WS 機能でサポートしていません。

19.2.2　クライアント API

ここでは，クライアント API のサポート範囲について説明します。

(1)　javax.xml.ws.BindingProvider インタフェース

javax.xml.ws.BindingProvider インタフェースのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19–4　javax.xml.ws.BindingProvider インタフェースのサポート範囲

項番	戻り値の型	メソッド名／説明		サポート
1	Binding	getBinding()		○
		説明	このバインディングプロバイダのバインディングを取得します。	
2	EndpointReference	getEndpointReference()		×
3	<T extends EndpointReference> T	getEndpointReference (java.lang.Class<T> clazz)		×
4	java.util.Map <java.lang.String, java.lang.Object>	getRequestContext()		○
		説明	要求メッセージのメッセージコンテキストを初期化するために使用されるコンテキストを取得します。	
5	java.util.Map <java.lang.String, java.lang.Object>	getResponseContext()		○
		説明	応答メッセージの処理によって得られたコンテキストを取得します。応答メッセージが処理される前に呼び出した場合は null を返します。one-way オペレーションの場合は，レスポンスメッセージが存在しないため，このメソッドを呼び出さないでください。one-way オペレーションで使用したときの動作は保証されません。	

（凡例）

○：Application Server の JAX-WS 機能でサポートしています。

×：Application Server の JAX-WS 機能でサポートしていません。

(2) javax.xml.ws.Dispatch インタフェース

javax.xml.ws.Dispatch インタフェースのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19-5 javax.xml.ws.Dispatch インタフェースのサポート範囲

項番	戻り値の型	メソッド名	サポート
1	T	invoke(T msg)	○
2	Response<T>	invokeAsync(T msg)	×
3	java.util.concurrent.Future<?>	invokeAsync(T msg, AsyncHandler<T> handler)	×
4	void	invokeOneWay(T msg)	○

(凡例)

○：Application Server の JAX-WS 機能でサポートしています。

×：Application Server の JAX-WS 機能でサポートしていません。

(3) javax.xml.ws.EndpointReference クラス

javax.xml.ws.EndpointReference クラスのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19-6 javax.xml.ws.EndpointReference クラスのサポート範囲

項番	戻り値の型	メソッド名	サポート
1	−	EndpointReference()	○
2	<T> T	getPort (java.lang.Class<T> serviceEndpointInterface, WebServiceFeature... features)	×
3	static EndpointReference	readFrom (javax.xml.transform.Source eprInfoset)	○
4	java.lang.String	toString()	×
5	abstract void	writeTo(javax.xml.transform.Result result)	×

(凡例)

−：戻り値の型がないことを示します。

○：Application Server の JAX-WS 機能でサポートしています。

×：Application Server の JAX-WS 機能でサポートしていません。

(4) javax.xml.ws.Service クラス

javax.xml.ws.Service クラスのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19–7　javax.xml.ws.Service クラスのサポート範囲

項番	戻り値の型	メソッド名／説明		サポート
1	−	Service(java.net.URL wsdlDocumentLocation, javax.xml.namespace.QName serviceName)		○
		説明	Service インスタンスを生成するコンストラクタです。	
		引数	wsdlDocumentLocation： WSDL 文書の位置です。null を指定した場合，Service コンストラクタを呼び出しているクラスに設定されている javax.xml.ws.WebServiceClient アノテーションの wsdlLocation 属性の値が設定されます。指定する URL の形式については，java.net.URL クラスの仕様に従ってください。なお，カタログ機能が有効な場合，この引数に指定した値を，カタログファイルで指定した別の WSDL ロケーションを示す URI にマッピングします。 serviceName： サービスの名前です。	
		例外	javax.xml.ws.WebServiceException： 次の場合に発生します。 • wsdlDocumentLocation に，存在しないローカルパス名を指定した場合 • wsdlDocumentLocation に，「?wsdl」付きの存在しない HTTP の URL を指定した場合 • wsdlDocumentLocation に，「?wsdl」付きではない存在しない HTTP の URL を指定した場合 • serviceName に null を指定した場合 • serviceName に，WSDL のサービス名（wsdl:service 要素の name 属性の値）以外の QName を指定した場合 • wsdlDocumentLocation に null を指定し，かつ Service コンストラクタを呼び出しているクラスで javax.xml.ws.WebServiceClient アノテーションを使用していない場合 • wsdlDocumentLocation に null を指定し，かつ Service コンストラクタを呼び出しているクラスの javax.jws.WebService アノテーションで，wsdlLocation 属性が設定されていない場合	
2	void	addPort(javax.xml.namespace.QName portName, java.lang.String bindingId, java.lang.String endpointAddress)		○
		例外	javax.xml.ws.WebServiceException： 生成済みのポート名を指定した場合に発生します。	
		注意	portName は，createDispatch()を呼び出すときに同じ QName を指定する必要があります。null は指定できません。 bindingId に null を指定した場合は，SOAP1.1/HTTP のバインディング ID が設定されます。	

項番	戻り値の型	メソッド名／説明		サポート
3	static Service	create(javax.xml.namespace.QName serviceName)		○
4	static Service	create(javax.xml.namespace.QName serviceName, javax.xml.ws.WebServiceFeature ... features)		×
5	static Service	create(java.net.URL wsdlDocumentLocation, javax.xml.namespace.QName serviceName)		○
		説明	Service インスタンスを生成します。	
		引数	wsdlDocumentLocation： WSDL 文書の位置です。指定する URL の形式については java.net.URL クラスの仕様に従ってください。 serviceName： WSDL のサービス名（wsdl:service 要素の name 属性値）です。	
		例外	javax.xml.ws.WebServiceException： 次の場合に発生します。 • wsdlDocumentLocation に存在しないローカルパス名を指定した場合 • wsdlDocumentLocation に，「?wsdl」付きの存在しない HTTP の URL を指定した場合 • wsdlDocumentLocation に，「?wsdl」付きではない存在しない HTTP の URL を指定した場合 • serviceName に，WSDL のサービス名（wsdl:service 要素の name 属性の値）以外を指定した場合（ただし，wsdlDocumentLocation に null を指定した場合を除く）	
6	static Service	create(java.net.URL wsdlDocumentLocation, javax.xml.namespace.QName serviceName, javax.xml.ws.WebServiceFeature ... features)		×
7	<T> Dispatch<T>	createDispatch (EndpointReference endpointReference, java.lang.Class<T> type, Service.Mode mode, WebServiceFeature... features)		×
8	Dispatch <java.lang.Object>	createDispatch (EndpointReference endpointReference, javax.xml.bind.JAXBContext context, Service.Mode mode, WebServiceFeature... features)		×
9	<T> Dispatch<T>	createDispatch (javax.xml.namespace.QName portName, java.lang.Class<T> type, Service.Mode mode)		○
		注意	portName は，addPort()の引数に指定するポート名と同じものを指定する必要があります。null は指定できません。	
10	<T> Dispatch<T>	createDispatch (javax.xml.namespace.QName portName, java.lang.Class<T> type, Service.Mode mode, WebServiceFeature... features)		×
11	Dispatch <java.lang.Object>	createDispatch (javax.xml.namespace.QName portName, javax.xml.bind.JAXBContext context, Service.Mode mode)		○

項番	戻り値の型	メソッド名／説明		サポート
		注意	portName は，addPort()の引数に指定するポート名と同じものを指定する必要があります。null は指定できません。	
12	Dispatch <java.lang.Object>	createDispatch (javax.xml.namespace.QName portName, javax.xml.bind.JAXBContext context, Service.Mode mode, WebServiceFeature... features)		×
13	java.util.concurrent. Executor	getExecutor()		×
14	HandlerResolver	getHandlerResolver()		○
		説明	この Service インスタンスによって使用されている HandlerResolver インスタンスを返します。存在しない場合は null を返します。	
15	<T> T	getPort (java.lang.Class<T> serviceEndpointInterface)		○
		説明	ポート（サービスにアクセスするためのプロキシ）を返します。	
		引数	serviceEndpointInterface： SEI の Class クラスです。	
		例外	javax.xml.ws.WebServiceException： 次の場合に発生します。 • serviceEndpointInterface が null の場合 • wsdlDocumentLocation に null を指定した Service.create()で生成した Service インスタンスから呼び出した場合 • javax.jws.WebService アノテーションを使用していない SEI を引数に指定した場合 • このメソッドを呼び出す前に，null を返す getHandlerChain()を実装した HandlerResolver オブジェクトを setHandlerResolver()で設定している場合 • このメソッドを呼び出す前に，論理ハンドラでも SOAP ハンドラでもない，ハンドラを含むハンドラチェインを返す getHandlerChain()を実装した HandlerResolver オブジェクトを setHandlerResolver()で設定している場合	
16	<T> T	getPort (java.lang.Class<T> serviceEndpointInterface, WebServiceFeature... features)		×
17	<T> T	getPort(EndpointReference endpointReference, java.lang.Class<T> serviceEndpointInterface, WebServiceFeature... features)		○※
		説明	ポート（サービスにアクセスするためのプロキシ）を返します。	
		引数	endpointReference： ポートによって呼び出されるサービスエンドポイントです。 serviceEndpointInterface： SEI の Class クラスです。	

項番	戻り値の型	メソッド名／説明		サポート
			features： ポート上で構成する WebServiceFeature のリストです。	
		例外	javax.xml.ws.WebServiceException： 次の場合に発生します。 • ポートの生成中に例外が発生した場合 • このメソッドの処理に必要な WSDL が存在しない場合 • endpointReference メタデータが Service インスタンスの serviceName に合っていない場合 • WSDL または endpointReference メタデータから portName を抽出できない場合 • 無効な endpointReference が指定された場合 • 無効な serviceEndpointInterface が指定された場合 • ポートと互換性がない，またはサポートされていない WebServiceFeature が指定された場合	
18	<T> T	getPort(javax.xml.namespace.QName portName, java.lang.Class<T> serviceEndpointInterface)		○
		説明	ポート（サービスにアクセスするためのプロキシ）を返します。	
		引数	portName： WSDL のポート名（wsdl:port 要素の name 属性の値）です。 serviceEndpointInterface： SEI の Class クラスです。	
		例外	javax.xml.ws.WebServiceException： 次の場合に発生します。 • portName に WSDL のポート名（wsdl:port 要素の name 属性の値）以外の QName を指定した場合 • portName が null の場合 • serviceEndpointInterface が null の場合	
19	<T> T	getPort(javax.xml.namespace.QName portName, java.lang.Class<T> serviceEndpointInterface, WebServiceFeature... features)		×
20	java.util.Iterator <javax.xml.namespace.QName>	getPorts()		○
		説明	サービスエンドポイント（WSDL に含まれるポート（wsdl:port 要素））を表す QName のリストの Iterator を返します。	
		例外	javax.xml.ws.WebServiceException： wsdlDocumentLocation に null を指定して生成した Service インスタンスに対して，Service.create()メソッドを呼び出した場合に発生します。	
21	javax.xml.namespace.QName	getServiceName()		○
		説明	このサービスの名前（WSDL のサービス名（wsdl:service 要素の name 属性の値））を返します。	

項番	戻り値の型	メソッド名／説明		サポート
22	java.net.URL	getWSDLDocumentLocation()		○
		説明	このサービスの WSDL ファイルの位置を返します。引数 wsdlDocumentLocation に null を指定して生成した Service インスタンスに対して，Service.create()メソッドを呼び出した場合には null を返します。	
23	void	setExecutor (java.util.concurrent.Executor executor)		×
24	void	setHandlerResolver (HandlerResolver handlerResolver)		○
		説明	この Service インスタンスの HandlerResolver インスタンスを設定してください。	

（凡例）

－：戻り値の型がないことを示します。

○：Application Server の JAX-WS 機能でサポートしています。

×：Application Server の JAX-WS 機能でサポートしていません。

注※

getPort メソッドで可変長引数を指定した場合の動作を次の表に示します。

表 19–8　可変長引数の指定方法と動作

項番	指定方法	指定例	動作
1	引数を省略	getPort(epr, sei)	可変長引数が指定されなかったと見なします。
2	引数を一つ指定	getPort(epr, sei, new FooFeatures())	可変長引数が一つ指定されたと見なします。
3	引数を二つ指定	getPort(epr, sei, new FooFeatures(), new BarFeatures())	可変長引数が二つ指定されたと見なします。
4	引数に null を指定	getPort(epr, sei, null)	可変長引数が指定されなかったと見なします。
5	引数に null を指定して WebServiceFeature 型にキャスト	getPort(epr, sei, (WebServiceFeature) null)	内容が null の可変長引数が指定されたと見なし，例外 NullPointerException が発生します。
6	引数に null を指定して WebServiceFeature 型の配列にキャスト	getPort(epr, sei, (WebServiceFeature[]) null)	可変長引数が指定されなかったと見なします。

(5)　javax.xml.ws.wsaddressing.W3CEndpointReference クラス

javax.xml.ws.wsaddressing.W3CEndpointReference クラスのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19-9　javax.xml.ws.wsaddressing.W3CEndpointReference クラスのサポート範囲

項番	戻り値の型	メソッド名	サポート
1	−	W3CEndpointReference()	○
2	−	W3CEndpointReference (javax.xml.transform.Source source)	○
3	void	writeTo(javax.xml.transform.Result result)	×

(凡例)

−：戻り値の型がないことを示します。

○：Application Server の JAX-WS 機能でサポートしています。

×：Application Server の JAX-WS 機能でサポートしていません。

19.2.3　サービス API

ここでは，サービス API のサポート範囲について説明します。

(1)　javax.xml.ws.Provider インタフェース

javax.xml.ws.Provider インタフェースのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19-10　javax.xml.ws.Provider インタフェースのサポート範囲

項番	戻り値の型	メソッド名	サポート
1	T	invoke(T request)	○

(凡例)

○：Application Server の JAX-WS 機能でサポートしています。

(2)　javax.xml.ws.WebServiceContext インタフェース

javax.xml.ws.WebServiceContext インタフェースのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19-11　javax.xml.ws.WebServiceContext インタフェースのサポート範囲

項番	戻り値の型	メソッド名／説明	サポート
1	<T extends EndpointReference> T	getEndpointReference (java.lang.Class<T> clazz, org.w3c.dom.Element... referenceParameters)	×
2	EndpointReference	getEndpointReference (org.w3c.dom.Element... referenceParameters)	×
3	MessageContext	getMessageContext()	○

項番	戻り値の型	メソッド名／説明		サポート
		説明	このメソッドが呼び出されたときに処理されているリクエストのメッセージコンテキストを取得します。	
		例外	java.lang.IllegalStateException： リクエストが処理されていないときにこのメソッドが呼び出された場合に発生します。	
4	java.security.Principal	getUserPrincipal()		×
5	boolean	isUserInRole(java.lang.String role)		×

（凡例）

○：Application Server の JAX-WS 機能でサポートしています。

×：Application Server の JAX-WS 機能でサポートしていません。

(3) javax.xml.ws.wsaddressing.W3CEndpointReferenceBuilder クラス

javax.xml.ws.wsaddressing.W3CEndpointReferenceBuilder クラスのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19-12 javax.xml.ws.wsaddressing.W3CEndpointReferenceBuilder クラスのサポート範囲

項番	戻り値の型	メソッド名	サポート
1	−	W3CEndpointReferenceBuilder()	○
2	W3CEndpointReferenceBuilder	address(java.lang.String address)	○
3	W3CEndpointReferenceBuilder	attribute(javax.xml.namespace.QName name, java.lang.String value)	○
4	W3CEndpointReference	build()	○※1
5	W3CEndpointReferenceBuilder	element(org.w3c.dom.Element element)	○
6	W3CEndpointReferenceBuilder	endpointName (javax.xml.namespace.QName endpointName)	○
7	W3CEndpointReferenceBuilder	interfaceName(javax.xml.namespace.QName interfaceName)	○
8	W3CEndpointReferenceBuilder	metadata (org.w3c.dom.Element metadataElement)	○
9	W3CEndpointReferenceBuilder	referenceParameter (org.w3c.dom.Element referenceParameter)	○
10	W3CEndpointReferenceBuilder	serviceName (javax.xml.namespace.QName serviceName)	○
11	W3CEndpointReferenceBuilder	wsdlDocumentLocation (java.lang.String wsdlDocumentLocation)	○※2

（凡例）

－：戻り値の型がないことを示します。

○：Application Server の JAX-WS 機能でサポートしています。

注※1

build メソッドは，スレッドセーフではありません。W3CEndpointReferenceBuilder のオブジェクトを複数スレッドで共有しないでください。共有した場合の動作は保証されません。

注※2

カタログ機能による引数 wsdlDocumentLocation のマッピングは，サポートしていません。

19.2.4　コア API

ここでは，コア API のサポート範囲について説明します。

(1)　com.sun.xml.ws.developer.StreamingAttachmentFeature クラス

com.sun.xml.ws.developer.StreamingAttachmentFeature クラスのサポート範囲を次の表に示します。

表 19–13　com.sun.xml.ws.developer.StreamingAttachmentFeature クラスのサポート範囲

項番	戻り値の型	メソッド名		サポート
1	－	StreamingAttachmentFeature()		×
2	－	StreamingAttachmentFeature(String dir,boolean parseEagerly,long memoryThreshold)		○
		説明	StreamingAttachmentFeature を作成します。	
		引数	dir： 一時ファイルの出力先ディレクトリパスです。存在しないディレクトリ，アクセス権のないディレクトリ，または存在するファイル名を指定した場合，メッセージを出力し，受信した添付ファイルを含む SOAP メッセージに含まれる MIME ボディをメモリ上に展開します（KDJW10026-W）。 parseEagerly： 受信した添付ファイルを含む SOAP メッセージを詳細に解析するかどうかを設定します。true の場合，添付ファイルを含む SOAP メッセージを詳細に解析します。 memoryThreshold： 添付ファイルを含む SOAP メッセージを受信する際に，SOAP メッセージに含まれる MIME ボディをメモリ上に展開するかどうかを判定するためのしきい値(単位：バイト)です。16KB（16384L）より大きい値または-1 を指定します。これら以外の値を指定した場合，動作は保証されません。-1 を指定した場合，常に MIME ボディをメモリ上に展開します。	
3	java.lang.String	getID()		○

項番	戻り値の型	メソッド名		サポート
		説明	StreamingAttachmentFeature の一意の識別子を取得します。	
4	org.jvnet.mimepull.MIMEConfig	getConfig()		○
		説明	ストリーミングの設定情報を表す MIMEConfig インスタンスを取得します。一度このメソッドを呼び出した場合，次の setter メソッドを用いてストリーミングの設定情報を変更することはできません。 • setDir(String dir) • setParseEagerly(boolean parseEagerly) • setMemoryThreshold(int memoryThreshold)	
5	void	setDir(String dir)		○
		説明	ストリーミングが有効の場合に，受信した添付ファイルを含む SOAP メッセージに含まれる MIME ボディを一時ファイルとして出力するときに使用するディレクトリを設定します。このメソッドを複数回呼び出した場合，最後に指定した値が有効になります。getConfig()メソッドを呼び出した場合，このメソッドによる再設定はできません。	
		引数	dir： 一時ファイルの出力先ディレクトリパスです。存在しないディレクトリ，アクセス権のないディレクトリ，または存在するファイル名を指定した場合，メッセージを出力し，受信した添付ファイルを含む SOAP メッセージに含まれる MIME ボディをメモリ上に展開します（KDJW10026-W）。	
6	void	setParseEagerly(boolean parseEagerly)		○
		説明	ストリーミングが有効の場合に，受信した添付ファイルを含む SOAP メッセージを詳細に解析するかどうかを設定します。このメソッドを複数回呼び出した場合，最後に指定した値が有効になります。getConfig()メソッドを呼び出した場合，このメソッドによる再設定はできません。	
		引数	parseEagerly： 受信した添付ファイルを含む SOAP メッセージを詳細に解析するかどうかを設定します。true の場合，添付ファイルを含む SOAP メッセージを詳細に解析します。	
7	void	setMemoryThreshold(int memoryThreshold)		○
		説明	ストリーミングが有効の場合に，添付ファイルを含む SOAP メッセージを受信するときに SOAP メッセージに含まれる MIME ボディをメモリ上に展開するかどうかを判定するためのしきい値を設定します。このメソッドを複数回呼び出した場合，最後に指定した値が有効になります。getConfig()メソッドを呼び出した場合，このメソッドによる再設定はできません。	
		引数	memoryThreshold： 添付ファイルを含む SOAP メッセージを受信する際に，SOAP メッセージに含まれる MIME ボディをメモリ上に展開するかどうかを判定するためのしきい値(単位：バイト)です。16KB（16384L）より大きい値または-1 を指定します。これら以外の値を指定した	

項番	戻り値の型	メソッド名		サポート
			場合，動作は保証されません。-1 を指定した場合，常に MIME ボディをメモリ上に展開します。	

（凡例）

－：戻り値の型がないことを示します。

○：Application Server の JAX-WS 機能でサポートしています。

×：Application Server の JAX-WS 機能でサポートしていません。

(2) com.sun.xml.ws.developer.StreamingDataHandler クラス

com.sun.xml.ws.developer.StreamingDataHandler クラスのサポート範囲を次の表に示します。

表 19-14 com.sun.xml.ws.developer.StreamingDataHandler クラスのサポート範囲

項番	戻り値の型	メソッド名		サポート
1	－	StreamingDataHandler(Object o, String s)		×
2	－	StreamingDataHandler(URL url)		×
3	－	StreamingDataHandler(DataSource dataSource)		×
4	java.io.InputStream	readOnce()		○
		説明	このオブジェクトの java.io.InputStream を取得します。	
		例外	IOException： このオブジェクトに対応する InputStream が取得できなかった場合に発生します。	
5	void	moveTo(File file)		○
		説明	このオブジェクトが示す添付ファイルを指定されたファイルに移動します。 • 引数 file に null を指定した場合，java.io.IOException が発生します（KDJW10023-E）。 • 引数 file に存在するファイルまたはディレクトリのパス，存在しない親ディレクトリを含むファイルパスを指定した場合，IOException が発生します（KDJW10027-E）。	
		引数	file： 出力先ファイルパスです。	
		例外	IOException： 次の場合に発生します。 • null を file に指定。 • 存在するファイルまたはディレクトリのパス，または存在しない親ディレクトリを含むファイルパスを file に指定。 • 入出力エラーが発生。	
6	void	close()		○

項番	戻り値の型	メソッド名		サポート
		説明	このオブジェクトが示す添付ファイルのリソースを解放します。	
		例外	IOException： 入出力エラーが発生した場合に発生します。	

（凡例）

－：戻り値の型がないことを示します。

○：Application Server の JAX-WS 機能でサポートしています。

×：Application Server の JAX-WS 機能でサポートしていません。

(3) org.jvnet.mimepull.MIMEConfig クラス

org.jvnet.mimepull.MIMEConfig クラスのサポート範囲を次の表に示します。

表 19-15 org.jvnet.mimepull.MIMEConfig クラスのサポート範囲

項番	戻り値の型	メソッド名		サポート
1	－	MIMEConfig()		×
2	void	setParseEagerly(boolean parseEagerly)		○
		説明	ストリーミングが有効の場合に，受信した添付ファイルを含む SOAP メッセージを詳細に解析するかどうかを設定します。このメソッドを複数回呼び出した場合，最後に指定した値が有効になります。	
		引数	parseEagerly： 受信した添付ファイルを含む SOAP メッセージを詳細に解析するかどうかを設定します。true の場合，添付ファイルを含む SOAP メッセージを詳細に解析します。	
3	void	setMemoryThreshold(long memoryThreshold)		○
		説明	ストリーミングが有効の場合に，添付ファイルを含む SOAP メッセージを受信するときに，SOAP メッセージに含まれる MIME ボディをメモリ上に展開するかどうかを判定するためのしきい値を設定します。このメソッドを複数回呼び出した場合，最後に指定した値が有効になります。	
		引数	memoryThreshold： 添付ファイルを含む SOAP メッセージを受信する際に，SOAP メッセージに含まれる MIME ボディをメモリ上に展開するかどうかを判定するためのしきい値(単位：バイト)です。16KB（16384L）より大きい値または-1 を指定します。これら以外の値を指定した場合，動作は保証されません。-1 を指定した場合，常に MIME ボディをメモリ上に展開します。	
4	void	setDir(String dir)		○
		説明	ストリーミングが有効の場合に，受信した添付ファイルを含む SOAP メッセージに含まれる MIME ボディを一時ファイルとして出力するときに使用するディレクトリを設定します。このメソッドを複数回呼び出した場合，null または空文字（""）以外の値で最初に指定したものが有効になります。	

項番	戻り値の型	メソッド名		サポート
		引数	dir： 一時ファイルの出力先ディレクトリパスです。	
5	void	validate()		○
		説明	一時ファイルを作成できるかを検証します。作成できない場合，添付ファイルをメモリ上に展開するように設定します。 存在しないディレクトリ，アクセス権のないディレクトリ，存在するファイル名を setDir(String dir)メソッドで設定し，このメソッドで検証しない場合，ストリーミングが一時ファイルを出力するときに org.jvnet.mimepull.MIMEParsingException が発生します。	

（凡例）

－：戻り値の型がないことを示します。

○：Application Server の JAX-WS 機能でサポートしています。

×：Application Server の JAX-WS 機能でサポートしていません。

(4) javax.xml.ws.Binding インタフェース

javax.xml.ws.Binding インタフェースのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19–16 javax.xml.ws.Binding インタフェースのサポート範囲

項番	戻り値の型	メソッド名／説明		サポート
1	java.lang.String	getBindingID()		×
2	java.util.List<Handler>	getHandlerChain()		○
		説明	プロトコルバインディングインスタンスのハンドラチェインのコピーを取得します。	
3	void	setHandlerChain (java.util.List<Handler> chain)		○
		説明	プロトコルバインディングインスタンスのハンドラチェインを設定します。	
		引数	chain： ハンドラチェインを構成するハンドラのリストです。	
		例外	javax.xml.ws.WebServiceException： ハンドラチェインの設定でエラーが起きた場合，または chain に null を指定した場合に発生します。	

（凡例）

○：Application Server の JAX-WS 機能でサポートしています。

×：Application Server の JAX-WS 機能でサポートしていません。

(5) javax.xml.ws.handler.Handler<C extends MessageContext>インタフェース

javax.xml.ws.handler.Handler<C extends MessageContext>インタフェースのサポート範囲を次の表に示します。詳細はJAX-WS 2.2仕様を参照してください。

表19-17 javax.xml.ws.handler.Handler<C extends MessageContext>インタフェースのサポート範囲

項番	戻り値の型	メソッド名／説明		サポート
1	MessageContext.Scope	close(MessageContext context)		○
		説明	メッセージ交換を完了するとき，JAX-WSのランタイムがメッセージ，フォルト，または例外をディスパッチする直前に呼び出されます。	
2	boolean	handleFault(C context)		○
		説明	フォルトメッセージを処理するために呼び出されます。	
3	boolean	handleMessage(C context)		○
		説明	インバウンドおよびアウトバウンドのメッセージで通常の処理をするために呼び出されます。	

（凡例）

○：Application ServerのJAX-WS機能でサポートしています。

(6) javax.xml.ws.handler.HandlerResolverインタフェース

javax.xml.ws.handler.HandlerResolverインタフェースのサポート範囲を次の表に示します。詳細はJAX-WS 2.2仕様を参照してください。

表19-18 javax.xml.ws.handler.HandlerResolverインタフェースのサポート範囲

項番	戻り値の型	メソッド名／説明		サポート
1	java.util.List<Handler>	getHandlerChain(PortInfo portInfo)		○
		説明	指定されたポートのハンドラチェインを取得します。このメソッドではnullを返さないでください。	
		引数	portInfo： アクセス対象のポートの情報です。	

（凡例）

○：Application ServerのJAX-WS機能でサポートしています。

(7)　javax.xml.ws.handler.LogicalMessageContext インタフェース

javax.xml.ws.handler.LogicalMessageContext インタフェースのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19–19　javax.xml.ws.handler.LogicalMessageContext インタフェースのサポート範囲

項番	戻り値の型	メソッド名／説明		サポート
1	LogicalMessage	getMessage()		○
		説明	このメッセージコンテキストからメッセージを取得します。	

（凡例）

○：Application Server の JAX-WS 機能でサポートしています。

(8)　javax.xml.ws.handler.MessageContext インタフェース

javax.xml.ws.handler.MessageContext インタフェースのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19–20　javax.xml.ws.handler.MessageContext インタフェースのサポート範囲

項番	戻り値の型	メソッド名／説明		サポート
1	MessageContext.Scope	getScope(java.lang.String name)		○※1
		説明	プロパティのスコープを取得します。	
		例外	java.lang.IllegalArgumentException： name に null，空の文字列，およびこのメッセージコンテキストに関連づけられていないプロパティ名を指定した場合に発生します。 標準のメッセージコンテキストのプロパティと，ユーザ定義メッセージコンテキストプロパティは，メッセージコンテキストに関連づけられているものと見なされます。 メッセージコンテキストのプロパティについては，「19.2.5(1)　メッセージコンテキストのプロパティのサポート範囲」を参照してください。	
2	void	setScope(java.lang.String name, MessageContext.Scope scope)		○※2
		説明	プロパティのスコープを設定します。	
		例外	java.lang.IllegalArgumentException name に null，空の文字列，およびこのメッセージコンテキストに関連づけられていないプロパティ名を指定した場合に発生します。標準のメッセージコンテキストのプロパティと，ユーザ定義メッセージコンテキストプロパティは，メッセージコンテキストに関連づけられているものと見なされます。	

項番	戻り値の型	メソッド名／説明		サポート
			メッセージコンテキストのプロパティについては，「19.2.5(1)　メッセージコンテキストのプロパティのサポート範囲」を参照してください。	

（凡例）

○：Application Server の JAX-WS 機能でサポートしています。

注※1

ハンドラで使用する場合だけサポートしています。

注※2

インバウンド時のサービス側ハンドラで，ユーザ定義メッセージコンテキストプロパティを追加する場合だけサポートします。ユーザ定義メッセージコンテキストプロパティ追加時の注意事項についての詳細は，「10.21.2(2)　ユーザ定義メッセージコンテキストプロパティ追加時の注意事項」を参照してください。

(9)　javax.xml.ws.handler.PortInfo インタフェース

javax.xml.ws.handler.PortInfo インタフェースのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19–21　javax.xml.ws.handler.PortInfo インタフェースのサポート範囲

項番	戻り値の型	メソッド名／説明		サポート
1	java.lang.String	getBindingID()		○
		説明	アクセス対象のポートが使用する，バインディングを識別する URI を取得します。	
2	javax.xml.namespace.QName	getPortName()		○
		説明	アクセス対象の WSDL のポートから QName を取得します。	
3	javax.xml.namespace.QName	getServiceName()		○
		説明	アクセス対象のポートを含む，WSDL のサービスの QName を取得します。	

（凡例）

○：Application Server の JAX-WS 機能でサポートしています。

(10)　javax.xml.ws.handler.soap.SOAPHandler<T extends SOAPMessageContext>インタフェース

javax.xml.ws.handler.soap.SOAPHandler<T extends SOAPMessageContext>インタフェースのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19-22　javax.xml.ws.handler.soap.SOAPHandler<T extends SOAPMessageContext>インタフェースのサポート範囲

項番	戻り値の型	メソッド名／説明		サポート
1	java.util.Set<javax.xml.namespace.QName>	getHeaders()		○
		説明	このハンドラインスタンスで処理できるヘッダを取得します。	

（凡例）

○：Application Server の JAX-WS 機能でサポートしています。

(11)　javax.xml.ws.handler.soap.SOAPMessageContext インタフェース

javax.xml.ws.handler.soap.SOAPMessageContext インタフェースのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19-23　javax.xml.ws.handler.soap.SOAPMessageContext インタフェースのサポート範囲

項番	戻り値の型	メソッド名／説明		サポート
1	java.lang.Object[]	getHeaders(javax.xml.namespace.QName header, javax.xml.bind.JAXBContext context, boolean allRoles)		○
		説明	このメッセージコンテキストが持つメッセージから特定の QName を持つヘッダを取得します。このメッセージコンテキストがメッセージを持っていない場合，または header で指定した QName に一致するヘッダが存在しない場合は，空の配列を返します。	
		例外	javax.xml.ws.WebServiceException： 次の場合に発生します。 • header に null を指定した場合 • context に null を指定した場合	
2	javax.xml.soap.SOAPMessage	getMessage()		○
		説明	このメッセージコンテキストから SOAP メッセージを取得します。	
3	java.util.Set<java.lang.String>	getRoles()		○
		説明	ハンドラチェインの実行に関連づけられた SOAP アクタおよびロールを取得します。	
4	void	setMessage(javax.xml.soap.SOAPMessage message)		×

（凡例）

○：Application Server の JAX-WS 機能でサポートしています。

×：Application Server の JAX-WS 機能でサポートしていません。

(12) javax.xml.ws.Holder<T>クラス

javax.xml.ws.Holder<T>クラスのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19-24 javax.xml.ws.Holder<T>クラスのサポート範囲

項番	戻り値の型	メソッド名／説明		サポート
1	−	Holder()		○
		説明	null 値を含む新しいホルダを作成します。	
2	−	Holder(T value)		○
		説明	指定した値を含む新しいホルダを作成します。	

（凡例）

−：戻り値の型がないことを示します。

○：Application Server の JAX-WS 機能でサポートしています。

(13) javax.xml.ws.LogicalMessage インタフェース

javax.xml.ws.LogicalMessage インタフェースのサポート範囲を次の表に示します。

表 19-25 javax.xml.ws.LogicalMessage インタフェースのサポート範囲

項番	戻り値の型	メソッド名／説明		サポート
1	javax.xml.transform.Source	getPayload()		○
		説明	XML ソースとしてメッセージのペイロードを取得します。	
2	java.lang.Object	getPayload(javax.xml.bind.JAXBContext context)		○
		説明	JAXB オブジェクトとしてメッセージのペイロードを取得します	
3	void	setPayload(java.lang.Object payload, javax.xml.bind.JAXBContext context)		×
4	void	setPayload(javax.xml.transform.Source payload)		×

（凡例）

○：Application Server の JAX-WS 機能でサポートしています。

×：Application Server の JAX-WS 機能でサポートしていません。

(14) javax.xml.ws.ProtocolException クラス

javax.xml.ws.ProtocolException クラスのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19-26　javax.xml.ws.ProtocolException クラスのサポート範囲

項番	戻り値の型	メソッド名／説明		サポート
1	−	ProtocolException()		○
		説明	詳細メッセージが null である新規例外を構築します。	
2	−	ProtocolException(java.lang.String message)		○
		説明	指定された詳細メッセージを持つ新規例外を構築します。	
3	−	ProtocolException(java.lang.String message, java.lang.Throwable cause)		○
		説明	指定された詳細メッセージおよび原因を使用して新規例外を構築します。	
4	−	ProtocolException(java.lang.Throwable cause)		○
		説明	指定された原因と詳細メッセージを持つ新規例外を構築します。	

（凡例）

−：戻り値の型がないことを示します。

○：Application Server の JAX-WS 機能でサポートしています。

(15)　javax.xml.ws.soap.AddressingFeature クラス

javax.xml.ws.soap.AddressingFeature クラスのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19-27　javax.xml.ws.soap.AddressingFeature クラスのサポート範囲

項番	戻り値の型	メソッド名		サポート
1	−	AddressingFeature()		○
		説明	AddressingFeature を作成します。	
2	−	AddressingFeature(boolean enabled)		○
		説明	AddressingFeature を作成します。	
		引数	enabled： WS-Addressing を有効にするか無効にするかを指定します。	
3	−	AddressingFeature(boolean enabled, boolean required)		○
		説明	AddressingFeature を作成します。	
		引数	enabled： WS-Addressing を有効にするか無効にするかを指定します。 required： WS-Addressing の使用を要求する場合に指定します。	

項番	戻り値の型	メソッド名		サポート
4	−	AddressingFeature(boolean enabled, boolean required, AddressingFeature.Responses responses)		○
		説明	AddressingFeature を作成します。	
		引数	enabled： WS-Addressing を有効にするか無効にするかを指定します。 required： WS-Addressing の使用を要求する場合に指定します。 responses： 要求する応答エンドポイントの種別を指定します。応答エンドポイントの種別は，次のどれかを指定できます。※ • すべての URI javax.xml.ws.soap.AddressingFeature.Responses.ALL • 匿名 URI だけ javax.xml.ws.soap.AddressingFeature.Responses.ANONYMOUS • 非匿名 URI だけ javax.xml.ws.soap.AddressingFeature.Responses.NON_ANONYMOUS	
5	java.lang.String	getID()		○
		説明	AdderssingFeature の一意の識別子を取得します。	
6	AddressingFeature.Responses	getResponses()		○
		説明	要求する応答エンドポイントの種別を取得します。	
7	Boolean	isRequired()		○
		説明	AdderssingFeature の使用を要求するかどうかの情報を取得します。	

（凡例）

−：戻り値の型がないことを示します。

○：Application Server の JAX-WS 機能でサポートしています。

注※

responses の指定は，クライアント側 JAX-WS エンジンの動作には影響しません。

(16) javax.xml.ws.soap.MTOMFeature クラス

javax.xml.ws.soap.MTOMFeature クラスのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19-28　javax.xml.ws.soap.MTOMFeature クラスのサポート範囲

項番	戻り値の型	メソッド名		サポート
1	−	MTOMFeature()		○
		説明	MTOMFeature を作成します。	
2	−	MTOMFeature(boolean enabled)		○
		説明	MTOMFeature を作成します。	
		引数	enabled： MTOM/XOP 仕様形式の添付ファイルを使用するかどうかを指定します。	
3	−	MTOMFeature(boolean enabled, int threshold)		○
		説明	MTOMFeature を作成します。	
		引数	enabled： MTOM/XOP 仕様形式の添付ファイルを使用するかどうかを指定します。 threshold： MTOM/XOP 仕様形式の添付ファイルとして送信するバイナリデータのサイズ（単位：バイト）です。threshold の指定値≦バイナリデータのサイズのとき，バイナリデータを MTOM/XOP 仕様形式の添付ファイルとして送信します。	
4	−	MTOMFeature(int threshold)		○
		説明	MTOMFeature を作成します。	
		引数	threshold： MTOM/XOP 仕様形式の添付ファイルとして送信するバイナリデータのサイズ（単位：バイト）です。threshold の指定値≦バイナリデータのサイズのとき，バイナリデータを MTOM/XOP 仕様形式の添付ファイルとして送信します。	
5	java.lang.String	getID()		○
		説明	MTOMFeature の一意の識別子を取得します。	
6	int	getThreshold()		○
		説明	バイナリデータを MTOM/XOP 仕様形式の添付ファイルとして送信するかどうかを判定するためのしきい値を取得します。	

（凡例）

−：戻り値の型がないことを示します。

○：Application Server の JAX-WS 機能でサポートしています。

(17)　javax.xml.ws.soap.SOAPBinding インタフェース

javax.xml.ws.soap.SOAPBinding インタフェースのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19-29　javax.xml.ws.soap.SOAPBinding インタフェースのサポート範囲

項番	戻り値の型	メソッド名		サポート
1	javax.xml.soap.MessageFactory	getMessageFactory()		×
2	java.util.Set<java.lang.String>	getRoles()		×
3	javax.xml.soap.SOAPFactory	getSOAPFactory()		×
4	boolean	isMTOMEnabled()		○
		説明	MTOM/XOP 仕様形式の添付ファイルが有効化されている場合，true を返します。	
5	void	setMTOMEnabled(boolean flag)		○
		説明	MTOM/XOP 仕様形式の添付ファイルを有効または無効にします。 MTOMFeature やこのメソッドを使用して MTOM/XOP 仕様形式の添付ファイルの有効または無効が設定されていない場合に，このメソッドで設定できます。すでに MTOM/XOP 仕様形式の添付ファイルの有効または無効が設定されている場合，このメソッドによる再設定はできません。	
		引数	flag： MTOM/XOP 仕様形式の添付ファイルを有効にするか無効にするかを指定します。	
6	void	setRoles(java.util.Set<java.lang.String> roles)		×

（凡例）

○：Application Server の JAX-WS 機能でサポートしています。

×：Application Server の JAX-WS 機能でサポートしていません。

(18)　javax.xml.ws.soap.SOAPFaultException クラス

javax.xml.ws.soap.SOAPFaultException クラスのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19-30　javax.xml.ws.soap.SOAPFaultException クラスのサポート範囲

項番	戻り値の型	メソッド名／説明		サポート
1	－	SOAPFaultException (javax.xml.soap.SOAPFault fault)		○
		説明	SOAPFaultException の新規例外を構築します。	
2	javax.xml.soap.SOAPFault	getFault()		○
		説明	埋め込まれた SOAPFault インスタンスを取得します。	

（凡例）

－：戻り値の型がないことを示します。

○：Application Server の JAX-WS 機能でサポートしています。

(19)　javax.xml.ws.WebServiceException クラス

javax.xml.ws.WebServiceException クラスのサポート範囲を次の表に示します。詳細は JAX-WS 2.2 仕様を参照してください。

表 19–31　javax.xml.ws.WebServiceException クラスのサポート範囲

項番	戻り値の型	メソッド名／説明		サポート
1	−	WebServiceException()		○
		説明	詳細メッセージが null である新規例外を構築します。	
2	−	WebServiceException (java.lang.String message)		○
		説明	指定された詳細メッセージを持つ新規例外を構築します。	
3	−	WebServiceException(java.lang.String message, java.lang.Throwable cause)		○
		説明	指定された詳細メッセージおよび原因を持つ新規例外を構築します。	
4	−	WebServiceException (java.lang.Throwable cause)		○
		説明	指定された原因および詳細メッセージを持つ新規例外を構築します。	

（凡例）

−：戻り値の型がないことを示します。

○：Application Server の JAX-WS 機能でサポートしています。

19.2.5　メッセージコンテキストの使用

JAX-WS API のサポート範囲内で，ハンドラ，Web サービスクライアント，および Web サービスからメッセージコンテキストにアクセスできます。詳細を次に示します。

ハンドラの場合

コールバックされるメソッド（handleMessage メソッドなど）のパラメタでメッセージコンテキストが渡されます。

Web サービスクライアントの場合

javax.xml.ws.BindingProvider インタフェースの getRequestContext メソッド，および getResponseContext メソッドでメッセージコンテキストのコピーにアクセスできます。

Web サービスの場合

javax.xml.ws.WebServiceContext インタフェースの getMessageContext メソッドでメッセージコンテキストにアクセスできます。

JAX-WS API については，JAX-WS 2.2 仕様を参照してください。また，Application Server の JAX-WS 機能での JAX-WS API のサポート範囲については，「19.2.1　インタフェースおよびクラスの一覧

(JAX-WS)」を参照してください。Application Server の JAX-WS 機能では，JAX-WS 2.2 仕様で定義された標準のプロパティ，およびベンダ固有のプロパティをサポートしています。

JAX-WS 2.2 仕様の 9 章に記載されている標準のプロパティについては，参照だけサポートしています。変更した場合の動作はサポートしていません。

(1) メッセージコンテキストのプロパティのサポート範囲

メッセージコンテキストのプロパティのサポート範囲を次の表に示します。

表 19-32 メッセージコンテキストのプロパティの一覧

項番	プロパティ名	記載個所 ※1	必須 ※2	サポート						
				Web サービスクライアント		Web サービス ※5	ハンドラ			
							Web サービスクライアント側		Web サービス側	
				out※3	in※4		out※6	in※7	in※8	out※9
javax.xml.ws.handler.message										
1	.outbound※10	9.4.1.1	○	×※10	×※10	×	△	△	△	△
javax.xml.ws.binding.attachments										
2	.inbound※10	9.4.1.1	○	×※10	×※10	△	△	△	△	△
3	.outbound※10	9.4.1.1	○	×※10	×※10	×	△	△	△	△
javax.xml.ws.reference										
4	.parameters※10	9.4.1.1	○	×※10	×※10	△	△	△	△	△
javax.xml.ws.wsdl										
5	.description※10，※12	9.4.1.1	－	×※10	×※10	×※11	×※11	×※11	×※11	×※11
6	.service※10，※12	9.4.1.1	－	×※10	×※10	△	△	△	△	△
7	.port※10，※12	9.4.1.1	－	×※10	×※10	△	△	△	△	△
8	.interface※10，※12	9.4.1.1	－	×※10	×※10	△	△	△	△	△
9	.operation※10，※12，※13	9.4.1.1	－	×※10	△※13	△	△	△	△	△
javax.xml.ws.http.request										
10	.headers	9.4.1.1	○	◎※14	×※10	△	◎※14	△※15	△	△
11	.method※10，※11	9.4.1.1	○	×※10	×※10	△	△※15	△※15	△	△
12	.querystring※10，※11	9.4.1.1	○	×※10	×※10	△	△※15	△※15	△	△
13	.headers	9.4.1.1	○	×※10	×※10	△※16	△※15	△※15	△※16	△※16

項番	プロパティ名	記載個所 ※1	必須 ※2	サポート						
				Web サービスクライアント		Web サービス ※5	ハンドラ			
							Web サービスクライアント側		Web サービス側	
				out※3	in※4		out※6	in※7	in※8	out※9
javax.xml.ws.http.response										
14	.headers※10	9.4.1.1	○	×※10	×※10	×	△※17	△	△※17	△※17
15	.code※10	9.4.1.1	○	×※10	×※10	×	△※18	△	△	△
javax.xml.ws.servlet										
16	.context※10, ※11	9.4.1.1	○	×※10	×※10	△	△※15	△※15	△	△
17	.request※10	9.4.1.1	○	×※10	×※10	△	△※15	△※15	△	△
18	.response※10	9.4.1.1	○	×※10	×※10	△	△※15	△※15	△	△
javax.xml.ws.service.endpoint										
19	.address※19	4.2.1.1	○	◎	△	×	◎	△	△	△
javax.xml.ws.security.auth										
20	.username	4.2.1.1	○	◎	△	×	◎	△	△	△
21	.password	4.2.1.1	○	◎	△	×	◎	△	△	△
javax.xml.ws.session										
22	.maintain	4.2.1.1	○	◎	△	×	◎	△	△	△
javax.xml.ws.soap.http.soapaction										
23	.use	4.2.1.1	−	×※20	×※20	×※20	×※20	×※20	×※20	×※20
24	.uri	4.2.1.1	−	×※20	×※20	×※20	×※20	×※20	×※20	×※20
com.cosminexus.jaxws										
25	.connect.timeout			◎	△	×	◎	△	△	△
26	.request.timeout			◎	△	×	◎	△	△	△
com.cosminexus.xml.ws.client.http										
27	.HostnameVerificationProperty			◎※21	×	×	×	×	×	×

（凡例）

○：必須であることを示します。

−：必須でないことを示します。

◎：参照および変更できます。

△：参照だけできます。変更した場合の動作は保証されません。

×：参照および変更できません。

空欄：Application Server の JAX-WS 機能が提供するプロパティであるため，該当しないことを示します。

注※1

JAX-WS 2.2 仕様で定義されている個所を示します。

注※2

JAX-WS 2.2 仕様で必須かどうかを示します。

注※3

javax.xml.ws.BindingProvider#getRequestContext で取得できる要求コンテキストで，参照または変更できるかどうかを示します。

注※4

javax.xml.ws.BindingProvider#getResponseContext で取得できる要求コンテキストで，参照または変更できるかどうかを示します。

注※5

Web サービスコンテキストのインジェクションについては，「10.21.2　Web サービスコンテキストのインジェクション」を，Web サービスにおけるメッセージコンテキストプロパティについては，「19.2.5(2)(l)　Web サービスでのメッセージコンテキストプロパティ」を参照してください。

注※6

Web サービスクライアントに関連づけられたハンドラで，アウトバウンド時（要求メッセージを送信するとき）に，参照または変更できるかどうかを示します。

注※7

Web サービスクライアントに関連づけられたハンドラで，インバウンド時（応答メッセージを受信するとき）に，参照または変更できるかどうかを示します。

注※8

Web サービス実装クラスまたはプロバイダ実装クラスに関連づけられたハンドラで，インバウンド時（要求メッセージを受信する時）に，参照または変更できるかどうかを示します。メッセージコンテキストプロパティ追加時の注意事項については，「10.21.2(2)　ユーザ定義メッセージコンテキストプロパティ追加時の注意事項」を参照してください。

注※9

Web サービス実装クラスまたはプロバイダ実装クラスに関連づけられたハンドラで，アウトバウンド時（応答メッセージを送信するとき）に，参照または変更できるかどうかを示します。メッセージコンテキストプロパティ追加時の注意事項については，「10.21.2(2)　ユーザ定義メッセージコンテキストプロパティ追加時の注意事項」を参照してください。

注※10

「19.2.5(2)(e)　Web サービスクライアントでの HANDLER スコープのメッセージコンテキストプロパティ」を参照してください。

注※11

常に null が返されます。

注※12

「19.2.5(2)(h)　WSDL に関連するメッセージコンテキストプロパティ」を参照してください。

注※13

「19.2.5(2)(i)　WSDL オペレーションの名前に関連するメッセージコンテキストプロパティ」を参照してください。

注※14

HTTP レスポンス圧縮機能との連携時に使用する HTTP ヘッダ「Accept-Encoding」，および HTTP リクエストボディの gzip 圧縮時に使用する HTTP ヘッダ「Content-Encoding」の追加と参照だけできます。「Accept-Encoding」については，「10.18　HTTP レスポンス圧縮機能との連携」を参照してください。「Content-Encoding」については「10.17　HTTP リクエストボディの gzip 圧縮」を参照してください。

注※15

「19.2.5(2)(a)　Web サービスクライアント側のハンドラで操作しても意味のないメッセージコンテキストプロパティ」を参照してください。

注※16

「19.2.5(2)(b)　パス情報」を参照してください。

注※17

「19.2.5(2)(c)　HTTP ヘッダ」を参照してください。

注※18

「19.2.5(2)(d)　HTTP ステータスコード」を参照してください。

注※19

「19.2.5(2)(g)　サービスエンドポイントのアドレスに指定するメッセージコンテキストのプロパティ」を参照してください。

注※20

「19.2.5(2)(f)　SOAPAction ヘッダに関連するメッセージコンテキストプロパティ」を参照してください。

注※21

「19.2.5(2)(k)　com.cosminexus.xml.ws.client.http.HostnameVerificationProperty プロパティの設定方法」を参照してください。

(2)　メッセージコンテキスト使用時の注意事項

メッセージコンテキスト使用時の注意事項について説明します。

(a)　Web サービスクライアント側のハンドラで操作しても意味のないメッセージコンテキストプロパティ

要求メッセージの HTTP メソッドのマップを保持するプロパティ（javax.xml.ws.http.request.method プロパティなど）は，Web サービス側のハンドラで取得して意味のあるプロパティです。したがって，Web サービスクライアント側のハンドラでこのプロパティを参照した場合は常に null が返されます。

(b)　パス情報

javax.xml.ws.http.request.pathinfo プロパティは常に null が保持されます。

(c)　HTTP ヘッダ

- 応答メッセージの HTTP ヘッダのマップを保持する javax.xml.ws.http.response.headers プロパティは，インバウンド時に Web サービスクライアント側のハンドラで取得して意味のあるプロパティです。したがって，Web サービス側のハンドラ，または Web サービスクライアント側のアウトバウンド処理のハンドラで参照した場合は，常に null が返されます。
- メッセージコンテキストの javax.xml.ws.http.request.headers プロパティ，および javax.xml.ws.http.response.headers プロパティをハンドラで参照した場合，取得されるマップ（Map<String,List<String>>オブジェクト）のキー値である HTTP ヘッダ名は，実際に送受信される HTTP メッセージ（SOAP メッセージを含む HTTP メッセージ）に関係なく，常に先頭文字だけが大文字になります。

  例）Content-type

(d) HTTP ステータスコード

Web サービスクライアント側のアウトバウンドのハンドラは，HTTP 通信が行われる前に処理されます。したがって，ハンドラから HTTP ステータスコードを保持する javax.xml.ws.http.response.code プロパティを参照した場合は，常に null が返されます。

(e) Web サービスクライアントでの HANDLER スコープのメッセージコンテキストプロパティ

標準のメッセージコンテキストプロパティとして APPLICATION スコープおよび HANDLER スコープがありますが，Web サービスクライアントからは，APPLICATION スコープのメッセージコンテキストプロパティだけ参照できます。したがって，Application Server の JAX-WS 機能では，「19.2.5(1) メッセージコンテキストのプロパティのサポート範囲」の表で注※9 が付けられたプロパティは，Web サービスクライアントでは使用できません。これらのプロパティを参照した場合の動作は保証されません。

(f) SOAPAction ヘッダに関連するメッセージコンテキストプロパティ

Application Server の JAX-WS 機能では SOAPAction ヘッダをサポートしていないため，javax.xml.ws.soap.http.soapaction.use プロパティと javax.xml.ws.soap.http.soapaction.uri プロパティは使用できません。これらのプロパティを参照した場合の動作は保証されません。

(g) サービスエンドポイントのアドレスに指定するメッセージコンテキストのプロパティ

サービスエンドポイントのアドレスを指定する javax.xml.ws.service.endpoint.address プロパティには，空白および空文字は設定できません。空白または空文字を設定した場合の動作は保証されません。javax.xml.ws.service.endpoint.address プロパティのその他の指定値については，「20.2.3 soap:address 要素または soap12:address 要素の location 属性に指定できる値」を参照してください。

(h) WSDL に関連するメッセージコンテキストプロパティ

ディスパッチベースの Web サービスクライアントおよびプロバイダベースの Web サービスでは WSDL ファイルがないので，WSDL に関連するメッセージコンテキストプロパティを参照した場合は常に null が返ります。

(i) WSDL オペレーションの名前に関連するメッセージコンテキストプロパティ

スタブベースの Web サービスクライアントでの javax.xml.ws.wsdl.operation プロパティの参照は，javax.xml.ws.BindingProvider#getResponseContext で取得できる要求コンテキストだけでできます。javax.xml.ws.BindingProvider#getRequestContext で取得できる要求コンテキストで参照した場合は常に null が返ります。また，javax.xml.ws.wsdl.operation プロパティに値を設定しても，送信する SOAP メッセージには影響を与えません。

(j) WS-RM 1.2 機能を使用する場合のサービスエンドポイントのアドレスの指定

WS-RM 1.2 機能を使用する Web サービスクライアントでは，最初の Web サービスの呼び出し前にサービスエンドポイントのアドレスを指定してください。最初の通信以降にサービスエンドポイントを変更すると，WS-RM の通信に失敗します。

(k) com.cosminexus.xml.ws.client.http.HostnameVerificationProperty プロパティの設定方法

com.cosminexus.xml.ws.client.http.HostnameVerificationProperty プロパティにメッセージコンテキストで true または false を設定する場合は，次の例のように java.lang.String 型の文字列で設定してください。java.lang.String 型の文字列以外で設定した場合の動作は保証されません。

```
context.put("com.cosminexus.xml.ws.client.http.HostnameVerificationProperty", "true");
```

(l) Web サービスでのメッセージコンテキストプロパティ

Web サービスでのメッセージコンテキストプロパティの，参照および変更について次に説明します。

- HANDLER スコープである JAX-WS 2.2 仕様の 9.4.1.1 項で定義されている標準のメッセージコンテキストプロパティのうち，一部のメッセージコンテキストプロパティを参照できます。参照できるメッセージコンテキストプロパティについては，「19.2.5(1) メッセージコンテキストのプロパティのサポート範囲」の「表 19-32 メッセージコンテキストのプロパティの一覧」を参照してください。
- APPLICATION スコープである JAX-WS 2.2 仕様の 4.2.1.1 項で定義されている標準のメッセージコンテキストプロパティや CJW 固有のメッセージコンテキストプロパティについては，参照および変更できません。
- APPLICATION スコープのユーザ定義メッセージコンテキストプロパティの参照および変更ができます。

19.3　アノテーションのサポート範囲

javax.xml.ws.WebServiceRef アノテーションは，サービスクラスおよびポートをインジェクトする場合に指定します。これら以外に指定したときの動作は保証されません。なお，フィールドと対応する setter メソッドには，同時に指定できません。

javax.xml.ws.WebServiceRef アノテーションは，J2EE サーバ上で実行される Web サービスクライアントの，フィールドおよび setter メソッドに指定できます。これら以外に指定した場合は無視されます。

19.3.1　javax.xml.ws.WebServiceRef アノテーション

javax.xml.ws.WebServiceRef アノテーション要素とサポート範囲を次の表に示します。

表 19–33　JAX-WS のアノテーションの要素の一覧

項番	アノテーション		サポート
	アノテーション名	要素名	
1	javax.xml.ws.WebServiceRef	lookup	×
2		mappedName	×
3		name	×
4		type	×
5		value	○
6		wsdlLocation	○

（凡例）

○：アノテーションおよび要素が指定できることを示します。

×：アノテーションおよび要素が指定できないことを示します（非サポート）。

要素ごとのサポート範囲について説明します。

(1)　value 要素（javax.xml.ws.WebServiceRef）

value 要素は，javax.xml.ws.Service を継承するサービスクラスを指定します。

サービスクラス型のフィールドまたはメソッドに対して javax.xml.ws.WebServiceRef アノテーションを指定した場合は，value 要素を指定できません。指定した場合の動作は保証されません。

ポート型のフィールドまたはメソッドに対して javax.xml.ws.WebServiceRef アノテーションを指定した場合は，必ず指定します。指定しない場合の動作は保証されません。

(2) wsdlLocation 要素（javax.xml.ws.WebServiceRef）

wsdlLocation 要素は，Web サービスの WSDL 文書の位置を指定します。

指定できる方法を次に示します。

- URL

  WSDL 文書の URL を指定します。指定形式は java.net.URL クラスの仕様に従います。プロトコルは http と https だけ指定できます。そのほかのプロトコルは指定できません。

- 相対パス

  WAR ファイルまたは EJB JAR ファイル内の WSDL 文書を，次のように相対パスで指定します。なお，RFC 2396 仕様の規則に従った文字列を指定できます。

  - javax.xml.ws.WebServiceRef アノテーションを指定した Web サービスクライアントを WAR ファイルに格納する場合は，WEB-INF で始まる相対パスを指定します。WSDL 文書はその WAR ファイル内の指定したパスに格納します。
  - javax.xml.ws.WebServiceRef アノテーションを指定した Web サービスクライアントを EJB JAR ファイルに格納する場合は，META-INF で始まる相対パスを指定します。WSDL 文書はその EJB JAR ファイル内の指定したパスに格納します。

- 絶対パス

  WSDL 文書の絶対パスを指定します。システムに依存する絶対パスで指定してください。指定形式は java.io.File クラスの仕様に従います。

wsdlLocation 要素を指定するときの注意事項について説明します。

- Web サービスクライアントのインスタンス生成時に，wsdlLocation 要素に指定された値を使用してサービスクラスを生成し，サービスクラスやポートのインジェクトを行います。このため，wsdlLocation 要素にはインスタンス生成時に有効な WSDL 文書の位置を指定する必要があります。存在しない WSDL や不正な内容の WSDL ロケーションを指定した場合は，サービスクラスやポートのインジェクションを行わないため，フィールドの値や setter メソッドの戻り値は null のままとなります。
- wsdlLocation 要素を指定しない場合は，サービスクラスのデフォルトコンストラクタと同様に，サービスクラスに指定された javax.xml.ws.WebServiceClient アノテーションの wsdlLocation 要素の値を使用して，サービスクラスまたはポートのインスタンスを生成します。

19.4　ハンドラチェイン設定ファイルのサポート範囲

ハンドラチェイン設定ファイルの構文のサポート範囲を，次に示す要素ごとに説明します。

- javaee:handler-chains 要素
- javaee:handler-chain 要素
- javaee:handler 要素
- javaee:handler-name 要素
- javaee:handler-class 要素
- javaee:soap-header 要素
- javaee:soap-role 要素

ここで説明する内容を除いたサポート範囲については，Java EE 5 仕様の"Java EE Web Services Metadata Handler Chain Schema"（標準のスキーマ）に従います。

19.4.1　javaee:handler-chains 要素

javaee:handler-chains 要素のサポート範囲を説明します。

- ハンドラチェイン設定ファイルのルート要素として，1 個だけ記述できます。省略できません。
- 子要素として javaee:handler-chain 要素を指定できます。javaee:handler-chain 要素以外を記述した場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW30019-E）。
- XML 仕様および XML Schema 仕様の標準の属性（xmlns 属性など）を除いて，属性を指定できません。XML 仕様および XML Schema 仕様の標準の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW30019-E）。

19.4.2　javaee:handler-chain 要素

javaee:handler-chain 要素のサポート範囲を説明します。

- javaee:handler-chains 要素の子要素として，1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW30019-E）。
- 子要素として javaee:handler 要素を指定できます。javaee:handler 要素以外を記述した場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW30019-E）。
- XML 仕様や XML Schema 仕様の標準の属性（xmlns 属性など）を除いて，属性を指定できません。XML 仕様および XML Schema 仕様の標準の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW30019-E）。

19.4.3　javaee:handler 要素

javaee:handler 要素のサポート範囲を説明します。

- javaee:handler-chain 要素の子要素として，1 個から 64 個まで記述できます。省略または 65 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW30019-E）。
- 子要素として下記の要素を指定できます。次に示す要素以外を記述した場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW30019-E）。
  - javaee:handler-name 要素
  - javaee:handler-class 要素
  - javaee:soap-header 要素
  - javaee:soap-role 要素
- javaee:handler-name 要素，javaee:handler-class 要素，javaee:soap-header 要素，および javaee:soap-role 要素は上記の順序で指定してください。
- XML 仕様および XML Schema 仕様の標準の属性（xmlns 属性など）を除いて，属性を指定できません。XML 仕様および XML Schema 仕様の標準の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW30019-E）。

19.4.4　javaee:handler-name 要素

javaee:handler-name 要素のサポート範囲を説明します。

- javaee:handler 要素の子要素として，0 個または 1 個記述できます。2 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW30019-E）※。
- 値には，ハンドラの名称を記述します。ただし，JAX-WS エンジンではこの値は無視されるので，XML Schema 仕様の xsd:token 型の制限に違反しない範囲で値を記述できます※。
- 子要素は記述できません。子要素を記述した場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW30019-E）。
- XML 仕様や XML Schema 仕様の標準の属性（xmlns 属性など）を除いて，属性を指定できません。XML 仕様および XML Schema 仕様の標準の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW30019-E）。

注※

標準のスキーマでは，javaee:handler-name 要素は必須です。動作定義ファイルで com.cosminexus.jaxws.validation.handlerchain.strict プロパティに true を設定すると，この要素の指定有無が JAX-WS エンジンでチェックされます。
com.cosminexus.jaxws.validation.handlerchain.strict プロパティで true を設定した場合に，javaee:handler 要素が省略されているときは，標準エラー出力とログにエラーメッセージが出力されます（KDJW30019-E）。

19.4.5 javaee:handler-class 要素

javaee:handler-class 要素のサポート範囲を説明します。

- javaee:handler 要素の子要素として，1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW30019-E）。
- 値には，ハンドラを実装したクラスの完全修飾名を記述してください。存在しないクラス名を記述した場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW00011-E）。
- 子要素は記述できません。子要素を記述した場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW30019-E）。
- XML 仕様および XML Schema 仕様の標準の属性（xmlns 属性など）を除いて，属性を指定できません。XML 仕様および XML Schema 仕様の標準の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW30019-E）。

19.4.6 javaee:soap-header 要素

javaee:soap-header 要素のサポート範囲を説明します。

- javaee:handler 要素の子要素として記述できます。省略することもできます。
- 値には，このハンドラが処理する SOAP ヘッダの名称を記述します。ただし，JAX-WS エンジンではこの値は無視されるので，XML Schema 仕様の xsd:QName 型の制限に違反しない範囲で値を記述できます。
- 子要素は記述できません。子要素を記述した場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW30019-E）。
- XML 仕様および XML Schema 仕様の標準の属性（xmlns 属性など）を除いて，属性を指定できません。XML 仕様および XML Schema 仕様の標準の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW30019-E）。

19.4.7 javaee:soap-role 要素

javaee:soap-role 要素のサポート範囲を説明します。

- javaee:handler 要素の子要素として記述できます。省略することもできます。
- 値には，このハンドラが振る舞う SOAP アクタまたはロールを記述してください。ただし，"http://www.w3.org/2003/05/soap-envelope/role/none"は記述できません。記述した場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW10007-E）。
- 子要素は記述できません。子要素を記述した場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW30019-E）。

- XML 仕様および XML Schema 仕様の標準の属性（xmlns 属性など）を除いて，属性を指定できません。XML 仕様および XML Schema 仕様の標準の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW30019-E）。

20 WSDL 仕様のサポート範囲

この章では，Web サービスを開発するときに注意が必要な，WSDL 仕様のサポート範囲について説明します。

20.1 WSDL 1.1 仕様のサポート範囲

WSDL の要素ごとに，WSDL 1.1 仕様のサポート範囲を説明します。

注意事項

- この節に記載していない要素や属性を指定している場合など，Application Server の JAX-WS 機能でサポートしている WSDL 仕様として文法的に不正な場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- この節では，WSDL 構文の制約について説明しています。WSDL 構文の問題がない場合でも，Java へのマッピングで問題があれば，別のエラーとなることがあります。WSDL 上で参照されない要素や属性についても不正がある場合は，エラーとなることがあります。

また，WSDL 拡張要素および拡張属性については次の注意事項があります。

- MIME バインディングのための拡張要素（mime:content，mime:mimeXml など）はサポートしていません。したがって，MIME 関連の WSDL 要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51188-E）。
- 次の仕様はサポートしています。

  ・SOAP 1.1 仕様の SOAP バインディングのための拡張要素（soap:header，soap:body など）

  ・SOAP 1.2 仕様の SOAP バインディングのための拡張要素

  ・WS-Addressing 1.0 仕様の拡張要素（wsaw:UsingAddressing など）および拡張属性（wsaw:Action など）

  ・JAX-WS 2.2 仕様のバインディング宣言（jaxws:bindings など）
- サポートしていない拡張要素または拡張属性を指定した場合は，無視されます。

20.1.1 wsdl:definitions 要素

wsdl:definitions 要素のサポート範囲を説明します。

- WSDL ファイルのルート要素として 1 個だけ記述できます。省略できません。省略または 2 個以上記述した場合は，XML Processor のエラーになります。
- 子要素として次に示す要素を指定できます。次に示す WSDL 要素以外の要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。cjwsimport コマンドで無視しない拡張要素のうち，次に示す要素以外の拡張要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51053-E）。
  - wsdl:documentation 要素
  - jaxws:bindings 要素（JAX-WS 2.2 仕様のバインディング宣言）
  - wsdl:import 要素

  - wsdl:types 要素
  - wsdl:message 要素
  - wsdl:portType 要素
  - wsdl:binding 要素
  - wsdl:service 要素
- wsdl:documentation 要素と jaxws:bindings 要素は，上記の順序で指定する必要があります。ただし，それ以外の要素は，指定する順序を問いません。

  wsdl:documentation 要素と jaxws:bindings 要素の指定順序を誤った場合，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- 次の属性を指定できます。次の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
  - name 属性
  - targetNamespace 属性

(1)　name 属性（wsdl:definitions 要素）

wsdl:definitions 要素に含まれる name 属性のサポート範囲を説明します。

- 0 個または 1 個記述できます。2 個以上記述した場合は，XML Processor のエラーになります。
- 指定できる値については，「20.2.1　NCName 型に指定できる値」を参照してください。

(2)　targetNamespace 属性（wsdl:definitions 要素）

wsdl:definitions 要素に含まれる targetNamespace 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 指定できる値については，「15.1.1(2)　名前空間の条件」を参照してください。

20.1.2　wsdl:import 要素

wsdl:import 要素のサポート範囲を説明します。

- wsdl:definitions 要素の子要素として，0 個から 255 個まで記述できます。256 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51052-E）。
- 子要素として wsdl:documentation 要素を指定できます。wsdl:documentation 要素以外の要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が

終了されます（KDJW51029-E）。cjwsimport コマンドで無視しない拡張要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51054-E）。

- 次の属性を指定できます。次の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
  - namespace 属性
  - location 属性

(1) namespace 属性（wsdl:import 要素）

wsdl:import 要素に含まれる namespace 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 指定できる値については，「26.3.1(1) namespace 属性（wsdl:import 要素）」を参照してください。

(2) location 属性（wsdl:import 要素）

wsdl:import 要素に含まれる location 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- xsd:anyURI を満たす任意の文字列を指定できます。
- 指定できる値については，「26.3.1(2) location 属性（wsdl:import 要素）」を参照してください。

20.1.3 wsdl:types 要素

wsdl:types 要素のサポート範囲を説明します。

- wsdl:definitions 要素の子要素として，1 個だけ記述できます。省略できません※。省略または 2 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51049-E）。

  注※

  WSDL のインポート機能を使用する場合は，インポート関係にあるすべての WSDL 内で 1 個あれば足りるので，この場合は省略できます。また，インポート関係にある WSDL の合計が上限になります。

- 子要素として次に示す要素を指定できます。次に示す WSDL 要素以外の要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます

（KDJW51029-E）。cjwsimport コマンドで無視しない拡張要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51059-E）。

  - wsdl:documentation 要素
  - xsd:schema 要素

- wsdl:documentation 要素と xsd:schema 要素は上記の順序で指定する必要があります。指定順序を誤った場合，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- 属性は指定できません。属性を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

20.1.4 wsdl:message 要素

wsdl:message 要素のサポート範囲を説明します。

- wsdl:definitions 要素の子要素として，1 個以上記述できます。省略できません※。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51050-E）。

  注※

  WSDL のインポート機能を使用する場合は，インポート関係にあるすべての WSDL 内で 1 個あれば足りるので，この場合は省略できます。また，インポート関係にある WSDL の合計が上限になります。

- 子要素として次に示す要素を指定できます。次に示す WSDL 要素以外の要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。cjwsimport コマンドで無視しない拡張要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51055-E）。
  - wsdl:documentation 要素
  - wsdl:part 要素
- wsdl:documentation 要素と wsdl:part 要素は上記の順序で指定する必要があります。指定順序を誤った場合，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- name 属性を指定できます。name 属性以外の属性は指定できません。name 属性以外の属性を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

(1) name 属性（wsdl:message 要素）

wsdl:message 要素に含まれる name 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 指定できる値については，「20.2.1　NCName 型に指定できる値」を参照してください。ただし，同じ wsdl:definitions 要素以下にあるほかの wsdl:message 要素の name 属性と同じ値は指定できません。同じ値を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

20.1.5　wsdl:part 要素

wsdl:part 要素のサポート範囲を説明します。

- wsdl:message 要素の子要素として，0 個から 255 個まで記述できます。256 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- 指定できる値については，「20.2　WSDL 作成時の注意事項」を参照してください。
- 子要素として wsdl:documentation 要素を指定できます。wsdl:documentation 要素以外の要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。cjwsimport コマンドで無視しない拡張要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51056-E）。
- 次の属性を指定できます。次の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
  - name 属性
  - element 属性

(1)　name 属性（wsdl:part 要素）

wsdl:part 要素に含まれる name 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 指定できる値については，「15.1.5(2)　パート名の条件」を参照してください。

(2)　element 属性（wsdl:part 要素）

wsdl:part 要素に含まれる element 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 参照できる wsdl:types 要素以下に宣言されている要素宣言を QName で指定してください。

20.1.6　wsdl:portType 要素

wsdl:portType 要素のサポート範囲を説明します。

- 要素の子要素として，1 個から 255 個まで記述できます。省略できません※。省略または 256 個以上記述した場合については，「15.1.2(3)　ポートタイプの記述個数」を参照してください。

  注※

  WSDL のインポート機能を使用する場合は，インポート関係にあるすべての WSDL 内で 1 個あれば足りるので，この場合は省略できます。また，インポート関係にあるすべての WSDL 内での合計が上限になります。
- 子要素として次に示す要素を指定できます。次に示す WSDL 要素以外の要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。cjwsimport コマンドで無視しない拡張要素のうち，次に示す要素以外の拡張要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51074-E）。
  - wsdl:documentation 要素
  - jaxws:bindings 要素（JAX-WS 2.2 仕様のバインディング宣言）
  - wsdl:operation 要素
- wsdl:documentation 要素，jaxws:bindings 要素，および wsdl:operation 要素は上記の順序で指定する必要があります。指定順序を誤った場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- name 属性を指定できます。name 属性以外の属性は指定できません。name 属性以外の属性を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

(1)　name 属性（portType 要素）

wsdl:portType 要素に含まれる name 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 指定できる値については，「15.1.2(2)　ポートタイプ名の条件」を参照してください。ただし，同じ wsdl:definitions 要素以下にある，ほかの wsdl:portType 要素の name 属性と同じ値は指定できませ

ん。同じ値を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

20.1.7　wsdl:operation 要素（wsdl:portType 要素の子要素の場合）

wsdl:operation 要素のサポート範囲を説明します。

- wsdl:portType 要素の子要素として，1 個から 255 個まで記述できます。省略できません。省略または 256 個以上記述した場合については，「15.1.3(3)　オペレーションとその子要素の記述個数」を参照してください。
- 子要素として次に示す要素を指定できます。次に示す WSDL 要素以外の要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。cjwsimport コマンドで無視しない拡張要素のうち，次に示す要素以外の拡張要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51080-E）。
  - wsdl:documentation 要素
  - jaxws:bindings 要素（JAX-WS 2.2 仕様のバインディング宣言）
  - wsdl:input 要素
  - wsdl:output 要素
  - wsdl:fault 要素
- wsdl:documentation 要素，jaxws:bindings 要素，wsdl:input 要素，wsdl:output 要素および wsdl:fault 要素は上記の順序で指定する必要があります。指定順序を誤った場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- name 属性を指定できます。name 属性以外の属性は指定できません。name 属性以外の属性を指定した場合の動作は保証されません。

(1)　name 属性（wsdl:operation 要素）

wsdl:operation 要素に含まれる name 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 指定できる値については，「15.1.3(2)　オペレーション名の条件」を参照してください。ただし，同じ wsdl:operation 要素以下にある，ほかの wsdl:portType 要素の name 属性と同じ値は指定できません。同じ値を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51083-E）。

20.1.8　wsdl:input 要素（wsdl:portType 要素の孫要素の場合）

wsdl:input 要素のサポート範囲を説明します。

- wsdl:operation 要素の子要素として，1 個だけ記述できます。省略できません。省略または 2 個以上記述した場合については，「15.1.3(3)　オペレーションとその子要素の記述個数」を参照してください。
- 子要素として wsdl:documentation 要素を指定できます。wsdl:documentation 要素以外の要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。cjwsimport コマンドで無視しない拡張要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51057-E）。
- 次の属性を指定できます。次の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
  - name 属性
  - message 属性
  - wsaw:Action 属性

  wsaw:Action 属性については，「37.5.6　wsa:Action 要素指定時の動作」を参照してください。

(1)　name 属性（wsdl:input 要素）

wsdl:input 要素に含まれる name 属性のサポート範囲を説明します。

- 0 個または 1 個記述できます。2 個以上記述した場合は，XML Processor のエラーになります。
- 指定できる値については，「20.2.1　NCName 型に指定できる値」を参照してください。

(2)　message 属性（wsdl:input 要素）

wsdl:input 要素に含まれる message 属性のサポート範囲を説明します。

- wsdl:input 要素の子要素として，1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 参照できる wsdl:definitions 要素以下に宣言されている wsdl:message を QName で指定してください。

20.1.9　wsdl:output 要素（wsdl:portType 要素の孫要素の場合）

wsdl:output 要素のサポート範囲を説明します。

- wsdl:operation 要素の子要素として，1 個だけ記述できます。0 個または 2 個以上記述した場合については，「15.1.3(3)　オペレーションとその子要素の記述個数」を参照してください。

- 子要素として wsdl:documentation 要素を指定できます。wsdl:documentation 要素以外の要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。cjwsimport コマンドで無視しない拡張要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51058-E）。
- 次の属性を指定できます。次の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
  - name 属性
  - message 属性
  - wsaw:Action 属性

  wsaw:Action 属性については，「37.5.6　wsa:Action 要素指定時の動作」を参照してください。

(1)　name 属性（wsdl:output 要素）

wsdl:output 要素に含まれる name 属性のサポート範囲を説明します。

- 0 個または 1 個記述できます。2 個以上記述した場合は，XML Processor のエラーになります。
- 指定できる値については，「20.2.1　NCName 型に指定できる値」を参照してください。

(2)　message 属性（wsdl:output 要素）

wsdl:output 要素に含まれる message 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 参照できる wsdl:definitions 要素以下に宣言されている wsdl:message を QName で指定してください。

20.1.10　wsdl:fault 要素（wsdl:portType 要素の孫要素の場合）

wsdl:fault 要素のサポート範囲を説明します。

- wsdl:operation 要素の子要素として，0 個から 255 個まで記述できます。省略または 256 個以上記述した場合については，「15.1.3(3)　オペレーションとその子要素の記述個数」を参照してください。
- 子要素として次に示す要素を指定できます。次に示す WSDL 要素以外の要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。cjwsimport コマンドで無視しない拡張要素のうち，次に示す拡張要素以外の拡張要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51096-E）。

  - wsdl:documentation 要素
  - jaxws:bindings 要素（JAX-WS 2.2 仕様のバインディング宣言）
- wsdl:documentation 要素と jaxws:bindings 要素は上記の順序で指定する必要があります。指定順序を誤った場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- 次の属性を指定できます。次の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
  - name 属性
  - message 属性
  - wsaw:Action 属性

  wsaw:Action 属性については，「37.5.6　wsa:Action 要素指定時の動作」を参照してください。

(1)　name 属性（wsdl:fault 要素）

wsdl:fault 要素に含まれる name 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 指定できる値については，「20.2.1　NCName 型に指定できる値」を参照してください。

(2)　message 属性（wsdl:fault 要素）

wsdl:fault 要素に含まれる message 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 参照できる wsdl:definitions 要素以下に宣言されている wsdl:message を QName で指定します。

20.1.11　wsdl:binding 要素

wsdl:binding 要素のサポート範囲を説明します。

- wsdl:definitions 要素の子要素として，1 個から 255 個まで記述できます。省略できません※。省略または 256 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51100-E）。

注※

WSDL のインポート機能を使用する場合は，インポート関係にあるすべての WSDL 内で 1 個あれば足りるので，この場合は省略できます。また，インポート関係にあるすべての WSDL 内での合計が上限になります。

- 子要素として次に示す要素を指定できます。次に示す WSDL 要素以外の要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。cjwsimport コマンドで無視しない拡張要素のうち，次に示す要素以外の拡張要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51102-E）。
  - wsdl:documentation 要素
  - soap:binding 要素（SOAP 1.1 仕様の SOAP バインディングのための拡張要素）※
  - soap12:binding 要素（SOAP 1.2 仕様の SOAP バインディングのための拡張要素）※
  - jaxws:bindings 要素（JAX-WS 2.2 仕様のバインディング宣言）
  - wsaw:UsingAddressing 要素（WS-Addressing 1.0 仕様の拡張要素）
  - wsdl:operation 要素

  注※

  soap:binding 要素と soap12:binding 要素は，どちらかを選択して指定してください。両方の要素を指定することはできません。
- wsdl:binding 要素の子要素は，上記の順序で指定してください。指定順序を誤った場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。ただし，次に示す要素の順序は入れ替わってもかまいません。
  - soap:binding 要素または soap12:binding 要素
  - jaxws:bindings 要素
  - wsaw:UsingAddressing 要素
- 次の属性を指定できます。次の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
  - name 属性
  - type 属性

(1) name 属性（wsdl:binding 要素）

wsdl:binding 要素に含まれる name 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 指定できる値については，「20.2.1　NCName 型に指定できる値」を参照してください。

(2) type 属性（wsdl:binding 要素）

wsdl:binding 要素に含まれる type 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 参照できる wsdl:definitions 要素以下に宣言されている wsdl:message を QName で指定します。

20.1.12 wsdl:operation 要素（wsdl:binding 要素の子要素の場合）

wsdl:operation 要素のサポート範囲を説明します。

- wsdl:binding 要素の子要素として，1 個から 255 個まで記述できます。省略できません。省略または 256 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- 子要素として次に示す要素を指定できます。次に示す WSDL 要素以外の要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。cjwsimport コマンドで無視しない拡張要素のうち，次に示す要素以外の拡張要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51108-E）。
  - wsdl:documentation 要素
  - soap:operation 要素（SOAP 1.1 仕様の SOAP バインディングのための拡張要素）※
  - soap12: operation 要素（SOAP 1.2 仕様の SOAP バインディングのための拡張要素）※
  - jaxws:bindings 要素（JAX-WS 2.2 仕様のバインディング宣言）
  - wsaw:Anonymous 要素（WS-Addressing 1.0 仕様の拡張要素）
  - wsdl:input 要素
  - wsdl:output 要素
  - wsdl:fault 要素

  注※

  soap: operation 要素と soap12: operation 要素は，どちらかを選択して指定してください。両方の要素を指定することはできません。
- wsdl:operation 要素（wsdl:binding 要素の子要素の場合）の子要素は，上記の順序で指定してください。指定順序を誤った場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。ただし，次に示す要素の順序は入れ替わってもかまいません。
  - soap:operation 要素または soap12: operation 要素

  - jaxws:bindings 要素
  - wsaw:Anonymous 要素
- name 属性を指定できます。name 属性以外の属性は指定できません。name 属性以外の属性を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- wsdl:binding 要素の wsdl:operation 要素は，wsdl:portType 要素で定義されている wsdl:operation 要素に対応するように定義してください。対応するように定義していない場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51112-E）。
- 子要素として wsdl:output 要素を指定する場合は，wsdl:portType 要素の wsdl:operation 要素で定義されている wsdl:output 要素に対応するように定義してください。
  - wsdl:portType 要素の wsdl:operation 要素に wsdl:output 要素を 1 個記述し，wsdl:binding 要素の wsdl:operation 要素から wsdl:output 要素を省略したときは，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51222-E）。
  - wsdl:portType 要素の wsdl:operation 要素から wsdl:output 要素を省略し，wsdl:binding 要素の wsdl:operation 要素に wsdl:output 要素を 1 個記述したときは，wsdl:output 要素が無視され，cjwsimport コマンドの処理が続行されます。

(1)　name 属性（wsdl:operation 要素）

wsdl:operation 要素に含まれる name 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 指定できる値については，「20.2.1　NCName 型に指定できる値」を参照してください。

20.1.13　wsdl:input 要素（wsdl:binding 要素の孫要素の場合）

wsdl:input 要素のサポート範囲を説明します。

- wsdl:operation 要素の子要素として 1 個だけ記述できます。省略できません。省略または 2 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- 子要素として次に示す要素を指定できます。次に示す WSDL 要素以外の要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。cjwsimport コマンドで無視しない拡張要素のうち，次に示す要素以外の拡張要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51114-E）。
  - wsdl:documentation 要素

- soap:header 要素（SOAP 1.1 仕様の SOAP バインディングのための拡張要素）※
- soap:body 要素（SOAP 1.1 仕様の SOAP バインディングのための拡張要素）※
- soap12:header 要素（SOAP 1.2 仕様の SOAP バインディングのための拡張要素）※
- soap12:body 要素（SOAP 1.2 仕様の SOAP バインディングのための拡張要素）※

注※

soap:header 要素と soap12:header 要素，および soap:body 要素と soap12:header 要素は，それぞれどちらかを選択して指定してください。SOAP 1.1 仕様と SOAP 1.2 仕様の要素を両方または混在して指定することはできません。

- wsdl:input 要素（wsdl:binding 要素の孫要素の場合）の子要素は，上記の順序で指定してください。指定順序を誤った場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。ただし，次に示す要素の順序は入れ替わってもかまいません。
  - soap:header 要素または soap12:header 要素
  - soap:body 要素または soap12:body 要素
- name 属性を指定できます。name 属性以外の属性は指定できません。name 属性以外の属性を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

(1) name 属性（wsdl:input 要素）

wsdl:input 要素に含まれる name 属性のサポート範囲を説明します。

- 0 個または 1 個記述できます。2 個以上記述した場合は，XML Processor のエラーになります。
- 指定できる値については，「20.2.1　NCName 型に指定できる値」を参照してください。

20.1.14　wsdl:output 要素（wsdl:binding 要素の孫要素の場合）

wsdl:output 要素のサポート範囲を説明します。

- wsdl:operation 要素の子要素として 1 個だけ記述できます。省略または 2 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- 子要素として次に示す要素を指定できます。次に示す WSDL 要素以外の要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。cjwsimport コマンドで無視しない拡張要素のうち，次に示す拡張要素以外の拡張要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51119-E）。
  - wsdl:documentation 要素

- soap:header 要素（SOAP 1.1 仕様の SOAP バインディングのための拡張要素）※
- soap:body 要素（SOAP 1.1 仕様の SOAP バインディングのための拡張要素）※
- soap12:header 要素（SOAP 1.2 仕様の SOAP バインディングのための拡張要素）※
- soap12:body 要素（SOAP 1.2 仕様の SOAP バインディングのための拡張要素）※

注※

soap:header 要素と soap12:header 要素，および soap:body 要素と soap12:header 要素は，それぞれどちらかを選択して指定してください。SOAP 1.1 仕様と SOAP 1.2 仕様の要素を両方または混在して指定することはできません。

- wsdl:output 要素（wsdl:binding 要素の孫要素の場合）の子要素は，上記の順序で指定してください。指定順序を誤った場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。ただし，次に示す要素の順序は入れ替わってもかまいません。
  - soap:header 要素または soap12:header 要素
  - soap:body 要素または soap12:body 要素
- name 属性を指定できます。name 属性以外の属性は指定できません。name 属性以外の属性を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

(1) name 属性（wsdl:output 要素）

wsdl:output 要素に含まれる name 属性のサポート範囲を説明します。

- 0 個または 1 個記述できます。2 個以上記述した場合は，XML Processor のエラーになります。
- 指定できる値については，「20.2.1　NCName 型に指定できる値」を参照してください。

20.1.15　wsdl:fault 要素（wsdl:binding 要素の孫要素の場合）

wsdl:fault 要素のサポート範囲を説明します。

- wsdl:operation 要素の子要素として，0 個から 255 個まで記述できます。256 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- 子要素として次に示す要素を指定できます。次に示す WSDL 要素以外の要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。cjwsimport コマンドで無視しない拡張要素のうち，次に示す拡張要素以外の拡張要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51123-E）。
  - wsdl:documentation 要素

- soap:fault 要素（SOAP 1.1 仕様の SOAP バインディングのための拡張要素）※
- soap12:fault 要素（SOAP 1.2 仕様の SOAP バインディングのための拡張要素）※

注※

soap:fault 要素と soap12:fault 要素は，どちらかを選択して指定してください。両方の要素を指定することはできません。

- wsdl:fault 要素（wsdl:binding 要素の孫要素の場合）の子要素は，上記の順序で指定してください。指定順序を誤った場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- name 属性を指定できます。name 属性以外の属性は指定できません。name 属性以外の属性を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

(1) name 属性（wsdl:fault 要素）

wsdl:fault 要素に含まれる name 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 指定できる値については，「20.2.1　NCName 型に指定できる値」を参照してください。ただし，同じ wsdl:operation 要素以下にある，ほかの wsdl:fault 要素の name 属性と同じ値を指定できません。同じ値を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

20.1.16　wsdl:service 要素

wsdl:service 要素のサポート範囲を説明します。

- wsdl:definitions 要素の子要素として，1 個から 255 個まで記述できます。省略できません※。省略または 256 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51127-E）。

  注※

  WSDL のインポート機能を使用する場合は，インポート関係にあるすべての WSDL 内で 1 個あれば足りるので，この場合は省略できます。また，インポート関係にあるすべての WSDL 内での合計が上限になります。

- 子要素として次に示す要素を指定できます。次に示す WSDL 要素以外の要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。cjwsimport コマンドで無視しない拡張要素のうち，次に示す拡張要素以外の拡張

要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51129-E）。

- wsdl:documentation 要素
- jaxws:bindings 要素（JAX-WS 2.2 仕様のバインディング宣言）
- wsdl:port 要素

- wsdl:service 要素の子要素は，上記の順序で指定する必要があります。指定順序を誤った場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- name 属性を指定できます。name 属性以外の属性は指定できません。name 属性以外の属性を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

(1) name 属性（wsdl:service 要素）

wsdl:service 要素に含まれる name 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 指定できる値については，「15.1.9(2) サービス名およびポート名の条件」を参照してください。ただし，同じ wsdl:definitions 要素以下にある，ほかの wsdl:service 要素の name 属性と同じ値は指定できません。同じ値を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

20.1.17 wsdl:port 要素

wsdl:port 要素のサポート範囲を説明します。

- wsdl:service 要素の子要素として，1 個から 255 個まで記述できます。省略できません。省略または 256 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- 子要素として次に示す要素を指定できます。次に示す WSDL 要素以外の要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。cjwsimport コマンドで無視しない拡張要素のうち，次に示す要素以外の拡張要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51135-E）。
  - wsdl:documentation 要素
  - soap:address 要素（SOAP 1.1 仕様の SOAP バインディングのための拡張要素）※
  - soap12:address 要素（SOAP 1.2 仕様の SOAP バインディングのための拡張要素）※

- jaxws:bindings 要素（JAX-WS 2.2 仕様のバインディング宣言）
- wsaw:UsingAddressing 要素（WS-Addressing 1.0 仕様の拡張要素）

注※

soap:address 要素と soap12:address 要素は，どちらかを選択して指定してください。両方の要素を指定することはできません。

- wsdl:port 要素の子要素は，上記の順序で指定してください。指定順序を誤った場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。ただし，次に示す要素の順序は入れ替わってもかまいません。
  - soap:address 要素または soap12:address 要素
  - jaxws:bindings 要素
  - wsaw:UsingAddressing 要素
- 次の属性を指定できます。次の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
  - name 属性
  - binding 属性

(1) name 属性（wsdl:port 要素）

wsdl:port 要素に含まれる name 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 指定できる値については，「15.1.9(2) サービス名およびポート名の条件」を参照してください。ただし，同じ wsdl:service 要素以下にある，ほかの wsdl:port 要素の name 属性と同じ値は指定できません。同じ値を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

(2) binding 属性（wsdl:port 要素）

wsdl:port 要素に含まれる binding 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 参照できる wsdl:definitions 要素以下に宣言されている wsdl:binding を QName で指定してください。

20.1.18　wsdl:documentation 要素

wsdl:documentation 要素のサポート範囲を説明します。

- 次に示す要素の子要素として，0 個または 1 個記述できます。2 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。WSDL 要素と cjwsimport コマンドで無視しない拡張要素のうち，次の要素の子要素として記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
  - wsdl:definitions 要素
  - wsdl:import 要素
  - wsdl:types 要素
  - wsdl:message 要素
  - wsdl:part 要素
  - wsdl:portType 要素
  - wsdl:operation 要素
  - wsdl:input 要素
  - wsdl:output 要素
  - wsdl:fault 要素
  - wsdl:binding 要素
  - wsdl:service 要素
  - wsdl:port 要素
- 子要素として任意の要素を指定できます。
- 属性は指定できません。属性を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

20.1.19　soap:binding 要素

soap:binding 要素のサポート範囲を説明します。

- wsdl:binding 要素として 1 個だけ記述できます。省略できません。省略または 2 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51143-E）。
- 子要素は記述できません。記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- 次の属性を指定できます。次に示す属性以外を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

- transport 属性
- style 属性

(1) transport 属性（soap:binding 要素）

soap:binding 要素に含まれる transport 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 値には"http://schemas.xmlsoap.org/soap/http"を記述してください。"http://schemas.xmlsoap.org/soap/http"以外を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51147-E）。

(2) style 属性（soap:binding 要素）

soap:binding 要素に含まれる style 属性のサポート範囲を説明します。

- 0 個または 1 個記述できます。2 個以上記述した場合は，XML Processor のエラーになります。
- 値には"document"を記述してください。"document"以外を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

20.1.20 soap:operation 要素

soap:operation 要素のサポート範囲を説明します。

- wsdl:operation 要素の子要素として 1 個だけ記述できます。省略できません。省略または 2 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51150-E）。
- 子要素は記述できません。子要素を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- 次の属性を指定できます。次に示す属性以外を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
  - soapAction 属性
  - style 属性

(1) soapAction 属性（soap:operation 要素）

soap:operation 要素に含まれる soapAction 属性のサポート範囲を説明します。

- 0 個または 1 個記述できます。2 個以上記述した場合は，XML Processor のエラーになります。

- xsd:anyURI を満たす任意の文字列を指定できます。
- この属性に指定した値は無視されます。

(2) style 属性（soap:operation 要素）

soap:operation 要素に含まれる style 属性のサポート範囲を説明します。

- 0 個または 1 個記述できます。2 個以上記述した場合は，XML Processor のエラーになります。
- 値には"document"を記述してください。"document"以外を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

20.1.21 soap:body 要素

soap:body 要素のサポート範囲を説明します。

- wsdl:binding 要素の孫要素となる wsdl:input 要素および wsdl:output の子要素として 1 個だけ記述できます。省略できません。省略または 2 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51156-E）。
- 子要素は記述できません。子要素を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- 次の属性を指定できます。次の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。ただし，namespace 属性を指定した場合は，標準エラー出力とログに異なるエラーメッセージが出力されます（KDJW51208-E）。
  - use 属性
  - parts 属性

(1) use 属性（soap:body 要素）

soap:body 要素に含まれる use 属性のサポート範囲を説明します。

- 0 個または 1 個記述できます。2 個以上記述した場合は，XML Processor のエラーになります。
- 値には"literal"を記述してください。"literal"以外を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

(2) parts 属性（soap:body 要素）

soap:body 要素に含まれる parts 属性のサポート範囲を説明します。

- 0 個または 1 個記述できます。2 個以上記述した場合は，XML Processor のエラーになります。

- soap:body 要素に参照される wsdl:message 要素以下に宣言されている wsdl:part 要素を 0 個または 1 個指定してください。2 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- parts 属性を指定するときの注意事項については，「20.2.2　SOAP ボディおよび SOAP ヘッダと参照先の wsdl:part 要素の記述について」を参照してください。

20.1.22　soap:header 要素

soap:header 要素のサポート範囲を説明します。

- wsdl:binding 要素の孫要素となる wsdl:input 要素および wsdl:output 要素の子要素として記述できます。省略することもできます。
- 子要素は記述できません。子要素を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- 次の属性を指定できます。次の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。ただし，namespace 属性を指定した場合は，標準エラー出力とログに警告メッセージが出力され，cjwsimport コマンドの処理が続行されます（KDJW51009-W）。
  - message 属性
  - part 属性
  - use 属性
- soap:header 要素を指定するときの注意事項については，「20.2.2　SOAP ボディおよび SOAP ヘッダと参照先の wsdl:part 要素の記述について」を参照してください。

(1)　message 属性（soap:header 要素）

soap:header 要素に含まれる message 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 参照できる wsdl:definitions 要素以下に宣言されている wsdl:message を QName で指定してください。
- message 属性を指定するときの注意事項については，「20.2.2　SOAP ボディおよび SOAP ヘッダと参照先の wsdl:part 要素の記述について」を参照してください。

(2)　part 属性（soap:header 要素）

soap:header 要素に含まれる part 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- message 属性に指定された wsdl:message 要素以下に宣言されている wsdl:part 要素を指定してください。
- part 属性を指定するときの注意事項については，「20.2.2　SOAP ボディおよび SOAP ヘッダと参照先の wsdl:part 要素の記述について」を参照してください。

(3)　use 属性（soap:header 要素）

soap:header 要素に含まれる use 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 値には"literal"を記述してください。"literal"以外を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

20.1.23　soap:fault 要素

soap:fault 要素のサポート範囲を説明します。

- wsdl:binding 要素の孫要素となる wsdl:fault 要素の子要素として 1 個だけ記述できます。省略または 2 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51051-E）。
- 子要素は記述できません。子要素を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- 次の属性を指定できます。次の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。ただし，namespace 属性を指定した場合は，標準エラー出力とログに異なるエラーメッセージが出力されます（KDJW51210-E）。
  - name 属性
  - use 属性

(1)　name 属性（soap:fault 要素）

soap:fault 要素に含まれる name 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。

- 親要素である wsdl:fault 要素の name 属性と同じ値を記述してください。異なる値を記述した場合は，標準エラー出力とログに警告メッセージが出力され，cjwsimport コマンドの処理は続行されます（KDJW51027-W）。
- 指定できる値については，「20.2.1　NCName 型に指定できる値」を参照してください。

(2)　use 属性（soap:fault 要素）

soap:fault 要素に含まれる use 属性のサポート範囲を説明します。

- 0 個または 1 個記述できます。2 個以上記述した場合は，XML Processor のエラーになります。
- 値には"literal"を記述してください。"literal"以外を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

20.1.24　soap:address 要素

soap:address 要素のサポート範囲を説明します。

- wsdl:port 要素の子要素として 1 個だけ記述できます。省略できません。省略または 2 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51175-E）。
- 子要素は記述できません。子要素を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- location 属性を指定できます。location 属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

(1)　location 属性（soap:address 要素）

soap:address 要素に含まれる location 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 指定できる値については，「20.2.3　soap:address 要素または soap12:address 要素の location 属性に指定できる値」を参照してください。

20.1.25　soap12:operation 要素

soap12:operation 要素のサポート範囲を説明します。

- wsdl:operation 要素の子要素として 1 個だけ記述できます。省略できません。省略または 2 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51150-E）。
- 子要素は記述できません。子要素を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- 次の属性を指定できます。次の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。ただし，soapActionRequired 属性を指定した場合は，cjwsimport コマンドの処理は続行されます。
  - soapAction 属性
  - style 属性

(1) soapAction 属性（soap12:operation 要素）

soap12:operation 要素に含まれる soapAction 属性のサポート範囲を説明します。

- 0 個または 1 個記述できます。2 個以上記述した場合は，XML Processor のエラーになります。
- xsd:anyURI を満たす任意の文字列を指定できます。
- JAX-WS エンジンでは soapAction 属性の指定は無視されます。

(2) style 属性（soap12:operation 要素）

soap12:operation 要素に含まれる style 属性のサポート範囲を説明します。

- 0 個または 1 個記述できます。2 個以上記述した場合は，XML Processor のエラーになります。
- 値には"document"を記述してください。"document"以外を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）

20.1.26 soap12:binding 要素

soap12:binding 要素のサポート範囲を説明します。

- wsdl:binding 要素の子要素として 1 個だけ記述できます。省略できません。省略または 2 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51143-E）。
- 子要素は記述できません。子要素を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- 次の属性を指定できます。次の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
  - transport 属性

- style 属性

(1) transport 属性（soap12:binding 要素）

soap12:binding 要素に含まれる transport 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーとなります。
- 値には"http://schemas.xmlsoap.org/soap/http"または"http://www.w3.org/2003/05/soap/bindings/HTTP/"を記述してください。"http://schemas.xmlsoap.org/soap/http"または"http://www.w3.org/2003/05/soap/bindings/HTTP/"以外を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51147-E）。

(2) style 属性（soap12:binding 要素）

soap12:binding 要素に含まれる style 属性のサポート範囲を説明します。

- 0 個または 1 個記述できます。2 個以上記述した場合は，XML Processor のエラーになります。
- 値には"document"を記述してください。"document"以外を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

20.1.27 soap12:body 要素

soap12:body 要素のサポート範囲を説明します。

- wsdl:binding 要素の孫要素である wsdl:input 要素，および wsdl:output 要素の子要素として，1 個だけ記述できます。省略できません。省略または 2 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51156-E）。
- 子要素は記述できません。子要素を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- 次の属性を指定できます。次の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。ただし，namespace 属性を指定した場合は，標準エラー出力とログに異なるエラーメッセージが出力されます（KDJW51208-E）。
  - use 属性
  - parts 属性

(1) use 属性（soap12:body 要素）

soap12:body 要素に含まれる use 属性のサポート範囲を説明します。

- 0 個または 1 個記述できます。2 個以上記述した場合は，XML Processor のエラーになります。
- 値には"literal"を記述してください。"literal"以外を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

(2) parts 属性（soap12:body 要素）

soap12:body 要素に含まれる parts 属性のサポート範囲を説明します。

- 0 個または 1 個記述できます。2 個以上記述した場合は，XML Processor のエラーになります。
- soap12:body 要素が参照する wsdl:message 要素以下に宣言されている wsdl:part 要素を，0 個または 1 個記述できます。2 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- parts 属性を指定するときの注意事項については，「20.2.2 SOAP ボディおよび SOAP ヘッダと参照先の wsdl:part 要素の記述について」を参照してください。

20.1.28 soap12:header 要素

soap12:header 要素のサポート範囲を説明します。

- wsdl:binding 要素の孫要素である wsdl:input 要素および wsdl:output 要素の子要素として記述できます。省略することもできます。
- 子要素は記述できません。子要素を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- 次の属性を指定できます。次の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。ただし，namespace 属性を指定した場合は，標準エラー出力とログに警告メッセージが出力され，cjwsimport コマンドの処理は続行されます（KDJW51009-W）。
  - message 属性
  - part 属性
  - use 属性
- soap12:header 要素を指定するときの注意事項については，「20.2.2 SOAP ボディおよび SOAP ヘッダと参照先の wsdl:part 要素の記述について」を参照してください。

(1) message 属性（soap12:header 要素）

soap12:header 要素に含まれる message 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。

- 参照できる wsdl:definitions 要素以下に宣言されている wsdl:message を QName で指定してください。
- message 属性を指定するときの注意事項については，「20.2.2　SOAP ボディおよび SOAP ヘッダと参照先の wsdl:part 要素の記述について」を参照してください。

(2)　part 属性（soap12:header 要素）

soap12:header 要素に含まれる part 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- message 属性に指定された wsdl:message 要素以下に宣言されている wsdl:part 要素を指定してください。
- part 属性を指定するときの注意事項については，「20.2.2　SOAP ボディおよび SOAP ヘッダと参照先の wsdl:part 要素の記述について」を参照してください。

(3)　use 属性（soap12:header 要素）

soap12:header 要素に含まれる use 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 値には"literal"を記述してください。"literal"以外を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

20.1.29　soap12:fault 要素

soap12:fault 要素のサポート範囲を説明します。

- wsdl:binding 要素の孫要素となる wsdl:fault 要素の子要素として 1 個だけ記述できます。省略または 2 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51051-E）。
- 子要素は記述できません。子要素を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- 次の属性を指定できます。次の属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
  - name 属性
  - use 属性

(1)　name 属性（soap12:fault 要素）

soap12:fault 要素に含まれる name 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 親要素である wsdl:fault 要素の name 属性と同じ値を記述してください。異なる値を記述した場合は，標準エラー出力とログに警告メッセージが出力され，cjwsimport コマンドの処理は続行されます（KDJW51027-W）。
- 指定できる値については，「20.2.1　NCName 型に指定できる値」を参照してください。

(2)　use 属性（soap12:fault 要素）

soap12:fault 要素に含まれる use 属性のサポート範囲を説明します。

- 0 個または 1 個記述できます。2 個以上記述した場合は，XML Processor のエラーになります。
- 値には"literal"を記述してください。"literal"以外を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

20.1.30　soap12:address 要素

soap12:address 要素のサポート範囲を説明します。

- wsdl:port 要素の子要素として 1 個だけ記述できます。省略できません。省略または 2 個以上記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51175-E）。
- 子要素は記述できません。子要素を記述した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。
- location 属性を指定できます。location 属性以外を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

(1)　location 属性（soap12:address 要素）

soap12:address 要素に含まれる location 属性のサポート範囲を説明します。

- 1 個だけ記述できます。省略できません。省略した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。2 個以上記述した場合は，XML Processor のエラーになります。
- 指定できる値については，「20.2.3　soap:address 要素または soap12:address 要素の location 属性に指定できる値」を参照してください。

20.1.31　xsd:schema 要素

xsd:schema 要素のサポート範囲を説明します。

- JAXB 2.2 仕様に従い Java 型にマッピングしますが，マッピングは XML Processor にすべて委譲しています。

(1)　xmime:expectedContentTypes 属性（xsd:schema 要素）

WSDL のスキーマ宣言である xsd:element 要素の type 属性に"xsd:base64Binary"を指定している場合，xmime:expectedContentTypes 属性を用いて MIME タイプを明示的に指定することで，Base64 形式のデータを MIME タイプに対応した Java 型に関連づけることができます。WSDL のスキーマ宣言である xsd:element 要素に対する xmime:expectedContentTypes 属性の記述可否を次の表に示します。

表 20–1　xmime:expectedContentTypes 属性の記述可否

項番	xsd:element 要素を参照する wsdl:message のパラメタ	xsd:element 要素に対する xmime:expectedContentTypes 属性の記述可否
1	wsdl:input	○※1
2	wsdl:output	○※1
3	wsdl:fault	×※2

（凡例）

○：記述できます。

×：記述できません。

注※1

WSDL パートの種類が inout の場合，wsdl:input 要素から参照する xsd:element 要素の xmime:expectedContentTypes 属性の値と wsdl:output 要素から参照する xsd:element 要素の xmime:expectedContentTypes 属性の値には，同じ MIME タイプを指定してください。異なる MIME タイプを指定した場合，動作は保証されません。

注※2

フォルトメッセージに xmime:expectedContentTypes 属性が記述された xsd:element 要素を指定した場合，動作は保証されません。

xsd:element 要素に xmime:expectedContentTypes 属性を指定した場合の WSDL から Java 型へのマッピング方法は，WSDL の xmime:expectedContentTypes 属性がある xsd:base64Binary 型を Java 型へマッピングします。WSDL から Java 型へのマッピング例を次の図に示します。

図 20-1　WSDL から Java 型へのマッピング例

●WSDL定義

```
<wsdl:definitions targetNamespace="http://localhost"
  xmlns:intf="http://localhost"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xmime="http://www.w3.org/2005/05/xmlmime" ・・・・・・>
  <wsdl:types>
    <xsd:schema targetNamespace="http://localhost">
      <xsd:element name="setPhotoData" type="intf:setPhotoData"/>
      <xsd:complexType name="setPhotoData">
        <xsd:sequence>
          <xsd:element name="in0" type="xsd:base64Binary"
xmime:expectedContentTypes="image/jpeg"/>
        </xsd:sequence>
      </xsd:complexType>
       ・・・・・・
    </xsd:schema>
  </wsdl:types>
   ・・・・・・
```

●WSDLからマッピング後のSEI

```
@WebService(・・・・・)
public interface PhotoData {
    public java.lang.String setPhotoData(java.awt.Image in0)
        throws UserException;
}
```

xmime:expectedContentTypes 属性に指定する MIME タイプには，"text/xml"および"application/xml"の charset パラメタを除いて，パラメタを記述しないでください。"text/xml"および"application/xml"の charset パラメタ以外のパラメタを記述した場合，動作は保証されません。

WSDL からマッピングする Java 型は，xmime:expectedContentTypes 属性に指定されている MIME タイプによって変化します。xmime:expectedContentTypes 属性に記述されている MIME タイプと関連づける Java 型の関係を次の表に示します。

表 20-2　xmime:expectedContentTypes 属性の値と関連づける Java 型

項番	xmime:expectedContentTypes 属性の値（MIME タイプ）	関連づける Java 型
1	application/xml	javax.xml.transform.Source
2	image/png※1	java.awt.Image※2
3	image/jpeg※1	
4	上記の MIME タイプをコンマ区切りで記述（例：image/png, image/jpeg）※3	
5	image/*※3	
6	text/plain	java.lang.String
7	text/*※4	javax.activation.DataHandler
8	text/xml※5	javax.xml.transform.Source
9	上記以外※4, ※6	javax.activation.DataHandler

注※1

表中にない image タイプを Java 型に関連づける場合は，xmime:expectedContentTypes 属性に"application/octet-stream"を指定し，javax.activation.DataHandler クラスに関連づけます。

注※2

JAXB 仕様に従います。java.awt.Image クラスは Java SE 仕様でのグラフィカルイメージを表現する抽象クラスで，データ形式の規定はありません。この関連づけを使用して画像データをインスタンス化した場合，具象クラスのインスタンスには復号化した情報だけが保持されることがあります。そのため，JPEG 形式のように符号化するときに情報が削減される可能性のある画像を添付ファイルとして送信する場合，受信側のインスタンスが送信側のインスタンスや元のデータと異なることがあります。

画像を元の形式のまま扱いたい場合は，javax.activation.DataHandler にマッピングされる MIME タイプ（application/octet-stream など）を使用してください。

注※3

MIME タイプに"image/png，image/jpeg"など同一タイプを指定した場合や"image/*"を指定した場合，送信する SOAP メッセージの添付ファイルパートにある MIME ヘッダの Content-Type フィールドの値は，java.awt.Image 型を使用したときの初期値（"image/png"）です。

注※4

MIME タイプに"text/*"や表中にない MIME タイプを指定した場合，送信する SOAP メッセージの添付ファイルパートにある MIME ヘッダの Content-Type フィールドの値は，javax.activation.DataHandler 型を使用したときの初期値（javax.activation.DataHandler オブジェクトの MIME タイプ）です。

注※5

MIME タイプに"text/xml"を指定した場合，送信する SOAP メッセージの添付ファイルパートにある MIME ヘッダの Content-Type フィールドの値は，javax.xml.transform.Source 型を使用したときの初期値（"application/xml"）です。

注※6

異なるタイプ名の MIME タイプをコンマ区切りで記述した場合も含まれます（例：image/png, text/plain）。

xmime:expectedContentTypes 属性に指定されている MIME タイプは，cjwsimport コマンドで自動生成される JavaBean クラスのうち，xmime:expectedContentTypes 属性を記述した要素に対応する JavaBean クラスにアノテートされた javax.xml.bind.annotation.XmlMimeType アノテーションの値へマッピングされます。WSDL から自動生成された JavaBean クラスへのマッピングの例を次の図に示します。

図 20-2　WSDL から自動生成された JavaBean クラスへのマッピング

●WSDL定義

```
<wsdl:definitions targetNamespace="http://localhost"
  xmlns:intf="http://localhost"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xmime="http://www.w3.org/2005/05/xmlmime" ・・・・・・>
  <wsdl:types>
    <xsd:schema targetNamespace="http://localhost">
      <xsd:element name="setPhotoData" type="intf:setPhotoData"/>
      <xsd:complexType name="setPhotoData">
        <xsd:sequence>
          <xsd:element name="in0" type="xsd:base64Binary"
xmime:expectedContentTypes="image/jpeg"/>
        </xsd:sequence>
      </xsd:complexType>
       ・・・・・・
    </xsd:schema>
  </wsdl:types>
   ・・・・・・
```

●WSDLからマッピング後のSEI

```
 ・・・・・

@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "setPhotoData", propOrder = {
    "in0"
})
public class PutPhotoData {

    @XmlElement(required = true)
    @XmlMimeType("image/jpeg")
    protected Image in0;

 ・・・・・
}
```

(a)　名前空間"xmime"のインポート

WSDL から Java 型へのマッピングでは，名前空間"xmime"に存在する属性"xmime:expectedContentTypes"を使用しますが，JAX-WS では xsd:import 要素による名前空間"xmime"のインポートは不要です。

hwsgen コマンドで作成した WSDL を使用する場合，必要に応じて名前空間"xmime"をインポートする必要があります。名前空間"xmime"のインポート例を次に示します。

```
<wsdl:definitions targetNamespace="http://localhost"
  xmlns:intf="http://localhost"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xmime="http://www.w3.org/2005/05/xmlmime" ・・・・・・>
  <wsdl:types>
    <xsd:schema targetNamespace="http://localhost">
      <xsd:import namespace="http://www.w3.org/2005/05/xmlmime"/>
      <xsd:element name="setPhotoData" type="intf:setPhotoData"/>
      <xsd:complexType name="setPhotoData">
        <xsd:sequence>
          <xsd:element name="in0" type="xsd:base64Binary" xmime:expectedContentTypes="image/
jpeg"/>
        </xsd:sequence>
      </xsd:complexType>
```

```
      . . . . . .
    </xsd:schema>
  </wsdl:types>
    . . . . . .
```

20.2 WSDL 作成時の注意事項

WSDL 作成時の注意事項について説明します。

20.2.1 NCName 型に指定できる値

Application Server の JAX-WS 機能では，XML Schema 仕様の xsd:NCName 型の制限に違反しないかぎり，半角英数字（0～9，A～Z，a～z）およびアンダースコア（_）を使用できます。半角英数字およびアンダースコア以外の文字を使用した場合，動作は保証されません。

20.2.2 SOAP ボディおよび SOAP ヘッダと参照先の wsdl:part 要素の記述について

wsdl:input 要素と wsdl:output 要素の子要素として記述する SOAP ボディおよび SOAP ヘッダの記述と，SOAP ボディおよび SOAP ヘッダから参照される wsdl:part 要素の記述について説明します。

以降では，SOAP 1.1 仕様の場合を例に説明します。SOAP 1.2 仕様の場合は，名前空間や要素名，属性値などを読み替えてください。

(1) SOAP ヘッダを定義しない場合

soap:body 要素に parts 属性を記述する場合でも，記述しない場合でも，親要素となる wsdl:input 要素または wsdl:output 要素から参照される wsdl:message 要素の子要素には，wsdl:part 要素を 1 個だけ記述してください。

soap:header 要素を記述しない場合の記述例を次の図に示します。

図 20-3　soap:header 要素を記述しない場合の記述例

(2)　SOAP ヘッダを定義する場合

soap:header 要素を記述する場合，次に示す内容に従って定義してください。

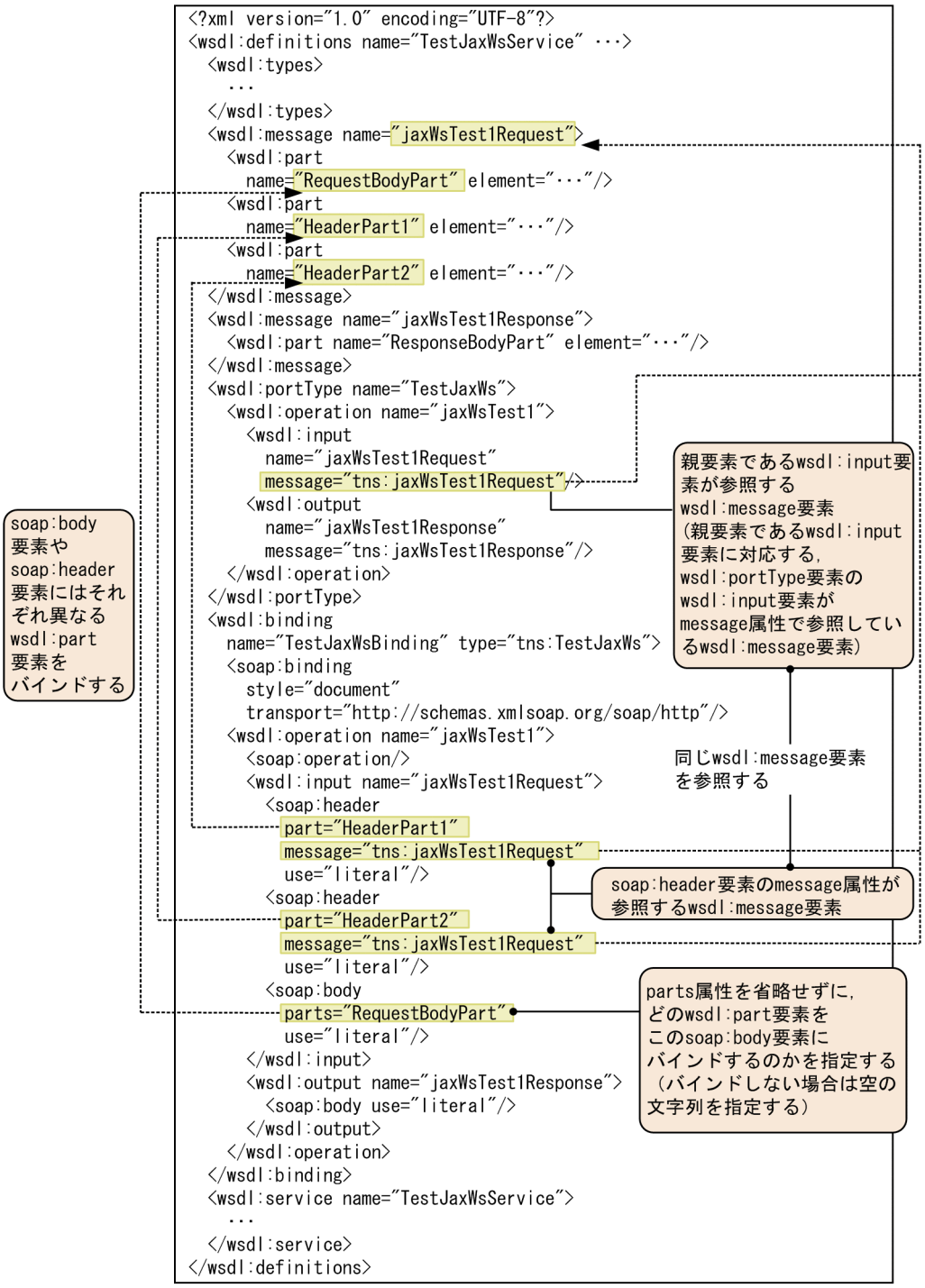
- soap:header 要素の message 属性には，親要素である wsdl:input 要素，または wsdl:output 要素から参照される wsdl:message 要素を指定してください。
- soap:header 要素の part 属性には，message 属性に指定した wsdl:message 要素以下に宣言されている wsdl:part 要素を指定してください。宣言されていない wsdl:part 要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます (KDJW51022-E)。
- soap:body 要素の parts 属性は省略できません。親要素となる wsdl:input 要素，または wsdl:output 要素から参照される wsdl:message 要素の複数の wsdl:part 子要素のうち，どの要素を soap:body 要素にバインドするのかを指定してください。バインドしない場合は，空の文字列を指定してください。

  soap:body 要素の parts 属性に参照されていない wsdl:message 要素の wsdl:part 要素，または存在しない wsdl:part 要素を指定した場合は，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51021-E)。また，soap:body 要素の parts 属性

を指定していない場合も，標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51179-E）。

soap:header 要素を記述する場合の記述例を次の図に示します。

図 20-4　soap:header 要素を記述する場合の記述例



20.2.3 soap:address 要素または soap12:address 要素の location 属性に指定できる値

soap:address 要素または soap12:address 要素の location 属性には，次に示す形式の URL を指定できます。

- <プロトコル※1>://<ホスト名※2>/<パス部分※4>
  （例）http://xxx.com/jaxws/service/UserInfoPort
- <プロトコル※1>://<ホスト名※2>:<ポート番号※3>/<パス部分※4>
  （例）http://xxx.com:80/jaxws/service/UserInfoPort

注※1

名前空間としては，http://または https://だけ指定できます。そのほかのプロトコルは指定できません。http://および https://以外のプロトコルを指定した場合，動作は保証されません。

注※2

RFC 2396 仕様の規則に従った文字列を指定できます。また，RFC 2732（IPv6 アドレス）仕様の規則に従った文字列も指定できます。

ただし，次の形式は指定できません。次の形式で指定した場合，動作は保証されません。

- クエリストリング （例）http://example.com/?a=b
- アンカー （例）http://example.com/index.html#anchor
- ポート番号 （例）http://example.com:8080/
- ユーザ名／パスワード （例）http://user:password@example.com
- パーセントエンコードされた文字 （例）http://%E4%BD%BF%E7%94%A8

注※3

半角数字 0～9 だけを使用した文字列を指定できます。そのほかの文字を指定した場合，動作は保証されません。

注※4

RFC 2396 仕様の規則に従った文字列を指定できます。また，パーセントエンコードされた文字（（例）http://%E4%BD%BF%E7%94%A8）も指定できます。

21 XML Catalogs 1.1 のサポート範囲

この章では，XML Catalogs 1.1 仕様のサポート範囲について説明します。

21.1 XML Catalogs 1.1 仕様のサポート範囲の一覧

XML Catalogs 1.1 仕様で定義されている，カタログファイルの構文のサポート範囲を次の表に示します。Application Server の JAX-WS 機能でサポートしていない構文で記述した場合の動作は保証されません。

表 21-1 カタログファイルの構文のサポート範囲の一覧（要素）

項番	要素名	サポート
1	er:catalog	○
2	er:group	×
3	er:public	○
4	er:system	○
5	er:rewriteSystem	×
6	er:systemSuffix	×
7	er:delegatePublic	×
8	er:delegateSystem	×
9	er:uri	×
10	er:rewriteURI	×
11	er:uriSuffix	×
12	er:delegateURI	×
13	er:nextCatalog	×

（凡例）

○：Application Server の JAX-WS 機能でサポートしています。

×：Application Server の JAX-WS 機能でサポートしていません。

表 21-2 カタログファイルの構文のサポート範囲の一覧（属性）

項番	要素名	属性名	サポート
1	er:catalog	prefer	○
		id	×
		xml:base	×
2	er:public	publicId	○
		uri	○
		id	×
		xml:base	×
3	er:system	systemId	○

項番	要素名	属性名	サポート
		uri	○
		id	×
		xml:base	×

（凡例）

○：Application Server の JAX-WS 機能でサポートしています。

×：Application Server の JAX-WS 機能でサポートしていません。

21.2 XML Catalogs 1.1 仕様のサポート範囲の詳細

XML Catalogs 1.1 仕様で定義されている，カタログファイルの構文のサポート範囲を要素ごとに説明します。

21.2.1 er:catalog 要素

- カタログファイルのルート要素として，1 個だけ記述できます。省略できません。省略した場合または 2 個以上記述した場合は警告メッセージ※が出力され，カタログ機能が無効な状態で処理が続行されます。
- 次の要素を子要素として指定できます。次の要素以外を記述した場合は警告メッセージ※が出力されます。
  - er:public 要素
  - er:system 要素

注※

カタログ機能を使用する場所によって，出力先と警告メッセージは次のように異なります。

- Web サービスクライアントの実行時：ログファイル（KDJW30023-W）
- Web サービスクライアントの開発時：標準エラー出力（KDJW51221-W）

(1) prefer 属性

- er:catalog 要素の属性として 0 個または 1 個記述できます。省略した場合の動作は，public を指定したときと同じになります。
- 次の値を属性値として指定できます。次の属性以外を記述した場合は警告メッセージ※が出力されます。
  - public
  - system
- WSDL または XML スキーマの xsd:import 要素に namespace 属性と schemalocation 属性が含まれる場合，prefer 属性に public を指定しても，er:public 要素によって xsd:import 要素の namespace 属性をマッピングすることはできません。

注※

カタログ機能を使用する場所によって，出力先と警告メッセージは次のように異なります。

- Web サービスクライアントの実行時：ログファイル（KDJW30023-W）
- Web サービスクライアントの開発時：標準エラー出力（KDJW51221-W）

21.2.2　er:public 要素

- publicId 属性に指定した XML スキーマの名前空間 URI を，uri 属性に指定した XML スキーマのロケーションを示す URI にマッピングします。
- er:catalog 要素の子要素として，0 個から 255 個まで記述できます。256 個以上記述した場合の動作は保証されません。

(1)　publicId 属性

- er:public 要素の属性として，1 個だけ記述できます。省略できません。省略した場合は警告メッセージ※が出力され，カタログ機能が無効な状態で処理が続行されます。
- 属性値は，xsd:import 要素の namespace 属性と同じ名前空間 URI を指定してください。

注※

カタログ機能を使用する場所によって，出力先と警告メッセージは次のように異なります。

- Web サービスクライアントの実行時：ログファイル（KDJW30023-W）
- Web サービスクライアントの開発時：標準エラー出力（KDJW51221-W）

(2)　uri 属性

- er:public 要素の属性として，1 個だけ記述できます。省略できません。省略した場合は警告メッセージ※1 が出力され，カタログ機能が無効な状態で処理が続行されます。
- 指定できる属性値の条件を次に示します。
  - マッピングする XML スキーマの相対パス※2 または URL を指定してください。指定する際は，マッピングする XML スキーマの対象の名前空間と，xsd:import 要素の namespace 属性の名前空間を一致させてください。名前空間が一致しない場合の動作は保証されません。
  - RFC 2396 で規定された xsd:anyURI を満たす文字で指定してください。また，RFC 2732（IPv6 アドレス）の規則に従った文字列も使用できます。
  - 大文字と小文字は区別されません。
  - 指定できる文字列の最大長の制限はありません。ただし，OS の制限を超えた場合は，エラーとなります。

注意

存在しない，またはアクセス権限のない WSDL または XML スキーマを指定した場合は警告メッセージ（KDJW30024-W）が出力され，カタログ機能が無効な状態で処理が続行されます。

注※1

カタログ機能を使用する場所によって，出力先と警告メッセージは次のように異なります。

- Web サービスクライアントの実行時：ログファイル（KDJW30023-W）
- Web サービスクライアントの開発時：標準エラー出力（KDJW51221-W）

注※2

カタログファイルを配置したディレクトリからの相対パスを示します。

21.2.3 er:system 要素

- systemId 属性に指定した WSDL または XML スキーマのロケーションを示す URI を，uri 属性に指定した別の WSDL または XML スキーマのロケーションを示す URI にマッピングします。
- er:catalog 要素の子要素として，0 個から 255 個まで記述できます。256 個以上記述した場合の動作は保証されません。

(1) systemId 属性

- er:system 要素の属性として，1 個だけ記述できます。省略できません。省略した場合は警告メッセージ※が出力され，カタログ機能が無効な状態で処理が続行されます。
- 指定できる属性値の条件を次に示します。
  - マッピングされる WSDL または XML スキーマの URL を，絶対 URL で指定してください。相対 URL および相対パスは指定できません。相対 URL および相対パスは，絶対 URL に変換して指定してください。
  - RFC 2396 で規定された xsd:anyURI を満たす文字で指定してください。また，RFC 2732（IPv6 アドレス）の規則に従った文字列も使用できます。
  - 大文字と小文字は区別されません。
  - 指定できる文字列の最大長の制限はありません。ただし，OS の制限を超えた場合は，エラーとなります。

注※

カタログ機能を使用する場所によって，出力先と警告メッセージは次のように異なります。

- Web サービスクライアントの実行時：ログファイル（KDJW30023-W）
- Web サービスクライアントの開発時：標準エラー出力（KDJW51221-W）

(2) uri 属性

- er:system 要素の属性として，1 個だけ記述できます。省略できません。省略した場合は警告メッセージ※1 が出力され，カタログ機能が無効な状態で処理が続行されます。
- 指定できる値の条件を次に示します。
  - マッピングする WSDL または XML スキーマの相対パス※2 または URL を指定してください。
  - RFC 2396 で規定された xsd:anyURI を満たす文字で指定してください。また，RFC 2732（IPv6 アドレス）の規則に従った文字列も使用できます。
  - 大文字と小文字は区別されません。

- 指定できる文字列の最大長の制限はありません。ただし，OS の制限を超えた場合は，エラーとなります。

注意

存在しない，またはアクセス権限のない WSDL または XML スキーマを指定した場合は警告メッセージ（KDJW30024-W）が出力され，カタログ機能が無効な状態で処理が続行されます。

注※1

カタログ機能を使用する場所によって，出力先と警告メッセージは次のように異なります。

- Web サービスクライアントの実行時：ログファイル（KDJW30023-W）
- Web サービスクライアントの開発時：標準エラー出力（KDJW51221-W）

注※2

カタログファイルを配置したディレクトリからの相対パスを示します。

22 SAAJ 仕様のサポート範囲

この章では，Web サービスを開発するときに注意が必要な，SAAJ 仕様のサポート範囲について説明します。

22.1　SAAJ 1.3 仕様のサポート範囲

ここでは，SAAJ 1.3 仕様のインタフェースおよびクラスのサポート範囲について説明します。また，ディスパッチベースの Web サービスクライアントで利用する場合の注意事項についても説明します。なお，SAAJ の各メソッドはスレッドセーフではありません。SAAJ のオブジェクトを複数スレッドで共有しないでください。共有した場合の動作は保証されません。

SAAJ 1.3 仕様のインタフェースのサポート範囲を次の表に示します。インタフェースについては，JDK のドキュメントを参照してください。

表 22–1　SAAJ 1.3 仕様のインタフェースのサポート範囲

項番	インタフェース名	メソッド名／フィールド名	サポート
1	Detail	addDetailEntry(Name name)	○
2		addDetailEntry(QName qname)	○
3		上記以外のメソッド	○
4	DetailEntry	メソッドなし	○
5	Name	すべてのメソッド	○
6	Node	getValue()	○
7		recycleNode()	○
8		setParentElement(SOAPElement parent)	○
9		setValue(String value)	○
10		上記以外のメソッド	○
11	SOAPBody	addBodyElement(Name name)	○
12		addBodyElement(QName qname)	○
13		addDocument(Document document)	○
14		addFault(Name faultCode, String faultString, Locale locale)	○
15		addFault(Name faultCode, String faultString)	○
16		addFault(QName faultCode, String faultString, Locale locale)	○
17		addFault(QName faultCode, String faultString)	○
18		上記以外のメソッド	○
19	SOAPBodyElement	メソッドなし	○
20	SOAPConstants	すべてのフィールド	○
21	SOAPElement	addAttribute(Name name, String value)	○
22		addAttribute(QName qname, String value)	○

項番	インタフェース名	メソッド名／フィールド名	サポート
23		addChildElement(Name name)	○
24		addChildElement(SOAPElement element)	○
25		addChildElement(String localName)	○
26		addChildElement(String localName, String prefix)	○
27		addChildElement(String localName, String prefix, String uri)	○
28		addChildElement(QName qname)	○
29		addNamespaceDeclaration(String prefix, String uri)	○
30		addTextNode(String text)	○
31		createQName(String localName, String prefix)	○
32		getAttributeValue(Name name)	○
33		getAttributeValue(QName qname)	○
34		getChildElements(Name name)	○
35		getChildElements(QName qname)	○
36		getEncodingStyle()	○
37		getNamespacePrefixes()	○
38		getNamespaceURI(String prefix)	○
39		removeAttribute(Name name)	○
40		removeAttribute(QName qname)	○
41		setElementQName(QName newName)	○
42		上記以外のメソッド	○
43	SOAPEnvelope	createName(String localName)	○
44		createName(String localName, String prefix, String uri)	○
45		上記以外のメソッド	○
46	SOAPFault	getFaultCode()	○
47		getFaultCodeAsName()	○
48		getFaultCodeAsQName()	○
49		getFaultString()	○
50		setFaultCode(Name faultCodeQName)	○
51		setFaultCode(QName faultCodeQName)	○
52		setFaultCode(String faultCode)	○
53		setFaultString(String faultString)	○

項番	インタフェース名	メソッド名／フィールド名	サポート
54		setFaultString(String faultString, Locale locale)	○
55		addFaultReasonText(String text, Locale locale)	○
56		getFaultReasonLocales()	○
57		getFaultReasonText(Locale locale)	○
58		getFaultReasonTexts()	○
59		getFaultStringLocale()	○
60		setFaultRole(String uri)	○
61		上記以外のメソッド	○
62	SOAPFaultElement	メソッドなし	○
63	SOAPHeader	addHeaderElement(Name name)	○
64		addHeaderElement(QName qname)	○
65		addUpgradeHeaderElement(String supportedSoapUri)	○
66		examineHeaderElements(String actor)	○
67		examineMustUnderstandHeaderElements(String actor)	○
68		extractHeaderElements(String actor)	○
69		上記以外のメソッド	○
70	SOAPHeaderElement	setActor(String actorURI)	○
71		setRole(String uri)	○
72		上記以外のメソッド	○
73	Text	すべてのメソッド	○

（凡例）

○：Application Server の JAX-WS 機能でサポートしています。

SAAJ 1.3 仕様のクラスのサポート範囲を次の表に示します。クラスについては，JDK のドキュメントを参照してください。

表 22–2　SAAJ 1.3 仕様のクラスのサポート範囲

項番	クラス名	メソッド名／フィールド名	サポート
1	AttachmentPart	addMimeHeader(String name, String value)	○
2		getAllMimeHeaders()	○
3		getContentLocation()	×
4		setContentLocation(String contentLocation)	×

項番	クラス名	メソッド名／フィールド名	サポート
5		setBase64Content(InputStream content, String contentType)	○
6		setContent(Object object, String contentType)	○
7		setContentId(String contentId)	○
8		setContentType(String contentType)	○
9		setMimeHeader(String name, String value)	○
10		setRawContent(InputStream content, String contentType)	○
11		setRawContentBytes(byte[] content, int offset, int len, String contentType)	○
12		上記以外のメソッド	○
13	MessageFactory	newInstance(String protocol)	○
14		上記以外のメソッド	○
15	MimeHeader	MimeHeader(String name, String value)コンストラクタ	○
16		上記以外のメソッド	○
17	MimeHeaders	addHeader(String name, String value)	○
18		setHeader(String name, String value)	○
19		上記以外のメソッド	○
20	SAAJMetaFactory	すべてのメソッド	○
21	SAAJResult	SAAJResult(SOAPMessage message)コンストラクタ	○
22		SAAJResult(SOAPElement rootNode)コンストラクタ	○
23		上記以外のメソッド	○
24	SOAPConnection	すべてのメソッド	○
25	SOAPConnectionFactory	すべてのメソッド	○
26	SOAPElementFactory※	すべてのメソッド	×
27	SOAPFactory	newInstance(String protocol)	○
28		createElement(Element domElement)	○
29		createElement(String localName, String prefix, String uri)	○
30		createFault(String reasonText, QName faultCode)	○
31		createName(String localName)	○
32		createName(String localName, String prefix, String uri)	○
33		上記以外のメソッド	○
34	SOAPMessage	addAttachmentPart(AttachmentPart AttachmentPart)	○

項番	クラス名	メソッド名／フィールド名	サポート
35		createAttachmentPart(Object content, String contentType)	○
36		getAttachment(SOAPElement element)	○
37		getAttachments(MimeHeaders headers)	×
38		getProperty(String property)	○
39		removeAttachments(MimeHeaders headers)	○
40		setContentDescription(String description)	○
41		setProperty(String property, Object value)	○
42		writeTo(OutputStream out)	○
43		上記以外のメソッド	○
44	SOAPPart	addMimeHeader(String name, String value)	○
45		getContentId()	×
46		getContentLocation()	×
47		getMimeHeader(String name)	○
48		setContent(Source source)	○
49		setContentId(String contentId)	×
50		setContentLocation(String contentLocation)	×
51		setMimeHeader(String name, String value)	○
52		上記以外のメソッド	○

（凡例）

○：Application Server の JAX-WS 機能でサポートしています。

×：Application Server の JAX-WS 機能でサポートしていません。

注※

非推奨クラスのため，使用した場合の動作は保証されません。

22.1.1 Detail インタフェース

Detail インタフェースのメソッドを使用する場合の注意事項を示します。

- addDetailEntry(Name name)メソッドおよび addDetailEntry(QName qname)メソッドの引数に null を指定しないでください。null を指定した場合の動作は保証されません。

22.1.2 Node インタフェース

Node インタフェースのメソッドを使用する場合の注意事項を示します。

- getValue()メソッドの対象ノードの子ノードの値は取得できません。対象ノードの子ノードの値を取得したい場合は，子ノードに対してメソッドを発行してください。
- 対象ノードに対して detachNode()を呼び出したことがなくても，recycleNode()メソッドを呼び出せます。メソッドを呼び出すと，detachNode()と同じ動作が実行されます。
- setParentElement(SOAPElement parent)メソッドでは，異なる DOM Document に属するノード同士を親子関係にできません。
- setValue(String value)メソッドの引数に null を指定すると，対象ノードの値に null が設定されます。

22.1.3 SOAPBody インタフェース

SOAPBody インタフェースのメソッドを使用する場合の注意事項を示します。

- 次に示すメソッドの引数には null を指定しないでください。null を指定した場合の動作は保証されません。
  - addBodyElement(Name name)
  - addBodyElement(QName qname)
  - addDocument(Document document)
  - addFault(Name faultCode, String faultString, Locale locale)
  - addFault(Name faultCode, String faultString)
  - addFault(QName faultCode, String faultString, Locale locale)
  - addFault(QName faultCode, String faultString)
- addFault(Name faultCode, String faultString, Locale locale)メソッドおよび addFault(QName faultCode, String faultString, Locale locale)メソッドの locale 引数には，null を指定しないでください。指定した場合の動作は保証されません。
- addFault(QName faultCode, String faultString, Locale locale)メソッドおよび addFault(QName faultCode, String faultString)メソッドの引数には，標準仕様で定義されているフォルトコードを指定してください。標準仕様で定義されていないフォルトコードを指定した場合の動作は保証されません。
- addFault(QName faultCode, String faultString, Locale locale)メソッドまたは addFault(QName faultCode, String faultString)メソッドの faultString 引数に空文字を設定して SOAP フォルトを送信した場合，受信した SOAP フォルトに対して getFaultString()メソッドおよび getFaultReasonTexts().next()を発行すると，null が取得されます。

22.1.4 SOAPElement インタフェース

SOAPElement インタフェースのメソッドを使用する場合の注意事項を示します。

- addAttribute(Name name, String value)メソッドの name 引数に null を指定しないでください。指定した場合の動作は保証されません。
- addAttribute(QName qname, String value)メソッドの qname 引数に null を指定しないでください。指定した場合の動作は保証されません。
- 次に示すメソッドの引数には null を指定しないでください。null を指定した場合の動作は保証されません。
  - addChildElement(Name name)
  - addChildElement(SOAPElement element)
  - addChildElement(QName qname)
  - addTextNode(String text)
  - getAttributeValue(Name name)
  - getAttributeValue(QName qname)
  - getChildElements(Name name)
  - getChildElements(QName qname)
  - removeAttribute(Name name)
  - removeAttribute(QName qname)
  - setElementQName(QName newName)
- 次に示すメソッドの localName 引数に null または空文字を指定しないでください。null または空文字を指定した場合の動作は保証されません。
  - addChildElement(String localName)
  - addChildElement(String localName, String prefix)
  - addChildElement(String localName, String prefix, String uri)
- addChildElement(String localName, String prefix, String uri)メソッドの uri 引数に null または空文字を指定しないでください。指定した場合の動作は保証されません。
- addNamespaceDeclaration(String prefix, String uri)メソッドの prefix 引数に null を指定しないでください。指定した場合の動作は保証されません。
- addNamespaceDeclaration(String prefix, String uri)メソッドの uri 引数に null または空文字を指定した場合，名前空間 URI が空文字の名前空間宣言が追加されます。
- createQName(String localName, String prefix)メソッドの localName 引数に null を指定しないでください。指定した場合の動作は保証されません。
- エンコードスタイルが設定されていない状態で getEncodingStyle()メソッドを発行した場合，null が返されます。

- デフォルト名前空間(xmlns="")に属する対象 SOAPElement に対して getNamespacePrefixes()メソッドを発行した場合，戻り値にデフォルト名前空間のプレフィクスは含まれません。
- getNamespaceURI(String prefix)メソッドで引数に指定したプレフィクスが対象の SOAPElement に宣言されていない場合，または引数に null または空文字を指定した場合，null が返されます。

22.1.5 SOAPEnvelope インタフェース

SOAPEnvelope インタフェースのメソッドを使用する場合の注意事項を示します。

- createName(String localName)メソッドおよび createName(String localName, String prefix, String uri)メソッドの localName 引数に null または空文字を指定しないでください。指定した場合の動作は保証されません。
- createName(String localName, String prefix, String uri)メソッドの uri 引数に null または空文字を指定した場合，名前空間 URI が空文字の Name オブジェクトが生成されます。

22.1.6 SOAPFault インタフェース

SOAPFault インタフェースのメソッドを使用する場合の注意事項を示します。

- フォルトコードが明示的に設定されていない SOAPFault オブジェクトに対して次のメソッドを発行した場合，Application Server の JAX-WS 機能が自動的に設定した値が返されます。
  - getFaultCode()
  - getFaultCodeAsName()
  - getFaultCodeAsQName()
- フォルトコードが明示的に設定されていない SOAPFault オブジェクトに対して次のメソッドを発行した場合，Application Server の JAX-WS 機能が自動的に設定したフォルト文字列"Fault string, and possibly fault code, not set"が返されます。
  - getFaultString()
  - getFaultReasonTexts()
- SOAP 1.1 形式の場合に，フォルトコードが明示的に設定されていない SOAPFault オブジェクトに対して getFaultStringLocale()メソッドを発行した場合，null が返されます。
- SOAP 1.2 形式の場合に，フォルトコードが明示的に設定されていない SOAPFault オブジェクトに対して getFaultReasonLocales()メソッドまたは getFaultStringLocale()メソッドを発行した場合，Application Server の JAX-WS 機能が自動的に設定した値が返されます。
- 次に示すメソッドの引数には null を指定しないでください。null を指定した場合の動作は保証されません。
  - setFaultCode(Name faultCodeQName)

  - setFaultCode(QName faultCodeQName)
  - setFaultCode(String faultCode)
  - setFaultString(String faultString)
  - setFaultString(String faultString, Locale locale)
  - setFaultRole(String uri)
- 次に示すメソッドの引数には，名前空間で修飾されたフォルトコードを指定してください。名前空間で修飾されていない，ローカル名だけのフォルトコードを指定した場合の動作は保証されません。
  - setFaultCode(Name faultCodeQName)
  - setFaultCode(QName faultCodeQName)
  - setFaultCode(String faultCode)
- setFaultString(String faultString, Locale locale)メソッドの locale 引数に null を指定しないでください。指定した場合の動作は保証されません。
- addFaultReasonText(String text, Locale locale)メソッドの text 引数に null を指定しないでください。指定した場合の動作は保証されません。
- getFaultReasonText(Locale locale)メソッドの引数に null を指定した場合，null が返されます。
- getFaultReasonLocales()メソッドまたは getFaultStringLocale()メソッドが取得したロケールは，受信した SOAP フォルトの xml:lang 属性に設定されているロケールと異なる場合があります。
- setFaultRole(String uri)メソッドの uri 引数には，URI 形式の文字列を指定してください。URI 形式以外の文字列を指定した場合の動作は保証されません。

22.1.7 SOAPHeader インタフェース

SOAPHeader インタフェースのメソッドを使用する場合の注意事項を示します。

- addHeaderElement(Name name)メソッドおよび addHeaderElement(QName qname)メソッドの引数に null は指定しないでください。null を指定した場合の動作は保証されません。
- addUpgradeHeaderElement(String supportedSoapUri)メソッドの supportedSoapUri 引数に null を指定しないでください。指定した場合の動作は保証されません。
- examineHeaderElements(String actor)メソッドおよび extractHeaderElements(String actor)メソッドの actor 引数に null を指定しないでください。指定した場合の動作は保証されません。
- examineMustUnderstandHeaderElements(String actor)メソッドの actor 引数に null を指定した場合，設定されているすべての SOAPHeaderElement を含む Iterator が返されます。

22.1.8 SOAPHeaderElement インタフェース

SOAPHeaderElement インタフェースのメソッドを使用する場合の注意事項を示します。

- setActor(String actorURI)メソッドの actorURI 引数に null を指定した場合，SOAP 1.1 形式の場合は値が空文字の actor 属性が，SOAP 1.2 形式の場合は値が空文字の role 属性が設定されます。
- SOAP 1.2 形式の場合，setRole(String uri)メソッドの uri 引数に null を指定すると，値が空文字の role 属性が設定されます。

22.1.9 AttachmentPart クラス

AttachmentPart クラスのメソッドを使用する場合の注意事項を示します。

- addMimeHeader(String name, String value)メソッドまたは setMimeHeader(String name, String value)メソッドで MIME ヘッダを設定しても，送受信時の SOAP メッセージ上に MIME ヘッダは現れません。
- addMimeHeader(String name, String value)メソッドまたは setMimeHeader(String name, String value)メソッドの value 引数に指定した値は，MIME ヘッダの値に設定されます。
- getAllMimeHeaders()メソッドでは Content-Transfer-Encoding ヘッダを取得できません。Application Server の JAX-WS 機能では，添付ファイルは常にバイナリ形式で送信されるため，AttachmentPart の Content-Transfer-Encoding ヘッダの値は binary になります。
- getContentLocation()メソッドを使用した場合の動作は保証されません。
- setContentLocation(String contentLocation)メソッドを使用した場合の動作は保証されません。Content-Location ヘッダの代わりに AttachmentPart#setContentId メソッドで設定した Content-Id ヘッダを使用する必要があります。
- 次に示すメソッドの contentType 引数に指定した値が Content-Type ヘッダの値に設定されます。そのため，添付ファイルの型に適合する MIME タイプを指定する必要があります。不正な MIME タイプを指定した場合の動作は保証されません。
  - setBase64Content(InputStream content, String contentType)
  - setContentType(String contentType)
  - setRawContent(InputStream content, String contentType)
  - setRawContentBytes(byte[] content, int offset, int len, String contentType)
- setContent(Object object, String contentType)メソッドの contentType 引数に null または標準仕様で定義されていない MIME タイプを指定しないでください。指定した場合の動作は保証されません。
- setContent(Object object, String contentType)メソッドの第 1 引数には，第 2 引数に指定した MIME タイプに適合するオブジェクトを指定してください。適合しないオブジェクトを指定した場合の動作は保証されません。また，null を指定した場合の動作も保証されません。

- setContentId(String contentId)メソッドの contentId 引数に null または空文字を指定した場合は，その値が Content-Id ヘッダの値に設定されます。
- setRawContentBytes(byte[] content, int offset, int len, String contentType)メソッドの offset 引数には正しいオフセットを，len 引数には正しいサイズを指定してください。不正な値を指定した場合の動作は保証されません。

22.1.10　MessageFactory クラス

MessageFactory クラスのメソッドを使用する場合の注意事項を示します。

- newInstance(String protocol)メソッドで DYNAMIC_SOAP_PROTOCOL は指定しないでください。指定した場合の動作は保証されません。

22.1.11　MimeHeader クラス

MimeHeader クラスのメソッドを使用する場合の注意事項を示します。

- RFC 822 や RFC 2045 などで，MIME ヘッダでは使用できないと定義されている文字は，MimeHeader(String name, String value)コンストラクタの引数に指定しないでください。指定した場合の動作は保証されません。

22.1.12　MimeHeaders クラス

MimeHeaders クラスのメソッドを使用する場合の注意事項を示します。

- addHeader(String name, String value)メソッドおよび setHeader(String name, String value)メソッドの value 引数に null を指定しないでください。指定した場合の動作は保証されません。
- addHeader(String name, String value)メソッドまたは setHeader(String name, String value)メソッドで Content-Length ヘッダまたは Content-Type ヘッダの値を設定しても，送受信時に値が上書きされます。
- SOAP ハンドラ内で SOAPMessage#getMimeHeaders()メソッドから取得した MimeHeaders オブジェクトは，HTTP ヘッダの取得および更新ができません。メッセージコンテキストを使用することにより HTTP ヘッダを取得することはできます。メッセージコンテキストについては，「19.2.5　メッセージコンテキストの使用」を参照してください。

22.1.13　SAAJResult クラス

SAAJResult クラスのメソッドを使用する場合の注意事項を示します。

- SAAJResult(SOAPMessage message)コンストラクタの引数に null は指定しないでください。null を指定した場合の動作は保証されません。
- SAAJResult(SOAPElement rootNode)コンストラクタの rootNode 引数に null を指定しないでください。指定した場合の動作は保証されません。

22.1.14 SOAPFactory クラス

SOAPFactory クラスのメソッドを使用する場合の注意事項を示します。

- newInstance(String protocol)メソッドで DYNAMIC_SOAP_PROTOCOL は指定しないでください。指定した場合の動作は保証されません。
- createElement(Element domElement)メソッドの引数に null を指定した場合，null が返されます。
- createElement(String localName, String prefix, String uri)メソッドの localName 引数に空文字を指定しないでください。指定した場合の動作は保証されません。
- createElement(String localName, String prefix, String uri)メソッドの prefix 引数に null または空文字を指定した場合，プレフィクスが null の SOAPElement オブジェクトが生成されます。
- createElement(String localName, String prefix, String uri)メソッドの uri 引数に null を指定しないでください。指定した場合の動作は保証されません。また，uri 引数に空文字を指定した場合は，名前空間 URI が null の SOAPElement オブジェクトが生成されます。
- createFault(String reasonText, QName faultCode)メソッドの引数に null は指定しないでください。null を指定した場合の動作は保証されません。
- createFault(String reasonText, QName faultCode)メソッドの faultCode 引数に標準仕様で定義されていないフォルトコードを指定しないでください。指定した場合の動作は保証されません。
- createName(String localName)メソッドまたは createName(String localName, String prefix, String uri)メソッドの localName 引数に空文字を指定しないでください。指定した場合の動作は保証されません。
- createName(String localName, String prefix, String uri)メソッドの uri 引数に null または空文字を指定した場合，名前空間 URI が空文字の Name オブジェクトが生成されます。

22.1.15 SOAPMessage クラス

SOAPMessage クラスのメソッドを使用する場合の注意事項を示します。

- 次に示すメソッドの引数に null は指定しないでください。null を指定した場合の動作は保証されません。
  - addAttachmentPart(AttachmentPart AttachmentPart)
  - getAttachment(SOAPElement element)
  - writeTo(OutputStream out)

- addAttachmentPart(AttachmentPart AttachmentPart)メソッドの AttachmentPart 引数に空の AttachmentPart オブジェクトを指定しないでください。指定した場合の動作は保証されません。
- createAttachmentPart(Object content, String contentType)メソッドの第 1 引数には，第 2 引数に指定した MIME タイプに適合するオブジェクトを指定してください。適合しないオブジェクトを指定した場合の動作は保証されません。また，null を指定した場合の動作も保証されません。
- getAttachment(SOAPElement element)メソッドの引数に指定した要素の値や href 属性から AttachmentPart を参照する場合は，存在する AttachmentPart を示す CID URL スキーム（RFC 2392 規定）を記述してください。存在しない AttachmentPart を示す CID URL スキームを記述した場合の動作は保証されません。
- getAttachments(MimeHeaders headers)メソッドを使用した場合の動作は保証されません。SOAPMessage#getAttachments()メソッドまたは getAttachment(SOAPElement)メソッドで AttachmentPart を取得する必要があります。
- getProperty(String property)メソッドの引数に null を指定した場合，null が返されます。
- Application Server の JAX-WS 機能が対応している SOAP メッセージの文字エンコードは，utf-8 だけです。ただし，getProperty(String property)メソッドの引数に SOAPMessage.CHARACTER_SET_ENCODING を指定してプロパティ値を取得する場合，null が返される場合があります。
- SOAPMessage#setProperty メソッドで対象プロパティを設定していない場合，getProperty(String property)メソッドには null が返されます。
- removeAttachments(MimeHeaders headers)メソッドの引数に null を指定した場合，AttachmentPart はすべて削除されます。
- setContentDescription(String description)メソッドの引数に null を指定した場合，Content-Description ヘッダには値が設定されません。空文字を指定した場合は，Content-Description ヘッダに空文字が設定されます。
- setProperty(String property, Object value)メソッドの property 引数には SOAPMessage.CHARACTER_SET_ENCODING または SOAPMessage.WRITE_XML_DECLARATION を指定してください。SOAPMessage.CHARACTER_SET_ENCODING または SOAPMessage.WRITE_XML_DECLARATION 以外のプロパティを指定しても，無視されます。
- setProperty(String property, Object value)メソッドの property 引数に SOAPMessage.CHARACTER_SET_ENCODING を指定する場合は，value 引数に utf-8 を指定してください。utf-8 以外の値を指定した場合の動作は保証されません。
- setProperty(String property, Object value)メソッドの property 引数に SOAPMessage.WRITE_XML_DECLARATION を指定する場合は，"true"または"false"を指定してください。"true"または"false"以外を指定した場合の動作は保証されません。
- setProperty(String property, Object value)メソッドの property 引数に null を指定しないでください。指定した場合の動作は保証されません。

- Dispatch／Provider で SOAP メッセージを送受信する場合，setProperty(String property, Object value)メソッドではプロパティを設定できません。プロパティを設定したい場合は，SOAPConnection で SOAP メッセージを送受信してください。

22.1.16　SOAPPart クラス

SOAPPart クラスのメソッドを使用する場合の注意事項を示します。

- addMimeHeader(String name, String value)メソッドまたは setMimeHeader(String name, String value)メソッドの value 引数に指定した値が MIME ヘッダの値に設定されます。
- addMimeHeader(String name, String value)メソッドまたは setMimeHeader(String name, String value)メソッドで SOAPPart に MIME ヘッダを設定しても，送受信時の SOAP メッセージ上に MIME ヘッダは現れません。
- getMimeHeader(String name)メソッドの引数に SOAPPart オブジェクトに設定されていない MIME ヘッダ名や null を指定した場合，null が返されます。
- setContent(Source source)メソッドの引数には XML としても SOAP としても正しい内容の Source オブジェクトを指定してください。不正な内容の Source オブジェクトを指定した場合の動作は保証されません。また，null を指定した場合の動作も保証されません。
- 次のメソッドを使用した場合の動作は保証されません。
  - getContentId()
  - getContentLocation()
  - setContentId(String contentId)
  - setContentLocation(String contentLocation)

22.1.17　添付ファイルを使用する場合のサポート範囲

プロバイダ実装クラスを使用して開発した Web サービス，またはディスパッチベースの Web サービスクライアントでは，SAAJ 仕様に従って添付ファイル付きの SOAP メッセージを生成，送受信できます。一度に送受信できる添付ファイルのサイズおよび個数は，実行環境のメモリ量によって変わりますが，制限はありません。メモリ量を増やせば，容量の大きい添付ファイルや多量の添付ファイルを一度に送受信することもできます。

添付ファイルを送受信するときのメモリの使用量については，「付録 C.3　添付ファイル使用時の 1 リクエスト当たりのメモリ使用量」を参照してください。

(1)　MIME タイプ

添付ファイルに対応する MIME タイプは，添付ファイルの拡張子によって決まります。添付ファイルの MIME タイプを明示しない場合，添付ファイルの拡張子によって自動的に適切な MIME タイプが設定され

ます。AttachmentPart#setContentType()メソッドなどによって MIME タイプを明示する場合は，添付ファイルの拡張子に対応した適切な MIME タイプを指定する必要があります。不正な MIME タイプを指定した場合の動作は保証されません。

添付ファイルの拡張子と MIME タイプの適切な組み合わせを次の表に示します。次の表に示した拡張子以外の場合，MIME タイプは application/octet-stream となります。

表 22–3　添付ファイルの拡張子と MIME タイプ

項番	添付ファイルの拡張子	対応する MIME タイプ
1	html，htm	text/html
2	txt，text	text/plain
3	gif，GIF	image/gif
4	ief	image/ief
5	jpeg，jpg，jpe，JPG	image/jpeg
6	tiff，tif	image/tiff
7	xwd	image/x-xwindowdump
8	ai，eps，ps	application/postscript
9	rtf	application/rtf
10	tex	application/x-tex
11	texinfo，texi	application/x-texinfo
12	t，tr，roff	application/x-troff
13	au	audio/basic
14	midi，mid	audio/midi
15	aifc	audio/x-aifc
16	aif，aiff	audio/x-aiff
17	wav	audio/x-wav
18	mpeg，mpg，mpe	video/mpeg
19	qt，mov	video/quicktime
20	avi	video/x-msvideo

(2)　添付ファイルを読み込む場合の注意事項

ファイルを読み込んで添付ファイルとして送受信する場合は，java.io.FileInputStream で読み込んだオブジェクトではなく，javax.activation.FileDataSource で読み込んだオブジェクトを添付ファイルに設定する必要があります。例を次に示します。

```
AttachmentPart apPart = request.createAttachmentPart();
FileDataSource source = new FileDataSource("D:¥¥attachment.txt");
apPart.setDataHandler(new DataHandler(source));
request.addAttachmentPart(apPart);
```

java.io.FileInputStream で読み込んだオブジェクトを設定した場合の動作は保証されません。

(3) DOM API を使用する場合の注意事項

DOM API を使用して SOAP メッセージを作成する場合は，次に示すメソッドは使用しないでください。使用した場合の動作は保証されません。

- org.w3c.dom.createEntityReference(String name)
- org.w3c.dom.createProcessingInstruction(String target, String data)

(4) 複数のファイルを添付する場合の注意事項

複数の添付ファイルを一度に送信する場合は，それぞれの AttachmentPart オブジェクトにユニークな Content-ID を設定する必要があります。Content-ID を設定しない，または重複する Content-ID を設定して複数の添付ファイルを送信しようとした場合，最後に設定した添付ファイルだけが送信されます。

複数の AttachmentPart オブジェクトにユニークな Content-ID を設定する例を次に示します。

```
AttachmentPart apPart1 = request.createAttachmentPart();
FileDataSource source1 = new FileDataSource("D:¥¥attachment1.txt");
apPart1.setDataHandler(new DataHandler(source1));
apPart1.setContentId("001");
request.addAttachmentPart(apPart1);

AttachmentPart apPart2 = request.createAttachmentPart();
FileDataSource source2 = new FileDataSource("D:¥¥attachment2.txt");
apPart2.setDataHandler(new DataHandler(source2));
apPart2.setContentId("002");
request.addAttachmentPart(apPart2);

AttachmentPart apPart3 = request.createAttachmentPart();
FileDataSource source3 = new FileDataSource("D:¥¥attachment3.txt");
apPart3.setDataHandler(new DataHandler(source3));
apPart3.setContentId("003");
request.addAttachmentPart(apPart3);
```

23 WS-RM 仕様のサポート範囲

この章では，WS-RM 仕様のサポート範囲について説明します。WS-RM 1.2 機能については，「34.　WS-RM 1.2 機能」を参照してください。

23.1　WS-RM 1.2 仕様のサポート範囲

WS-RM 1.2 仕様のサポート範囲を次の表に示します。なお，表中の大分類は WS-RM 1.2 仕様の該当個所（章節項）を，小分類は WS-RM 1.2 仕様の該当個所に記載されている内容を示します。

表 23–1　WS-RM 1.2 仕様のサポート範囲

分類				サポート
大分類	小分類			
2.4	送達保証	AtLeastOnce		×
		AtMostOnce		×
		ExactlyOnce		○
		InOrder		×
3	RM 要素			○
3.1	拡張要素／拡張属性の考慮※1			○
3.2	Piggy-Backing			○
3.3	WS-Addressing の利用			○
3.4	シーケンス生成			○
3.4	シーケンス生成リクエスト	wsrm:CreateSequence		○
		wsrm:AcksTo※2		△
		wsrm:Expires※3		×
		wsrm:Offer		○
		拡張要素／拡張属性※1		○
3.4	シーケンス生成レスポンス	wsrm:CreateSequenceResponse		○
		wsrm:Identifier		○
		wsrm:Expires※3		×
		wsrm:IncompleteSequenceBehavior	DiscardEntireSequence	×
			DiscardFollowingFirstGap	×
			NoDiscard	○
		wsrm:Accept		○
		拡張要素／拡張属性※1		○
3.5	シーケンスクローズ			○
3.5	シーケンスクローズリクエスト	wsrm:CloseSequence		○

分類			サポート
大分類	小分類		
		wsrm:Identifier	○
		wsrm:LastMsgNumber	○
		拡張要素／拡張属性※1	○
3.5	シーケンスクローズレスポンス	wsrm:CloseSequenceResponse	○
		wsrm:Identifier	○
		拡張要素／拡張属性※1	○
3.6	シーケンス終了		○
3.6	シーケンス終了リクエスト	wsrm:TerminateSequence	○
		wsrm:Identifier	○
		wsrm:LastMsgNumber	○
		拡張要素／拡張属性※1	○
3.6	シーケンス終了レスポンス	wsrm:TerminateSequenceResponse	○
		wsrm:Identifier	○
		拡張要素／拡張属性※1	○
3.7	シーケンス		○
	シーケンス要素	wsrm:Sequence	○
		wsrm:Identifier	○
		wsrm:MessageNumber	○
3.8	Ack リクエスト		○
3.8	Ack リクエスト要素	wsrm:AckRequested	○
		wsrm:Identifier	○
		拡張要素／拡張属性※1	○
3.9	Ack		○
3.9	Ack 要素	wsrm:SequenceAcknowledgement	○
		wsrm:Identifier	○
		wsrm:AcknowledgementRange	○
		wsrm:None※4	△
		wsrm:Final	○
		wsrm:Nack	×

分類			サポート
大分類	小分類		
		拡張要素／拡張属性※1	○
4	フォルト		○
4	SOAP 1.1 対応		○
4	SOAP 1.2 対応		○
4.1	wsrm:SequenceFault フォルト		○
4.2	wsrm:SequenceTerminated フォルト		○
4.3	wsrm:UnknownSequence フォルト		○
4.4	wsrm:InvalidAcknowledgement フォルト		○
4.5	wsrm:MessageNumberRollover フォルト		○
4.6	wsrm:CreateSequenceRefused フォルト		○
4.7	wsrm:SequenceClosed フォルト		○
4.8	wsrm:WSRMRequired フォルト		×
5	セキュリティの脅威と対策		×
6	セキュアなシーケンス		×

（凡例）

○：Application Server の WS-RM 1.2 機能でサポートしています。

×：Application Server の WS-RM 1.2 機能でサポートしていません。

△：Application Server の WS-RM 1.2 機能でサポートしていますが，一部制限があります。

注※1

Application Server の WS-RM 1.2 機能では，拡張要素および拡張属性を付加しません。受信メッセージに含まれる拡張要素および拡張属性は無視されます。

注※2

使用できる要素値は，匿名 URI だけです。

注※3

wsrm:Expires 要素によるシーケンス有効期限の設定はサポートしていません。シーケンスの有効期限は，WSDL に net35rmpInactivityTimeout を指定して設定します。

注※4

Application Server の WS-RM 1.2 機能では，wsrm:None 要素を送信しません。返す Ack がない場合，HTTP ステータスコード 202 を返します。受信メッセージに含まれる場合は正常に処理されます。

23.2　WS-RM Policy 1.2 仕様のサポート範囲

WS-RM Policy 1.2 仕様のサポート範囲を次の表に示します。なお，表中の大分類は WS-RM Policy 1.2 仕様の該当個所（章節項）を，小分類は WS-RM Policy 1.2 仕様の該当個所に記載されている内容を示します。

表 23–2　WS-RM Policy 1.2 仕様のサポート範囲

分類			サポート
大分類	小分類		
2.2	アサーション要素	/wsrmp:RMAssertion	○
		/wsrmp:RMAssertion/@wsp:Optional	×
		/wsrmp:RMAssertion/wsp:Policy	○
		/wsrmp:RMAssertion/wsp:Policy/wsrmp:SequenceSTR	×
		/wsrmp:RMAssertion/wsp:Policy/ wsrmp:SequenceTransportSecurity	×
		/wsrmp:RMAssertion/wsp:Policy/wsrmp:DeliveryAssurance	○
		/wsrmp:RMAssertion/wsp:Policy/wsrmp:DeliveryAssurance/ wsp:Policy	○
		/wsrmp:RMAssertion/wsp:Policy/wsrmp:DeliveryAssurance/ wsp:Policy/wsrmp:ExactlyOnce	○
		/wsrmp:RMAssertion/wsp:Policy/wsrmp:DeliveryAssurance/ wsp:Policy/wsrmp:AtLeastOnce	×
		/wsrmp:RMAssertion/wsp:Policy/wsrmp:DeliveryAssurance/ wsp:Policy/wsrmp:AtMostOnce	×
		/wsrmp:RMAssertion/wsp:Policy/wsrmp:DeliveryAssurance/ wsp:Policy/wsrmp:InOrder	×
		拡張要素/拡張属性※	○
2.3	アサーション添付	/wsdl:definitions/wsdl:service/wsdl:port	×
		/wsdl:definitions/wsdl:binding	○
		/wsdl:definitions/wsdl:binding/wsdl:operation/wsdl:input	×
		/wsdl:definitions/wsdl:binding/wsdl:operation/wsdl:output	×
		/wsdl:definitions/wsdl:binding/wsdl:operation/wsdl:fault	×
2.5	シーケンスセキュリティポリシー		×

（凡例）

○：Application Server の WS-RM Policy 1.2 機能でサポートしています。

×：Application Server の WS-RM Policy 1.2 機能でサポートしていません。

注※

Application Server の WS-RM Policy 1.2 機能では，拡張要素および拡張属性を付加しません。受信メッセージに含まれる拡張要素および拡張属性は無視されます。

23.3　com.sun.xml.ws.Closeable クラス

com.sun.xml.ws.Closeable クラスのサポート範囲を次の表に示します。

表 23–3　com.sun.xml.ws.Closeable クラスのメソッドの一覧

項番	戻り値の型	メソッド名／説明	
1	void	close()	
		説明	シーケンスをクローズし，終了させるために，クライアント側でポートのオブジェクトを com.sun.xml.ws.Closeable 型にキャストして，close()メソッドを呼び出す必要があります。close()メソッドを呼び出したあとは，Web サービスメソッドを呼び出すことはできません。再度，通信したい場合は，ポートのオブジェクトを取得し直す必要があります。
		例外	javax.xml.ws.WebServiceException： close()メソッドを呼び出したあとに Web サービスメソッドを使用した場合に発生します。

23.4　WS-Policy による設定

WS-RM 1.2 機能には，WS-RM Policy での設定のほかに，WS-Policy として WSDL に記述する独自の設定があります。ここでは WSDL に追加するプロパティについて説明します。

WS-RM Policy の追加方法については，「34.4　WS-RM Policy の追加方法」を参照してください。

表 23–4　WSDL に追加するプロパティ

項番	プロパティ	説明	単位	範囲	デフォルト値
1	<net35rmp:InactivityTimeout Milliseconds = "設定値"/>	シーケンスの有効期限を設定します。アプリケーションメッセージの再送間隔を設定する場合，アプリケーションメッセージの再送間隔以上の値を設定してください。 設定した期間，通信されない場合 シーケンスの有効期限が切れ，シーケンスは自動的に終了します。 有効期限が切れたあとにメッセージを送信した場合 WebServiceException の子クラスである SequenceTerminatedException 例外または UnknownSequenceException 例外が発生します。 有効期限が切れたあとにメッセージを受信した場合 SequenceTerminated フォルトまたは UnknownSequence フォルトを返信します。 通信を続ける場合 再度ポートのオブジェクトを取得し，シーケンスを生成し直す必要があります。 範囲外の値を指定した場合 警告メッセージを出力し，デフォルト値で動作します（KDJR16017-W）。	ミリ秒	1～9,223,372,036,854,775,807	600,000
2	<cwsrm:MaxMessageNumber value="設定値"/>	一つのシーケンスで扱える最大のメッセージ数を設定します。 設定したメッセージ数を超えてメッセージを送信した場合 WebServiceException の子クラスである MessageNumberRolloverException 例外が発生します。	−	1～100,000	10,000

項番	プロパティ	説明	単位	範囲	デフォルト値
		設定したメッセージ数を超えてメッセージを受信した場合 MessageNumberRollover フォルトを返信します。 通信を続ける場合 既存のシーケンスを終了するため，ポートのオブジェクトを com.sun.xml.ws.Closeable にキャストし，close メソッドを呼び出します。その後，再度ポートのオブジェクトを取得し，シーケンスを生成し直す必要があります。 範囲外の値を指定した場合 警告メッセージを出力し，デフォルト値で動作します（KDJR16017-W）。			
3	<metro:AckRequestInterval Milliseconds="設定値"/>	WS-RM 1.2 機能が自動的にバックグラウンドで送信する Ack メッセージの送信間隔を設定します。タイミングによっては，設定値の 2 倍程度の間隔が開く場合があります。 範囲外の値を指定した場合，警告メッセージを出力し，デフォルト値で動作します（KDJR16017-W）。	ミリ秒	1～9,223,372,036,854,775,807	2,000
4	<metro:RetransmissionConfig> <metro:Interval Milliseconds="設定値"/> </metro:RetransmissionConfig>	アプリケーションメッセージの再送間隔を設定します。 範囲外の値を設定した場合，警告メッセージを出力し，デフォルト値で動作します（KDJR16017-W）。	ミリ秒	1～9,223,372,036,854,775,807	2,000
5	<metro:RetransmissionConfig> <metro:MaxRetries>設定値</metro:MaxRetries> </metro:RetransmissionConfig>	アプリケーションメッセージの再送回数を設定します。設定した再送回数を超えると，再送を停止し，アプリケーションに発生したエラーを返します。 0 を指定した場合 再送回数は無限になります。 範囲外の値を設定した場合 警告メッセージを出力し，デフォルト値で動作します（KDJR16017-W）。	－	0～9,223,372,036,854,775,807	3

（凡例）

－：なし

シーケンスの有効期限を 300,000 ミリ秒（5 分）に，シーケンスの最大メッセージ数を 1,000 に設定する WS-RM Policy の例を次に示します。

```
<wsp:Policy wsu:Id="WSRM_policy">
  <wsp:ExactlyOne>
    <wsp:All>
      <wsrmp:RMAssertion>
        <wsp:Policy>
          <wsrmp:DeliveryAssurance>
            <wsp:Policy>
              <wsrmp:ExactlyOnce/>
            </wsp:Policy>
          </wsrmp:DeliveryAssurance>
        </wsp:Policy>
      </wsrmp:RMAssertion>
      <wsaw:UsingAddressing/>
      <net35rmp:InactivityTimeout Milliseconds="300000"/>
      <cwsrm:MaxMessageNumber value="1000"/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

24 JAX-RS 仕様のサポート範囲

この章では，RESTful Web サービス（Web リソース）を開発するときに留意が必要な，JAX-RS 仕様のサポート範囲について説明します。

24.1　JAX-RS 1.1 仕様のサポート範囲

ここでは，JAX-RS 1.1 仕様のサポート範囲について説明します。JAX-RS 1.1 仕様のサポート範囲を次の表に示します。

表 24–1　JAX-RS 1.1 仕様のサポート範囲

分類			対応	備考
大分類※	小分類			
2	*Application*	ビルトイン実装	○	Application Server の JAX-RS 機能は，サーブレットベースのメカニズム（JAX-RS 1.1 仕様の 2.3.2 項の最終段落に記述されています）をサポートしています。また，JAX-RS 機能はデフォルトのビルトイン *Application* 実装を提供しているので，RESTful Web サービスを実装する際に *Application* の実装は不要です（*Application* のカスタム実装はサポートしていません）。ビルトイン *Application* 実装やデプロイについては，「3.5.1　WAR ファイルの構成」を参照してください。
		カスタム実装	×	
3.1，3.3	要求メソッド識別子	ビルトイン実装	○	Application Server の JAX-RS 機能は，JAX-RS 1.1 仕様で定義されている次の標準の要求メソッド識別子をサポートしています。 • GET アノテーション • POST アノテーション • PUT アノテーション • DELETE アノテーション • HEAD アノテーション JAX-RS 1.1 仕様でオプションとなっている要求メソッド識別子のカスタム実装はサポートしていません。
		カスタム実装	×	
3.1	ルートリソースクラス		○	ルートリソースクラスについては，「17.1.1　ルートリソースクラス」を参照してください。
3.1	ルートリソースクラスのライフサイクル		○	ルートリソースクラスのライフサイクルについては，「17.1.1(1)　ライフサイクル」を参照してください。
3.1.2	ルートリソースクラスのコンストラクタパラメタへのインジェクション		○	ルートリソースクラスのコンストラクタパラメタへのインジェクションについては，「17.1.1(2)　コンストラクタ」を参照してください。 インジェクトできる型や，DefaultValue アノテーションとの組み合わせについては，「17.1.4　パラメタ型」を参照してください。
3.2	フィールドと bean プロパティへのインジェクション		○	フィールドと bean プロパティへのインジェクションについては，「17.1.1(3)　フィールドおよび bean プロパティ」を参照してください。 インジェクトできる型や，DefaultValue アノテーションとの組み合わせについては，「17.1.4　パラメタ型」を参照してください。
3.3	リソースメソッド		○	リソースメソッドについては，「17.1.1(4)　リソースメソッド」を参照してください。

分類			対応	備考
大分類※	小分類			
3.3.2	リソースメソッドのパラメタへのインジェクション		○	リソースメソッドのパラメタへのインジェクションについては，「17.1.1(4)(b)　パラメタのアノテーション」を参照してください。 インジェクトできる型や，DefaultValue アノテーションとの組み合わせについては，「17.1.4　パラメタ型」を参照してください。
3.3.2(1)	HTTP リクエストのエンティティボディからエンティティパラメタへのマッピング		○	エンティティパラメタについては，「17.1.1(4)(c)　エンティティパラメタ」を参照してください。
3.3.3	戻り値から HTTP レスポンスのエンティティボディへのマッピング		○	戻り値については，「17.1.1(4)(d)　戻り値」を参照してください。
3.2，3.3.4	例外のハンドリング		○	例外については，「17.1.5　例外のマッピング」を参照してください。
3.3.5	HEAD HTTP リクエストおよび OPTIONS HTTP リクエストの処理		○	HEAD HTTP リクエストの処理については，「24.3.2(3) javax.ws.rs.HEAD アノテーション」を参照してください。 OPTIONS HTTP リクエストの処理については，「24.3.2(4) javax.ws.rs.OPTIONS アノテーション」を参照してください。
3.4，3.7.3	URI テンプレートと正規表現		○	URI テンプレートについては，「17.1.6　URI テンプレート」を参照してください。
3.4.1	サブリソースメソッド		○	サブリソースメソッドについては，「17.1.1(5)　サブリソースメソッド」を参照してください。
3.4.1	サブリソースロケータとサブリソース		○	サブリソースロケータについては，「17.1.1(6)　サブリソースロケータ」を参照してください。サブリソースクラスについては，「17.1.7　サブリソースクラス」を参照してください。
3.5	メディアタイプの宣言		○	メディアタイプの宣言については，「17.1.9　メディアタイプ宣言」を参照してください。
3.6	アノテーションの継承		○	アノテーションの継承については，「17.1.11　アノテーションの継承」を参照してください。
3.7	HTTP リクエストからリソースメソッドへのマッチング		○	－
3.8	HTTP レスポンスのメディアタイプの決定		○	－
4.2	エンティティプロバイダ（メッセージボディリーダおよびライタ）	ビルトイン実装	○	Application Server の JAX-RS 機能は，ビルトインのエンティティプロバイダを提供しているので，RESTful Web サービスを実装する際にエンティティプロバイダの実装は不要です（エンティティプロバイダのカスタム実装はサポートしていません）。 ビルトインのエンティティプロバイダには，JAX-RS 1.1 仕様でサポートが必須とされているエンティティプロバイダのほか，JAX-RS 機能で追加にサポートするエンティティプロバイダが含まれます。ビルトインのエンティティプロバイダとサポートされる型については，「17.1.1(4)(c)　エンティティパラメタ」を参照してください。
		カスタム実装	×	

分類		対応	備考
大分類※	小分類		
4.3	コンテキストプロバイダ	×	コンテキストプロバイダを実装しなくても，Application Server の JAX-RS 機能は JAX-RS 1.1 仕様の標準のコンテキストを適切に処理します。
4.4	例外マッピングプロバイダ	○	例外マッピングプロバイダについては，「17.2.2　例外マッピングプロバイダ」を参照してください。
5	コンテキスト	○	Application Server の JAX-RS 機能は標準のコンテキスト型をサポートしています。コンテキストについては，「24.4　コンテキスト」を参照してください。
5.2.1	javax.ws.rs.core.Application	×	−
5.2.2	javax.ws.rs.core.UriInfo	○	javax.ws.rs.core.UriInfo については，「24.4.1 javax.ws.rs.core.UriInfo」を参照してください。
5.2.3	javax.ws.rs.core.HttpHeaders	○	javax.ws.rs.core.HttpHeaders については，「24.4.2 javax.ws.rs.core.HttpHeaders」を参照してください。
5.2.4	javax.ws.rs.core.Request	○	javax.ws.rs.core.Request については，「24.4.3 javax.ws.rs.core.Request」を参照してください。
5.2.5	javax.ws.rs.core.SecurityContext	○	javax.ws.rs.core.SecurityContext については，「24.4.4 javax.ws.rs.core.SecurityContext」を参照してください。
5.2.6	javax.ws.rs.ext.Providers	○	javax.ws.rs.ext.Providers については，「24.4.5 javax.ws.rs.core.ext.Providers」を参照してください。
6.1	javax.servlet.ServletConfig	○	javax.servlet.ServletConfig については，「24.4.6 javax.servlet.ServletConfig」を参照してください。
6.1	javax.servlet.ServletContext	○	javax.servlet.ServletContext については，「24.4.7 javax.servlet.ServletContext」を参照してください。
6.1	javax.servlet.http.HttpServletRequest	○	javax.servlet.http.HttpServletRequest については，「24.4.8 javax.servlet.http.HttpServletRequest」を参照してください。
6.1	javax.servlet.http.HttpServletResponse	○	javax.servlet.http.HttpServletResponse については，「24.4.9　javax.servlet.http.HttpServletResponse」を参照してください。
6.1	サーブレットコンテナ（Web コンテナ）による実行環境	○	JAX-RS 機能は，サーブレットコンテナによる RESTful Web サービスの実行環境を提供しています。 「1.4.2(2)(c)　*Application*」も参照してください。
6.1	サーブレット仕様で定義されている型のインジェクション	○	次の項目も参照してください。 • javax.servlet.ServletConfig • javax.servlet.ServletContext • javax.servlet.http.HttpServletRequest • javax.servlet.http.HttpServletResponse

分類		対応	備考
大分類※	小分類		
6.1	要求エンティティのストリーミング処理とメソッド内での応答のコミット	×	Application Server の JAX-RS 機能は，JAX-RS 1.1 仕様の 6.1 節の後半で説明されている動作をサポートしていません。
6.2	Java EE コンテナ（EJB コンテナ）による実行環境	×	−
7	ランタイムデリゲート	○	Application Server の JAX-RS 実装は，JAX-RS 1.1 仕様のランタイムデリゲートのメカニズムに従って実装されています。
Apx.A	アノテーション	○	API については，「24.2　API のサポート範囲」を参照してください。
Apx.B	HTTP ヘッダ	○	−
JavaDoc	API	○	API については，「24.2　API のサポート範囲」を参照してください。

（凡例）

○：対応しています。

×：対応していません。

−：該当しません。

注※

JAX-RS 1.1 仕様の該当個所（章節項）を示します。

24.2　API のサポート範囲

ここでは，JAX-RS API のインタフェースおよびクラスのサポート範囲について説明します。また，JAX-RS 1.1 仕様のインタフェースおよびクラスを利用する場合の注意事項についても説明します。

JAX-RS API のインタフェースおよびクラスのサポート範囲を次の表に示します。インタフェースおよびクラスについては，JAX-RS API のドキュメントを参照してください。

表 24–2　JAX-RS 1.1 仕様のインタフェースおよびクラスのサポート範囲

項番	インタフェースまたはクラス	コンストラクタ／メソッド／フィールド	サポート
javax.ws.rs パッケージ			
1	WebApplicationException	すべてのメソッド	○
2	ApplicationPath	−	×
3	Consumes	−	○
4	CookieParam	−	○
5	DefaultValue	−	○
6	DELETE	−	○
7	Encoded	−	○
8	FormParam	−	○
9	GET	−	○
10	HEAD	−	○
11	HeaderParam	−	○
12	HttpMethod	−	×
13	MatrixParam	−	○
14	OPTIONS	−	○
15	Path	−	○
16	PathParam	−	○
17	POST	−	○
18	Produces	−	○
19	PUT	−	○
20	QueryParam	−	○
javax.ws.rs.core パッケージ			
21	HttpHeaders	getAcceptableMediaTypes()	○
22		getCookies()	○

項番	インタフェースまたはクラス	コンストラクタ／メソッド／フィールド	サポート
23		getLanguages()	○
24		getRequestHeader(String name)	○
25		上記以外のメソッド	○
26	MultivaluedMap<K,V>	すべてのメソッド	○
27	PathSegment	getPath()	○
28		上記以外のメソッド	○
29	Request	evaluatePreconditions(java.util.Date lastModified)	○
30		evaluatePreconditions(EntityTag eTag)	○
31		evaluatePreconditions(java.util.Date lastModified, EntityTag eTag)	○
32		getMethod()	○
33		selectVariant(java.util.List<Variant> variants)	○
34		上記以外のメソッド	○
35	Response.StatusType	すべてのメソッド	○
36	SecurityContext	isUserInRole(String role)	○
37		上記以外のメソッド	○
38	StreamingOutput	すべてのメソッド	○
39	UriInfo	getMatchedResources()	×
40		getMatchedURIs()	×
41		getMatchedURIs(boolean decode)	×
42		getPath()	○
43		getPath(boolean decode)	○
44		getPathParameters()	○
45		getPathParameters(boolean decode)	○
46		getPathSegments()	○
47		getPathSegments(boolean decode)	○
48		getQueryParameters()	○
49		getQueryParameters(boolean decode)	○
50		getRequestUri()	○
51		getRequestUriBuilder()	○
52		上記以外のメソッド	○

項番	インタフェースまたはクラス	コンストラクタ／メソッド／フィールド	サポート
53	Application	すべてのメソッド	×
54	CacheControl	すべてのメソッド	○
55	Cookie	valueOf(String value)	○
56		上記以外のメソッド	○
57	EntityTag	Entity(String value)	○
58		valueOf(String value)	○
59		上記以外のメソッド	○
60	GenericEntity<T>	すべてのメソッド	○
61	MediaType	MediaType(String type, String subtype, java.util.Map<String,String> parameters)	○
62		equals(Object obj)	○
63		getParameters()	○
64		isCompatible(MediaType other)	○
65		valueOf(String type)	○
66		上記以外のメソッド	○
67	NewCookie	valueOf(String value)	○
68		上記以外のメソッド	○
69	Response	created(URI location)	○
70		fromResponse(Response response)	○
71		notModified(EntityTag tag)	○
72		notModified(String tag)	○
73		ok(Object entity, String type)	○
74		seeOther(URI location)	○
75		status(int status)	○
76		temporaryRedirect(URI location)	○
77		上記以外のメソッド	○
78	Response.ResponseBuilder	build()	○
79		status(int status)	○
80		上記以外のメソッド	○
81	UriBuilder	build(Object... values)	○
82		clone()	○

項番	インタフェースまたはクラス	コンストラクタ／メソッド／フィールド	サポート
83		fragment(String fragment)	○
84		fromPath(String path)	○
85		fromUri(String uri)	○
86		fromUri(java.net.URI uri)	○
87		host(String host)	○
88		newInstance()	○
89		path(String path)	○
90		port(int port)	○
91		queryParam(String name, Object... values)	○
92		replacePath(String path)	○
93		replaceQuery(String query)	○
94		replaceQueryParam(String name, Object... values)	○
95		scheme(String scheme)	○
96		schemeSpecificPart(String ssp)	○
97		segment(String... segments)	○
98		uri(java.net.URI uri)	○
99		userInfo(String ui)	○
100		上記以外のメソッド	×
101	Variant	すべてのメソッド	○
102	Variant.VariantListBuilder	すべてのメソッド	×
103	Response.Status	すべてのメソッド	○
104	Response.Status.Family	すべてのメソッド	○
105	UriBuilderException	すべてのメソッド	○
106	Context	－	○
javax.ws.rs.ext パッケージ			
107	ContextResolver<T>	すべてのメソッド	×
108	ExceptionMapper<E extends Throwable>	すべてのメソッド	○
109	MessageBodyReader<T>	すべてのメソッド	×
110	MessageBodyWriter<T>	すべてのメソッド	×
111	Providers	すべてのメソッド	○

項番	インタフェースまたはクラス	コンストラクタ／メソッド／フィールド	サポート
112	RuntimeDelegate.HeaderDelegate<T>	すべてのメソッド	○※
113	RuntimeDelegate	すべてのメソッド	○※
114	Provider	−	○

（凡例）

○：Application Server の JAX-RS 機能でサポートしています。

×：Application Server の JAX-RS 機能でサポートしていません。

−：該当するメソッドおよびフィールドはありません。

注※

ユーザが直接使用することはありません。

24.2.1 HttpHeaders インタフェース

HttpHeaders インタフェースのメソッドを使用する場合の注意事項を次に示します。

- getCookies()メソッドの戻り値に，Cookie のバージョン情報（例："cookieName=$Version=0;cookieName=cookieValue"）も含まれます。
- 受信した HTTP メッセージの"Content-Language"ヘッダに，ISO 639 で規定された言語コード以外を指定すると，getLanguages()メソッドの動作は保証されません。
- getRequestHeader(String name)メソッドを使用する場合，Http リクエストに存在しないヘッダの値を取得しようとすると，null が返却されます。

24.2.2 PathSegment インタフェース

PathSegment インタフェースのメソッドを使用する場合の注意事項を次に示します。

- getPath()メソッドを使用するとき，対象パスにある文字コード（例："%20"など）は，デコードされて返されます。

24.2.3 Request インタフェース

Request インタフェースのメソッドを使用する場合の注意事項を次に示します。

- 次に示すメソッドの引数には null を指定しないでください。null を指定したときの動作は保証されません。
  - evaluatePreconditions(java.util.Date lastModified)
  - evaluatePreconditions(EntityTag eTag)

  - evaluatePreconditions(java.util.Date lastModified, EntityTag eTag)
- 次に示すメソッドの引数 EntityTag eTag に weak の EntityTag を指定しないでください。weak の EntityTag を指定すると，戻り値の ResponseBuilder インスタンスの HTTP Status は"412"（Precondition Failed）になります。
  - evaluatePreconditions(EntityTag eTag)
  - evaluatePreconditions(java.util.Date lastModified, EntityTag eTag)
- If-None-Match ヘッダが設定されて，かつマッチするリソースがないときは，If-Modified-Since ヘッダが設定されても無視されます。
- リクエストのスコープ外から getMethod()メソッドを呼び出さないでください。getMethod()メソッドが呼び出された場合の動作は保証されません。
- 引数が異なる四つの evaluatePreconditions()メソッドのどれかを呼び出す前に selectVariant(java.util.List<Variant> variants)メソッドを呼び出していても，JAX-RS 1.1 標準仕様の API ドキュメントの記載と異なり，Response.ResponseBuilder オブジェクトには vary HTTP ヘッダは含まれません。

24.2.4 SecurityContext インタフェース

SecurityContext インタフェースのメソッドを使用する場合の注意事項を次に示します。

- isUserInRole(String role)メソッドの引数には null を指定しないでください。null を指定したときの動作は保証されません。

24.2.5 UriInfo インタフェース

UriInfo インタフェースのメソッドを使用する場合の注意事項を次に示します。

- getAbsolutePath()メソッドは，JAX-RS 1.1 標準仕様の API ドキュメントの記載と異なり，UriInfo#getBase().resolve(uriInfo.getPath())のショートカットではなく，UriInfo#getBaseUri().resolve(uriInfo.getPath())のショートカットです。getBase()というメソッドは存在しません。
- 次に示すメソッドを使用したときの動作は保証されません。
  - getMatchedResources()
  - getMatchedURIs()
  - getMatchedURIs(boolean decode)
- getPath()メソッドおよび getPath(boolean decode)メソッドの戻り値に，マトリクスパラメタ情報は含まれますが，クエリパラメタ情報は含まれません。

- getPathParameters()メソッドおよびgetPathParameters(boolean decode)メソッドで取得したMultivaluedMap<String,String>は，JAX-RS 1.1 標準仕様の API ドキュメントの記載と異なり，変更できます。
- getPathSegments()メソッドおよびgetPathSegments(boolean decode)メソッドで取得したjava.util.List<PathSegment>は，JAX-RS 1.1 標準仕様の API ドキュメントの記載と異なり，変更できます。
- getQueryParameters()メソッドおよびgetQueryParameters(boolean decode)メソッドで取得したMultivaluedMap<String,String>は，JAX-RS 1.1 標準仕様の API ドキュメントの記載と異なり，変更できます。

24.2.6 Cookie クラス

Cookie クラスのメソッドを使用する場合の注意事項を次に示します。

- 次に示すようにvalueOf(String value)メソッドの引数を指定してください。次に示す以外の記述形式で引数を指定したとき，動作は保証されません。
  - Cookie.valueOf("$Version=xxxx;name=xxxx;$Path=xxxx;$Domain=xxxx;");※
  - Cookie.valueOf("name=xxxx;$Domain=xxxx;$Path=xxxx");※
  - Cookie.valueOf("$Version=xxxx;name=xxxx;");※
  - Cookie.valueOf("name=xxxx");※

  注※

  "xxxx"の部分に，プロパティの値を入れます。

24.2.7 EntityTag クラス

EntityTag クラスのメソッドを使用する場合の注意事項を次に示します。

- EntityTag(String value)コンストラクタでは，weak の EntityTag インスタンスを生成できません。EntityTag(String value)コンストラクタの引数に，"W/"tagValue""のような文字列を指定しないでください。
- valueOf(String value)メソッドを使用するときは，指定したいタグを引用符で囲んで，メソッド引数に指定してください。引用符がないときの動作は保証されません。指定例を次に示します。

  String value = "¥"entityTag¥"";

  EntityTag entityTag = EntityTag.valueOf(value);
- valueOf(String value)メソッドを使用して weak の EntityTag インスタンスを生成するときは，指定したいタグの前に大文字の"W"を追加して引数に指定してください。指定例を次に示します。

  String value = "W/¥"weakEntityTag¥"";

EntityTag entityTag = EntityTag.valueOf(value);

24.2.8 MediaType クラス

MediaType クラスのメソッドを使用する場合の注意事項を次に示します。

- 標準仕様に従った値を使用してください。それ以外の値を指定したときの動作は保証されません。
- MediaType(String type,String subtype, java.util.Map<String,String> parameters)コンストラクタの parameters 引数に指定するマップには，同一のキー（大文字と小文字が異なるだけの文字列も含む）を複数登録しないでください。同一のキーを複数登録したときの動作は保証されません。
- equals(Object obj)メソッドを使用するとき，比較対象となる MediaType オブジェクトの属性である"type"，"subtype"および"parameters"のキーは，大文字と小文字を区別しません。"parameters"の値については大文字と小文字を区別します。
- isCompatible(MediaType other)メソッドを使用するとき，引数の MediaType オブジェクトの属性である"type"および"subtype"は，大文字と小文字を区別しません。
- valueOf(String type)メソッドの引数には，次の文字は使用しないでください。使用した場合の動作は保証されません。

  : ( ) < > @ , ; : / " [ ] ? = { }

  スペース，ラインフィード，キャリッジリターン，水平タブ

24.2.9 NewCookie クラス

NewCookie クラスのメソッドを使用する場合の注意事項を次に示します。

- 次に示すように valueOf(String value)メソッドの引数を指定してください。次に示す以外の記述形式で引数を指定したとき，動作は保証されません。
  - NewCookie.valueOf("name=xxxx;Version=xxxx;Comment=xxxx;Domain=xxxx;Path=xxxx;Max-Age=xxxx;Secure");※
  - NewCookie.valueOf("name=xxxx;Path=xxxx;Domain=xxxx;Comment=xxxx;Max-Age=xxxx;Secure");※
  - NewCookie.valueOf("name=xxxx;Comment=xxxx;Max-Age=xxxx;Secure");※
  - NewCookie.valueOf("name=xxxx;Path=xxxx;Domain=xxxx;Version=xxxx;");※
  - NewCookie.valueOf("name=xxxx;Path=xxxx;Domain=xxxx;"); ※
  - NewCookie.valueOf("name=xxxx;"); ※

注※

"xxxx"の部分に，プロパティの値を入れます。

24.2.10　Response クラス

Response クラスのメソッドを使用する場合の注意事項を次に示します。

- fromResponse(Response response)メソッドの引数には null を指定しないでください。null を指定したときの動作は保証されません。
- 次に示すのメソッドの引数には null を指定しないでください。null を指定したときは，JAX-RS 1.1 標準仕様の API ドキュメントの記載と異なり，IllegalArgumentException は発生しません。
  - created(URI location)
  - notModified(EntityTag tag)
  - notModified(String tag)
  - seeOther(URI location)
  - temporaryRedirect(URI location)
- ok(Object entity, String type)の引数 type には，標準仕様に従った値を指定してください。それ以外の値を指定したときの動作は保証されません。
- status(int status)メソッドの引数に，100 未満または 599 を超える数値を指定しないでください。指定したときは，JAX-RS 1.1 標準仕様の API ドキュメントの記載と異なり，IllegalArgumentException は発生しません。

24.2.11　Response.ResponsBuilder クラス

Response.ResponseBuilder クラスのメソッドを使用する場合の注意事項を次に示します。

- build()メソッドを呼び出したあと，ResponseBuilder インスタンスは，JAX-RS 1.1 標準仕様の API ドキュメントの記載と異なり，ok のステータスではなく no content のステータスにリセットされます。
- status(int status)メソッドの引数に，100 未満または 599 を超える数値を指定しないでください。指定したときは，JAX-RS 1.1 標準仕様の API ドキュメントの記載と異なり，IllegalArgumentException は発生しません。

24.2.12　UriBuilder クラス

UriBuilder クラスのメソッドを使用する場合の注意事項を次に示します。

なお，説明中の「UriBuilder オブジェクトの URI」とは，それぞれのメソッドを呼び出すときに，それまでのほかのメソッドの呼び出しですでに UriBuilder オブジェクトに構築されている URI のことを指します。

- UriBuilder クラスのメソッドを使用する場合は，JAX-RS API のドキュメントや RFC 2396，2732 を参照の上，各メソッドの引数に使用できる文字および形式を使用してください。引数の文字や形式が不正な場合，そのメソッドを呼び出したり，build(Object... values)メソッドを呼び出したりしたときに

例外がスローされることがあります。なお，次のメソッドでは，不正な文字は自動的にパーセントエンコードされます（すでにパーセントエンコードされている場合はそれ以上パーセントエンコードされません）。

- fragment(java.lang.String fragment)
- host(String host)
- path(String path)
- queryParam(java.lang.String name, java.lang.Object... values)
- replacePath(String path)
- replaceQuery(java.lang.String query)
- replaceQueryParam(java.lang.String name, java.lang.Object... values)
- segment(String... segments)
- userInfo(String ui)

- 不透明な URI はサポートされません。階層的な URI に対してだけ使用してください。
- テンプレートパラメタの使用はサポートされません。メソッドの引数にテンプレートパラメタが含まれる場合，動作は保証されません。
- build(Object... values)メソッドには引数を指定しないでください。引数が指定されて呼び出された場合，動作は保証されません。
- クエリパラメタの順序は保持されません。
- port(int port)メソッドを使用する場合，65535 を超える数値を指定しても，例外にはなりません。-1 より小さい数値を指定すると，IllegalArgumentException がスローされます。
- replaceQuery(String query)メソッドを使用する場合，UriBuilder オブジェクトの URI に置き換える対象が存在しないときの動作は保証されません。
- replaceQueryParam(String name, Object... values)メソッドの引数に，UriBuilder オブジェクトの URI に存在しないクエリパラメタを指定した場合，指定されたパラメタが URI に追加されます。
- replaceQueryParam(String name, Object... values)メソッドの values 引数に null を指定した場合，JAX-RS 1.1 標準仕様の API ドキュメントの記載と異なり，既存のクエリパラメタは取り除かれず，IllegalArgumentException がスローされます。
- replacePath(String path)メソッドの引数に null を指定した場合，JAX-RS 1.1 標準仕様の API ドキュメントの記載と異なり，既存のパスは取り除かれず，IllegalArgumentException がスローされます。

24.2.13 Provider アノテーション

Provider アノテーションを使用する場合の注意事項を次に示します。

- ExceptionMapper インタフェースを実装したクラスだけに使用できます。

24.3　アノテーション

ここでは，Application Server がサポートする JAX-RS 1.1 仕様のアノテーションについて説明します。

24.3.1　インジェクション用アノテーション

ここでは，JAX-RS エンジンがサポートするインジェクション用アノテーションについて説明します。

(1)　注意事項

(a)　複数のアノテーションを使用する場合

インジェクション用アノテーションは，パラメタ，フィールド，または bean プロパティそれぞれにつき一つだけ使用できます。一つのパラメタ，フィールド，または bean プロパティにつき複数のアノテーションを同時に指定した場合は，最も右のアノテーションだけが有効になります。最も右のアノテーションが JAX-RS 1.1 仕様でない場合は，ほかのすべてのアノテーションが無視されてインジェクションは実行されません。

(b)　bean プロパティに使用する場合

インジェクション用アノテーションを bean プロパティに使用する場合は，setter メソッドに対してメソッドレベルでアノテートしてください。

(c)　フィールドに使用する場合

ルートリソースクラスおよび例外マッピングプロバイダでは，コンストラクタおよび bean プロパティの setter メソッドから，インジェクション用アノテーションでアノテートされたフィールドを参照または変更しないでください。

コンストラクタ，フィールド，および bean プロパティへのインジェクションは，ルートリソースクラスおよび例外マッピングプロバイダが JAX-RS エンジンによってインスタンス化されるときに同時に行われます。このため，コンストラクタおよび bean プロパティの setter メソッドから，インジェクション用アノテーションでアノテートされたフィールドを参照した場合，取得される値は不定です。また，インジェクション用アノテーションでアノテートされたフィールドを変更した場合，フィールドへのインジェクションが失敗するか，または成功しても値が不定になります。

(2)　javax.ws.rs.HeaderParam アノテーション

javax.ws.rs.HeaderParam アノテーションは HTTP ヘッダの値を取得するために使用します。アノテーションの値には，HTTP リクエストに含まれる HTTP ヘッダの名称を指定します。

javax.ws.rs.HeaderParam アノテーションを指定できる対象を，次の表に示します。

表 24–3　javax.ws.rs.HeaderParam アノテーションを指定できる対象

Web リソースまたはプロバイダ	コンストラクタのパラメタ	フィールド	bean プロパティ	メソッドのパラメタ
ルートリソースクラス	○	○	○	○
サブリソースクラス	×	×	×	○
例外マッピングプロバイダ	×	×	×	−

（凡例）

○：指定できることを示します。

×：指定できないことを示します。

−：該当するパラメタがないことを示します。

Web リソースまたはプロバイダについては，「17.　Web リソースとプロバイダ」の各項目を参照してください。

(3)　javax.ws.rs.CookieParam アノテーション

javax.ws.rs.CookieParam アノテーションは HTTP Cookie の値を取得するために使用します。アノテーションの値は，HTTP リクエストに含まれる Cookie の名称を指定します。

javax.ws.rs.CookieParam アノテーションを指定できる対象を，次の表に示します。

表 24–4　javax.ws.rs.CookieParam アノテーションを指定できる対象

Web リソースまたはプロバイダ	コンストラクタのパラメタ	フィールド	bean プロパティ	メソッドのパラメタ
アプリケーションサブクラス	×	×	×	−
ルートリソースクラス	○	○	○	○
サブリソースクラス	×	×	×	○
エンティティプロバイダ	×	×	×	−
コンテキストプロバイダ	×	×	×	−
例外マッピングプロバイダ	×	×	×	−

（凡例）

○：指定できることを示します。

×：指定できないことを示します。

−：該当するパラメタがないことを示します。

Web リソースまたはプロバイダについては，「17.　Web リソースとプロバイダ」の各項目を参照してください。

(4)　javax.ws.rs.MatrixParam アノテーション

javax.ws.rs.MatrixParam アノテーションは URI マトリクスパラメタの値を取得するために使用します。アノテーションの値は，HTTP リクエストに含まれるマトリクスパラメタの名称を指定します。

javax.ws.rs.MatrixParam アノテーションを指定できる対象を，次の表に示します。

表 24–5　javax.ws.rs.Matrix parameter アノテーションを指定できる対象

Web リソースまたはプロバイダ	コンストラクタのパラメタ	フィールド	bean プロパティ	メソッドのパラメタ
アプリケーションサブクラス	×	×	×	−
ルートリソースクラス	○	○	○	○
サブリソースクラス	×	×	×	○
エンティティプロバイダ	×	×	×	−
コンテキストプロバイダ	×	×	×	−
例外マッピングプロバイダ	×	×	×	−

（凡例）

○：指定できることを示します。

×：指定できないことを示します。

−：該当するパラメタがないことを示します。

Web リソースまたはプロバイダについては，「17.　Web リソースとプロバイダ」の各項目を参照してください。

(5)　javax.ws.rs.QueryParam アノテーション

javax.ws.rs.QueryParam アノテーションは URI クエリパラメタの値を取得するために使用します。アノテーションの値は，HTTP リクエストに含まれるクエリパラメタの名称を指定します。

javax.ws.rs.QueryParam アノテーションを指定できる対象を，次の表に示します。

表 24–6　javax.ws.rs.QueryParam を指定できる対象

Web リソースまたはプロバイダ	コンストラクタのパラメタ	フィールド	bean プロパティ	メソッドのパラメタ
アプリケーションサブクラス	×	×	×	−
ルートリソースクラス	○	○	○	○
サブリソースクラス	×	×	×	○
エンティティプロバイダ	×	×	×	−
コンテキストプロバイダ	×	×	×	−

Web リソースまたはプロバイダ	コンストラクタのパラメタ	フィールド	bean プロパティ	メソッドのパラメタ
例外マッピングプロバイダ	×	×	×	−

（凡例）

○：指定できることを示します。

×：指定できないことを示します。

−：該当するパラメタがないことを示します。

Web リソースまたはプロバイダについては，「17.　Web リソースとプロバイダ」の各項目を参照してください。

(6)　javax.ws.rs.PathParam アノテーション

javax.ws.rs.PathParam アノテーションは URI のパスの値を取得するために使用します。アノテーションの値には，テンプレートパラメタを指定します。

javax.ws.rs.PathParam アノテーションを指定できる対象を，次の表に示します。

表 24–7　javax.ws.rs.PathParam アノテーションを指定できる対象

Web リソースまたはプロバイダ	コンストラクタのパラメタ	フィールド	bean プロパティ	メソッドのパラメタ
アプリケーションサブクラス	×	×	×	−
ルートリソースクラス	○	○	○	○
サブリソースクラス	×	×	×	○
エンティティプロバイダ	×	×	×	−
コンテキストプロバイダ	×	×	×	−
例外マッピングプロバイダ	×	×	×	−

（凡例）

○：指定できることを示します。

×：指定できないことを示します。

−：該当するパラメタがないことを示します。

Web リソースまたはプロバイダについては，「17.　Web リソースとプロバイダ」の各項目を参照してください。

(7)　javax.ws.rs.FormParam アノテーション

javax.ws.rs.FormParam アノテーションは，HTTP リクエストのエンティティボディに含まれるフォームパラメタの値を取得するために使用します。アノテーションの値はフォームパラメタの名称を指定します。

javax.ws.rs.FormParam アノテーションを指定できる対象を，次の表に示します。

表 24–8　javax.ws.rs.FormParam アノテーションを指定できる対象

Web リソースまたはプロバイダ	コンストラクタのパラメタ	フィールド	bean プロパティ	メソッドのパラメタ
ルートリソースクラス	○	○	○	○
サブリソースクラス	×	×	×	○
例外マッピングプロバイダ	×	×	×	−

（凡例）

○：指定できることを示します。

×：指定できないことを示します。

−：該当するパラメタがないことを示します。

Web リソースまたはプロバイダについては，「17.　Web リソースとプロバイダ」の各項目を参照してください。

エンティティボディに含まれるフォームパラメタ数の上限値は，デフォルトでは 10,000 です。リクエストのパラメタ数が指定した値を超えた場合，エラーとなり（KDJJ10042-E），HTTP ステータスコードに 413 を設定した，例外マッピングプロバイダで処理できる javax.ws.rs.WebApplicationException がスローされます。必要に応じて J2EE サーバ用ユーザプロパティファイル（usrconf.properties）の webserver.connector.limit.max_parameter_count プロパティで変更してください。J2EE サーバ用ユーザプロパティファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「2.2.3　usrconf.properties（J2EE サーバ用ユーザプロパティファイル)」を参照してください。

(8)　javax.ws.rs.core.Context アノテーション

javax.ws.rs.Context アノテーションはコンテキストの値をインジェクトするために使用します。javax.ws.rs.Context アノテーションを指定できる対象を，次の表に示します。

表 24–9　javax.ws.rs.Context アノテーションを指定できる対象（1）

Web リソースまたはプロバイダ		Context Application 型	Context UriInfo 型	Context HttpHeaders 型	Context Request 型	Context SecurityContext 型	Context Providers 型
ルートリソースクラス	コンストラクタのパラメタ	×	○	○	○	○	○
	フィールド	×	○	○	○	○	○
	bean プロパティ	×	○	○	○	○	○
	メソッドのパラメタ	×	○	○	○	○	○
サブリソースクラス	コンストラクタのパラメタ	×	×	×	×	×	×
	フィールド	×	×	×	×	×	×
	bean プロパティ	×	×	×	×	×	×

Web リソースまたはプロバイダ		Context Application 型	Context UriInfo 型	Context HttpHeaders 型	Context Request 型	Context SecurityContext 型	Context Providers 型
	メソッドのパラメタ	×	○	○	○	○	○
例外マッピングプロバイダ	コンストラクタのパラメタ	×	×	×	×	×	○
	フィールド	×	○	○	○	○	○
	bean プロパティ	×	○	○	○	○	○

（凡例）

○：指定できることを示します。

×：指定できないことを示します。

表 24–10　javax.ws.rs.Context アノテーションを指定できる対象（2）

Web リソースまたはプロバイダ		Context HttpServletRequest 型	HttpServletResponse 型	ServletContext 型	ServletConfig 型
ルートリソースクラス	コンストラクタのパラメタ	○※	○	○	○
	フィールド	○※	○	○	○
	bean プロパティ	○※	○	○	○
	メソッドのパラメタ	○※	○	○	○
サブリソースクラス	コンストラクタのパラメタ	×	×	×	×
	フィールド	×	×	×	×
	bean プロパティ	×	×	×	×
	メソッドのパラメタ	○※	○	○	○
例外マッピングプロバイダ	コンストラクタのパラメタ	○※	○	○	○
	フィールド	○※	○	○	○
	bean プロパティ	○※	○	○	○

（凡例）

○：指定できることを示します。

×：指定できないことを示します。

注※

エンティティボディおよびクエリパラメタは取得できません。エンティティパラメタを持つリソースメソッドで，Context アノテーションでインジェクトされた HttpServletRequest インスタンスの getReader()メソッドを呼び出した場合，例外マッピングプロバイダで処理できる java.lang.IllegalStateException がスローされます。

(9) javax.ws.rs.DefaultValue アノテーション

javax.ws.rs.DefaultValue アノテーションは，次に示すアノテーションと組み合わせて使用できます。javax.ws.rs.DefaultValue アノテーションを使用すると，それぞれのアノテーションでアノテートされたパラメタにインジェクトする値が，HTTP リクエストにない場合のデフォルト値を指定できます。

- MatrixParam アノテーション
- QueryParam アノテーション
- CookieParam アノテーション
- HeaderParam アノテーション
- FormParam アノテーション

例えば，インスタンスに対してクエリもしくはマトリクスパラメタが URI の要求に存在しないとき，または対象のフォームパラメタが要求のエンティティボディに存在しないときは，DefaultValue アノテーションに指定されている初期値が使用されます。

(10) javax.ws.rs.Encoded アノテーション

javax.ws.rs.Encoded アノテーションは，URL エンコードされた値が自動で URL デコードされるのを無効化するために使用します。URL エンコードされた値が自動で URL デコードされるのは，次に示すインジェクション用アノテーションでアノテートされたパラメタ，フィールド，および bean プロパティです。

- javax.ws.rs.MatrixParam アノテーション
- javax.ws.rs.QueryParam アノテーション
- javax.ws.rs.PathParam アノテーション

javax.ws.rs.Encoded アノテーションは，次に示す場所で使用できます。

- ルートリソースクラス（クラスレベル）
- サブリソースクラス（クラスレベル）
- ルートリソースクラスのコンストラクタ（コンストラクタレベル）
- ルートリソースクラスのコンストラクタのパラメタ（パラメタレベル）
- リソースクラスのリソースメソッド，サブリソースメソッド，およびサブリソースロケータ（メソッドレベル）
- リソースクラスのリソースメソッド，サブリソースメソッド，およびサブリソースロケータのそれぞれのパラメタ（パラメタレベル）
- ルートリソースクラスのフィールド，および bean プロパティ（フィールドレベル，プロパティレベル）

パラメタレベル，フィールドレベル，プロパティレベルでは，前述のインジェクション用アノテーション（javax.ws.rs.MatrixParam，javax.ws.rs.QueryParam，javax.ws.rs.PathParam）と組み合わせて使用してください。

クラスレベルで javax.ws.rs.Encoded アノテーションを使用した場合は，そのクラスのすべてのパラメタ，フィールド，および bean プロパティで，URL エンコードされた値が自動で URL デコードされるのを無効化できます。

コンストラクタレベルで javax.ws.rs.Encoded アノテーションを使用した場合は，そのコンストラクタのすべてのパラメタで，URL エンコードされた値が自動で URL デコードされるのを無効化できます。

メソッドレベルで javax.ws.rs.Encoded アノテーションを使用した場合は，そのリソースメソッド，サブリソースメソッド，またはサブリソースロケータのすべてのパラメタで，URL エンコードされた値が自動で URL デコードされるのを無効化できます。

24.3.2 ビルトイン要求メソッド識別子

ここでは，JAX-RS エンジンがサポートする要求メソッド識別子について説明します。

(1) javax.ws.rs.DELETE アノテーション

javax.ws.rs.DELETE アノテーションは，アノテートされたメソッドが HTTP DELETE リクエストを処理することを指定します。javax.ws.rs.DELETE アノテーションが使用できる対象を次に示します。

- ルートリソースクラスの public メソッド
- サブリソースクラスの public メソッド

(2) javax.ws.rs.GET アノテーション

javax.ws.rs.GET アノテーションは，アノテートされたメソッドが HTTP GET リクエストを処理することを示します。javax.ws.rs.GET アノテーションが使用できる対象を次に示します。

- ルートリソースクラスの public メソッド
- サブリソースクラスの public メソッド

(3) javax.ws.rs.HEAD アノテーション

javax.ws.rs.HEAD アノテーションは，アノテートされたメソッドが HTTP HEAD リクエストを処理することを示します。javax.ws.rs.HEAD アノテーションが使用できる対象を次に示します。

- ルートリソースクラスの public メソッド
- サブリソースクラスの public メソッド

HTTP HEAD リクエストを受けた場合，JAX-RS エンジンは次の優先順位で動作します。

1. HEAD アノテーションでアノテートされたメソッドがあれば呼び出す。

2. HEAD アノテーションでアノテートされたメソッドがなければ，GET アノテーションでアノテートされたメソッドを呼び出す。なお，そのメソッドが戻り値を返す場合は無視される。

HTTP HEAD リクエストを処理するルートリソースクラスの例を次に示します。

```
package com.sample.resources;

import javax.ws.rs.HEAD;
import javax.ws.rs.Path;
import javax.ws.rs.core.Response;

//ルートリソースクラス
@Path(”/root”)
public class Resource {

  //リソースメソッド
  @HEAD
  public Response getValue() {

    String customHeader = ”foo”;
    String customHeaderValue = ”bar”;
    int httpStatus = 200;

    //ResponseBuilderを使用してResponseオブジェクトを構築する

    return Response.status(httpStatus).header(customHeader,
        customHeaderValue).build();
  }
}
```

ルートリソースクラスcom.sample.resources.Resource を含む Web アプリケーション（WAR ファイル）のコンテキストルートが"example"で，Web アプリケーションが"sample.com"というホストで公開されているとします。この例では，URL"http://sample.com/example/root"に対する HTTP HEAD リクエストは，getValue()メソッドにディスパッチされます。getValue()メソッドが返す Response オブジェクトがエンティティボディを含んでいても無視されます。

(4) javax.ws.rs.OPTIONS アノテーション

javax.ws.rs.OPTIONS アノテーションは，アノテートされたメソッドが HTTP OPTIONS リクエストを処理することを示します。javax.ws.rs.OPTIONS アノテーションが使用できる対象を次に示します。

- ルートリソースクラスの public メソッド
- サブリソースクラスの public メソッド

HTTP OPTIONS リクエストのための JAX-RS エンジンの動作を次に示します。

1. javax.ws.rs.OPTIONS アノテーションでアノテートされたメソッドを呼び出す。
2. javax.ws.rs.OPTIONS アノテーションでアノテートされたメソッドがなければ，Web リソースのアノテーションの情報を使用して，JAX-RS エンジンが自動的に応答する。

HTTP OPTIONS リクエストを処理するルートリソースクラスの例を次に示します。

```
package com.sample.resources;

import javax.ws.rs.OPTIONS;
import javax.ws.rs.Path;
import javax.ws.rs.core.Response;

//ルートリソースクラス
@Path(”/root”)
public class Resource {

  //リソースメソッド
  @OPTIONS
  public Response getValue() {
    String entity = ”Some Contents”;
    String customHeader = ”foo”;
    String customHeaderValue = ”bar”;
    int httpStatus = 200;

    //ResponseBuilderを使用してResponseオブジェクトを構築する

    return Response.status(httpStatus).header(customHeader,
        customHeaderValue).entity(entity).build();

  }
}
```

ルートリソースクラスcom.sample.resources.Resource を含む Web アプリケーション（WAR ファイル）のコンテキストルートが"example"で，Web アプリケーションが"sample.com"というホストで公開されているとします。この例では，URL"http://sample.com/example/root"に対する HTTP OPTIONS リクエストは，getValue()メソッドにディスパッチされます。

(5) javax.ws.rs.POST アノテーション

javax.ws.rs.POST アノテーションは，アノテートされたメソッドが HTTP POST リクエストを処理することを示します。javax.ws.rs.POST アノテーションが使用できる対象を次に示します。

- ルートリソースクラスの public メソッド
- サブリソースクラスの public メソッド

(6) javax.ws.rs.PUT アノテーション

javax.ws.rs.PUT アノテーションは，アノテートされたメソッドが HTTP PUT リクエストを処理することを示します。javax.ws.rs.PUT アノテーションが使用できる対象を次に示します。

- ルートリソースクラスの public メソッド
- サブリソースクラスの public メソッド

24.3.3　パス指定用アノテーション

ここでは，JAX-RS エンジンがサポートするパス指定用のアノテーションについて説明します。

(1)　javax.ws.rs.Path アノテーション

javax.ws.rs.Path アノテーションは，リソースのパスを指定します。クラスレベルで使用される場合，アノテートされたクラスはルートリソースクラスと見なされます。メソッドレベルで使用される場合，アノテートされたメソッドはサブリソースメソッドまたはサブリソースロケータと見なされます。
javax.ws.rs.Path アノテーションは次に示す場所で使用できます。

- ルートリソースクラス（クラスレベル）
- リソースクラス（ルートリソースクラスまたはサブリソースクラス）の要求メソッド識別子でアノテートされた public メソッド（メソッドレベル）：このメソッドはサブリソースメソッドです。
- リソースクラス（ルートリソースクラスまたはサブリソースクラス）の要求メソッド識別子でアノテートされていない public メソッド（メソッドレベル）：このメソッドはサブリソースロケータです。

24.3.4　メディアタイプ宣言用アノテーション

ここでは，JAX-RS エンジンがサポートするメディアタイプ宣言用アノテーションについて説明します。

(1)　javax.ws.rs.Consumes アノテーション

javax.ws.rs.Consumes アノテーションは，Web リソースが HTTP リクエストでサポートする MIME メディアタイプのリストを指定します。javax.ws.rs.Consumes アノテーションが使用できる対象を次に示します。

- ルートリソースクラス（クラスレベル）
- リソースメソッド（メソッドレベル）
- サブリソースメソッド（メソッドレベル）
- サブリソースクラス（クラスレベル）

javax.ws.rs.Consumes アノテーションが例外マッピングプロバイダに使用されている場合，無視されます。

(2)　javax.ws.rs.Produces アノテーション

javax.ws.rs.Produces アノテーションは，Web リソースが HTTP レスポンスでサポートする MIME メディアタイプのリストを指定します。javax.ws.rs.Produces アノテーションが使用できる対象を次に示します。

- ルートリソースクラス（クラスレベル）
- リソースメソッド（メソッドレベル）

- サブリソースメソッド（メソッドレベル）
- サブリソースクラス（クラスレベル）

javax.ws.rs.Produces アノテーションが例外マッピングプロバイダに使用されている場合，無視されます。

24.4　コンテキスト

ルートリソースクラス，サブリソースクラス，および例外マッピングプロバイダでは，Context アノテーションを使用して JAX-RS 1.1 仕様で定義されるコンテキストを取得できます。JAX-RS エンジンでサポートしているコンテキストの種類は次のとおりです。

- javax.ws.rs.core.UriInfo
- javax.ws.rs.core.HttpHeaders
- javax.ws.rs.core.Request
- javax.ws.rs.core.SecurityContext
- javax.ws.rs.core.ext.Providers
- javax.servlet.ServletConfig
- javax.servlet.ServletContext
- javax.servlet.http.HttpServletRequest
- javax.servlet.http.HttpServletResponse

コンテキストに含まれる情報は HTTP リクエストごとに異なりますが，プロバイダのインスタンスはシングルトンであるため，javax.ws.rs.core.ext.Providers に含まれる情報は WAR ファイルの単位で常に同一です。

ここでは JAX-RS エンジンでサポートしているコンテキストの種類について説明します。

24.4.1　javax.ws.rs.core.UriInfo

javax.ws.rs.core.UriInfo は，URI を構成する各コンポーネント（クエリパラメタやマトリクスパラメタなど）を保持するコンテキストです。HTTP リクエストごとの情報を提供します。なお，javax.ws.rs.core.UriInfo の各メソッドで取得できるのは，正規化後の情報です。

ルートリソースクラスのフィールドにインジェクトされる javax.ws.rs.core.UriInfo コンテキストの使用例を次に示します。

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.UriInfo;

//ルートリソースクラス
@Path("/root")
public class Resource {
  //Contextアノテーションを使用してUriInfoをインジェクトするフィールド
  private @Context UriInfo uriInfo;
```

```
  //リソースメソッド
  @GET
  public String getValue() {
    String value = this.uriInfo.getQueryParameters().getFirst(”query”);
    return value;
  }
}
```

ルートリソースクラスcom.sample.resources.Resource を含む Web アプリケーション（WAR ファイル）のコンテキストルートが"example"で，Web アプリケーションが"sample.com"というホストで公開されているとします。この例で，URL"http://sample.com/example/root?query=10"に対する HTTP GET リクエストでは，まず uriInfo フィールドに javax.ws.rs.core.UriInfo コンテキストがインジェクトされ，その後 HTTP GET リクエストを処理できる getValue()メソッドが呼び出されます。このため，getValue()メソッドで uriInfo フィールドからクエリパラメタ"query"を取得すると"10"という値が取得されます。

24.4.2 javax.ws.rs.core.HttpHeaders

javax.ws.rs.core.HttpHeaders は，HTTP リクエストの HTTP ヘッダを保持するコンテキストです。

ルートリソースクラスのフィールドにインジェクトされる javax.ws.rs.core.HttpHeaders の使用例を次に示します。

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.HttpHeaders;

//ルートリソースクラス
@Path(”/root”)
public class Resource {
  //Contextアノテーションを使用してHttpHeadersをインジェクトするフィールド
  private @Context HttpHeaders httpHeaders;

//リソースメソッド
  @GET
  public String getValue () {
    String value = this.httpHeaders.getRequestHeader(”Accept”).get(0);
    return value;
  }
}
```

ルートリソースクラスcom.sample.resources.Resource を含む Web アプリケーション（WAR ファイル）のコンテキストルートが"example"で，Web アプリケーションが"sample.com"というホストで公開されているとします。この例で，Accept HTTP ヘッダに"application/xml"を指定した URL"http://sample.com/example/root"に対する HTTP GET リクエストでは，まず httpHeaders フィールドに javax.ws.rs.core.HttpHeaders コンテキストがインジェクトされ，その後 HTTP GET リクエストを処理

できる getValue()メソッドが呼び出されます。このため，getValue()メソッドで httpHeaders フィールドから Accept HTTP ヘッダの値を取得すると"application/xml"という値が取得されます。

24.4.3 javax.ws.rs.core.Request

javax.ws.rs.core.Request は，RFC 2616 で規定される「コンテントネゴシエーション（Content Negotiation)」に必要な機能を提供するコンテキストです。

ルートリソースクラスのフィールドにインジェクトされる javax.ws.rs.core.Request の使用例を次に示します。

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.EntityTag;
import javax.ws.rs.core.Request;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;
//ルートリソースクラス
@Path(”/root”)
public class Resource {
  //Contextアノテーションを使用してRequestをインジェクトするフィールド
  private @Context Request request;
  //コンテントネゴシエーションで利用するHTTP Entity Tag
  private EntityTag eTag = new EntityTag(”a-resource-status-specific-tag”);
  //リソースメソッド
  @GET
  public Response getData() {
ResponseBuilder rb = null;
    //コンテントネゴシエーション(プレコンディションの評価)を行う
    //  プレコンディションに合えばnull，合わなければ
    //  適切なETag HTTPヘッダやステータスコード(412: Precondition Failed) が
    //  設定されたResponseBuilderオブジェクトが取得される
    rb = request.evaluatePreconditions(this.eTag);
    if (rb != null) {
      //プレコンディションに合わない場合，ResponseBuilderオブジェクトから
      //HTTPレスポンスを生成してそのまま返す
      return rb.build();
    } else {
      //プレコンディションに合う場合，要求のデータを返す
      String data = ”Some Information”;
      return Response.ok().entity(data).build();
    }
  }
}
```

ルートリソースクラスcom.sample.resources.Resource を含む Web アプリケーション（WAR ファイル）のコンテキストルートが"example"で，Web アプリケーションが"sample.com"というホストで公開されているとします。この例で，If-Match HTTP ヘッダに"a-resource-status-specific-tag"を指定した

URL"http://sample.com/example/root"に対するHTTP GETリクエストでは，まずrequestフィールドにjavax.ws.rs.core.Requestコンテキストがインジェクトされ，その後HTTP GETリクエストを処理できるgetData()メソッドが呼び出されます。このため，getData()メソッドでは，コンテントネゴシエーション（この例ではHTTPリクエストのIf-Match HTTPヘッダとリソースが保持するHTTP EntityTagの比較）が行われ，"Some Information"という値が取得されます。

24.4.4 javax.ws.rs.core.SecurityContext

javax.ws.rs.core.SecurityContextは，処理中のHTTPリクエストに関連するセキュリティ情報を保持するコンテキストです。

ルートリソースクラスのフィールドにインジェクトされるjavax.ws.rs.core.SecurityContextの使用例を次に示します。

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.SecurityContext;

//ルートリソースクラス
@Path(”/root”)
public class Resource {
  //Contextアノテーションを使用してSecurityContextをインジェクトするフィールド
  private @Context SecurityContext securityContext;

  //リソースメソッド
  @GET
  public String getValue () {
    String value = ”Authentication Scheme: ”
        + this.securityContext.getAuthenticationScheme()
        + ”, User Principal: ” + this.securityContext.getUserPrincipal()
        + ”, Is secure: ” + this.securityContext.isSecure()
        + ”, Is user in role: ” + this.securityContext.isUserInRole(”admin”);

    return value;
  }
}
```

セキュリティ情報を含むweb.xmlの例を次に示します。

```
<?xml version=”1.0” encoding=”UTF-8”?>
<web-app ...>
  ...
  <security-constraint>
     <web-resource-collection>
       <web-resource-name>Test Resource</web-resource-name>
       <url-pattern>/*</url-pattern>
       <http-method>GET</http-method>
     </web-resource-collection>
```

```
    <auth-constraint>
      <role-name>admin</role-name>
    </auth-constraint>
  </security-constraint>
  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>jaxrs_server</realm-name>
  </login-config>
  <security-role>
    <role-name>admin</role-name>
  </security-role>
</web-app>
```

ルートリソースクラスcom.sample.resources.Resource を含む Web アプリケーション（WAR ファイル）のコンテキストルートが"example"で，Web アプリケーションが"sample.com"というホストで公開されているとします。この例で，適切な認証情報を指定した URL"http://sample.com/example/root"に対する HTTP GET リクエストでは，まず securityContext フィールドに javax.ws.rs.core.SecurityContext コンテキストがインジェクトされ，その後 HTTP GET リクエストを処理できる getValue()メソッドが呼び出されます。getValue()メソッドでは，web.xml の設定と実際の認証情報に基づいてセキュリティ情報が取得されます。

24.4.5 javax.ws.rs.core.ext.Providers

javax.ws.rs.core.ext.Providers は，デプロイされた Web リソースで動作するプロバイダを保持するコンテキストです。

ルートリソースクラスのフィールドにインジェクトされる javax.ws.rs.core.ext.Providers の使用例を次に示します。

```
package com.sample.resources;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Context;
import javax.ws.rs.ext.Providers;

//ルートリソースクラス
@Path(”root”)
public class Resource{

  //Contextアノテーションを使用してProvidersをインジェクトするフィールド
  private @Context Providers providers;

  //リソースメソッド
  @GET
  public String getValue() {
    //providersフィールドから例外マッピングプロバイダを取得する
    return this.providers.getExceptionMapper(RuntimeException.class);
  }
}
```

ルートリソースクラス com.sample.resources.Resource と java.lang.RuntimeException を処理できる例外マッピングプロバイダ com.sample.providers.EntityProviderReader を含む Web アプリケーション（WAR ファイル）のコンテキストルートが"example"で，Web アプリケーションが"sample.com"というホストで公開されているとします。この例で，URL"http://sample.com/example/root"に対する HTTP GET リクエストでは，まず providers フィールドに javax.ws.rs.core.ext.Providers コンテキストがインジェクトされ，その後 HTTP GET リクエストを処理できる getValue()メソッドが呼び出されます。このため，getValue()メソッドで providers フィールドから java.lang.RuntimeException を処理できる例外マッピングプロバイダを取得すると com.sample.providers.EntityProviderReader インスタンスが取得されます。

24.4.6 javax.servlet.ServletConfig

javax.servlet.ServletConfig は Servlet 仕様で定義されているクラスです。

ルートリソースクラスのフィールドにインジェクトされる javax.servlet.ServletConfig の使用例を次に示します。

```
package com.sample.resources;

import javax.servlet.ServletConfig;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Context;

//ルートリソースクラス
@Path(”/root”)
public class Resource {
  //Contextアノテーションを使用してServletConfigをインジェクトするフィールド
  private @Context ServletConfig config;

  //リソースメソッド
  @GET
  public String getValue() {
    return this.config.getInitParameter(”TestParam”);
  }
}
```

追加の初期化パラメタ（init-param 要素）を含む web.xml の例を次に示します。

```
<?xml version=”1.0” encoding=”UTF-8”?>
<web-app ...>
  <servlet>
    <servlet-name>CosminexusJaxrsServlet</servlet-name>
    <servlet-class>com.cosminexus.jersey.spi.container.servlet.ServletContainer</servlet-cla
ss>
    <init-param>
      <param-name>com.cosminexus.jersey.config.property.packages</param-name>
      <param-value>com.sample.resources</param-value>
    </init-param>
    <init-param>
```

```
      <param-name>TestParam</param-name>
      <param-value>TestValue</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxrsServlet</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>
```

ルートリソースクラスcom.sample.resources.Resource を含む Web アプリケーション（WAR ファイル）のコンテキストルートが"example"で，Web アプリケーションが"sample.com"というホストで公開されているとします。この例で，URL"http://sample.com/example/root"に対する HTTP GET リクエストでは，まず config フィールドに javax.servlet.ServletConfig コンテキストがインジェクトされ，その後 HTTP GET リクエストを処理できる getValue()メソッドが呼び出されます。このため，getValue()メソッドで config フィールドから初期化パラメタ"TestParam"を取得すると"TestValue"という値が取得されます。

24.4.7 javax.servlet.ServletContext

javax.servlet.ServletContext は Servlet 仕様で定義されているクラスです。

ルートリソースクラスのフィールドにインジェクトされる javax.servlet.ServletContext の使用例を次に示します。

```
package com.sample.resources;

import javax.servlet.ServletContext;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Context;

//ルートリソースクラス
@Path("/root")
public class Resource {

  //Contextアノテーションを使用してServletContextをインジェクトするフィールド
  private @Context ServletContext context;

  //リソースメソッド
  @GET
  public String getValue() {
    return this.context.getInitParameter("Hitachi");
  }
}
```

コンテキストスコープの初期化パラメタ（context-param 要素）を含む web.xml を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
```

```
  ...
  <context-param>
    <param-name>TestParam</param-name>
    <param-value>TestValue</param-value>
  </context-param>
</web-app>
```

ルートリソースクラスcom.sample.resources.Resource を含む Web アプリケーション（WAR ファイル）のコンテキストルートが"example"で，Web アプリケーションが"sample.com"というホストで公開されているとします。この例で，URL"http://sample.com/example/root"に対する HTTP GET リクエストでは，まず context フィールドに javax.servlet.ServletContext コンテキストがインジェクトされ，その後 HTTP GET リクエストを処理できる getValue()メソッドが呼び出されます。このため，getValue()メソッドで context はフィールドからコンテキストスコープの初期化パラメタ"TestParam"を取得すると"TestValue"という値が取得されます。

24.4.8 javax.servlet.http.HttpServletRequest

javax.servlet.http.HttpServletRequest は Servlet 仕様で定義されているクラスです。

ルートリソースクラスのフィールドにインジェクトされる javax.servlet.http.HttpServletRequest の使用例を次に示します。

```
package com.sample.resources;

import javax.servlet.http.HttpServletRequest;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Context;

//ルートリソースクラス
@Path(”/root”)
public class Resource {

  //Contextアノテーションを使用してHttpServletRequestをインジェクトするフィールド
  private @Context HttpServletRequest httpRequest;

  //リソースメソッド
  @GET
  public String getValue() {
    return this.httpRequest.getParameter(”Hitachi”);

  }
}
```

ルートリソースクラスcom.sample.resources.Resource を含む Web アプリケーション（WAR ファイル）のコンテキストルートが"example"で，Web アプリケーションが"sample.com"というホストで公開されているとします。この例で，URL"http://sample.com/example/root?TestParam=TestValue"に対する HTTP GET リクエストでは，まず HttpRequest フィールドに javax.servlet.http.HttpServletRequest コンテキストがインジェクトされ，その後 HTTP GET リクエストを処理できる getValue()メソッドが呼

び出されます。このため，getValue()メソッドで httpRequest フィールドからリクエストパラメタ"TestParam"を取得すると，"TestValue"という値が取得されます。

24.4.9 javax.servlet.http.HttpServletResponse

javax.servlet.http.HttpServletResponse は Servlet 仕様で定義されているクラスです。

ルートリソースクラスのフィールドにインジェクトされる javax.servlet.http.HttpServletResponse の使用例を次に示します。

```
package com.sample.resources;

import java.io.IOException;
import javax.servlet.http.HttpServletResponse;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Context;

//ルートリソースクラス
@Path(”/root”)
public class Resource {

  //Contextアノテーションを使用してHttpServletResponseをインジェクトするフィールド
  private @Context HttpServletResponse httpResponse;

  //リソースメソッド
  @GET
  public void getValue() throws IOException {

    String entity = ”Response mentioned using HttpServletResponse”;

    httpResponse.setHeader(”abc”,”xyz”);
    httpResponse.getOutputStream().write(entity.getBytes());
    httpResponse.getOutputStream().flush();
    httpResponse.getOutputStream().close();

    return ;
  }
}
```

ルートリソースクラスcom.sample.resources.Resource を含む Web アプリケーション（WAR ファイル）のコンテキストルートが"example"で，Web アプリケーションが"sample.com"というホストで公開されているとします。この例で，URL"http://sample.com/example/root"に対する HTTP GET リクエストでは，まず HttpResponse フィールドに javax.servlet.http.HttpServletResponse がインジェクトされ，その後 HTTP GET リクエストを処理できる getValue()メソッドが呼び出されます。このため，HttpResponse クラスが応答の構築に使用される個所と，ルートリソースクラス com.sample.resources.Resource の getValue()メソッドは同じになります。

25 RESTful Web サービス用クライアント API のサポート範囲

この章では，RESTful Web サービス用クライアント API の仕様およびサポート範囲について説明します。

なお，この章では RESTful Web サービス用クライアント API を「クライアント API」と呼びます。

25.1　クライアント API のインタフェースおよびクラスのサポート範囲

ここでは，クライアント API のインタフェースおよびクラスのサポート範囲について説明します。クライアント API のインタフェースおよびクラスのサポート範囲を次の表に示します。

表 25–1　クライアント API のインタフェースおよびクラスのサポート範囲

項番	インタフェースまたはクラス	コンストラクタ／メソッド／フィールド
com.cosminexus.jersey.api.client パッケージ		
1	Client	create()
2		create(ClientConfig cc)
3		destroy()
4		getProperties()
5		handle(ClientRequest request)
6		resource(String u)
7		resource(URI u)
8		setChunkedEncodingSize(Integer chunkSize)
9		setConnectTimeout(Integer interval)
10		setFollowRedirects(Boolean redirect)
11		setReadTimeout(Integer interval)
12	ClientHandlerException	親クラスのメソッド※1
13	ClientRequest	clone()
14		create()
15		getEntity()
16		getHeaders()
17		getHeaderValue(Object headerValue)
18		getMethod()
19		getProperties()
20		getPropertyAsFeature(String name)
21		getPropertyAsFeature(String name, boolean defaultValue)
22		getURI()
23		setEntity(Object entity)
24		setMethod(String method)
25		setURI(java.net.URI uri)

項番	インタフェースまたはクラス	コンストラクタ／メソッド／フィールド
26	ClientRequest.Builder	accept(MediaType... types)
27		accept(String... types)
28		acceptLanguage(Locale... locales)
29		acceptLanguage(String... locales)
30		build(URI uri, String method)
31		cookie(Cookie cookie)
32		entity(Object entity)
33		entity(Object entity, MediaType type)
34		entity(Object entity, String type)
35		header(String name, Object value)
36		type(MediaType type)
37		type(String type)
38	ClientResponse	bufferEntity()
39		close()
40		getAllow()
41		getClient()
42		getClientResponseStatus()
43		getCookies()
44		getEntity(Class<T> c)
45		getEntity(GenericType<T> gt)
46		getEntityInputStream()
47		getEntityTag()
48		getHeaders()
49		getLanguage()
50		getLastModified()
51		getLength()
52		getLocation()
54		getResponseDate()
55		getStatus()
56		getType()
57		hasEntity()

項番	インタフェースまたはクラス	コンストラクタ／メソッド／フィールド
58	ClientResponse.Status	ACCEPTED
59		BAD_GATEWAY
60		BAD_REQUEST
61		CONFLICT
62		CREATED
63		EXPECTATION_FAILED
64		FORBIDDEN
65		FOUND
66		GATEWAY_TIMEOUT
67		GONE
68		HTTP_VERSION_NOT_SUPPORTED
69		INTERNAL_SERVER_ERROR
70		LENGTH_REQUIRED
71		METHOD_NOT_ALLOWED
72		MOVED_PERMANENTLY
73		NO_CONTENT
74		NON_AUTHORITIVE_INFORMATION
75		NOT_ACCEPTABLE
76		NOT_FOUND
77		NOT_IMPLEMENTED
78		NOT_MODIFIED
79		OK
80		PARTIAL_CONTENT
81		PAYMENT_REQUIRED
82		PRECONDITION_FAILED
83		PROXY_AUTHENTICATION_REQUIRED
84		REQUEST_ENTITY_TOO_LARGE
85		REQUEST_TIMEOUT
86		REQUEST_URI_TOO_LONG
87		REQUESTED_RANGE_NOT_SATIFIABLE
88		RESET_CONTENT

項番	インタフェースまたはクラス	コンストラクタ／メソッド／フィールド
89		SEE_OTHER
90		SERVICE_UNAVAILABLE
91		TEMPORARY_REDIRECT
92		UNAUTHORIZED
93		UNSUPPORTED_MEDIA_TYPE
94		USE_PROXY
95		fromStatusCode(int statusCode)
96		getFamily()
97		getReasonPhrase()
98		getStatusCode()
99		toString()
100		valueOf(String name)
101		values()
102	GenericType	GenericType()
103		GenericType(Type genericType)
104		getRawClass()
105		getType()
106	UniformInterfaceException	getResponse()
107	WebResource	accept(MediaType... types)
108		accept(String... types)
109		acceptLanguage(Locale... locales)
110		acceptLanguage(String... locales)
111		cookie(Cookie cookie)
112		delete()
113		delete(Class<T> c)
114		delete(Class<T> c, Object requestEntity)
115		delete(GenericType<T> gt)
116		delete(GenericType<T> gt, Object requestEntity)
117		delete(Object requestEntity)
118		entity(Object entity)
119		entity(Object entity, MediaType type)

項番	インタフェースまたはクラス	コンストラクタ／メソッド／フィールド
120		entity(Object entity, String type)
121		get(Class<T> c)
122		get(GenericType<T> gt)
123		getRequestBuilder()
124		getURI()
125		getUriBuilder()
126		head()
127		header(String name, Object value)
128		method(String method)
129		method(String method, Class<T> c)
130		method(String method, Class<T> c, Object requestEntity)
131		method(String method, GenericType<T> gt)
132		method(String method, GenericType<T> gt, Object requestEntity)
133		method(String method, Object requestEntity)
134		options(Class<T> c)
135		options(GenericType<T> gt)
136		path(String path)
137		post()
138		post(Class<T> c)
139		post(Class<T> c, Object requestEntity)
140		post(GenericType<T> gt)
141		post(GenericType<T> gt, Object requestEntity)
142		post(Object requestEntity)
143		put()
144		put(Class<T> c)
145		put(Class<T> c, Object requestEntity)
146		put(GenericType<T> gt)
147		put(GenericType<T> gt, Object requestEntity)
148		put(Object requestEntity)
149		queryParam(String key, String value)

項番	インタフェースまたはクラス	コンストラクタ／メソッド／フィールド
150		queryParams(MultivaluedMap<String, String> params)
151		type(MediaType type)
152		type(String type)
153		uri(java.net.URI uri)
154	WebResource.Builder	accept(MediaType... types)
155		accept(String... types)
156		acceptLanguage(Locale... locales)
157		acceptLanguage(String... locales)
158		cookie(Cookie cookie)
159		delete()
160		delete(Class<T> c)
161		delete(Class<T> c, Object requestEntity)
162		delete(GenericType<T> gt)
163		delete(GenericType<T> gt, Object requestEntity)
164		delete(Object requestEntity)
165		entity(Object entity)
166		entity(Object entity, MediaType type)
167		entity(Object entity, String type)
168		get(Class<T> c)
169		get(GenericType<T> gt)
170		head()
171		header(String name, Object value)
172		method(String method)
173		method(String method, Class<T> c)
174		method(String method, Class<T> c, Object requestEntity)
175		method(String method, GenericType<T> gt)
176		method(String method, GenericType<T> gt, Object requestEntity)
177		method(String method, Object requestEntity)
178		options(Class<T> c)
179		options(GenericType<T> gt)

項番	インタフェースまたはクラス	コンストラクタ／メソッド／フィールド
180		post()
181		post(Class<T> c)
182		post(Class<T> c, Object requestEntity)
183		post(GenericType<T> gt)
184		post(GenericType<T> gt, Object requestEntity)
185		post(Object requestEntity)
186		put()
187		put(Class<T> c)
188		put(Class<T> c, Object requestEntity)
189		put(GenericType<T> gt)
190		put(GenericType<T> gt, Object requestEntity)
191		put(Object requestEntity)
192		type(MediaType type)
193		type(String type)
com.cosminexus.jersey.api.client.config パッケージ		
194	DefaultClientConfig	PROPERTY_BUFFER_RESPONSE_ENTITY_ON_EXCEPTION
195		PROPERTY_CHUNKED_ENCODING_SIZE
196		PROPERTY_CONNECT_TIMEOUT
197		PROPERTY_FOLLOW_REDIRECTS
198		PROPERTY_READ_TIMEOUT
199		getPropertyAsFeature(String featureName)
200		getFeatures()
201		getFeature(String featureName)
202		getProperties()
203		getProperty(String propertyName)
com.cosminexus.jersey.client.urlconnection パッケージ		
204	HTTPSProperties	PROPERTY_HTTPS_PROPERTIES
205		HTTPSProperties()
206		HTTPSProperties(HostnameVerifier hv)
207		HTTPSProperties(HostnameVerifier hv, SSLContext c)
208		getHostnameVerifier()

項番	インタフェースまたはクラス	コンストラクタ／メソッド／フィールド
209		getSSLContext()
com.cosminexus.jersey.core.util パッケージ		
210	MultivaluedMapImpl※2	MultivaluedMapImpl()
211		add(String key, String value)
212		getFirst(String key)
213		putSingle(String key, String value)

注※1

ClientHandlerException クラスは RuntimeException クラスの派生クラスです。ClientHandlerException クラスは RuntimeException クラスのメソッドの範囲で使用してください。

注※2

MultivaluedMapImpl クラスのコンストラクタおよびメソッドの仕様については，JAX-RS API のドキュメントを参照してください。注意事項については「25.14　MultivaluedMapImpl クラスのコンストラクタおよびメソッド仕様と注意事項」を参照してください。

25.1.1　サポートするプロパティとフィーチャ

ここでは，クライアント API によってサポートされているプロパティとフィーチャについて説明します。

(1)　フィーチャ

JAX-RS エンジンがサポートしているフィーチャとデータ型を次の表に示します。

表 25–2　JAX-RS エンジンがサポートしているフィーチャとデータ型

項番	フィーチャ名	データ型
1	com.sun.jersey.api.json.POJOMappingFeature (JSONConfiguration.FEATURE_POJO_MAPPING)	Boolean

このフィーチャは変更可能なフィーチャマップに追加して使用します。使用方法の詳細は，「getFeatures() メソッド」を参照してください。

(2)　プロパティ

JAX-RS エンジンがサポートしているプロパティとデータ型を次の表に示します。

表 25–3　JAX-RS エンジンがサポートしているプロパティとデータ型

項番	プロパティ	データ型
1	com.sun.jersey.client.property.followRedirects (ClientConfig.PROPERTY_FOLLOW_REDIRECTS)	Boolean
2	com.sun.jersey.client.property.readTimeout (ClientConfig.PROPERTY_READ_TIMEOUT)	Integer
3	com.sun.jersey.client.property.connectTimeout (ClientConfig.PROPERTY_CONNECT_TIMEOUT)	Integer
4	com.sun.jersey.client.property.chunkedEncodingSize (ClientConfig.PROPERTY_CHUNKED_ENCODING_SIZE)	Integer
5	com.sun.jersey.client.property .bufferResponseEntityOnException (ClientConfig.PROPERTY_BUFFER_RESPONSE_ENTITY_ON_EXCEPTION)	Boolean
6	com.sun.jersey.client.impl.urlconnection .httpsProperties (HTTPSProperties.PROPERTY_HTTPS_PROPERTIES)	HTTPSProperties

これらのプロパティは変更可能なプロパティマップに追加して使用します。使用方法の詳細は次のメソッドを参照してください。

- Client クラス
  getProperties()メソッド
- ClientRequest クラス
  getProperties()メソッド
- DefaultClientConfig クラス
  getProperties()メソッド

25.1.2　ClientRequest クラスと Web リソースクラスに含まれる情報

ClientRequest クラスと Web リソースクラスには次の情報が含まれます。

表 25–4　ClientRequest クラスと Web リソースクラスに含まれる情報

クラス	含まれる情報
ClientRequest クラス	Web リソースの URI
	HTTP メソッド名
	HTTP リクエストのエンティティボディ

クラス	含まれる情報
	HTTP ヘッダ
	プロパティマップ
Web リソースクラス	Web リソースの URI
	HTTP リクエストのエンティティボディ
	HTTP ヘッダ

25.2　Client クラスのメソッド仕様と注意事項

ここでは，Client クラスのメソッドの仕様，および利用する場合の注意事項について説明します。

create()メソッド

説明

クライアント（Client オブジェクト）を生成します。

構文

public static Client create()

パラメタ

ありません。

戻り値

生成されたクライアントを返します。

注意事項

- このメソッドの実行結果は，生成しただけの DefaultClientConfig オブジェクトを cc パラメタに指定して create(ClientConfig cc)メソッドを呼び出した場合と同じです。

create(ClientConfig cc)メソッド

説明

指定された設定でクライアント（Client オブジェクト）を生成します。

構文

public static Client create(ClientConfig cc)

パラメタ

cc

クライアントを設定します。

戻り値

生成されたクライアントを返します。

注意事項

- 特定のプロパティ・フィーチャで Client オブジェクトを初期化したい場合に，このメソッドを使用します。プロパティとフィーチャについては「25.1.1　サポートするプロパティとフィーチャ」を参照してください。プロパティとフィーチャを設定する方法の詳細については，「getFeatures()メソッド」および「getProperties()メソッド」を参照してください。
- cc パラメタには DefaultClientConfig オブジェクトだけを指定してください。

destroy()メソッド

説明

クライアントを破棄します。クライアントに関連するすべてのシステムリソースが破棄されます。

このメソッドは，受信途中の応答がないときに呼び出してください。それ以外の場合，動作は保証されません。

このメソッドを呼び出したあとは，Client オブジェクトを再利用してはいけません。再利用した場合，動作は保証されません。

構文

public void destroy()

パラメタ

ありません。

戻り値

ありません。

注意事項

ありません。

getProperties()メソッド

説明

変更可能なプロパティマップを取得します。

構文

public Map<String,Object> getProperties()

パラメタ

ありません。

戻り値

プロパティマップを返します。

注意事項

- 戻り値のプロパティマップには，プロパティを追加または変更できます。追加または変更できるプロパティについては，「25.1.1　サポートするプロパティとフィーチャ」を参照してください。プロパティについては「11.4.3　プロパティとフィーチャの設定」も参照してください。

getProperties()メソッドの使用例を次に示します。

```
// Clientオブジェクトを生成する
Client client = Client.create();
// 読み込みタイムアウト値を設定する
client.getProperties().put(ClientConfig.PROPERTY_READ_TIMEOUT, 10000);

// HTTPリクエストを生成する
ClientRequest cRequest = ClientRequest.create().build(new URI("http://test.com/"), "GET");
try{
  // HTTPレスポンスをClientResponseオブジェクトとして受信する
  ClientResponse cResponse = client.handle(cRequest);
} catch(ClientHandlerException e){
  // 適切な処理を実行する
}
```

この例では，まず，getProperties メソッドで変更可能なプロパティマップを取得し，読み込みタイムアウト値を 10,000 ミリ秒に設定しています。次に ClientRequest オブジェクトを作成し，Client クラスの handle メソッドを利用して HTTP 通信を行い，HTTP レスポンスを ClientResponse オブジェクトとして受信しています。HTTP レスポンスが完全に読み込まれるまでに読み込みタイムアウトが発生した場合，エラーとなり（KDJJ18888-E），SocketTimeoutException をラップした ClientHandlerException がスローされます。

handle(ClientRequest request)メソッド

説明

指定された ClientRequest オブジェクトの内容で HTTP リクエストを処理し，ClientResponse オブジェクトとして HTTP レスポンスを返します。

構文

public ClientResponse handle(ClientRequest request)

throws ClientHandlerException

パラメタ

request

HTTP リクエストです。

戻り値

HTTP レスポンスを返します。

例外

ClientHandlerException

処理中に問題が発生した場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

注意事項

- HTTP リクエストおよび HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。例外が発生する条件は環境などによって異なります。次に示すそれぞれのメソッドの注意事項も参照してください。

  Client クラス

  - getProperties()メソッド
  - resource(String u)メソッド
  - resource(URI u)メソッド
  - setChunkedEncodingSize(Integer chunkSize)メソッド
  - setConnectTimeout(Integer interval)メソッド
  - setFollowRedirects(Boolean redirect)メソッド
  - setReadTimeout(Integer interval)メソッド

  ClientRequest クラス

  - getProperties()メソッド

setEntity(Object entity)メソッド
setMethod(String method)メソッド
setURI(java.net.URI uri)メソッド

ClientRequest.Builder クラス
accept(MediaType... types)メソッド
accept(String... types)メソッド
acceptLanguage(Locale... locales)メソッド
acceptLanguage(String... locales)メソッド
build(URI uri, String method)メソッド
cookie(Cookie cookie)メソッド
entity(Object entity)メソッド
entity(Object entity, MediaType type)メソッド
entity(Object entity, String type)メソッド
header(String name, Object value)メソッド
type(MediaType type)メソッド
type(String type)メソッド

- handle(ClientRequest request)メソッドの使用例については「getProperties()メソッド」を参照してください。

resource(String u)メソッド

説明

クライアント（Client オブジェクト）から WebResource オブジェクトを生成します。

構文

public WebResource resource(String u)

パラメタ

u

Web リソースの URI です。

戻り値

WebResource オブジェクトを返します。

注意事項

- u パラメタには null 以外の有効な URI を指定してください。null を指定した場合，NullPointerException がスローされます。それ以外の不正な値を指定した場合の動作は，Java SE の java.net.URI クラスの create()メソッドと同じです。詳細については，JDK のドキュメントを参照してください。

resource(URI u)メソッド

説明

クライアント（Client オブジェクト）から Web リソースを生成します。

構文

public WebResource resource(URI u)

パラメタ

u

Web リソースの URI です。

戻り値

WebResource オブジェクトを返します。

注意事項

- u パラメタには有効な URI を指定してください。また，URI には RFC 2396 で定義されている文字を使用してください。u パラメタは自動では URI エンコーディングされません。必要に応じて URI エンコードした値を u パラメタに指定してください。

setChunkedEncodingSize(Integer chunkSize)メソッド

説明

HTTP リクエストのエンティティを，指定されたチャンクサイズでチャンク転送エンコーディングして送信するように設定します。

このメソッドは，getProperties()メソッドで取得したプロパティのマップに ClientConfig.PROPERTY_CHUNKED_ENCODING_SIZE プロパティを設定する場合と機能は同じです。

構文

public void setChunkedEncodingSize(Integer chunkSize)

パラメタ

chunkSize

チャンクサイズを指定します。0 以下の値が指定された場合，デフォルト値が適用されます。

戻り値

ありません。

注意事項

- chunkSize パラメタで設定した値によるクライアントの動作は，HttpURLConnection クラスの setChunkedStreamingMode()メソッドと同じです。詳細については，JDK のドキュメントを参照してください。
- JAX-RS エンジンは chunkSize パラメタに設定された値を検証しません。HTTP 通信に先立って，そのまま HttpURLConnectioin にコピーします。chunkSize パラメタには，Java SE の HttpURLConnection クラスの setChunkedStreamingMode()メソッドの説明に従って値を設定してください。
- クライアントのプロパティマップに，すでに ClientConfig.PROPERTY_CHUNKED_ENCODING_SIZE で指定した値が含まれている場合，chunkSize パラメタで指定された値が上書きされます。
- chunkSize パラメタで指定された値によって HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

setConnectTimeout(Integer interval)メソッド

説明

接続タイムアウト値をミリ秒単位で設定します。

このメソッドは，getProperties()メソッドで取得したプロパティのマップに ClientConfig.PROPERTY_CONNECT_TIMEOUT プロパティを設定する場合と機能は同じです。

構文

public void setConnectTimeout(Integer interval)

パラメタ

interval

接続タイムアウト値です。null または 0 が指定された場合，タイムアウトされません。

戻り値

ありません。

注意事項

- interval パラメタで設定した値によるクライアントの動作は，HttpURLConnection クラスの setConnectTimeout()メソッドと同じです。詳細については，JDK のドキュメントを参照してください。
- JAX-RS エンジンは interval パラメタに設定された値を検証しません。HTTP 通信に先立って，そのまま HttpURLConnectioin にコピーします。interval パラメタには，Java SE の HttpURLConnection クラスの setConnectTimeout()メソッドの説明に従って値を設定してください。
- クライアントのプロパティマップに，すでに ClientConfig.PROPERTY_CONNECT_TIMEOUT で指定した値が含まれている場合，interval パラメタで指定された値が上書きされます。
- interval パラメタで指定された値によって HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

setFollowRedirects(Boolean redirect)メソッド

説明

HTTP リダイレクトに自動的に従うべきかどうかを設定します。

このメソッドは，getProperties()メソッドで取得したプロパティのマップに ClientConfig.PROPERTY_FOLLOW_REDIRECTS プロパティを設定する場合と機能は同じです。

構文

public void setFollowRedirects(Boolean redirect)

パラメタ

redirect

true の場合，クライアントはステータスコードが 300 番台の HTTP レスポンスに記述された URI に自動的にリダイレクトされます。

戻り値

ありません。

注意事項

- redirect パラメタで設定した値によるクライアントの動作は，HttpURLConnection クラスの setInstanceFollowRedirects()メソッドと同じです。詳細については，JDK のドキュメントを参照してください。
- JAX-RS エンジンは redirect パラメタに設定された値を検証しません。HTTP 通信に先立って，そのまま HttpURLConnectioin にコピーします。redirect パラメタには，Java SE の HttpURLConnection クラスの setInstanceFollowRedirects()メソッドの説明に従って値を設定してください。
- クライアントのプロパティマップに，すでに ClientConfig.PROPERTY_FOLLOW_REDIRECTS で指定した値が含まれている場合，redirect パラメタで指定された値が上書きされます。
- redirect パラメタで指定された値によって HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

setReadTimeout(Integer interval)メソッド

説明

読み込みタイムアウト値をミリ秒単位で設定します。

このメソッドは，getProperties()メソッドで取得したプロパティのマップに ClientConfig.PROPERTY_READ_TIMEOUT プロパティを設定する場合と機能は同じです。

構文

public void setReadTimeout(Integer interval)

パラメタ

interval

読み込みタイムアウト値です。null または 0 が指定された場合，タイムアウトされません。

戻り値

ありません。

注意事項

- interval パラメタで設定した値によるクライアントの動作は，HttpURLConnection クラスの setReadTimeout()メソッドと同じです。詳細については，JDK のドキュメントを参照してください。

- JAX-RS エンジンは interval パラメタに設定された値を検証しません。HTTP 通信に先立って，そのまま HttpURLConnectioin にコピーします。interval パラメタには，Java SE の HttpURLConnection クラスの setReadTimeout()メソッドの説明に従って値を設定してください。
- クライアントのプロパティマップに，すでに ClientConfig.PROPERTY_READ_TIMEOUT で指定した値が含まれている場合，interval パラメタで指定された値が上書きされます。
- interval パラメタで指定された値によって HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

25.3　ClientHandlerException クラスのメソッド仕様と注意事項

ClientHandlerException クラスは RuntimeException クラスの派生クラスです。
ClientHandlerException クラスは RuntimeException クラスのメソッドの範囲で使用してください。

25.4　ClientRequest クラスのメソッド仕様と注意事項

ここでは，ClientRequest クラスのメソッドの仕様，および利用する場合の注意事項について説明します。

clone()メソッド

説明

HTTP リクエスト（ClientRequest オブジェクト）を複製します。

構文

public abstract ClientRequest clone()

パラメタ

ありません。

戻り値

複製された HTTP リクエスト（ClientRequest オブジェクト）を返します。

注意事項

- このメソッドによって複製される ClientRequest オブジェクトのエンティティは次のとおりです。
  - Web リソースの URI
  - HTTP メソッド名
  - HTTP リクエスト
  - HTTP ヘッダ
- ClientRequest オブジェクトのプロパティマップは複製されません。必要に応じて，複製された ClientRequest オブジェクトに追加してください。

create()メソッド

説明

ClientRequest オブジェクトを構築するためのビルダ（ClientRequest.Builder オブジェクト）を生成します。

構文

public static final ClientRequest.Builder create()

パラメタ

ありません。

戻り値

ビルダ（ClientRequest.Builder オブジェクト）を返します。

注意事項

- 使用例については「getProperties()メソッド」を参照してください。

getEntity()メソッド

説明

HTTP リクエストのエンティティを取得します。

構文

public abstract Object getEntity()

パラメタ

ありません。

戻り値

HTTP リクエストのエンティティを返します。

注意事項

- ClientRequest クラスの setEntity()メソッドで設定したエンティティの値が返ります。
- ClientRequest クラスの setEntity()メソッドでエンティティが設定されていなかったり，null が指定されていたりした場合，null が返ります。

getHeaders()メソッド

説明

HTTP ヘッダのマップを取得します。

構文

public abstract MultivaluedMap<String,Object> getHeaders()

パラメタ

ありません。

戻り値

HTTP ヘッダのマップを返します。

注意事項

- ClientRequest オブジェクトに設定されている HTTP ヘッダが MultivaluedMap<java.lang.String,java.lang.Object>の形式で返ります。HTTP ヘッダは ClientRequest オブジェクトの次のメソッドでも設定できます。
  - accept(MediaType... types)メソッド
  - accept(String... types)メソッド
  - acceptLanguage(Locale... locales)メソッド
  - acceptLanguage(String... locales)メソッド
  - cookie(Cookie cookie)メソッド
  - entity(Object entity, MediaType type)メソッド
  - entity(Object entity, String type)メソッド
  - header(String name, Object value)メソッド
  - type(MediaType type)メソッド
  - type(String type)メソッド
- HTTP ヘッダが一つも設定されていない場合，空のマップが返ります。

getHeaderValue(Object headerValue)メソッド

説明

指定された HTTP ヘッダの値を文字列に変換します。

構文

public static String getHeaderValue(Object headerValue)

パラメタ

headerValue

HTTP ヘッダの値です。

戻り値

変換された文字列を返します。

注意事項

- 戻り値は，パラメタに指定されたオブジェクトの toString()メソッドが返す値です。

getMethod()メソッド

説明

HTTP メソッドを取得します。

構文

public abstract String getMethod()

パラメタ

ありません。

戻り値

HTTP メソッドを返します。

注意事項

- ClientRequest オブジェクトに設定されている HTTP メソッドの名前が文字列で返ります。HTTP メソッドは次のメソッドで設定できます。
  - ClientRequest クラスの setMethod(String method)メソッド
  - ClientRequest.Builder クラスの build(URI uri, String method)メソッド
- HTTP メソッドに null が設定されている場合，null が返ります。

getProperties()メソッド

説明

変更可能なプロパティマップを取得します。

構文

public abstract Map<String,Object> getProperties()

パラメタ

ありません。

戻り値

プロパティマップを返します。

注意事項

- 戻り値のプロパティマップにはプロパティを追加または変更できます。追加または変更できるプロパティについては，「25.1.1　サポートするプロパティとフィーチャ」を参照してください。プロパティについては「11.4.3　プロパティとフィーチャの設定」も参照してください。

getProperties()メソッドの使用例を次に示します。

```
// Clientオブジェクトを生成する
Client client = Client.create();
// HTTPリクエストを生成する
ClientRequest cRequest = ClientRequest.create().build(new URI("http://test.com/"), "GET");
// 読み込みタイムアウト値を設定する
cRequest.getProperties().put(ClientConfig.PROPERTY_READ_TIMEOUT, 10000);
try{
  // HTTPレスポンスをClientResponseオブジェクトとして受信する
  ClientResponse cResponse = client.handle(cRequest);
} catch(ClientHandlerException e){
  // 適切な処理を実行する
}
```

この例では，ClientRequest オブジェクトを生成したあと，ClientRequest オブジェクトの getProperties() メソッドで変更可能なプロパティマップを取得し，読み込みタイムアウト値を 10,000 ミリ秒に設定しています。次に，その ClientRequest オブジェクトを指定して Client クラスの handle メソッドを呼び出します。ClientRequest オブジェクトのプロパティは Client オブジェクトにコピーされます。ClientResponse オブジェクトとして HTTP レスポンスを受信するとき，完全に読み込まれるまでに読み込みタイムアウトが発生した場合，エラーとなり（KDJJ18888-E），SocketTimeoutException をラップした ClientHandlerException がスローされます。

getPropertyAsFeature(String name)メソッド

説明

プロパティマップから Boolean のプロパティをフィーチャとして取得します。

構文

public boolean getPropertyAsFeature(String name)

パラメタ

name
　プロパティ名です。

戻り値

指定されたプロパティが true 値を持つ Boolean の場合 true，そうでない場合は false が返ります。

注意事項

ありません。

getPropertyAsFeature(String name, boolean defaultValue)メソッド

説明

プロパティマップから Boolean のプロパティをフィーチャとして取得します。

構文

public boolean getPropertyAsFeature(String name,

boolean defaultValue)

パラメタ

name
　プロパティ名です。

defaultValue
　指定されたプロパティが存在しない場合に仮定するデフォルト値です。

戻り値

指定されたプロパティが true 値を持つ Boolean の場合 true，そうでない場合は指定されたデフォルト値が返ります。

注意事項

ありません。

getURI()メソッド

説明

HTTP リクエストの URI を取得します。URI にはリクエストの対象となるリソースを識別する情報が含まれます。

構文

public abstract URI getURI()

パラメタ

ありません。

戻り値

HTTP リクエストの URI を返します。

注意事項

- ClientRequest オブジェクトに設定されている URI が返ります。URI は次のメソッドで設定できます。
  - ClientRequest クラスの setURI(java.net.URI uri)メソッド
  - ClientRequest.Builder クラスの build(URI uri, String method)メソッド
- URI に null が設定されている場合，null が返ります。

setEntity(Object entity)メソッド

説明

エンティティを設定します。

構文

public abstract void setEntity(Object entity)

パラメタ

entity

エンティティです。

戻り値

ありません。

注意事項

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。

setMethod(String method)メソッド

説明

HTTP メソッドを設定します。

構文

public abstract void setMethod(String method)

パラメタ

method

HTTP メソッド名です。

戻り値

ありません。

注意事項

- method パラメタには，すべて大文字で「GET」「HEAD」「POST」「OPTIONS」「PUT」または「DELETE」のどれかを指定してください。

setURI(java.net.URI uri)メソッド

説明

HTTP リクエストの URI を設定します。URI にはリクエストの対象となるリソースを識別する情報を正しく指定してください。

構文

public abstract void setURI(URI uri)

パラメタ

uri

HTTP リクエストの URI です。

戻り値

ありません。

注意事項

- uri パラメタには有効な URI を指定してください。また，URI には RFC 2396 で定義されている文字を使用してください。uri パラメタは自動では URI エンコーディングされません。必要に応じて URI エンコードした値を uri パラメタに指定してください。

25.5　ClientRequest.Builder クラスのメソッド仕様と注意事項

ここでは，ClientRequest.Builder クラスのメソッドの仕様，および利用する場合の注意事項について説明します。

accept(MediaType... types)メソッド

説明

受信できる MIME メディアタイプを追加します。

構文

public T accept(MediaType... types)

パラメタ

types

受信できる MIME メディアタイプの配列です。

戻り値

WebResource オブジェクトを構築するためのビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- types パラメタに含まれる null でない値が Accept HTTP ヘッダに追加されます。null の値は無視され Accept HTTP ヘッダには追加されません。
- Accept HTTP ヘッダには，MediaType オブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは toString()メソッドによって返される値を検証しません。標準仕様に従って値を指定してください。
- Accept HTTP ヘッダの値は次のメソッドでも追加できます。
  - ClientRequest.Builder クラス
    accept(String... types)メソッド
    header(String name, Object value)メソッド
  - WebResource クラス
    accept(MediaType... types)メソッド
    accept(String... types)メソッド
    header(String name, Object value)メソッド
  - WebResource.Builder クラス
    accept(MediaType... types)メソッド

accept(String... types)メソッド
header(String name, Object value)メソッド

これらのメソッドおよびaccept(MediaType... types)メソッドで，受信できるMIMEメディアタイプが一つも追加されない場合の動作は，HTTP通信を行う前に，HttpURLConnectionオブジェクトにAccept HTTPヘッダを追加しない場合の動作と同様です。

accept(String... types)メソッド

説明

受信できるMIMEメディアタイプを追加します。

構文

public T accept(String... types)

パラメタ

types
受信できるMIMEメディアタイプの配列です。

戻り値

WebResourceオブジェクトを構築するためのビルダ（WebResource.Builderオブジェクト）を返します。

注意事項

- typesパラメタに含まれるnullでない値がAccept HTTPヘッダに追加されます。nullの値は無視されAccept HTTPヘッダには追加されません。
- Accept HTTPヘッダには，指定された値がそのまま設定されます。JAX-RSエンジンは指定された値を検証しません。標準仕様に従って値を指定してください。
- Accept HTTPヘッダの値は次のメソッドでも追加できます。
  - ClientRequest.Builde クラス
    accept(MediaType... types)メソッド
    header(String name, Object value)メソッド
  - WebResource クラス
    accept(MediaType... types)メソッド
    accept(String... types)メソッド
    header(String name, Object value)メソッド
  - WebResource.Builder クラス

accept(MediaType... types)メソッド
accept(String... types)メソッド
header(String name, Object value)メソッド

これらのメソッドおよびaccept(String... types)メソッドで，受信できるMIMEメディアタイプが一つも追加されない場合の動作は，HTTP通信を行う前に，HttpURLConnectionオブジェクトにAccept HTTPヘッダを追加しない場合の動作と同様です。

acceptLanguage(Locale... locales)メソッド

説明

受信できる言語を追加します。

構文

public T acceptLanguage(Locale... locales)

パラメタ

locales

受信できる言語の配列です。

戻り値

WebResourceオブジェクトを構築するためのビルダ（WebResource.Builderオブジェクト）を返します。

注意事項

- localesパラメタに含まれるnullでない値がAccept-Language HTTPヘッダに追加されます。nullの値は無視されAccept-Language HTTPヘッダには追加されません。
- Accept-Language HTTPヘッダには，LocaleオブジェクトのtoString()メソッドによって返される値が設定されます。JAX-RSエンジンはtoString()メソッドによって返される値を検証しません。標準仕様に従って値を指定してください。
- Accept-Language HTTPヘッダの値は次のメソッドでも追加できます。
  - ClientRequest.Builderクラス
    header(String name, Object value)メソッド
    acceptLanguage(String... locales)メソッド
  - WebResourceクラス
    acceptLanguage(Locale... locales)メソッド
    acceptLanguage(String... locales)メソッド
    header(String name, Object value)メソッド

- WebResource.Builder クラス
  acceptLanguage(Locale... locales)メソッド
  acceptLanguage(String... locales)メソッド
  header(String name, Object value)メソッド

これらのメソッドおよびacceptLanguage(Locale... locales)メソッドで，受信できる言語が一つも追加されない場合の動作は，HTTP 通信を行う前に，HttpURLConnection オブジェクトに Accept-Language HTTP ヘッダを追加しない場合の動作と同様です。

acceptLanguage(String... locales)メソッド

説明

受信できる言語を追加します。

構文

public T acceptLanguage(String... locales)

パラメタ

locales

受信できる言語の配列です。

戻り値

WebResource オブジェクトを構築するためのビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- locales パラメタに含まれる null でない値が Accept-Language HTTP ヘッダに追加されます。null の値は無視され Accept-Language HTTP ヘッダには追加されません。
- Accept-Language HTTP ヘッダには，指定された値がそのまま設定されます。JAX-RS エンジンは指定された値を検証しません。標準仕様に従って値を指定してください。
- Accept-Language HTTP ヘッダの値は次のメソッドでも追加できます。
  - ClientRequest.Builder クラス
    acceptLanguage(Locale... locales)メソッド
    header(String name, Object value)メソッド
  - WebResource クラス
    acceptLanguage(Locale... locales)メソッド
    acceptLanguage(String... locales)メソッド
    header(String name, Object value)メソッド

- WebResource.Builder クラス
  acceptLanguage(Locale... locales)メソッド
  acceptLanguage(String... locales)メソッド
  header(String name, Object value)メソッド

これらのメソッドおよび acceptLanguage(String... locales)メソッドで，受信できる言語が一つも追加されない場合の動作は，HTTP 通信を行う前に，HttpURLConnection オブジェクトに Accept-Language HTTP ヘッダを追加しない場合の動作と同様です。

build(URI uri, String method)メソッド

説明

ClientRequest オブジェクトを構築します。

構文

public ClientRequest build(URI uri,

String method)

パラメタ

uri
HTTP リクエストの URI です。

method
HTTP メソッド名です。

戻り値

ClientRequest オブジェクトを返します。

注意事項

- uri パラメタには有効な URI を指定してください。また，URI には RFC 2396 で定義されている文字を使用してください。uri パラメタは自動では URI エンコーディングされません。必要に応じて URI エンコードした値を uri パラメタに指定してください。
- method パラメタには，すべて大文字で「GET」「HEAD」「POST」「OPTIONS」「PUT」または「DELETE」のどれかを指定してください。

cookie(Cookie cookie)メソッド

説明

Cookie を設定します。

構文

public T cookie(Cookie cookie)

パラメタ

cookie

設定する Cookie です。

戻り値

WebResource オブジェクトを構築するためのビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- null 以外の値が Cookie HTTP ヘッダに追加されます。null の値は無視され Cookie HTTP ヘッダには追加されません。
- Cookie HTTP ヘッダには，Cookie オブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは toString()メソッドによって返される値を検証しません。標準仕様に従って値を指定してください。
- Cookie HTTP ヘッダの値は次のメソッドでも追加できます。
  - ClientRequest.Builder クラス
    header(String name, Object value)メソッド
  - WebResource クラス
    cookie(Cookie cookie)メソッド
    header(String name, Object value)メソッド
  - WebResource.Builder クラス
    cookie(Cookie cookie)メソッド
    header(String name, Object value)メソッド

  これらのメソッドおよび cookie(Cookie cookie)メソッドで，Cookie が追加されない場合の動作は，HTTP 通信を行う前に，HttpURLConnection オブジェクトに Cookie HTTP ヘッダを追加しない場合の動作と同様です。

entity(Object entity)メソッド

説明

HTTP リクエストのエンティティを設定します。

HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。ジェネリクスをエンティティに指定する場合は，GenericEntity オブジェクトを使用できます。

構文

public T entity(Object entity)

パラメタ

entity

　HTTP リクエストのエンティティです。

戻り値

ビルダ（ClientRequest.Builder オブジェクト）を返します。

注意事項

ありません。

entity(Object entity, MediaType type)メソッド

説明

HTTP リクエストのエンティティとその MIME メディアタイプを設定します。

HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。ジェネリクスをエンティティに指定する場合は，GenericEntity オブジェクトを使用できます。

構文

public T entity(Object entity,

MediaType type)

パラメタ

entity

HTTP リクエストのエンティティです。

type

MIME メディアタイプです。

戻り値

ビルダ（ClientRequest.Builder オブジェクト）を返します。

注意事項

- type パラメタに null が指定された場合，無視されます。Content-Type HTTP ヘッダには設定されません。
- Content-Type HTTP ヘッダには，MediaType オブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは toString()メソッドによって返される値を検証しません。標準仕様に従って値を指定してください。
- Content-Type HTTP ヘッダの値が次のメソッドですでに設定されていた場合，type パラメタの値で上書きされます。
  - ClientRequest.Builder クラス
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド
  - WebResource クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド
  - WebResource.Builder クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド

これらのメソッドおよびentity(Object entity, MediaType type)メソッドで，MIMEメディアタイプが設定されない場合の動作は，HTTP通信を行う前に，HttpURLConnectionオブジェクトにContent-Type HTTPヘッダを設定しない場合の動作と同様です。

entity(Object entity, String type)メソッド

説明

HTTPリクエストのエンティティとそのMIMEメディアタイプを設定します。

HTTPリクエストのエンティティに使用できるJavaの型については「25.15.1　HTTPリクエストのエンティティに使用できるJavaの型とMIMEメディアタイプの組み合わせ」を参照してください。ジェネリクスをエンティティに指定する場合は，GenericEntityオブジェクトを使用できます。

構文

public T entity(Object entity,

String type)

パラメタ

entity
　HTTPリクエストのエンティティです。

type
　MIMEメディアタイプです。

戻り値

ビルダ（ClientRequest.Builderオブジェクト）を返します。

注意事項

- typeパラメタにnullまたは空の文字列が設定された場合，IllegalArgumentExceptionがスローされます。
- Content-Type HTTPヘッダには，MediaTypeクラスのvalueOf(String)スタティックメソッドのパラメタにtypeパラメタを指定して構築したMediaTypeオブジェクトのtoString()メソッドによって返される値が設定されます。JAX-RSエンジンはtypeパラメタに指定される値を検証しません。標準仕様に従って値を指定してください。
- Content-Type HTTPヘッダの値が次のメソッドですでに設定されていた場合，typeパラメタの値で上書きされます。
  - ClientRequest.Builderクラス
    entity(Object entity, MediaType type)メソッド

header(String name, Object value)メソッド
type(MediaType type)メソッド
type(String type)メソッド

- WebResource クラス
  entity(Object entity, MediaType type)メソッド
  entity(Object entity, String type)メソッド
  header(String name, Object value)メソッド
  type(MediaType type)メソッド
  type(String type)メソッド
- WebResource.Builder クラス
  entity(Object entity, MediaType type)メソッド
  entity(Object entity, String type)メソッド
  header(String name, Object value)メソッド
  type(MediaType type)メソッド
  type(String type)メソッド

これらのメソッドおよび entity(Object entity, String type)メソッドで，MIME メディアタイプが設定されない場合の動作は，HTTP 通信を行う前に，HttpURLConnection オブジェクトに Content-Type HTTP ヘッダを設定しない場合の動作と同様です。

header(String name, Object value)メソッド

説明

HTTP ヘッダを追加します。

構文

public T header(String name,

Object value)

パラメタ

name
　HTTP ヘッダ名です。

value
　HTTP ヘッダの値です。

戻り値

ビルダ（ClientRequest.Builder オブジェクト）を返します。

注意事項

- name パラメタと value パラメタ両方の値が null の場合，呼び出しは無視されます。
- name パラメタの値が null ではない場合も，value パラメタの値が null のとき，呼び出しは無視されます。
- name パラメタの値が null で，かつ value パラメタの値が null ではない場合，エラーとなり（KDJJ18888-E），NullPointerException をラップした ClientHandlerException がスローされます。
- このメソッドで Content-Length，Connection，Host の各 HTTP ヘッダを設定することはできません。name パラメタにこれらが指定された場合，value パラメタの値が null ではないときでも，呼び出しは無視されます。なお，それぞれの HTTP ヘッダは HttpURLConnection によって設定されます。
- name パラメタで指定された HTTP ヘッダの値は value パラメタで指定された null ではないオブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは toString()メソッドによって返される値を検証しません。value パラメタには標準仕様に従って値を指定してください。
- accept，acceptLanguage，cookie の各 HTTP ヘッダは，次のメソッドでも追加できます。
  - ClientRequest.Builder クラス
    accept(MediaType... types)メソッド
    accept(String... types)メソッド
    acceptLanguage(Locale... locales)メソッド
    acceptLanguage(String... locales)メソッド
    cookie(Cookie cookie)メソッド
  - WebResource クラス
    accept(MediaType... types)メソッド
    accept(String... types)メソッド
    acceptLanguage(Locale... locales)メソッド
    acceptLanguage(String... locales)メソッド
    cookie(Cookie cookie)メソッド
  - WebResource.Builder クラス
    accept(MediaType... types)メソッド
    accept(String... types)メソッド
    acceptLanguage(Locale... locales)メソッド
    acceptLanguage(String... locales)メソッド
    cookie(Cookie cookie)メソッド

これらのメソッドおよびheader(String name, Object value)メソッドで，各HTTPヘッダが追加されていない場合の動作は，HttpURLConnectionオブジェクトに各HTTPヘッダを追加しない場合と同様です。

- Content-Type HTTPヘッダが次のメソッドですでに設定されていた場合，valueパラメタの値で上書きされます。
  - ClientRequest.Builderクラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド
  - WebResourceクラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド
  - WebResource.Builderクラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド

  これらのメソッドおよびheader(String name, Object value)メソッドで，MIMEメディアタイプが設定されない場合の動作は，HTTP通信を行う前に，HttpURLConnectionオブジェクトにContent-Type HTTPヘッダを設定しない場合の動作と同様です。

type(MediaType type)メソッド

説明

MIMEメディアタイプを設定します。

構文

public T type(MediaType type)

パラメタ

type

MIME メディアタイプです。

戻り値

ビルダ（ClientRequest.Builder オブジェクト）を返します。

注意事項

- type パラメタに指定された null でない値が Content-Type HTTP ヘッダに設定されます。null の値は無視され Content-Type HTTP ヘッダには設定されません。
- Content-Type HTTP ヘッダには，MediaType オブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは toString()メソッドによって返される値を検証しません。標準仕様に従って値を指定してください。
- Content-Type HTTP ヘッダが次のメソッドですでに設定されていた場合，値は上書きされます。
  - ClientRequest.Builder クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(String type)メソッド
  - WebResource クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド
  - WebResource.Builder クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド

  これらのメソッドおよび type(MediaType type)メソッドで，MIME メディアタイプが設定されない場合の動作は，HTTP 通信を行う前に HttpURLConnection オブジェクトに Content-Type HTTP ヘッダを設定しない場合の動作と同様です。

type(String type)メソッド

説明

MIME メディアタイプを設定します。

構文

public T type(String type)

パラメタ

type
　MIME メディアタイプです。

戻り値

ビルダ（ClientRequest.Builder オブジェクト）を返します。

注意事項

- type パラメタに null または空の文字列が設定された場合，IllegalArgumentException がスローされます。
- Content-Type HTTP ヘッダには，MediaType クラスの valueOf(String)スタティックメソッドのパラメタに type パラメタを指定して構築した MediaType オブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは type パラメタに指定される値を検証しません。標準仕様に従って値を指定してください。
- Content-Type HTTP ヘッダが次のメソッドですでに設定されていた場合，type パラメタの値で上書きされます。
  - ClientRequest.Builder クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
  - WebResource クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド
  - WebResource.Builder クラス

entity(Object entity, MediaType type)メソッド

entity(Object entity, String type)メソッド

header(String name, Object value)メソッド

type(MediaType type)メソッド

type(String type)メソッド

これらのメソッドおよびtype(String type)メソッドで，MIMEメディアタイプが設定されない場合の動作は，HTTP通信を行う前に，HttpURLConnectionオブジェクトにContent-Type HTTPヘッダを設定しない場合の動作と同様です。

25.6　ClientResponse クラスのメソッド仕様と注意事項

ここでは，ClientResponse クラスのメソッドの仕様，および利用する場合の注意事項について説明します。

bufferEntity()メソッド

説明

エンティティをバッファリングします。HTTP レスポンスのエンティティの入力となるストリームからすべてのデータが読み込まれ，メモリに格納されます。バッファリング後，入力ストリームはクローズされます。

構文

public void bufferEntity()

throws ClientHandlerException

パラメタ

ありません。

戻り値

ClientHandlerException

処理中に問題が発生した場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

注意事項

ありません。

close()メソッド

説明

HTTP レスポンスを終了します。エンティティの入力ストリームはクローズされます。

構文

public void close()

throws ClientHandlerException

パラメタ

ありません。

戻り値

ClientHandlerException

処理中に問題が発生した場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

注意事項

ありません。

getAllow()メソッド

説明

Allow HTTP ヘッダから，リソースでサポートされる HTTP メソッドの集合を取得します。なお，Allow HTTP ヘッダは，HTTP OPTIONS リクエストの場合に取得できます。

構文

public Set<String> getAllow()

パラメタ

ありません。

戻り値

リソースでサポートされる HTTP メソッドの集合を返します。すべてのメソッドは大文字で出力されます。

注意事項

- HTTP レスポンスに Allow HTTP ヘッダが存在しない場合，空の集合を返します。

getClient()メソッド

説明

クライアント（Client オブジェクト）を取得します。

構文

public Client getClient()

パラメタ

ありません。

戻り値

クライアント（Client オブジェクト）を返します。

注意事項

- このメソッドで返される Client オブジェクトは，ClientResponse オブジェクトを生成した Client オブジェクトです。

getClientResponseStatus()メソッド

説明

ステータスコードを取得します。

構文

public ClientResponse.Status getClientResponseStatus()

パラメタ

ありません。

戻り値

ステータスコードを返します。

HTTP レスポンスのステータスコードが Response.Status 列挙型で定義される定数値に対応していれば，その Response.Status 列挙型定数値を返します。対応していなければ，null を返します。

注意事項

ありません。

getCookies()メソッド

説明

Cookie のリストを取得します。HTTP レスポンスの Set-Cookie HTTP ヘッダに値が設定されている場合，NewCookie オブジェクトのリストが返ります。

構文

public List<NewCookie> getCookies()

パラメタ

ありません。

戻り値

Cookie（NewCookie オブジェクト）のリストを返します。

注意事項

- HTTP レスポンスに Set-Cookie HTTP ヘッダが存在しない場合，空のリストを返します。

getEntity(Class<T> c)メソッド

説明

HTTP レスポンスのエンティティを取得します。取得されるエンティティが Closeable インタフェースを実装したクラスのインスタンスでない場合，入力ストリームはクローズされます。

構文

public <T> T getEntity(Class<T> c)

throws ClientHandlerException,

UniformInterfaceException

パラメタ

c

エンティティの型です。

戻り値

指定された型のオブジェクトを返します。

例外

ClientHandlerException

処理中に問題が発生した場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

UniformInterfaceException

HTTP ステータスコードが 204 No Content の場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

注意事項

- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- このメソッドは一度だけ呼び出せます。二度以上呼び出された場合，動作は保証されません。

getEntity(GenericType<T> gt)メソッド

説明

HTTP レスポンスのエンティティを取得します。取得されるエンティティが Closeable インタフェースを実装したクラスのインスタンスでない場合，入力ストリームはクローズされます。

構文

public <T> T getEntity(GenericType<T> gt)

throws ClientHandlerException,

UniformInterfaceException

パラメタ

gt

エンティティの型を表現する GenericType オブジェクトです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

ClientHandlerException

処理中に問題が発生した場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

UniformInterfaceException

HTTP ステータスコードが 204 No Content の場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

注意事項

- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- このメソッドは一度だけ呼び出せます。このメソッドが二度以上呼び出された場合，動作は保証されません。

getEntityInputStream()メソッド

説明

HTTP レスポンスの入力ストリームを取得します。

構文

public InputStream getEntityInputStream()

パラメタ

ありません。

戻り値

HTTP レスポンスの入力ストリームを返します。

注意事項

ありません。

getEntityTag()メソッド

説明

ETag HTTP ヘッダの値を取得します。

構文

public EntityTag getEntityTag()

パラメタ

ありません。

戻り値

ETag HTTP ヘッダの値を返します。

注意事項

- HTTP レスポンスに ETag HTTP ヘッダがない場合，または HttpURLConnection が ETag HTTP ヘッダの値として null を返す場合には null を返します。
- ETag HTTP ヘッダの値が空の文字列であるか，または標準仕様に準拠していない場合は，IllegalArgumentException がスローされます。

getHeaders()メソッド

説明

HTTP レスポンスの HTTP ヘッダのマップを取得します。

構文

public MultivaluedMap<String,String> getHeaders()

パラメタ

ありません。

戻り値

HTTP レスポンスの HTTP ヘッダのマップを返します。

注意事項

- HTTP レスポンスに HTTP ヘッダがない場合，空のマップを返します。

getLanguage()メソッド

説明

Content-Language HTTP ヘッダの値を取得します。

構文

public String getLanguage()

パラメタ

ありません。

戻り値

Content-Language HTTP ヘッダの値を返します。

注意事項

- HTTP レスポンスに Content-Language HTTP ヘッダがない場合，または HttpURLConnection が Content-Language HTTP ヘッダの値として null を返す場合には null を返します。

getLastModified()メソッド

説明

Last-Modified HTTP ヘッダの値を取得します。

構文

public Date getLastModified()

パラメタ

ありません。

戻り値

Last-Modified HTTP ヘッダの値を返します。

注意事項

- HTTP レスポンスに Last-Modified HTTP ヘッダがない場合，または HttpURLConnection が Last-Modified HTTP ヘッダの値として null を返す場合には null を返します。
- Last-Modified HTTP ヘッダの値が空の文字列の場合，または次に示す形式に従っていない場合，IllegalArgumentException がスローされます。
  - EEE, dd MMM yyyy HH:mm:ss zzz
  - EEEE, dd-MMM-yy HH:mm:ss zzz
  - EEE MMM d HH:mm:ss yyyy

Date のフォーマットにおける各パターン文字，特にロケールに依存する「E」および「M」については，Java SE 仕様の SimpleDateFormat クラスに関する情報を参照してください。

getLength()メソッド

説明

Content-Length HTTP ヘッダの値を取得します。

構文

public int getLength()

パラメタ

ありません。

戻り値

Content-Length HTTP ヘッダの値が有効な数値の場合に整数値で返します。

注意事項

- HTTP レスポンスに Content-Length HTTP ヘッダが存在しない場合，HttpURLConnection が Content-Length HTTP ヘッダの値として null を返す場合，または Content-Length HTTP ヘッダの値が無効な整数値の場合には，-1 を返します。

getLocation()メソッド

説明

Location HTTP ヘッダの値を取得します。

構文

public URI getLocation()

パラメタ

ありません。

戻り値

Location HTTP ヘッダの値を返します。

注意事項

- HTTP レスポンスに Location HTTP ヘッダがない場合，または HttpURLConnection が Location HTTP ヘッダの値として null を返す場合には null を返します。
- Location HTTP ヘッダの値が不正な URI の場合は，IllegalArgumentException がスローされます。

getResponseDate()メソッド

説明

Date HTTP ヘッダの値を取得します。

構文

public Date getResponseDate()

パラメタ

ありません。

戻り値

Date HTTP ヘッダの値を返します。

注意事項

- HTTP レスポンスに Date HTTP ヘッダがない場合，または HttpURLConnection が Date HTTP ヘッダの値として null を返す場合には null を返します。
- Date HTTP ヘッダの値が空の文字列の場合，または次に示す日付の形式に従っていない場合，IllegalArgumentException がスローされます。
  - EEE, dd MMM yyyy HH:mm:ss zzz
  - EEEE, dd-MMM-yy HH:mm:ss zzz
  - EEE MMM d HH:mm:ss yyyy

  Date のフォーマットにおける各パターン文字，特にロケールに依存する「E」および「M」については，Java SE 仕様の SimpleDateFormat クラスに関する情報を参照してください。

getStatus()メソッド

説明

HTTP ステータスコードを取得します。

構文

public int getStatus()

パラメタ

ありません。

戻り値

HTTP ステータスコードを整数値で返します。

注意事項

ありません。

getType()メソッド

説明

Content-Type HTTP ヘッダの値を取得します。

構文

public MediaType getType()

パラメタ

ありません。

戻り値

Content-Type HTTP ヘッダの値を返します。

注意事項

- HTTP レスポンスに Content-Type HTTP ヘッダがない場合，または HttpURLConnection が Content-Type HTTP ヘッダの値として null を返す場合には null を返します。
- Content-Type HTTP ヘッダの値から MediaType オブジェクトを構築できない場合，IllegalArgumentException がスローされます。

hasEntity()メソッド

説明

エンティティが存在するかどうかを確認します。

構文

public boolean hasEntity()

パラメタ

ありません。

戻り値

HTTP レスポンスにエンティティが存在する場合，true を返します。

注意事項

- HTTP HEAD リクエストで HTTP レスポンスを取得した場合は false を返します。
- HttpURLConnection が Content-Length HTTP ヘッダの値として 0 より大きいか，または-1 を返すときに true を返します。それ以外の場合は false を返します。
- 一般に，HttpURLConnection は，チャンク転送エンコーディングでない場合で HTTP レスポンスにエンティティが存在するときに 0 より大きい整数値を返し，チャンク転送エンコーディングである場合に-1 を返します。チャンク転送エンコーディングの詳細については，RFC 2616 を参照してください。
- 次の場合，不正な HTTP レスポンスとなるため動作は保証されません。
  - チャンク転送エンコーディングである場合で，HTTP レスポンスに Content-Length HTTP ヘッダがあるとき
  - チャンク転送エンコーディングでない場合で，Content-Length HTTP ヘッダの値が HTTP レスポンスのエンティティに基づく適切な値でないとき

25.7 ClientResponse.Status クラスの列挙型定数とメソッドの仕様

ここでは，ClientResponse.Status クラスの列挙型定数，およびメソッドの仕様について説明します。

ClientResponse.Status クラスの列挙型定数

ClientResponse.Status クラスでは，HTTP ステータスコードとして，次の列挙型定数が定義されています。定数ごとの構文および HTTP ステータスコードについて説明します。

表 25-5 ClientResponse.Status クラスで定義されている列挙型定数

項番	定数名	構文	HTTP ステータスコード
1	OK	public static final ClientResponse.Status OK	200 OK
2	CREATED	public static final ClientResponse.Status CREATED	201 Created
3	ACCEPTED	public static final ClientResponse.Status ACCEPTED	202 Accepted
4	NON_AUTHORITIVE_INFORMATION	public static final ClientResponse.Status NON_AUTHORITIVE_INFORMATION	203 Non-Authoritative Information
5	NO_CONTENT	public static final ClientResponse.Status NO_CONTENT	204 No Content
6	RESET_CONTENT	public static final ClientResponse.Status RESET_CONTENT	205 Reset Content
7	PARTIAL_CONTENT	public static final ClientResponse.Status PARTIAL_CONTENT	206 Partial Content
8	MOVED_PERMANENTLY	public static final ClientResponse.Status MOVED_PERMANENTLY	301 Moved Permantly
9	FOUND	public static final ClientResponse.Status FOUND	302 Found
10	SEE_OTHER	public static final ClientResponse.Status SEE_OTHER	303 See Other
11	NOT_MODIFIED	public static final ClientResponse.Status NOT_MODIFIED	304 Not Modified
12	USE_PROXY	public static final ClientResponse.Status USE_PROXY	305 Use Proxy
13	TEMPORARY_REDIRECT	public static final ClientResponse.Status TEMPORARY_REDIRECT	307 Temporary Redirect
14	BAD_REQUEST	public static final ClientResponse.Status BAD_REQUEST	400 Bad Request
15	UNAUTHORIZED	public static final ClientResponse.Status UNAUTHORIZED	401 Unauthorized

項番	定数名	構文	HTTP ステータスコード
16	PAYMENT_REQUIRED	public static final ClientResponse.Status PAYMENT_REQUIRED	402 Payment Required
17	FORBIDDEN	public static final ClientResponse.Status FORBIDDEN	403 Forbidden
18	NOT_FOUND	public static final ClientResponse.Status NOT_FOUND	404 Not Found
19	METHOD_NOT_ALLOWED	public static final ClientResponse.Status METHOD_NOT_ALLOWED	405 Method Not Allowed
20	NOT_ACCEPTABLE	public static final ClientResponse.Status NOT_ACCEPTABLE	406 Not Acceptable
21	PROXY_AUTHENTICATION_REQUIRED	public static final ClientResponse.Status PROXY_AUTHENTICATION_REQUIRED	407 Proxy Authentication Required
22	REQUEST_TIMEOUT	public static final ClientResponse.Status REQUEST_TIMEOUT	408 Request Timeout
23	CONFLICT	public static final ClientResponse.Status CONFLICT	409 Conflict
24	GONE	public static final ClientResponse.Status GONE	410 Gone
25	LENGTH_REQUIRED	public static final ClientResponse.Status LENGTH_REQUIRED	411 Length Required
26	PRECONDITION_FAILED	public static final ClientResponse.Status PRECONDITION_FAILED	412 Precondition Failed
27	REQUEST_ENTITY_TOO_LARGE	public static final ClientResponse.Status REQUEST_ENTITY_TOO_LARGE	413 Request Entity Too Large
28	REQUEST_URI_TOO_LONG	public static final ClientResponse.Status REQUEST_URI_TOO_LONG	414 Request-URI Too Long
29	UNSUPPORTED_MEDIA_TYPE	public static final ClientResponse.Status UNSUPPORTED_MEDIA_TYPE	415 Unsupported Media Type
30	REQUESTED_RANGE_NOT_SATIFIABLE	public static final ClientResponse.Status REQUESTED_RANGE_NOT_SATIFIABLE	416 Requested Range Not Satisfiable
31	EXPECTATION_FAILED	public static final ClientResponse.Status EXPECTATION_FAILED	417 Expectation Failed
32	INTERNAL_SERVER_ERROR	public static final ClientResponse.Status INTERNAL_SERVER_ERROR	500 Internal Server Error
33	NOT_IMPLEMENTED	public static final ClientResponse.Status NOT_IMPLEMENTED	501 Not Implemented

項番	定数名	構文	HTTP ステータスコード
34	BAD_GATEWAY	public static final ClientResponse.Status BAD_GATEWAY	502 Bad Gateway
35	SERVICE_UNAVAILABLE	public static final ClientResponse.Status SERVICE_UNAVAILABLE	503 Service Unavailable
36	GATEWAY_TIMEOUT	public static final ClientResponse.Status GATEWAY_TIMEOUT	504 Gateway Timeout
37	HTTP_VERSION_NOT_SUPPORTED	public static final ClientResponse.Status HTTP_VERSION_NOT_SUPPORTED	505 HTTP Version Not Supported

fromStatusCode(int statusCode)メソッド

説明

数値で表現されたステータスコードを対応する ClientResponse.Status オブジェクトに変換します。

構文

public static ClientResponse.Status fromStatusCode(int statusCode)

パラメタ

statusCode

数値で表現されたステータスコードです。

戻り値

ステータスコードに対応する ClientResponse.Status オブジェクトを返します。対応するものがない場合は null を返します。

注意事項

ありません。

getFamily()メソッド

説明

ステータスコードの分類を取得します。

構文

public Response.Status.Family getFamily()

パラメタ

ありません。

戻り値

ステータスコードの分類を返します。

注意事項

ありません。

getReasonPhrase()メソッド

説明

説明文字列（Reason Phrase）を取得します。

構文

public String getReasonPhrase()

パラメタ

ありません。

戻り値

説明文字列（Reason Phrase）を返します。

注意事項

ありません。

getStatusCode()メソッド

説明

対応するステータスコードを取得します。

構文

public int getStatusCode()

パラメタ

ありません。

戻り値

ステータスコードを返します。

注意事項

ありません。

toString()メソッド

説明

説明文字列（Reason Phrase）を取得します。

構文

public String toString()

パラメタ

ありません。

戻り値

説明文字列（Reason Phrase）を返します。

注意事項

ありません。

valueOf(String name)メソッド

説明

指定された名称の列挙型定数を取得します。

この列挙型で宣言されている列挙型定数の識別子を厳密に指定してください。余分なホワイトスペースも許容されません。

構文

public static ClientResponse.Status valueOf(String name)

パラメタ

name

取得する列挙型定数名です。

戻り値

指定された名称の列挙型定数を返します。

例外

IllegalArgumentException

指定した名称の列挙型定数がない場合にスローされます。

NullPointerException

name パラメタに null が指定された場合にスローされます。

注意事項

ありません。

values()メソッド

説明

この列挙型で宣言されている列挙型定数を宣言されている順に含む配列を返します。このメソッドは例えば次のように繰り返し処理する場合に利用できます。

```
for (ClientResponse.Status c : ClientResponse.Status.values())
    System.out.println(c);
```

構文

public static ClientResponse.Status[] values()

パラメタ

ありません。

戻り値

この列挙型で宣言されている列挙型定数を宣言されている順に含む配列を返します。

注意事項

ありません。

25.8　GenericType クラスのコンストラクタおよびメソッド仕様と注意事項

ここでは，GenericType クラスのコンストラクタおよびメソッドの仕様，および利用する場合の注意事項について説明します。GenericType クラスは，型パラメタが解決されたジェネリクスの型情報を保持します。

GenericType()コンストラクタ

説明

GenericType オブジェクトを新規に構築します。対象となる型パラメタが解決されたジェネリックの型情報，およびそのジェネリックを宣言したクラスは型パラメタ T から取得されます。このコンストラクタは protected スコープであるため，ユーザはサブクラスを作成する必要があります（多くの場合，匿名のサブクラスを作成します）。

構文

protected GenericType()

パラメタ

ありません。

戻り値

ありません。

注意事項

ありません。

GenericType(Type genericType)コンストラクタ

説明

GenericType オブジェクトを新規に構築します。対象となる型パラメタが解決されたジェネリックの型，およびそのジェネリックを宣言したクラスは genericType パラメタから取得されます。

構文

public GenericType(Type genericType)

パラメタ

genericType

型パラメタが解決されたジェネリックを表現する java.lang.Type オブジェクトです。

戻り値

ありません。

例外

IllegalArgumentException

genericType パラメタが null である場合，または genericType パラメタが java.lang.Class オブジェクトでも，その型を宣言した型がクラスである java.lang.reflect.ParameterizedType オブジェクトでもない場合にスローされます。

注意事項

ありません。

getRawClass()メソッド

説明

保持する型パラメタが解決されたジェネリクスを宣言したクラスを取得します。

構文

public final Class<T> getRawClass()

パラメタ

ありません。

戻り値

保持する型パラメタが解決されたジェネリクスを宣言したクラスを返します。

注意事項

ありません。

getType()メソッド

説明

保持する型パラメタが解決されたジェネリクスの型を取得します。

構文

public final Type getType()

パラメタ

ありません。

戻り値

保持する型パラメタが解決されたジェネリクスの型を返します。

注意事項

ありません。

25.9　UniformInterfaceException クラスのメソッド仕様と注意事項

ここでは，UniformInterfaceException クラスのメソッドの仕様，および利用する場合の注意事項について説明します。UniformInterfaceException クラスは，ここで説明するメソッドおよび親クラスのメソッドの範囲で使用してください。

getResponse()メソッド

説明

例外に関連する ClientResponse オブジェクトを取得します。

構文

public ClientResponse getResponse()

パラメタ

ありません。

戻り値

ClientResponse オブジェクトを返します。

注意事項

ありません。

25.10　WebResource クラスのメソッド仕様と注意事項

ここでは，WebResource クラスのメソッドの仕様，および利用する場合の注意事項について説明します。

accept(MediaType... types)メソッド

説明

受信できる MIME メディアタイプを追加します。

構文

public WebResource.Builder accept(MediaType... types)

パラメタ

types

受信できる MIME メディアタイプの配列です。

戻り値

WebResource オブジェクトを構築するためのビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- types パラメタに含まれる null でない値が Accept HTTP ヘッダに追加されます。null の値は無視され Accept HTTP ヘッダには追加されません。
- Accept HTTP ヘッダには，MediaType オブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは toString()メソッドによって返される値を検証しません。標準仕様に従って値を指定してください。
- Accept HTTP ヘッダの値は次のメソッドでも追加できます。
  - ClientRequest.Builder クラス
    accept(MediaType... types)メソッド
    accept(String... types)メソッド
    header(String name, Object value)メソッド
  - WebResource クラス
    accept(String... types)メソッド
    header(String name, Object value)メソッド
  - WebResource.Builder クラス
    accept(MediaType... types)メソッド

accept(String... types)メソッド
header(String name, Object value)メソッド

これらのメソッドおよびaccept(MediaType... types)メソッドで，受信できるMIMEメディアタイプが一つも追加されない場合の動作は，HTTP通信を行う前に，HttpURLConnectionオブジェクトにAccept HTTPヘッダを追加しない場合の動作と同様です。

accept(String... types)メソッド

説明

受信できるMIMEメディアタイプを追加します。

構文

public WebResource.Builder accept(String... types)

パラメタ

types
受信できるMIMEメディアタイプの配列です。

戻り値

WebResourceオブジェクトを構築するためのビルダ（WebResource.Builderオブジェクト）を返します。

注意事項

- typesパラメタに含まれるnullでない値がAccept HTTPヘッダに追加されます。nullの値は無視されAccept HTTPヘッダには追加されません。
- Accept HTTPヘッダには，指定された値がそのまま設定されます。JAX-RSエンジンは指定された値を検証しません。標準仕様に従って値を指定してください。
- Accept HTTPヘッダの値は次のメソッドでも追加できます。
  - ClientRequest.Builde クラス
    accept(MediaType... types)メソッド
    accept(String... types)メソッド
    header(String name, Object value)メソッド
  - WebResource クラス
    accept(MediaType... types)メソッド
    header(String name, Object value)メソッド
  - WebResource.Builder クラス

accept(MediaType... types)メソッド
accept(String... types)メソッド
header(String name, Object value)メソッド

これらのメソッドおよびaccept(String... types)メソッドで，受信できるMIMEメディアタイプが一つも追加されない場合の動作は，HTTP通信を行う前に，HttpURLConnectionオブジェクトにAccept HTTPヘッダを追加しない場合の動作と同様です。

acceptLanguage(Locale... locales)メソッド

説明

受信できる言語を追加します。

構文

public WebResource.Builder acceptLanguage(Locale... locales)

パラメタ

locales

受信できる言語の配列です。

戻り値

WebResourceオブジェクトを構築するためのビルダ（WebResource.Builderオブジェクト）を返します。

注意事項

- localesパラメタに含まれるnullでない値がAccept-Language HTTPヘッダに追加されます。nullの値は無視されAccept-Language HTTPヘッダには追加されません。
- Accept-Language HTTPヘッダには，LocaleオブジェクトのtoString()メソッドによって返される値が設定されます。JAX-RSエンジンはtoString()メソッドによって返される値を検証しません。標準仕様に従って値を指定してください。
- Accept-Language HTTPヘッダの値は次のメソッドでも変更できます。
  - ClientRequest.Builderクラス
    acceptLanguage(Locale... locales)メソッド
    acceptLanguage(String... locales)メソッド
    header(String name, Object value)メソッド
  - WebResourceクラス
    acceptLanguage(String... locales)メソッド
    header(String name, Object value)メソッド

- WebResource.Builder クラス
  acceptLanguage(Locale... locales)メソッド
  acceptLanguage(String... locales)メソッド
  header(String name, Object value)メソッド

これらのメソッドおよびacceptLanguage(Locale... locales)メソッドで，受信できる言語が一つも追加されない場合の動作は，HTTP 通信を行う前に，HttpURLConnection オブジェクトに Accept-Language HTTP ヘッダを設定しない場合の動作と同様です。

acceptLanguage(String... locales)メソッド

説明

受信できる言語を追加します。

構文

public WebResource.Builder acceptLanguage(String... locales)

パラメタ

locales
受信できる言語の配列です。

戻り値

WebResource オブジェクトを構築するためのビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- locales パラメタに含まれる null でない値が Accept-Language HTTP ヘッダに追加されます。null の値は無視され Accept-Language HTTP ヘッダには追加されません。
- Accept-Language HTTP ヘッダには，指定された値がそのまま設定されます。JAX-RS エンジンは指定された値を検証しません。標準仕様に従って値を指定してください。
- Accept-Language HTTP ヘッダの値は次のメソッドでも追加できます。
  - ClientRequest.Builder クラス
    acceptLanguage(Locale... locales)メソッド
    acceptLanguage(String... locales)メソッド
    header(String name, Object value)メソッド
  - WebResource クラス
    acceptLanguage(Locale... locales)メソッド
    header(String name, Object value)メソッド

- WebResource.Builder クラス
  acceptLanguage(Locale... locales)メソッド
  acceptLanguage(String... locales)メソッド
  header(String name, Object value)メソッド

これらのメソッドおよび acceptLanguage(String... locales)メソッドで，受信できる言語が一つも追加されない場合の動作は，HTTP 通信を行う前に，HttpURLConnection オブジェクトに Accept-Language HTTP ヘッダを追加しない場合の動作と同様です。

cookie(Cookie cookie)メソッド

説明

Cookie を設定します。

構文

public WebResource.Builder cookie(Cookie cookie)

パラメタ

cookie
　設定する Cookie です。

戻り値

WebResource オブジェクトを構築するためのビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- null 以外の値が Cookie HTTP ヘッダに追加されます。null の値は無視され Cookie HTTP ヘッダには追加されません。
- Cookie HTTP ヘッダには，Cookie オブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは toString()メソッドによって返される値を検証しません。標準仕様に従って値を指定してください。
- Cookie HTTP ヘッダの値は次のメソッドでも追加できます。
  - ClientRequest.Builder クラス
    cookie(Cookie cookie)メソッド
    header(String name, Object value)メソッド
  - WebResource クラス
    header(String name, Object value)メソッド
  - WebResource.Builder クラス

cookie(Cookie cookie)メソッド

header(String name, Object value)メソッド

これらのメソッドおよびcookie(Cookie cookie)メソッドで，Cookieが追加されない場合の動作は，HTTP通信を行う前に，HttpURLConnectionオブジェクトにCookie HTTPヘッダを追加しない場合の動作と同様です。

delete()メソッド

説明

HTTP DELETEメソッドを呼び出します。リクエストにはエンティティを含めません。また，レスポンスのエンティティも受け取りません。

ステータスコードが300より小さい場合でレスポンスにエンティティが含まれるとき，そのエンティティは無視されます。

構文

public void delete()

throws UniformInterfaceException

パラメタ

ありません。

戻り値

ありません。

例外

UniformInterfaceException

HTTPレスポンスのステータスコードが300以上の場合にスローされます。ログにはKDJJ18888-Eのエラーメッセージが出力されます。

注意事項

- HTTPリクエストまたはHTTPレスポンスの処理中に例外が発生した場合，エラーとなり(KDJJ18888-E)，その例外をラップしたClientHandlerExceptionがスローされます。

delete(Class<T> c)メソッド

説明

HTTP DELETE メソッドを呼び出します。リクエストにエンティティを含めません。なお，レスポンスのエンティティは受け取ります。

構文

public <T> T delete(Class<T> c)

throws UniformInterfaceException

パラメタ

c

HTTP レスポンスのエンティティの型です。

戻り値

指定された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，c パラメタが ClientResponse 型でない

注意事項

- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

delete(Class<T> c, Object requestEntity)メソッド

説明

HTTP DELETE メソッドを呼び出します。リクエストにエンティティを含めます。また，レスポンスのエンティティも受け取ります。

構文

public <T> T delete(Class<T> c,

Object requestEntity)

throws UniformInterfaceException

パラメタ

c

HTTP レスポンスのエンティティの型です。

requestEntity

HTTP リクエストのエンティティです。

戻り値

指定された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，c パラメタが ClientResponse 型でない

注意事項

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

delete(GenericType<T> gt)メソッド

説明

HTTP DELETE メソッドを呼び出します。リクエストにエンティティを含めません。なお，レスポンスのエンティティは受け取ります。

構文

public <T> T delete(GenericType<T> gt)

throws UniformInterfaceException

パラメタ

gt

HTTP レスポンスのエンティティの型を表現する GenericType オブジェクトです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，gt パラメタが ClientResponse 型を表現していない

注意事項

- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

delete(GenericType<T> gt, Object requestEntity)メソッド

説明

HTTP DELETE メソッドを呼び出します。リクエストにエンティティを含めます。また，レスポンスのエンティティも受け取ります。

構文

public <T> T delete(GenericType<T> gt,

Object requestEntity)

throws UniformInterfaceException

パラメタ

gt

HTTP レスポンスのエンティティの型を表現する GenericType オブジェクトです。

requestEntity

HTTP リクエストのエンティティです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，gt パラメタが ClientResponse 型を表現していない

注意事項

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

delete(Object requestEntity)メソッド

説明

HTTP DELETE メソッドを呼び出します。リクエストにエンティティを含めます。レスポンスのエンティティは受け取りません。

ステータスコードが 300 より小さい場合でレスポンスにエンティティが含まれるとき，そのエンティティは無視されます。

構文

public void delete(Object requestEntity)

throws UniformInterfaceException

パラメタ

requestEntity
　HTTP リクエストのエンティティです。

戻り値

ありません。

例外

UniformInterfaceException
　HTTP レスポンスのステータスコードが 300 以上の場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

注意事項

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

entity(Object entity)メソッド

説明

HTTP リクエストのエンティティを設定します。

構文

public WebResource.Builder entity(Object entity)

パラメタ

entity
　HTTP リクエストのエンティティです。

戻り値

ビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。

entity(Object entity, MediaType type)メソッド

説明

HTTP リクエストのエンティティの MIME メディアタイプを設定します。

HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。ジェネリクスをエンティティに指定する場合は，GenericEntity オブジェクトを使用できます。

構文

public WebResource.Builder entity(Object entity,

MediaType type)

パラメタ

entity
　HTTP リクエストのエンティティです。

type
　MIME メディアタイプです。

戻り値

ビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- type パラメタに指定された null でない値が Content-Type HTTP ヘッダに設定されます。null の値は無視され Content-Type HTTP ヘッダには設定されません。
- Content-Type HTTP ヘッダには，MediaType オブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは toString()メソッドによって返される値を検証しません。標準仕様に従って値を指定してください。
- Content-Type HTTP ヘッダの値は次のメソッドでも変更できます。

- ClientRequest.Builder クラス
  entity(Object entity, MediaType type)メソッド
  entity(Object entity, String type)メソッド
  header(String name, Object value)メソッド
  type(MediaType type)メソッド
  type(String type)メソッド
- WebResource クラス
  entity(Object entity, String type)メソッド
  header(String name, Object value)メソッド
  type(MediaType type)メソッド
  type(String type)メソッド
- WebResource.Builder クラス
  entity(Object entity, MediaType type)メソッド
  entity(Object entity, String type)メソッド
  header(String name, Object value)メソッド
  type(MediaType type)メソッド
  type(String type)メソッド

これらのメソッドおよび entity(Object entity, MediaType type)メソッドで，MIME メディアタイプが設定されない場合の動作は，HTTP 通信を行う前に，HttpURLConnection オブジェクトに Content-Type HTTP ヘッダを設定しない場合の動作と同様です。

entity(Object entity, String type)メソッド

説明

HTTP リクエストのエンティティの MIME メディアタイプを設定します。

HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。ジェネリクスをエンティティに指定する場合は，GenericEntity オブジェクトを使用できます。

構文

public WebResource.Builder entity(Object entity,

String type)

パラメタ

entity

HTTP リクエストのエンティティです。

type

MIME メディアタイプです。

戻り値

ビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- type パラメタに null または空の文字列が設定された場合，IllegalArgumentException がスローされます。
- type パラメタに指定された null でない値が Content-Type HTTP ヘッダに設定されます。
- Content-Type HTTP ヘッダには，MediaType オブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは toString()メソッドによって返される値を検証しません。標準仕様に従って値を指定してください。
- Content-Type HTTP ヘッダの値は次のメソッドでも変更できます。
  - ClientRequest.Builder クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド
  - WebResource クラス
    entity(Object entity, MediaType type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド
  - WebResource.Builder クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド

これらのメソッドおよびentity(Object entity, String type)メソッドで，MIMEメディアタイプが設定されない場合の動作は，HTTP通信を行う前に，HttpURLConnectionオブジェクトにContent-Type HTTPヘッダを設定しない場合の動作と同様です。

get(Class<T> c)メソッド

説明

HTTP GETメソッドを呼び出します。

構文

public <T> T get(Class<T> c)

throws UniformInterfaceException

パラメタ

c

HTTPレスポンスのエンティティの型です。

戻り値

指定された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログにはKDJJ18888-Eのエラーメッセージが出力されます。

- HTTPレスポンスのステータスコードが204である
- HTTPレスポンスのステータスコードが300以上で，cパラメタがClientResponse型でない

注意事項

- HTTPレスポンスのエンティティに使用できるJavaの型については「25.15.2　HTTPレスポンスのエンティティに使用できるJavaの型とMIMEメディアタイプの組み合わせ」を参照してください。
- HTTPリクエストまたはHTTPレスポンスの処理中に例外が発生した場合，エラーとなり(KDJJ18888-E)，その例外をラップしたClientHandlerExceptionがスローされます。

get(GenericType<T> gt)メソッド

説明

HTTP GET メソッドを呼び出します。

構文

public <T> T get(GenericType<T> gt)

throws UniformInterfaceException

パラメタ

gt

HTTP レスポンスのエンティティの型を表現する GenericType オブジェクトです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，gt パラメタが ClientResponse 型を表現していない

注意事項

- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

getRequestBuilder()メソッド

説明

ビルダ（WebResource.Builder オブジェクト）を取得します。

構文

public WebResource.Builder getRequestBuilder()

パラメタ

ありません。

戻り値

ビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

ありません。

getURI()メソッド

説明

Web リソースの URI を取得します。URI にはリクエストの対象となるリソースを識別する情報が含まれます。

構文

public URI getURI()

パラメタ

ありません。

戻り値

Web リソースの URI を返します。

注意事項

ありません。

getUriBuilder()メソッド

説明

URI ビルダ（URIBuilder オブジェクト）を取得します。

構文

public UriBuilder getUriBuilder()

パラメタ

ありません。

戻り値

URI ビルダ（URIBuilder オブジェクト）を返します。

注意事項

- この WebResource オブジェクトが保持する Web リソースの URI から作成された URI ビルダ（URIBuilder オブジェクト）が返ります。

head()メソッド

説明

HTTP HEAD メソッドを呼び出します。

構文

public ClientResponse head()

パラメタ

ありません。

戻り値

HTTP レスポンスを返します。

注意事項

- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

header(String name, Object value)メソッド

説明

HTTP ヘッダを追加します。

構文

public WebResource.Builder header(String name,

Object value)

パラメタ

name

HTTP ヘッダ名です。

value

HTTP ヘッダの値です。

戻り値

ビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- name パラメタと value パラメタ両方の値が null の場合，呼び出しは無視されます。
- name パラメタの値が null ではない場合も，value パラメタの値が null のとき，呼び出しは無視されます。
- name パラメタの値が null で，かつ value パラメタの値が null ではない場合，エラーとなり（KDJJ18888-E），NullPointerException をラップした ClientHandlerException がスローされます。
- このメソッドで Content-Length，Connection，Host の各 HTTP ヘッダを設定することはできません。name パラメタにこれらが指定された場合，value パラメタの値が null ではないときでも，呼び出しは無視されます。なお，それぞれの HTTP ヘッダは HttpURLConnection によって設定されます。
- name パラメタで指定された HTTP ヘッダの値は value パラメタで指定された null ではないオブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは toString()メソッドによって返される値を検証しません。value パラメタには標準仕様に従って値を指定してください。
- accept，acceptLanguage，cookie の各 HTTP ヘッダは，次のメソッドでも追加できます。
  - ClientRequest.Builder クラス
    accept(MediaType... types)メソッド
    accept(String... types)メソッド
    acceptLanguage(Locale... locales)メソッド
    acceptLanguage(String... locales)メソッド
    cookie(Cookie cookie)メソッド
  - WebResource クラス
    accept(MediaType... types)メソッド
    accept(String... types)メソッド

acceptLanguage(Locale... locales)メソッド
acceptLanguage(String... locales)メソッド
cookie(Cookie cookie)メソッド

- WebResource.Builder クラス
  accept(MediaType... types)メソッド
  accept(String... types)メソッド
  acceptLanguage(Locale... locales)メソッド
  acceptLanguage(String... locales)メソッド
  cookie(Cookie cookie)メソッド

これらのメソッドおよび header(String name, Object value)メソッドで，各 HTTP ヘッダが追加されていない場合の動作は，HttpURLConnection オブジェクトに各 HTTP ヘッダを追加しない場合と同様です。

- Content-Type HTTP ヘッダが次のメソッドですでに設定されていた場合，value パラメタの値で上書きされます。
  - ClientRequest.Builder クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド
  - WebResource クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド
  - WebResource.Builder クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド

  これらのメソッドおよび header(String name, Object value)メソッドで，MIME メディアタイプが設定されない場合の動作は，HTTP 通信を行う前に，HttpURLConnection オブジェクトに Content-Type HTTP ヘッダを設定しない場合の動作と同様です。

method(String method)メソッド

説明

HTTP メソッドを呼び出します。リクエストにはエンティティを含めません。また，レスポンスのエンティティも受け取りません。

ステータスコードが 300 より小さい場合でレスポンスにエンティティが含まれるとき，そのエンティティは無視されます。

構文

public void method(String method)

throws UniformInterfaceException

パラメタ

method
: HTTP メソッド名です。

戻り値

ありません。

例外

UniformInterfaceException
: HTTP レスポンスのステータスコードが 300 以上の場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

注意事項

- method パラメタには，すべて大文字で「GET」「HEAD」「POST」「OPTIONS」「PUT」または「DELETE」のどれかを指定してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

method(String method, Class<T> c)メソッド

説明

HTTP メソッドを呼び出します。リクエストにエンティティを含めません。なお，レスポンスのエンティティは受け取ります。

構文

public <T> T method(String method,

Class<T> c)

throws UniformInterfaceException

パラメタ

method

HTTP メソッドです。

c

HTTP レスポンスのエンティティの型です。

戻り値

指定された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，c パラメタが ClientResponse 型でない

注意事項

- method パラメタには，すべて大文字で「GET」「HEAD」「POST」「OPTIONS」「PUT」または「DELETE」のどれかを指定してください。
- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

method(String method, Class<T> c, Object requestEntity)メソッド

説明

HTTP メソッドを呼び出します。リクエストにエンティティを含めません。なお，レスポンスのエンティティは受け取ります。

構文

public <T> T method(String method,

Class<T> c,

Object requestEntity)

throws UniformInterfaceException

パラメタ

method

HTTP メソッドです。

c

HTTP レスポンスのエンティティの型です。

requestEntity

HTTP リクエストのエンティティです。

戻り値

指定された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，c パラメタが ClientResponse 型でない

注意事項

- method パラメタには，すべて大文字で「GET」「HEAD」「POST」「OPTIONS」「PUT」または「DELETE」のどれかを指定してください。
- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

method(String method, GenericType<T> gt)メソッド

説明

HTTP メソッドを呼び出します。リクエストにエンティティを含めません。なお，レスポンスのエンティティは受け取ります。

構文

public <T> T method(String method,

GenericType<T> gt)

throws UniformInterfaceException

パラメタ

method

HTTP メソッドです。

gt

HTTP レスポンスのエンティティの型を表現する GenericType オブジェクトです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，gt パラメタが ClientResponse 型を表現していない

注意事項

- method パラメタには，すべて大文字で「GET」「HEAD」「POST」「OPTIONS」「PUT」または「DELETE」のどれかを指定してください。
- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

method(String method, GenericType<T> gt, Object requestEntity)メソッド

説明

HTTP メソッドを呼び出します。リクエストにエンティティを含めません。なお，レスポンスのエンティティは受け取ります。

構文

public <T> T method(String method,

GenericType<T> gt,

Object requestEntity)

throws UniformInterfaceException

パラメタ

method

HTTP メソッドです。

gt

HTTP レスポンスのエンティティの型を表現する GenericType オブジェクトです。

requestEntity

HTTP リクエストのエンティティです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，gt パラメタが ClientResponse 型を表現していない

注意事項

- method パラメタには，すべて大文字で「GET」「HEAD」「POST」「OPTIONS」「PUT」または「DELETE」のどれかを指定してください。

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

method(String method, Object requestEntity)メソッド

説明

HTTP メソッドを呼び出します。リクエストにエンティティを含めません。なお，レスポンスのエンティティは受け取ります。

ステータスコードが 300 より小さい場合でレスポンスにエンティティが含まれるとき，そのエンティティは無視されます。

構文

public void method(String method,

Object requestEntity)

throws UniformInterfaceException

パラメタ

method
　HTTP メソッドです。

requestEntity
　HTTP リクエストのエンティティです。

戻り値

ありません。

例外

UniformInterfaceException
　HTTP レスポンスのステータスコードが 300 以上の場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

注意事項

- method パラメタには，すべて大文字で「GET」「HEAD」「POST」「OPTIONS」「PUT」または「DELETE」のどれかを指定してください。
- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

options(Class<T> c)メソッド

説明

HTTP OPTIONS メソッドを呼び出します。

構文

public <T> T options(Class<T> c)

throws UniformInterfaceException

パラメタ

c

HTTP レスポンスのエンティティの型です。

戻り値

指定された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，c パラメタが ClientResponse 型でない

注意事項

- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。

- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

options(GenericType<T> gt)メソッド

説明

HTTP OPTIONS メソッドを呼び出します。

構文

public <T> T options(GenericType<T> gt)

throws UniformInterfaceException

パラメタ

gt

HTTP レスポンスのエンティティの型を表現する GenericType オブジェクトです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，gt パラメタが ClientResponse 型を表現していない

注意事項

- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

path(String path)メソッド

説明

このオブジェクト（WebResource オブジェクト）から新しく WebResource オブジェクトを生成します。生成されるオブジェクトが保持する Web リソースの URI は，このオブジェクトが保持する Web リソースの URI にパラメタで指定されたパスが追加された URI です。

構文

public WebResource path(String path)

パラメタ

path

追加するパスです。

戻り値

新しく生成した WebResource オブジェクトを返します。

注意事項

- path パラメタに null が設定された場合，IllegalArgumentException がスローされます。
- path パラメタには null 以外の正しいパスを指定してください。無効な文字は，標準仕様に従って自動的に URL エンコードされます。
- クエリストリングはパスの一部ではないため，疑問符（?）は無効な文字と見なされて自動的に URL エンコードされることに注意してください。エンコードされない文字は次のとおりです。

  ! $ & ' ( ) * + - / ; = @ _ ~ . ,

  半角英数字（0～9，A～Z，a～z）

  すでに URL エンコードされたトークンは正しく認識されるため，二重に URL エンコードされることはありません。
- このオブジェクト（WebResource オブジェクト）が保持する Web リソースの URI と path パラメタで指定されたパスの間には，必要に応じて自動的にスラッシュ（/）が挿入されます。またスラッシュ（/）が二重になるような場合は自動的に補正されます。
- このオブジェクト（WebResource オブジェクト）のエンティティは，このメソッドで新しく生成された WebResource オブジェクトにはコピーされません。

post()メソッド

説明

HTTP POST メソッドを呼び出します。リクエストにはエンティティを含めません。また，レスポンスのエンティティも受け取りません。

ステータスコードが 300 より小さい場合でレスポンスにエンティティが含まれるとき，そのエンティティは無視されます。

構文

public void post()

throws UniformInterfaceException

パラメタ

ありません。

戻り値

ありません。

例外

UniformInterfaceException

HTTP レスポンスのステータスコードが 300 以上の場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

注意事項

- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

post(Class<T> c)メソッド

説明

HTTP POST メソッドを呼び出します。リクエストにエンティティを含めません。なお，レスポンスのエンティティは受け取ります。

構文

public <T> T post(Class<T> c)

throws UniformInterfaceException

パラメタ

c

HTTP レスポンスのエンティティの型です。

戻り値

指定された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，c パラメタが ClientResponse 型でない

注意事項

- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

post(Class<T> c, Object requestEntity)メソッド

説明

HTTP POST メソッドを呼び出します。リクエストにエンティティを含めます。また，レスポンスのエンティティも受け取ります。

構文

public <T> T post(Class<T> c,

Object requestEntity)

throws UniformInterfaceException

パラメタ

c

HTTP レスポンスのエンティティの型です。

requestEntity

HTTP リクエストのエンティティです。

戻り値

指定された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，c パラメタが ClientResponse 型でない

注意事項

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

post(GenericType<T> gt)メソッド

説明

HTTP POST メソッドを呼び出します。リクエストにエンティティを含めます。また，レスポンスのエンティティも受け取ります。

構文

public <T> T post(GenericType<T> gt)

throws UniformInterfaceException

パラメタ

gt

HTTP レスポンスのエンティティの型を表現する GenericType オブジェクトです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，gt パラメタが ClientResponse 型を表現していない

注意事項

- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

post(GenericType<T> gt, Object requestEntity)メソッド

説明

HTTP POST メソッドを呼び出します。リクエストにエンティティを含めます。また，レスポンスのエンティティも受け取ります。

構文

public <T> T post(GenericType<T> gt,

Object requestEntity)

throws UniformInterfaceException

パラメタ

gt

HTTP レスポンスのエンティティの型を表現する GenericType オブジェクトです。

requestEntity

HTTP リクエストのエンティティです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，gt パラメタが ClientResponse 型を表現していない

注意事項

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

post(Object requestEntity)メソッド

説明

HTTP POST メソッドを呼び出します。リクエストにエンティティを含めます。レスポンスのエンティティは受け取りません。

ステータスコードが 300 より小さい場合でレスポンスにエンティティが含まれるとき，そのエンティティは無視されます。

構文

public void post(Object requestEntity)

throws UniformInterfaceException

パラメタ

requestEntity
　HTTP リクエストのエンティティです。

戻り値

ありません。

例外

UniformInterfaceException
　HTTP レスポンスのステータスコードが 300 以上の場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

注意事項

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

put()メソッド

説明

HTTP PUT メソッドを呼び出します。リクエストにはエンティティを含めません。また，レスポンスのエンティティも受け取りません。

ステータスコードが 300 より小さい場合でレスポンスにエンティティが含まれるとき，そのエンティティは無視されます。

構文

public void put()

throws UniformInterfaceException

パラメタ

ありません。

戻り値

ありません。

例外

UniformInterfaceException

HTTP レスポンスのステータスコードが 300 以上の場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

注意事項

- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

put(Class<T> c)メソッド

説明

HTTP PUT メソッドを呼び出します。リクエストにエンティティを含めます。また，レスポンスのエンティティも受け取ります。

構文

public <T> T put(Class<T> c)

throws UniformInterfaceException

パラメタ

c

HTTP レスポンスのエンティティの型です。

戻り値

指定された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，c パラメタが ClientResponse 型でない

注意事項

- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

put(Class<T> c, Object requestEntity)メソッド

説明

HTTP PUT メソッドを呼び出します。リクエストにエンティティを含めます。また，レスポンスのエンティティも受け取ります。

構文

public <T> T put(Class<T> c,

Object requestEntity)

throws UniformInterfaceException

パラメタ

c

HTTP レスポンスのエンティティの型です。

requestEntity

HTTP リクエストのエンティティです。

戻り値

指定された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である

HTTP レスポンスのステータスコードが 300 以上で，c パラメタが ClientResponse 型でない

注意事項

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

put(GenericType<T> gt)メソッド

説明

HTTP PUT メソッドを呼び出します。リクエストにエンティティを含めます。また，レスポンスのエンティティも受け取ります。

構文

public <T> T put(GenericType<T> gt)

throws UniformInterfaceException

パラメタ

gt
　HTTP レスポンスのエンティティの型を表現する GenericType オブジェクトです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException
　次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，gt パラメタが ClientResponse 型を表現していない

注意事項

- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

put(GenericType<T> gt, Object requestEntity)メソッド

説明

HTTP PUT メソッドを呼び出します。リクエストにエンティティを含めます。また，レスポンスのエンティティも受け取ります。

構文

public <T> T put(GenericType<T> gt,

Object requestEntity)

throws UniformInterfaceException

パラメタ

gt
　HTTP レスポンスのエンティティの型を表現する GenericType オブジェクトです。

requestEntity
　HTTP リクエストのエンティティです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException
　次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，gt パラメタが ClientResponse 型を表現していない

注意事項

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

put(Object requestEntity)メソッド

説明

HTTP PUT メソッドを呼び出します。リクエストにエンティティを含めます。また，レスポンスのエンティティも受け取ります。

ステータスコードが 300 より小さい場合でレスポンスにエンティティが含まれるとき，そのエンティティは無視されます。

構文

public void put(Object requestEntity)

throws UniformInterfaceException

パラメタ

requestEntity
　HTTP リクエストのエンティティです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException
　HTTP レスポンスのステータスコードが 300 以上の場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

注意事項

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。

- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

queryParam(String key, String value)メソッド

説明

このオブジェクト（WebResource オブジェクト）から新しく WebResource オブジェクトを生成します。生成されるオブジェクトが保持する Web リソースの URI は，このオブジェクトが保持する Web リソースの URI に，パラメタに指定されたクエリパラメタが追加された URI です。

構文

public WebResource queryParam(String key,

String value)

パラメタ

key

クエリパラメタ名です。

value

クエリパラメタの値です。

戻り値

新しく生成した WebResource オブジェクトを返します。

注意事項

- key パラメタまたは value パラメタのどちらかに null が設定された場合，IllegalArgumentException がスローされます。
- key パラメタおよび value パラメタには正しいクエリパラメタの名称および値を指定してください。無効な文字は，標準仕様に従って自動的に URL エンコードされます。
  - エンコードされない文字は次のとおりです。
    ! $ ' ( ) * - / ; ? @ _ ~ . ,
    半角英数字（0～9，A～Z，a～z）
  - すでに URL エンコードされたトークンは正しく認識されるため，二重に URL エンコードされることはありません。
  - 空白は「+」にエンコードされます。

- 新しく生成される WebResource オブジェクトのクエリパラメタは，このオブジェクト（WebResource オブジェクト）のクエリパラメタに，パラメタに指定されたクエリパラメタが追加されたものです。なお，同じ名称のクエリパラメタがこのオブジェクトに存在している場合も含みます。

- このオブジェクト（WebResource オブジェクト）のエンティティは，このメソッドで新しく生成された WebResource オブジェクトにはコピーされません。

queryParams(MultivaluedMap<String, String> params)メソッド

説明

このオブジェクト（WebResource オブジェクト）から新しく WebResource オブジェクトを生成します。生成されるオブジェクトが保持する Web リソースの URI は，このオブジェクトが保持する Web リソースの URI に，パラメタに指定されたクエリパラメタが追加された URI です。

構文

public WebResource queryParams(MultivaluedMap<String,String> params)

パラメタ

params

クエリパラメタのマップです。

戻り値

新しく生成した WebResource オブジェクトを返します。

注意事項

- params パラメタに null が指定された場合，NullPointerException がスローされます。
- params パラメタで指定されたクエリパラメタのマップに null であるキーが含まれている場合，IllegalArgumentException がスローされます。
- params パラメタで指定されたクエリパラメタのマップに含まれるキーと値の組み合わせのうち，キーが null 以外で値が null であるものについては，キーだけが新しく生成された WebResource オブジェクトのクエリパラメタとして追加されます。
- params パラメタには正しいクエリパラメタを指定してください。無効な文字は，標準仕様に従って自動的に URL エンコードされます。
  - エンコードされない文字は次のとおりです。

    ! $ ' ( ) * - / ; ? @ _ ~ . ,

    半角英数字（0～9，A～Z，a～z）

- すでに URL エンコードされたトークンは正しく認識されるため，二重に URL エンコードされることはありません。
- 空白は「+」にエンコードされます。
- 新しく生成される WebResource オブジェクトのクエリパラメタは，このオブジェクト（WebResource オブジェクト）のクエリパラメタに，パラメタに指定されたクエリパラメタが追加されたものです。なお，同じ名称のクエリパラメタがこのオブジェクトに存在している場合も含みます。

- このオブジェクト（WebResource オブジェクト）のエンティティは，このメソッドで新しく生成された WebResource オブジェクトにはコピーされません。

type(MediaType type)メソッド

説明

MIME メディアタイプを設定します。

構文

public WebResource.Builder type(MediaType type)

パラメタ

type
MIME メディアタイプです。

戻り値

ビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- type パラメタに指定された null でない値が Content-Type HTTP ヘッダに設定されます。null の値は無視され Content-Type HTTP ヘッダには設定されません。
- Content-Type HTTP ヘッダには，MediaType オブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは toString()メソッドによって返される値を検証しません。標準仕様に従って値を指定してください。
- Content-Type HTTP ヘッダが次のメソッドですでに設定されていた場合，値は上書きされます。
  - ClientRequest.Builder クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド

type(MediaType type)メソッド
type(String type)メソッド

- WebResource クラス
  entity(Object entity, MediaType type)メソッド
  entity(Object entity, String type)メソッド
  header(String name, Object value)メソッド
  type(String type)メソッド
- WebResource.Builder クラス
  entity(Object entity, MediaType type)メソッド
  entity(Object entity, String type)メソッド
  header(String name, Object value)メソッド
  type(MediaType type)メソッド
  type(String type)メソッド

これらのメソッドおよび type(MediaType type)メソッドで，MIME メディアタイプが設定されない場合の動作は，HTTP 通信を行う前に HttpURLConnection オブジェクトに Content-Type HTTP ヘッダを設定しない場合の動作と同様です。

type(String type)メソッド

説明

MIME メディアタイプを設定します。

構文

public WebResource.Builder type(String type)

パラメタ

type

MIME メディアタイプです。

戻り値

ビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- type パラメタに null または空の文字列が設定された場合，IllegalArgumentException がスローされます。

- Content-Type HTTP ヘッダには，MediaType クラスの valueOf(String)スタティックメソッドのパラメタに type パラメタを指定して構築した MediaType オブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは type パラメタに指定される値を検証しません。標準仕様に従って値を指定してください。
- Content-Type HTTP ヘッダが次のメソッドですでに設定されていた場合，type パラメタの値で上書きされます。
  - ClientRequest.Builder クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド
  - WebResource クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
  - WebResource.Builder クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド

  これらのメソッドおよび type(String type)メソッドで，MIME メディアタイプが設定されない場合の動作は，HTTP 通信を行う前に，HttpURLConnection オブジェクトに Content-Type HTTP ヘッダを設定しない場合の動作と同様です。

uri(java.net.URI uri)メソッド

説明

このオブジェクト（WebResource オブジェクト）から新しく WebResource オブジェクトを生成します。

パラメタに指定された URI にスラッシュ（/）で始まるパスコンポーネントが含まれていた場合，新しく生成される WebResource オブジェクトが保持する Web リソースの URI のパスは置き換えられます。スラッシュ（/）で始まるパスコンポーネントが含まれない場合，新しく生成される WebResource オブジェクトが保持する Web リソースの URI のパスは，このオブジェクト（WebResource オブジェクト）のパスにそのパスコンポーネントが追加されたパスになります。

パラメタに指定された URI にクエリパラメタが含まれていた場合，新しく生成される WebResource オブジェクトが保持する Web リソースの URI のクエリパラメタはそのクエリパラメタになります。このオブジェクトが保持する Web リソースの URI にクエリパラメタがあれば置き換えられます。

構文

public WebResource uri(URI uri)

パラメタ

uri

Web リソースの URI です。

戻り値

新しく生成した WebResource オブジェクトを返します。

注意事項

- uri パラメタには正しい URI を指定してください。uri パラメタは自動では URL エンコーディングされません。必要に応じて URL エンコードした値を uri パラメタに指定してください。
- このオブジェクト（WebResource オブジェクト）のエンティティは，このメソッドで新しく生成された WebResource オブジェクトにはコピーされません。
- uri パラメタに null が指定された場合，NullPointerException がスローされます。

25.11　WebResource.Builder クラスのメソッド仕様と注意事項

ここでは，WebResource.Builder クラスのメソッドの仕様，および利用する場合の注意事項について説明します。

accept(MediaType... types)メソッド

説明

受信できる MIME メディアタイプを追加します。

構文

public T accept(MediaType... types)

パラメタ

types

受信できる MIME メディアタイプの配列です。

戻り値

WebResource オブジェクトを構築するためのビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- types パラメタに含まれる null でない値が Accept HTTP ヘッダに追加されます。null の値は無視され Accept HTTP ヘッダには追加されません。
- Accept HTTP ヘッダには，MediaType オブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは toString()メソッドによって返される値を検証しません。標準仕様に従って値を指定してください。
- Accept HTTP ヘッダの値は次のメソッドでも追加できます。
  - ClientRequest.Builder クラス
    accept(MediaType... types)メソッド
    accept(String... types)メソッド
    header(String name, Object value)メソッド
  - WebResource クラス
    accept(MediaType... types)メソッド
    accept(String... types)メソッド
    header(String name, Object value)メソッド
  - WebResource.Builder クラス

accept(String... types)メソッド
header(String name, Object value)メソッド

これらのメソッドおよびaccept(MediaType... types)メソッドで，受信できるMIMEメディアタイプが一つも追加されない場合の動作は，HTTP通信を行う前に，HttpURLConnectionオブジェクトにAccept HTTPヘッダを追加しない場合の動作と同様です。

accept(String... types)メソッド

説明

受信できるMIMEメディアタイプを追加します。

構文

public T accept(String... types)

パラメタ

types
受信できるMIMEメディアタイプの配列です。

戻り値

WebResourceオブジェクトを構築するためのビルダ（WebResource.Builderオブジェクト）を返します。

注意事項

- typesパラメタに含まれるnullでない値がAccept HTTPヘッダに追加されます。nullの値は無視されAccept HTTPヘッダには追加されません。
- Accept HTTPヘッダには，指定された値がそのまま設定されます。JAX-RSエンジンは指定された値を検証しません。標準仕様に従って値を指定してください。
- Accept HTTPヘッダの値は次のメソッドでも追加できます。
  - ClientRequest.Builde クラス
    accept(MediaType... types)メソッド
    accept(String... types)メソッド
    header(String name, Object value)メソッド
  - WebResource クラス
    accept(MediaType... types)メソッド
    accept(String... types)メソッド
    header(String name, Object value)メソッド

- WebResource.Builder クラス
  accept(MediaType... types)メソッド
  header(String name, Object value)メソッド

これらのメソッドおよび accept(String... types)メソッドで，受信できる MIME メディアタイプが一つも追加されない場合の動作は，HTTP 通信を行う前に，HttpURLConnection オブジェクトに Accept HTTP ヘッダを追加しない場合の動作と同様です。

acceptLanguage(Locale... locales)メソッド

説明

受信できる言語を追加します。

構文

public T acceptLanguage(Locale... locales)

パラメタ

locales

受信できる言語の配列です。

戻り値

WebResource オブジェクトを構築するためのビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- locales パラメタに含まれる null でない値が Accept-Language HTTP ヘッダに追加されます。null の値は無視され Accept-Language HTTP ヘッダには追加されません。
- Accept-Language HTTP ヘッダには，Locale オブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは toString()メソッドによって返される値を検証しません。標準仕様に従って値を指定してください。
- Accept-Language HTTP ヘッダの値は次のメソッドでも変更できます。
  - ClientRequest.Builder クラス
    acceptLanguage(Locale... locales)メソッド
    acceptLanguage(String... locales)メソッド
    header(String name, Object value)メソッド
  - WebResource クラス
    acceptLanguage(Locale... locales)メソッド
    acceptLanguage(String... locales)メソッド

header(String name, Object value)メソッド

- WebResource.Builder クラス
  acceptLanguage(String... locales)メソッド
  header(String name, Object value)メソッド

これらのメソッドおよびacceptLanguage(Locale... locales)メソッドで，受信できる言語が一つも追加されない場合の動作は，HTTP 通信を行う前に，HttpURLConnection オブジェクトに Accept-Language HTTP ヘッダを設定しない場合の動作と同様です。

acceptLanguage(String... locales)メソッド

説明

受信できる言語を追加します。

構文

public T acceptLanguage(String... locales)

パラメタ

locales

受信できる言語の配列です。

戻り値

WebResource オブジェクトを構築するためのビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- locales パラメタに含まれる null でない値が Accept-Language HTTP ヘッダに追加されます。null の値は無視され Accept-Language HTTP ヘッダには追加されません。
- Accept-Language HTTP ヘッダには，指定された値がそのまま設定されます。JAX-RS エンジンは指定された値を検証しません。標準仕様に従って値を指定してください。
- Accept-Language HTTP ヘッダの値は次のメソッドでも追加できます。
  - ClientRequest.Builder クラス
    acceptLanguage(Locale... locales)メソッド
    acceptLanguage(String... locales)メソッド
    header(String name, Object value)メソッド
  - WebResource クラス
    acceptLanguage(Locale... locales)メソッド

acceptLanguage(String... locales)メソッド
header(String name, Object value)メソッド

- WebResource.Builder クラス
  acceptLanguage(Locale... locales)メソッド
  header(String name, Object value)メソッド

これらのメソッドおよび acceptLanguage(String... locales)メソッドで，受信できる言語が一つも追加されない場合の動作は，HTTP 通信を行う前に，HttpURLConnection オブジェクトに Accept-Language HTTP ヘッダを追加しない場合の動作と同様です。

cookie(Cookie cookie)メソッド

説明

Cookie を設定します。

構文

public T cookie(Cookie cookie)

パラメタ

cookie
設定する Cookie です。

戻り値

WebResource オブジェクトを構築するためのビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- null 以外の値が Cookie HTTP ヘッダに追加されます。null の値は無視され Cookie HTTP ヘッダには追加されません。
- Cookie HTTP ヘッダには，Cookie オブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは toString()メソッドによって返される値を検証しません。標準仕様に従って値を指定してください。
- Cookie HTTP ヘッダの値は次のメソッドでも追加できます。
  - ClientRequest.Builder クラス
    cookie(Cookie cookie)メソッド
    header(String name, Object value)メソッド
  - WebResource クラス
    cookie(Cookie cookie)メソッド

header(String name, Object value)メソッド

- WebResource.Builder クラス
  header(String name, Object value)メソッド

これらのメソッドおよび cookie(Cookie cookie)メソッドで，Cookie が追加されない場合の動作は，HTTP 通信を行う前に，HttpURLConnection オブジェクトに Cookie HTTP ヘッダを追加しない場合の動作と同様です。

delete()メソッド

説明

HTTP DELETE メソッドを呼び出します。リクエストにはエンティティを含めません。また，レスポンスのエンティティも受け取りません。

ステータスコードが 300 より小さい場合でレスポンスにエンティティが含まれるとき，そのエンティティは無視されます。

構文

public void delete()

throws UniformInterfaceException

パラメタ

ありません。

戻り値

ありません。

例外

UniformInterfaceException

HTTP レスポンスのステータスコードが 300 以上の場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

注意事項

- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

delete(Class<T> c)メソッド

説明

HTTP DELETE メソッドを呼び出します。リクエストにエンティティを含めません。なお，レスポンスのエンティティは受け取ります。

構文

public <T> T delete(Class<T> c)

throws UniformInterfaceException

パラメタ

c

HTTP レスポンスのエンティティの型です。

戻り値

指定された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，c パラメタが ClientResponse 型でない

注意事項

- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

delete(Class<T> c, Object requestEntity)メソッド

説明

HTTP DELETE メソッドを呼び出します。リクエストにエンティティを含めます。また，レスポンスのエンティティも受け取ります。

構文

public <T> T delete(Class<T> c,

Object requestEntity)

throws UniformInterfaceException

パラメタ

c

HTTP レスポンスのエンティティの型です。

requestEntity

HTTP リクエストのエンティティです。

戻り値

指定された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，c パラメタが ClientResponse 型でない

注意事項

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

delete(GenericType<T> gt)メソッド

説明

HTTP DELETE メソッドを呼び出します。リクエストにエンティティを含めません。なお，レスポンスのエンティティは受け取ります。

構文

public <T> T delete(GenericType<T> gt)

throws UniformInterfaceException

パラメタ

gt

HTTP レスポンスのエンティティの型を表現する GenericType オブジェクトです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，gt パラメタが ClientResponse 型を表現していない

注意事項

- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

delete(GenericType<T> gt, Object requestEntity)メソッド

説明

HTTP DELETE メソッドを呼び出します。リクエストにエンティティを含めます。また，レスポンスのエンティティも受け取ります。

構文

public <T> T delete(GenericType<T> gt,

Object requestEntity)

throws UniformInterfaceException

パラメタ

gt

HTTP レスポンスのエンティティの型を表現する GenericType オブジェクトです。

requestEntity

HTTP リクエストのエンティティです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，gt パラメタが ClientResponse 型を表現していない

注意事項

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

delete(Object requestEntity)メソッド

説明

HTTP DELETE メソッドを呼び出します。リクエストにエンティティを含めます。レスポンスのエンティティは受け取りません。

ステータスコードが 300 より小さい場合でレスポンスにエンティティが含まれるとき，そのエンティティは無視されます。

構文

public void delete(Object requestEntity)

throws UniformInterfaceException

パラメタ

requestEntity
　HTTP リクエストのエンティティです。

戻り値

ありません。

例外

UniformInterfaceException
　HTTP レスポンスのステータスコードが 300 以上の場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

注意事項

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

entity(Object entity)メソッド

説明

HTTP リクエストのエンティティを設定します。

構文

public T entity(Object entity)

パラメタ

entity
　HTTP リクエストのエンティティです。

戻り値

ビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- エンティティパラメタに使用できる Java の型については，「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。

entity(Object entity, MediaType type)メソッド

説明

HTTP リクエストのエンティティの MIME メディアタイプを設定します。

HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。ジェネリクスをエンティティに指定する場合は，GenericEntity オブジェクトを使用できます。

構文

public T entity(Object entity,

MediaType type)

パラメタ

entity
　HTTP リクエストのエンティティです。

type
　MIME メディアタイプです。

戻り値

ビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- type パラメタに指定された null でない値が Content-Type HTTP ヘッダに設定されます。null の値は無視され Content-Type HTTP ヘッダには設定されません。
- Content-Type HTTP ヘッダには，MediaType オブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは toString()メソッドによって返される値を検証しません。標準仕様に従って値を指定してください。
- Content-Type HTTP ヘッダの値は次のメソッドでも変更できます。

- ClientRequest.Builder クラス
  entity(Object entity, MediaType type)メソッド
  entity(Object entity, String type)メソッド
  header(String name, Object value)メソッド
  type(MediaType type)メソッド
  type(String type)メソッド
- WebResource クラス
  entity(Object entity, MediaType type)メソッド
  entity(Object entity, String type)メソッド
  header(String name, Object value)メソッド
  type(MediaType type)メソッド
  type(String type)メソッド
- WebResource.Builder クラス
  entity(Object entity, String type)メソッド
  header(String name, Object value)メソッド
  type(MediaType type)メソッド
  type(String type)メソッド

これらのメソッドおよび entity(Object entity, MediaType type)メソッドで，MIME メディアタイプが設定されない場合の動作は，HTTP 通信を行う前に，HttpURLConnection オブジェクトに Content-Type HTTP ヘッダを設定しない場合の動作と同様です。

entity(Object entity, String type)メソッド

説明

HTTP リクエストのエンティティの MIME メディアタイプを設定します。

HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。ジェネリクスをエンティティに指定する場合は，GenericEntity オブジェクトを使用できます。

構文

public T entity(Object entity,

String type)

パラメタ

entity

HTTP リクエストのエンティティです。

type

MIME メディアタイプです。

戻り値

ビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- type パラメタに null または空の文字列が設定された場合，IllegalArgumentException がスローされます。
- type パラメタに指定された null でない値が Content-Type HTTP ヘッダに設定されます。
- Content-Type HTTP ヘッダには，MediaType オブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは toString()メソッドによって返される値を検証しません。標準仕様に従って値を指定してください。
- Content-Type HTTP ヘッダの値は次のメソッドでも変更できます。
  - ClientRequest.Builder クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド
  - WebResource クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド
  - WebResource.Builder クラス
    entity(Object entity, MediaType type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド

これらのメソッドおよびentity(Object entity, String type)メソッドで，MIMEメディアタイプが設定されない場合の動作は，HTTP通信を行う前に，HttpURLConnectionオブジェクトにContent-Type HTTPヘッダを設定しない場合の動作と同様です。

get(Class<T> c)メソッド

説明

HTTP GETメソッドを呼び出します。

構文

public <T> T get(Class<T> c)

throws UniformInterfaceException

パラメタ

c

HTTPレスポンスのエンティティの型です。

戻り値

指定された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログにはKDJJ18888-Eのエラーメッセージが出力されます。

- HTTPレスポンスのステータスコードが204である
- HTTPレスポンスのステータスコードが300以上で，cパラメタがClientResponse型でない

注意事項

- HTTPレスポンスのエンティティに使用できるJavaの型については「25.15.2　HTTPレスポンスのエンティティに使用できるJavaの型とMIMEメディアタイプの組み合わせ」を参照してください。
- HTTPリクエストまたはHTTPレスポンスの処理中に例外が発生した場合，エラーとなり(KDJJ18888-E)，その例外をラップしたClientHandlerExceptionがスローされます。

get(GenericType<T> gt)メソッド

説明

HTTP GET メソッドを呼び出します。

構文

public <T> T get(GenericType<T> gt)

throws UniformInterfaceException

パラメタ

gt

HTTP レスポンスのエンティティの型を表現する GenericType オブジェクトです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，gt パラメタが ClientResponse 型を表現していない

注意事項

- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

head()メソッド

説明

HTTP HEAD メソッドを呼び出します。

構文

public ClientResponse head()

パラメタ

ありません。

戻り値

HTTP レスポンスを返します。

注意事項

- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

header(String name, Object value)メソッド

説明

HTTP ヘッダを追加します。

構文

public T header(String name,

Object value)

パラメタ

name
　HTTP ヘッダ名です。

value
　HTTP ヘッダの値です。

戻り値

ビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- name パラメタと value パラメタ両方の値が null の場合，呼び出しは無視されます。
- name パラメタの値が null ではない場合も，value パラメタの値が null のとき，呼び出しは無視されます。

- name パラメタの値が null で，かつ value パラメタの値が null ではない場合，エラーとなり（KDJJ18888-E），NullPointerException をラップした ClientHandlerException がスローされます。
- このメソッドで Content-Length，Connection，Host の各 HTTP ヘッダを設定することはできません。name パラメタにこれらが指定された場合，value パラメタの値が null ではないときでも，呼び出しは無視されます。なお，それぞれの HTTP ヘッダは HttpURLConnection によって設定されます。
- name パラメタで指定された HTTP ヘッダの値は value パラメタで指定された null ではないオブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは toString()メソッドによって返される値を検証しません。value パラメタには標準仕様に従って値を指定してください。
- accept，acceptLanguage，cookie の各 HTTP ヘッダは，次のメソッドでも追加できます。
  - ClientRequest.Builder クラス
    accept(MediaType... types)メソッド
    accept(String... types)メソッド
    acceptLanguage(Locale... locales)メソッド
    acceptLanguage(String... locales)メソッド
    cookie(Cookie cookie)メソッド
  - WebResource クラス
    accept(MediaType... types)メソッド
    accept(String... types)メソッド
    acceptLanguage(Locale... locales)メソッド
    acceptLanguage(String... locales)メソッド
    cookie(Cookie cookie)メソッド
  - WebResource.Builder クラス
    accept(MediaType... types)メソッド
    accept(String... types)メソッド
    acceptLanguage(Locale... locales)メソッド
    acceptLanguage(String... locales)メソッド
    cookie(Cookie cookie)メソッド

  これらのメソッドおよび header(String name, Object value)メソッドで，各 HTTP ヘッダが追加されていない場合の動作は，HttpURLConnection オブジェクトに各 HTTP ヘッダを追加しない場合と同様です。
- Content-Type HTTP ヘッダが次のメソッドですでに設定されていた場合，value パラメタの値で上書きされます。
  - ClientRequest.Builder クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド

type(MediaType type)メソッド
type(String type)メソッド

- WebResource クラス
  entity(Object entity, MediaType type)メソッド
  entity(Object entity, String type)メソッド
  header(String name, Object value)メソッド
  type(MediaType type)メソッド
  type(String type)メソッド
- WebResource.Builder クラス
  entity(Object entity, MediaType type)メソッド
  entity(Object entity, String type)メソッド
  type(MediaType type)メソッド
  type(String type)メソッド

これらのメソッドおよびheader(String name, Object value)メソッドで，MIMEメディアタイプが設定されない場合の動作は，HTTP通信を行う前に，HttpURLConnectionオブジェクトにContent-Type HTTPヘッダを設定しない場合の動作と同様です。

method(String method)メソッド

説明

HTTPメソッドを呼び出します。リクエストにはエンティティを含めません。また，レスポンスのエンティティも受け取りません。

ステータスコードが300より小さい場合でレスポンスにエンティティが含まれるとき，そのエンティティは無視されます。

構文

public void method(String method)

throws UniformInterfaceException

パラメタ

method

HTTPメソッド名です。

戻り値

ありません。

例外

UniformInterfaceException

HTTP レスポンスのステータスコードが 300 以上の場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

注意事項

- method パラメタには，すべて大文字で「GET」「HEAD」「POST」「OPTIONS」「PUT」または「DELETE」のどれかを指定してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

method(String method, Class<T> c)メソッド

説明

HTTP メソッドを呼び出します。リクエストにエンティティを含めません。なお，レスポンスのエンティティは受け取ります。

構文

public <T> T method(String method,

Class<T> c)

throws UniformInterfaceException

パラメタ

method

HTTP メソッドです。

c

HTTP レスポンスのエンティティの型です。

戻り値

指定された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，c パラメタが ClientResponse 型でない

注意事項

- method パラメタには，すべて大文字で「GET」「HEAD」「POST」「OPTIONS」「PUT」または「DELETE」のどれかを指定してください。
- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

method(String method, Class<T> c, Object requestEntity)メソッド

説明

HTTP メソッドを呼び出します。リクエストにエンティティを含めません。なお，レスポンスのエンティティは受け取ります。

構文

public <T> T method(String method,

Class<T> c,

Object requestEntity)

throws UniformInterfaceException

パラメタ

method
　HTTP メソッドです。

c
　HTTP レスポンスのエンティティの型です。

requestEntity
　HTTP リクエストのエンティティです。

戻り値

指定された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，c パラメタが ClientResponse 型でない

注意事項

- method パラメタには，すべて大文字で「GET」「HEAD」「POST」「OPTIONS」「PUT」または「DELETE」のどれかを指定してください。
- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

method(String method, GenericType<T> gt)メソッド

説明

HTTP メソッドを呼び出します。リクエストにエンティティを含めません。なお，レスポンスのエンティティは受け取ります。

構文

public <T> T method(String method,

GenericType<T> gt)

throws UniformInterfaceException

パラメタ

method

HTTP メソッドです。

gt

HTTP レスポンスのエンティティの型を表現する GenericType オブジェクトです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，gt パラメタが ClientResponse 型を表現していない

注意事項

- method パラメタには，すべて大文字で「GET」「HEAD」「POST」「OPTIONS」「PUT」または「DELETE」のどれかを指定してください。
- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

method(String method, GenericType<T> gt, Object requestEntity)メソッド

説明

HTTP メソッドを呼び出します。リクエストにエンティティを含めません。なお，レスポンスのエンティティは受け取ります。

構文

public <T> T method(String method,

GenericType<T> gt,

Object requestEntity)

throws UniformInterfaceException

パラメタ

method
HTTP メソッドです。

gt
HTTP レスポンスのエンティティの型を表現する GenericType オブジェクトです。

requestEntity
HTTP リクエストのエンティティです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException
次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，gt パラメタが ClientResponse 型を表現していない

注意事項

- method パラメタには，すべて大文字で「GET」「HEAD」「POST」「OPTIONS」「PUT」または「DELETE」のどれかを指定してください。
- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

method(String method, Object requestEntity)メソッド

説明

HTTP メソッドを呼び出します。リクエストにエンティティを含めません。なお，レスポンスのエンティティは受け取ります。

ステータスコードが 300 より小さい場合でレスポンスにエンティティが含まれるとき，そのエンティティは無視されます。

構文

public void method(String method,

Object requestEntity)

throws UniformInterfaceException

パラメタ

method
　HTTP メソッドです。

requestEntity
　HTTP リクエストのエンティティです。

戻り値

ありません。

例外

UniformInterfaceException
　HTTP レスポンスのステータスコードが 300 以上の場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

注意事項

- method パラメタには，すべて大文字で「GET」「HEAD」「POST」「OPTIONS」「PUT」または「DELETE」のどれかを指定してください。
- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

options(Class<T> c)メソッド

説明

HTTP OPTIONS メソッドを呼び出します。

構文

public <T> T options(Class<T> c)

throws UniformInterfaceException

パラメタ

c

HTTP レスポンスのエンティティの型です。

戻り値

指定された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，c パラメタが ClientResponse 型でない

注意事項

- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

options(GenericType<T> gt)メソッド

説明

HTTP OPTIONS メソッドを呼び出します。

構文

public <T> T options(GenericType<T> gt)

throws UniformInterfaceException

パラメタ

gt

HTTP レスポンスのエンティティの型を表現する GenericType オブジェクトです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，gt パラメタが ClientResponse 型を表現していない

注意事項

- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

post()メソッド

説明

HTTP POST メソッドを呼び出します。リクエストにはエンティティを含めません。また，レスポンスのエンティティも受け取りません。

ステータスコードが 300 より小さい場合でレスポンスにエンティティが含まれるとき，そのエンティティは無視されます。

構文

public void post()

throws UniformInterfaceException

パラメタ

ありません。

戻り値

ありません。

例外

UniformInterfaceException

HTTP レスポンスのステータスコードが 300 以上の場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

注意事項

- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

post(Class<T> c)メソッド

説明

HTTP POST メソッドを呼び出します。リクエストにエンティティを含めません。なお，レスポンスのエンティティは受け取ります。

構文

public <T> T post(Class<T> c)

throws UniformInterfaceException

パラメタ

c

HTTP レスポンスのエンティティの型です。

戻り値

指定された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，c パラメタが ClientResponse 型でない

注意事項

- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

post(Class<T> c, Object requestEntity)メソッド

説明

HTTP POST メソッドを呼び出します。リクエストにエンティティを含めます。また，レスポンスのエンティティも受け取ります。

構文

public <T> T post(Class<T> c,

Object requestEntity)

throws UniformInterfaceException

パラメタ

c

HTTP レスポンスのエンティティの型です。

requestEntity

HTTP リクエストのエンティティです。

戻り値

指定された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，c パラメタが ClientResponse 型でない

注意事項

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

post(GenericType<T> gt)メソッド

説明

HTTP POST メソッドを呼び出します。リクエストにエンティティを含めます。また，レスポンスのエンティティも受け取ります。

構文

public <T> T post(GenericType<T> gt)

throws UniformInterfaceException

パラメタ

gt
　HTTP レスポンスのエンティティの型を表現する GenericType オブジェクトです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException
　次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，gt パラメタが ClientResponse 型を表現していない

注意事項

- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

post(GenericType<T> gt, Object requestEntity)メソッド

説明

HTTP POST メソッドを呼び出します。リクエストにエンティティを含めます。また，レスポンスのエンティティも受け取ります。

構文

public <T> T post(GenericType<T> gt,

Object requestEntity)

throws UniformInterfaceException

パラメタ

gt
　HTTP レスポンスのエンティティの型を表現する GenericType オブジェクトです。

requestEntity
　HTTP リクエストのエンティティです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException
　次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，gt パラメタが ClientResponse 型を表現していない

注意事項

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

post(Object requestEntity)メソッド

説明

HTTP POST メソッドを呼び出します。リクエストにエンティティを含めます。レスポンスのエンティティは受け取りません。

ステータスコードが 300 より小さい場合でレスポンスにエンティティが含まれるとき，そのエンティティは無視されます。

構文

public void post(Object requestEntity)

throws UniformInterfaceException

パラメタ

requestEntity
　HTTP リクエストのエンティティです。

戻り値

ありません。

例外

UniformInterfaceException
　HTTP レスポンスのステータスコードが 300 以上の場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

注意事項

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。

- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

put()メソッド

説明

HTTP PUT メソッドを呼び出します。リクエストにはエンティティを含めません。また，レスポンスのエンティティも受け取りません。

ステータスコードが 300 より小さい場合でレスポンスにエンティティが含まれるとき，そのエンティティは無視されます。

構文

public void put()

throws UniformInterfaceException

パラメタ

ありません。

戻り値

ありません。

例外

UniformInterfaceException

HTTP レスポンスのステータスコードが 300 以上の場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

注意事項

- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

put(Class<T> c)メソッド

説明

HTTP PUT メソッドを呼び出します。リクエストにエンティティを含めます。また，レスポンスのエンティティも受け取ります。

構文

public <T> T put(Class<T> c)

throws UniformInterfaceException

パラメタ

c

HTTP レスポンスのエンティティの型です。

戻り値

指定された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，c パラメタが ClientResponse 型でない

注意事項

- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

put(Class<T> c, Object requestEntity)メソッド

説明

HTTP PUT メソッドを呼び出します。リクエストにエンティティを含めます。また，レスポンスのエンティティも受け取ります。

構文

public <T> T put(Class<T> c,

Object requestEntity)

throws UniformInterfaceException

パラメタ

c

HTTP レスポンスのエンティティの型です。

requestEntity

HTTP リクエストのエンティティです。

戻り値

指定された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，c パラメタが ClientResponse 型でない

注意事項

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

put(GenericType<T> gt)メソッド

説明

HTTP PUT メソッドを呼び出します。リクエストにエンティティを含めます。また，レスポンスのエンティティも受け取ります。

構文

public <T> T put(GenericType<T> gt)

throws UniformInterfaceException

パラメタ

gt

HTTP レスポンスのエンティティの型を表現する GenericType オブジェクトです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，gt パラメタが ClientResponse 型を表現していない

注意事項

- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

put(GenericType<T> gt, Object requestEntity)メソッド

説明

HTTP PUT メソッドを呼び出します。リクエストにエンティティを含めます。また，レスポンスのエンティティも受け取ります。

構文

public <T> T put(GenericType<T> gt,

Object requestEntity)

throws UniformInterfaceException

パラメタ

gt

HTTP レスポンスのエンティティの型を表現する GenericType オブジェクトです。

requestEntity

HTTP リクエストのエンティティです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException

次のどちらかの条件を満たした場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

- HTTP レスポンスのステータスコードが 204 である
- HTTP レスポンスのステータスコードが 300 以上で，gt パラメタが ClientResponse 型を表現していない

注意事項

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP レスポンスのエンティティに使用できる Java の型については「25.15.2　HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

put(Object requestEntity)メソッド

説明

HTTP PUT メソッドを呼び出します。リクエストにエンティティを含めます。また，レスポンスのエンティティも受け取ります。

ステータスコードが 300 より小さい場合でレスポンスにエンティティが含まれるとき，そのエンティティは無視されます。

構文

public void put(Object requestEntity)

throws UniformInterfaceException

パラメタ

requestEntity
　HTTP リクエストのエンティティです。

戻り値

指定された GenericType オブジェクトで表現された型のオブジェクトを返します。

例外

UniformInterfaceException
　HTTP レスポンスのステータスコードが 300 以上の場合にスローされます。ログには KDJJ18888-E のエラーメッセージが出力されます。

注意事項

- HTTP リクエストのエンティティに使用できる Java の型については「25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ」を参照してください。
- HTTP リクエストまたは HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

type(MediaType type)メソッド

説明

MIME メディアタイプを設定します。

構文

public T type(MediaType type)

パラメタ

type
　MIME メディアタイプです。

戻り値

ビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- type パラメタに指定された null でない値が Content-Type HTTP ヘッダに設定されます。null の値は無視され Content-Type HTTP ヘッダには設定されません。
- Content-Type HTTP ヘッダには，MediaType オブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは toString()メソッドによって返される値を検証しません。標準仕様に従って値を指定してください。
- Content-Type HTTP ヘッダが次のメソッドですでに設定されていた場合，値は上書きされます。
  - ClientRequest.Builder クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド
  - WebResource クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド
  - WebResource.Builder クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(String type)メソッド

  これらのメソッドおよび type(MediaType type)メソッドで，MIME メディアタイプが設定されない場合の動作は，HTTP 通信を行う前に HttpURLConnection オブジェクトに Content-Type HTTP ヘッダを設定しない場合の動作と同様です。

type(String type)メソッド

説明

MIME メディアタイプを設定します。

構文

public T type(String type)

パラメタ

type

MIME メディアタイプです。

戻り値

ビルダ（WebResource.Builder オブジェクト）を返します。

注意事項

- type パラメタに null または空の文字列が設定された場合，IllegalArgumentException がスローされます。
- Content-Type HTTP ヘッダには，MediaType クラスの valueOf(String)スタティックメソッドのパラメタに type パラメタを指定して構築した MediaType オブジェクトの toString()メソッドによって返される値が設定されます。JAX-RS エンジンは type パラメタに指定される値を検証しません。標準仕様に従って値を指定してください。
- Content-Type HTTP ヘッダが次のメソッドですでに設定されていた場合，type パラメタの値で上書きされます。
  - ClientRequest.Builder クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド
  - WebResource クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド
    type(String type)メソッド
  - WebResource.Builder クラス
    entity(Object entity, MediaType type)メソッド
    entity(Object entity, String type)メソッド
    header(String name, Object value)メソッド
    type(MediaType type)メソッド

  これらのメソッドおよび type(String type)メソッドで，MIME メディアタイプが設定されない場合の動作は，HTTP 通信を行う前に，HttpURLConnection オブジェクトに Content-Type HTTP ヘッダを設定しない場合の動作と同様です。

25.12　DefaultClientConfig クラスの定数およびメソッドの仕様と注意事項

ここでは，DefaultClientConfig クラスの定数とメソッドの仕様，および利用する場合の注意事項について説明します。

PROPERTY_BUFFER_RESPONSE_ENTITY_ON_EXCEPTION 定数

説明

例外発生時のレスポンスエンティティのバッファリングを設定するためのプロパティです。

構文

static final java.lang.String PROPERTY_BUFFER_RESPONSE_ENTITY_ON_EXCEPTION

注意事項

- 例外発生時のレスポンスエンティティのバッファリングの設定については，「13.1.2　共通定義ファイルの設定項目」を参照してください。プロパティについては「25.1.1　サポートするプロパティとフィーチャ」を参照してください。

PROPERTY_CHUNKED_ENCODING_SIZE 定数

説明

チャンク転送エンコーディングを設定するためのプロパティです。

構文

static final java.lang.String PROPERTY_CHUNKED_ENCODING_SIZE

注意事項

- チャンク転送エンコーディングの設定については，「13.1.2　共通定義ファイルの設定項目」を参照してください。プロパティについては「25.1.1　サポートするプロパティとフィーチャ」を参照してください。

PROPERTY_CONNECT_TIMEOUT 定数

説明

クライアントソケットの接続タイムアウト値を設定するためのプロパティです。

構文

static final java.lang.String PROPERTY_CONNECT_TIMEOUT

注意事項

- クライアントソケットの接続タイムアウト値の設定については，「13.1.2　共通定義ファイルの設定項目」を参照してください。プロパティについては「25.1.1　サポートするプロパティとフィーチャ」を参照してください。

PROPERTY_FOLLOW_REDIRECTS 定数

説明

自動リダイレクトを設定するためのプロパティです。

構文

static final java.lang.String PROPERTY_FOLLOW_REDIRECTS

注意事項

- 自動リダイレクトの設定については，「13.1.2　共通定義ファイルの設定項目」を参照してください。プロパティについては「25.1.1　サポートするプロパティとフィーチャ」を参照してください。

PROPERTY_READ_TIMEOUT 定数

説明

クライアントソケットの読み込みタイムアウト値を設定するためのプロパティです。

構文

static final java.lang.String PROPERTY_READ_TIMEOUT

注意事項

- クライアントソケットの接続タイムアウト値の設定については，「13.1.2　共通定義ファイルの設定項目」を参照してください。プロパティについては「25.1.1　サポートするプロパティとフィーチャ」を参照してください。

getPropertyAsFeature(String featureName)メソッド

説明

Boolean 型のプロパティをフィーチャとしてプロパティマップから取得します。

構文

public boolean getPropertyAsFeature(String name)

パラメタ

featurename
　取得するプロパティ名（フィーチャ名）です。

戻り値

プロパティマップに指定された名称の Boolean 型のプロパティがある場合でその値が true のときは true を，そのほかのときは false を返します。

注意事項

ありません。

getFeatures()メソッド

説明

すべてのフィーチャを含むマップを取得します。

構文

public Map<String,Boolean> getFeatures()

パラメタ

ありません。

戻り値

フィーチャのマップを返します。null を返すことはありません。

注意事項

getFeatures()メソッドの使用例を次に示します。

```
// ClientConfigオブジェクトを生成する
ClientConfig cc = new DefaultClientConfig();
// ClientConfigオブジェクトにJSON POJOマッピングを有効にするための
// フィーチャを追加する
cc.getFeatures().put(JSONConfiguration.FEATURE_POJO_MAPPING, true);
// 上記で生成したClientConfigオブジェクトを指定して
// Clientオブジェクトを生成する
Client client = Client.create(cc);
// 読み込みタイムアウト値を設定する
client.getProperties().put(ClientConfig.PROPERTY_READ_TIMEOUT, 10000);

// JSONフォーマットにマッピングするPOJOオブジェクトを生成する
Pojo pojo = new Pojo();

// HTTPリクエストを生成し，マッピングされたPOJOオブジェクトを送信する
ClientRequest cRequest = ClientRequest.create().entity(pojo).type("application/json").build(
new URI("http://example.com/example"), "POST");
try{
  // HTTPレスポンスをClientResponseオブジェクトとして受信する
  ClientResponse cResponse = client.handle(cRequest);
} catch(ClientHandlerException e){
  // 適切な処理を実行する
}
```

この例では，まず，getFeatures メソッドで変更可能なプロパティマップを取得し，JSON POJO マッピングを有効にするためのフィーチャを追加しています。次に ClientRequest オブジェクトを作成し，Client クラスの handle()メソッドを利用して HTTP 通信を行い，HTTP レスポンスを ClientResponse オブジェクトとして受信しています。HTTP リクエストのエンティティとして送信した POJO オブジェクトは，JSON POJO マッピングによって JSON フォーマットでマッピングされています。これは JSON POJO マッピングを有効にするためのフィーチャを追加したことで実現されています。

getFeature(String featureName)メソッド

説明

フィーチャの値を取得します。

構文

public boolean getFeature(String featureName)

パラメタ

featureName

フィーチャ名です。

戻り値

指定された名称のフィーチャが存在する場合で値が true のときは true を，そのほかのときは false を返します。

注意事項

- null または空の文字列が featureName パラメタに設定された場合は false を返します。

getProperties()メソッド

説明

クライアントに関連するすべてのフィーチャのマップを取得します。

構文

public Map<String,Object> getProperties()

パラメタ

ありません。

戻り値

フィーチャのマップを返します。null を返すことはありません。

注意事項

getProperties()メソッドの使用例を次に示します。

```
// クライアントを設定する
ClientConfig cc = new DefaultClientConfig();

// 読み込みタイムアウト値を設定する
cc.getProperties().put(ClientConfig.PROPERTY_READ_TIMEOUT, 10000);

// クライアントの設定を利用してClientオブジェクトを生成する
Client client = Client.create(cc);

// HTTPリクエストを生成する
ClientRequest cRequest = ClientRequest.create().build(new URI("http://example.com/example"),
 "GET");
try{
  // HTTPレスポンスをClientResponseオブジェクトとして受信する
  ClientResponse cResponse = client.handle(cRequest);
} catch(ClientHandlerException e){
  // 適切な処理を実行する
}
```

この例では，まず，getProperties メソッドで変更可能なプロパティマップを取得し，読み込みタイムアウト値を 10,000 ミリ秒に設定しています。Client クラスの create(ClientConfig cc)メソッドではこのクライアント設定を使って，クライアントインスタンスを作成します。次に ClientRequest オブジェクトを作成し，Client クラスの handle メソッドを利用して HTTP 通信を行い，HTTP レスポンスを ClientResponse オブジェクトとして受信しています。HTTP レスポンスが完全に読み込まれるまでに読み込みタイムアウトが発生した場合，エラーとなり（KDJJ18888-E），SocketTimeoutException をラップした ClientHandlerException がスローされます。

getProperty(String propertyName)メソッド

説明

プロパティの値を取得します。

構文

public Object getProperty(String propertyName)

パラメタ

propertyName

プロパティ名です。

戻り値

プロパティの値を返します。propertyName パラメタに指定された名称のパラメタが存在しない場合は null を返します。

注意事項

- null または空の文字列が propertyName パラメタに指定された場合は null を返します。

25.13　HTTPSProperties クラスの定数，コンストラクタおよびメソッドの仕様と注意事項

ここでは，HTTPSProperties クラスの定数，コンストラクタおよびメソッドの仕様，および利用する場合の注意事項について説明します。

PROPERTY_HTTPS_PROPERTIES 定数

説明

HTTPSProperties オブジェクトを設定するためのプロパティです。変更可能なプロパティマップに HTTPSProperties オブジェクトを追加するときに使用します。プロパティについては「25.1.1　サポートするプロパティとフィーチャ」を参照してください。

構文

public static final java.lang.String PROPERTY_HTTPS_PROPERTIES

HTTPSProperties()コンストラクタ

説明

javax.net.ssl.SSLContext クラスの getInstance()メソッドを引数に"SSL"を指定して生成した SSLContext オブジェクトから HTTPSProperties オブジェクトを構築します。構築される HTTPSProperties オブジェクトは，javax.net.ssl.HostnameVerifier オブジェクトを含みません。

構文

public HTTPSProperties()

throws java.security.NoSuchAlgorithmException

パラメタ

ありません。

戻り値

ありません。

例外

NoSuchAlgorithmException

SSLContext オブジェクトが生成できなかった場合にスローされます。

注意事項

ありません。

HTTPSProperties(HostnameVerifier hv)コンストラクタ

説明

引数に指定された javax.net.ssl.HostnameVerifier オブジェクトと javax.net.ssl.SSLContext クラスの getInstance()メソッドを引数に"SSL"を指定して生成した SSLContext オブジェクトから HTTPSProperties オブジェクトを構築します。

構文

public HTTPSProperties(HostnameVerifier hv)

throws NoSuchAlgorithmException

パラメタ

hv

構築する HTTPSProperties オブジェクトに設定する HostnameVerifier オブジェクトです。

戻り値

ありません。

例外

NoSuchAlgorithmException

SSLContext オブジェクトが生成できなかった場合にスローされます。

注意事項

- hv パラメタに null 値が設定された場合の動作は，HTTP リクエストを生成する前に HttpsURLConnection クラスに HostnameVerifier オブジェクトを設定しない場合と同じです。
- null 以外の値を hv パラメタに指定した場合の動作は，同じ値を HttpsURLConnection クラスの setHostnameVerifier()メソッドに指定した場合と同じです。なお，JAX-RS エンジンはパラメタに指定された値を検証しません。パラメタには標準仕様に従って値を指定してください。

HTTPSProperties(HostnameVerifier hv, SSLContext c)コンストラクタ

説明

引数に指定された javax.net.ssl.HostnameVerifier および javax.net.ssl.SSLContext オブジェクトから HTTPSProperties オブジェクトを構築します。

構文

public HTTPSProperties(HostnameVerifier hv,

SSLContext c)

パラメタ

hv

構築する HTTPSProperties オブジェクトに設定する HostnameVerifier です。

c

構築する HTTPSProperties オブジェクトに設定する SSLContext です。null は指定できません。

戻り値

ありません。

注意事項

- c パラメタに null が設定された場合，IllegalArgumentException がスローされます。
- hv パラメタに null が設定された場合の動作は，HTTP リクエストを生成する前に HttpsURLConnection クラスに HostnameVerifier オブジェクトを設定しない場合と同じです。
- null 以外の値を hv パラメタに指定した場合の動作は，同じ値を HttpsURLConnection クラスの setHostnameVerifier()メソッドに指定した場合と同じです。
- null 以外の値を c パラメタに指定した場合の動作は，c パラメタに対して getSocketFactory()メソッドを呼び出して取得した値を HttpsURLConnection クラスの setSSLSocketFactory()メソッドに指定した場合と同じです。
- JAX-RS エンジンはパラメタに指定された値を検証しません。パラメタには標準仕様に従って値を指定してください。

getHostnameVerifier()メソッド

説明

HostnameVerifier オブジェクトを取得します。

構文

public HostnameVerifier getHostnameVerifier()

パラメタ

ありません。

戻り値

HostnameVerifier オブジェクトを返します。HostnameVerifier オブジェクトがこのオブジェクト（HTTPSProperties オブジェクト）に設定されていない場合は null を返します。

注意事項

- HostnameVerifier オブジェクトは HTTPSProperties(HostnameVerifier hv)コンストラクタおよび HTTPSProperties(HostnameVerifier hv, SSLContext c)コンストラクタで設定できます。

getSSLContext()メソッド

説明

SSLContext オブジェクトを取得します。

構文

public SSLContext getSSLContext()

パラメタ

ありません。

戻り値

SSLContext オブジェクトを返します。SSLContext オブジェクトがこのオブジェクト (HTTPSProperties オブジェクト) に設定されていない場合は null を返します。

注意事項

- SSLContext オブジェクトは HTTPSProperties(HostnameVerifier hv, SSLContext)コンストラクタで設定できます。

25.14 MultivaluedMapImpl クラスのコンストラクタおよびメソッド仕様と注意事項

MultivaluedMapImpl クラスは，javax.ws.rs.core.MultivaluedMap インタフェースの実装クラスです。MultivaluedMapImpl クラスのメソッドの仕様については JAX-RS API のドキュメントを参照してください。

ここでは，MultivaluedMapImpl クラスのコンストラクタおよびメソッドを利用する場合の注意事項について説明します。

- MultivaluedMapImpl クラスはスレッドセーフではありません。
- MultivaluedMapImpl クラスは簡易的な実装クラスであり，最小限の機能だけを提供しています。javax.ws.rs.core.MultivaluedMap インタフェースの add メソッド，getFirst メソッド，および putSingle メソッドだけを使用できます。

  java.util.Map インタフェースのほかのメソッド（put メソッドや remove メソッドなど）は呼び出すことができません。add メソッド，getFirst メソッド，および putSingle メソッド以外のメソッドを使用した場合，JAX-RS エンジンの動作は保証されません。
- MultivaluedMapImpl クラスの機能が不足している場合は，javax.ws.rs.core.MultivaluedMap インタフェースを実装したクラスを個別に作成してください。
- add(String key, String value)メソッド，および putSingle(String key, String value)メソッドで value パラメタに null が指定された場合，null の代わりに空の文字列（""）が値として追加されます。

25.15　使用できる Java の型と MIME メディアタイプの組み合わせ

この節では，RESTful Web サービス用クライアント API のエンティティパラメタおよび戻り値に使用できる Java の型と MIME メディアタイプの組み合わせについて説明します。なお，POJO に JAXB 仕様のアノテーションは使用しないでください。使用した場合，説明とは異なる動作になるおそれがあります。

25.15.1　HTTP リクエストのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ

HTTP リクエストのエンティティボディに使用できる Java 型と MIME メディアタイプの組み合わせを次の表に示します。なお，これ以外の組み合わせの場合，JAX-RS エンジンの動作は保証されません。

表 25-6　エンティティボディに使用できる Java の型と MIME メディアタイプの組み合わせ

項番	Java の型	charset※1	Content-Type HTTP ヘッダ	Content-Type HTTP ヘッダ※2（JAX-RS エンジン）
1	byte[]	×	任意(*/*)	application/octet-stream
2	java.lang.String	○	任意(*/*)	text/plain
3	java.io.InputStream	×	任意(*/*)	application/octet-stream
4	java.io.Reader	○	任意(*/*)	text/plain
5	java.io.File	×	任意(*/*)	application/octet-stream
6	javax.activation.DataSource	×	任意(*/*)	application/octet-stream
7	javax.xml.transform.Source※3	×	任意(*/*)	application/xml
8	javax.xml.bind.JAXBElement<String>※4	○	ext/xml, application/xml, application/*+xml	application/xml
9	XmlRootElement アノテーションでアノテートされた JAXB クラス※4	○	text/xml, application/xml, application/*+xml	application/xml
10	javax.ws.rs.core.MultivaluedMap<String,String>	○	application/x-www-form-urlencoded	application/x-www-form-urlencoded
11	javax.ws.rs.core.StreamingOutput	×	任意(*/*)	application/octet-stream
12	org.w3c.dom.Document	×	任意(*/*)	application/xml
13	java.awt.image.RenderedImage	×	image/jpeg	application/octet-stream

項番	Java の型	charset※1	Content-Type HTTP ヘッダ	Content-Type HTTP ヘッダ※2（JAX-RS エンジン）
14	javax.ws.rs.core.GenericEntity<T>※5	△	T に指定した型と同じ MIME メディアタイプです。	T に指定した型と同じ MIME メディアタイプです。
15	POJO※6	×※7	application/json	application/octet-stream

（凡例）

任意(*/*)：すべての MIME メディアタイプをサポートしていることを示します。

注※1

Content-Type HTTP ヘッダに charset パラメタが含まれる場合，HTTP リクエストに変換するときにその情報が考慮されるかどうかを示します。

○：考慮されます。Content-Type HTTP ヘッダに charset パラメタが含まれない場合は，UTF-8 が仮定されます。

△：T に指定した型に依存します。

×：考慮されません。

注※2

ClientRequest オブジェクトまたは Web リソースオブジェクトで，Content-Type HTTP ヘッダが指定されていない場合，エンティティが null 以外のときは，このカラムで指定した Content-Type を考慮して，HTTP リクエストの Content-Type HTTP ヘッダを設定します。

エンティティが null のときの動作は，HTTP リクエストをする前に，HttpURLConnection オブジェクトに Content-Type HTTP ヘッダを設定しないときの動作と同じです。

注※3

次に示す実装クラスを使用できます。

・javax.xml.transform.stream.StreamSource

・javax.xml.transform.sax.SAXSource

・javax.xml.transform.dom.DOMSource

注※4

MIME メディアタイプが application/fastinfoset または application/json の場合，エラーにならないで正常終了します。

注※5

T にはこの表の項番 1 から項番 13，および項番 15 の型を指定できます。

注※6

JSON POJO マッピングを有効にしてください。JSON POJO マッピングが有効でない場合の動作は，サポートされない Java 型がエンティティパラメタに指定された場合の動作と同じです。JSON POJO マッピングを有効にする方法については「18. JSON POJO マッピング」を参照してください。

注※7

Content-Type HTTP ヘッダに charset パラメタを追加しないでください。

指定した Java の型がサポートされていない型で，値が null でない場合，エラーとなり（KDJJ10032-E および KDJJ18888-E），ClientHandlerException がスローされます。ただし，指定した Java の型が javax.mail.internet.MimeMultipart で，MIME メディアタイプが multipart/*のときはエラーにならずに正常終了します。

次に示す Java の型では，HTTP リクエストのエンティティボディが使用できない MIME メディアタイプの場合，エラーとなり（KDJJ10032-E および KDJJ18888-E），ClientHandlerException がスローされます。

1. javax.xml.bind.JAXBElement<String>
2. XmlRootElement アノテーションでアノテートされた JAXB クラス
3. javax.ws.rs.core.MultivaluedMap<String, String>
4. java.awt.image.RenderedImage
5. POJO

ただし，1.または 2.で HTTP リクエストのエンティティボディの MIME メディアタイプが application/fastinfoset または application/json のときは，エラーにならないで正常終了します。

HTTP リクエストおよび HTTP レスポンスの処理中に例外が発生した場合，エラーとなり（KDJJ18888-E），その例外をラップした ClientHandlerException がスローされます。

また，HTTP メソッドの種類によっては次の注意事項があります。

- HTTP メソッドが DELETE，HEAD，または OPTIONS のどれかの場合，HTTP リクエストのエンティティボディには null でない値を指定しないでください。null でない値が指定されたときはエラーとなり（KDJJ18888-E），ClientHandlerException がスローされます。
- HTTP メソッドが GET の場合，HTTP リクエストのエンティティボディには null でない値を指定しないでください。null でない値が指定されたときは無視されます。
- HTTP メソッドが POST または PUT で HTTP リクエストのエンティティボディに null が指定された場合，送信される HTTP リクエストにエンティティボディは含まれません。

25.15.2 HTTP レスポンスのエンティティに使用できる Java の型と MIME メディアタイプの組み合わせ

戻り値に使用できる Java 型と MIME メディアタイプの組み合わせを次の表に示します。

表 25–7 エンティティボディの戻り値に使用できる Java の型と MIME メディアタイプの組み合わせ

項番	Java の型	charset※1	MIME メディアタイプ
1	byte[]	×	任意(*/*)
2	java.lang.String	○	任意(*/*)
3	java.io.InputStream	×	任意(*/*)
4	java.io.Reader	○	任意(*/*)
5	java.io.File※2	×	任意(*/*)
6	javax.activation.DataSource	×	任意(*/*)
7	javax.xml.transform.Source※3	×	任意(*/*)

項番	Java の型	charset※1	MIME メディアタイプ
8	javax.xml.bind.JAXBElement<String>※4, ※5	×	text/xml, application/xml, application/*+xml
9	XmlRootElement アノテーションおよび/または XmlType アノテーションでアノテートされた JAXB クラス※5	×	text/xml, application/xml, application/*+xml
10	javax.ws.rs.core.MultivaluedMap<String,String>	○	application/x-www-form-urlencoded
11	org.w3c.dom.Document	×	任意(*/*)
12	java.awt.image.RenderedImage	×	application/octet-stream, image/jpeg
13	com.cosminexus.jersey.core.provider.EntityHolder<T>※6	△	T に指定した型と同じ MIME メディアタイプです。
14	com.cosminexus.jersey.api.client.GenericType<T>※7	△	T に指定した型と同じ MIME メディアタイプです。
15	POJO※8	○	application/json

（凡例）

任意(*/*)：すべての MIME メディアタイプをサポートしていることを示します。

注※1

Content-Type HTTP ヘッダに charset パラメタが含まれる場合，HTTP レスポンスにインジェクトするときにその情報が考慮されるかどうかを示します。

○：考慮されます。Content-Type HTTP ヘッダに charset パラメタが含まれない場合は，UTF-8 が仮定されます。

△：T に指定した型に依存します。

×：考慮されません。

注※2

JAX-RS エンジンは，ローカルマシンに temp ディレクトリを作成し，一時ファイルを保存します。

注※3

次に示す実装クラスを使用できます。

・javax.xml.transform.stream.StreamSource

・javax.xml.transform.sax.SAXSource

・javax.xml.transform.dom.DOMSource

なお，javax.xml.transform.stream.StreamSource，または javax.xml.transform.sax.SAXSource を使用する場合は，汎用型（ClientResponse）で HTTP レスポンスを受信し，ClientResponse クラスの getEntity()メソッドでエンティティのオブジェクトを取得してください。またその際，getEntity()メソッドを呼び出す前に，必ず bufferEntity()メソッドを呼び出してください。汎用型（ClientResponse）で HTTP レスポンスを受信する方法については「11.4.1　Web リソースクライアントのユースケース」を参照してください。

注※4

この表の項番 14 の T として使用します。これ以外の使用は保証されません。

注※5

MIME メディアタイプが application/fastinfoset または application/json の場合，エラーにならないで正常終了します。

注※6

この表の項番 14 の T として使用します。T にはこの表の項番 2 から項番 12，および項番 15 の型を指定できます。これ以外の使用は保証されません。

注※7

T にはこの表の項番 2 から項番 13，および項番 15 の型を指定できます。

注※8

JSON POJO マッピングを有効にしてください。JSON POJO マッピングが有効でない場合の動作は，サポートされない Java 型がエンティティパラメタに指定された場合の動作と同じです。JSON POJO マッピングを有効にする方法については「18. JSON POJO マッピング」を参照してください。

指定した Java の型がサポートされない型の場合，エラーとなり（KDJJ10031-E および KDJJ18888-E），ClientHandlerException がスローされます。

戻り値が次に示す Java の型で，HTTP レスポンスのエンティティボディが使用できない MIME メディアタイプの場合，エラーとなり（KDJJ10031-E または KDJJ18888-E），ClientHandlerException がスローされます。

- javax.xml.bind.JAXBElement<String>
- XmlRootElement アノテーションおよび/または XmlType アノテーションでアノテートされた JAXB クラス
- javax.ws.rs.core.MultivaluedMap<String,String>
- java.awt.image.RenderedImage
- POJO

HTTP レスポンスのエンティティボディからの変換で IOException が発生した場合，エラーとなり（KDJJ18888-E），ClientHandlerException がスローされます。

HTTP レスポンスのエンティティボディからの変換で IOException 以外の例外がスローされた場合は，対応する例外がスローされます。なお，ログを確認するときは，JAX-RS 機能のログファイルではなく J2EE サーバのログファイルを確認してください

java.awt.image.RenderedImage のエンティティパラメタでは，Content-Type HTTP ヘッダが image/*の場合，エラーとなり（KDJJ18888-E），java.io.IOException をラップした ClientHandlerException がスローされます。

com.cosminexus.jersey.api.client.GenericType<T>のエンティティパラメタでは，HTTP レスポンスにエンティティボディがある場合，T が表の項番 1 のとき，エラーとなり（KDJJ10031-E および KDJJ18888-E），ClientHandlerException がスローされます。

com.cosminexus.jersey.core.provider.EntityHolder<T>のエンティティパラメタでは，HTTP レスポンスにエンティティボディがある場合，次に示すどちらかのときにエラーとなり（KDJJ10003-E），javax.ws.rs.WebApplicationException がスローされます。

- T がサポートされない型（表の項番 1，項番 13，および項番 14）のとき

- T がサポートされる型（表の項番 2 から項番 12，および項番 15）だが，HTTP レスポンスのエンティティボディが使用できない MIME メディアタイプのとき

クライアント API が J2EE サーバ上で動作する場合，エンティティボディに含まれるフォームパラメタ数の上限値のデフォルトは 10,000 です。エンティティパラメタの型が javax.ws.rs.core.MultivaluedMap<String,String>，または com.cosminexus.jersey.core.provider.GenericType<EntityHolder<javax.ws.rs.core.MultivaluedMap<String,String>>>のとき，レスポンスのパラメタ数が指定した値を超えたときは，RuntimeException がスローされます。必要に応じて J2EE サーバ用ユーザプロパティファイル（usrconf.properties）の webserver.connector.limit.max_parameter_count プロパティで変更してください。なお，ログを確認するときは，JAX-RS 機能のログファイルではなく J2EE サーバのログファイルを確認してください。

HTTP レスポンスに Content-Type HTTP ヘッダが存在しない場合，MIME メディアタイプは application/octet-stream であると見なされます。

25.16　RESTful Web サービス用クライアント API のスレッドセーフ性

RESTful Web サービス用クライアント API のスレッドセーフ性について次の表に示します。

表 25–8　RESTful Web サービス用クライアント API のスレッドセーフ性

項番	インタフェースまたはクラス	コンストラクタ／メソッド／フィールド	スレッドセーフ性※1
com.cosminexus.jersey.api.client パッケージ			
1	Client	create()	○
2		create(ClientConfig cc)	○※2
3		destroy()	×
4		getProperties()	×
5		handle(ClientRequest request)	○※2
6		resource(String u)	○
7		resource(URI u)	○
8		setChunkedEncodingSize(Integer chunkSize)	×
9		setConnectTimeout(Integer interval)	×
10		setFollowRedirects(Boolean redirect)	×
11		setReadTimeout(Integer interval)	×
12	ClientRequest	すべてのメソッド	×
13	ClientRequest.Builder	すべてのメソッド	×
14	ClientResponse	すべてのメソッド	×
15	GenericType	すべてのメソッド	○
16	WebResource	accept(MediaType... types)	○
17		accept(String... types)	○
18		acceptLanguage(Locale... locales)	○
19		acceptLanguage(String... locales)	○
20		cookie(Cookie cookie)	○
21		delete()	○
22		delete(Class<T> c)	○
23		delete(Class<T> c, Object requestEntity)	△
24		delete(GenericType<T> gt)	○

項番	インタフェースまたはクラス	コンストラクタ／メソッド／フィールド	スレッドセーフ性※1
25		delete(GenericType<T> gt, Object requestEntity)	△
26		delete(Object requestEntity)	△
27		entity(Object entity)	○
28		entity(Object entity, MediaType type)	○
29		entity(Object entity, String type)	○
30		get(Class<T> c)	○
31		get(GenericType<T> gt)	○
32		getRequestBuilder()	○
33		getURI()	○
34		getUriBuilder()	○
35		head()	○
36		header(String name, Object value)	○
37		method(String method)	○
38		method(String method, Class<T> c)	○
39		method(String method, Class<T> c, Object requestEntity)	△
40		method(String method, GenericType<T> gt)	○
41		method(String method, GenericType<T> gt, Object requestEntity)	△
42		method(String method, Object requestEntity)	△
43		options(Class<T> c)	○
44		options(GenericType<T> gt)	○
45		path(String path)	○
46		post()	○
47		post(Class<T> c)	○
48		post(Class<T> c, Object requestEntity)	△
49		post(GenericType<T> gt)	○
50		post(GenericType<T> gt, Object requestEntity)	△
51		post(Object requestEntity)	△
52		put()	○

項番	インタフェースまたはクラス	コンストラクタ／メソッド／フィールド	スレッドセーフ性※1
53		put(Class<T> c)	○
54		put(Class<T> c, Object requestEntity)	△
55		put(GenericType<T> gt)	○
56		put(GenericType<T> gt, Object requestEntity)	△
57		put(Object requestEntity)	△
58		queryParam(String key, String value)	○
59		queryParams(MultivaluedMap<String,String> params)	○
60		type(MediaType type)	○
61		type(String type)	○
62		uri(URI uri)	○
63	WebResource.Builder	すべてのメソッド	×
com.cosminexus.jersey.api.client.config パッケージ			
64	DefaultClientConfig	すべてのメソッド	×
65	Provider	−	○

（凡例）

○：スレッドセーフです。

△：要求エンティティ（requestEntity パラメタ）に指定するインスタンスがスレッドセーフの場合，スレッドセーフです。

×：スレッドセーフではありません。

−：メソッドはありません。

注※1

スレッドセーフではない場合，同じオブジェクトのこのメソッドを複数スレッドから呼び出さないでください。

注※2

引数に指定するクラスはスレッドセーフではありません。引数に指定するクラスのインスタンスはそれぞれのスレッドで生成してください。

26 WSDL インポート機能

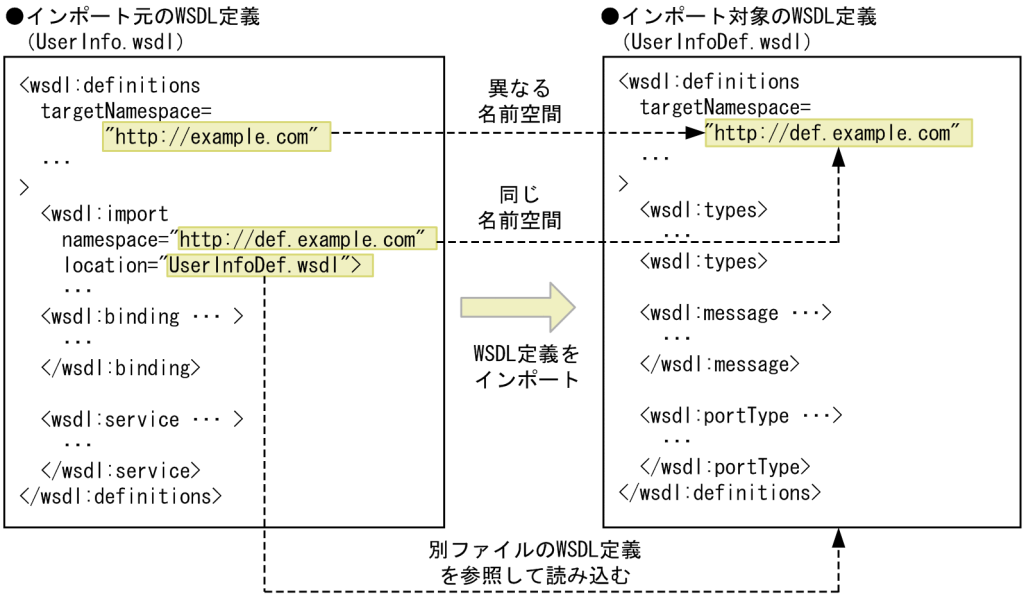
この章では，WSDL のインポート機能について説明します。

26.1　WSDL インポート機能とは

WSDL インポート機能とは，wsdl:import 要素を使用して，共通部品として別ファイルの WSDL 定義を読み込む機能です。

WSDL インポート機能のイメージを次の図に示します。

図 26-1　WSDL インポート機能のイメージ



26.2　インポートできる WSDL 定義

インポート対象の WSDL 定義の条件，およびインポート時の留意点について説明します。

26.2.1　インポート対象の WSDL 定義の条件

インポート対象の WSDL 定義は，次の条件を満たしている必要があります。

- ルート要素に wsdl:definitions が指定されていること。
  ルート要素に wsdl:definitions 以外の形式のファイルを指定した場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51200-E）。
- WSDL 定義にアクセス権限があること。
  アクセス権限がない WSDL 定義を指定した場合，JDK のエラーとなり，処理が終了されます。

インポート対象の WSDL 定義の拡張子は任意です。

26.2.2　複数の WSDL 定義のインポート

複数の WSDL 定義を組み合わせてインポートできます。ただし，最初の開始点となるインポート元の WSDL 定義に，wsdl:service 要素を定義する必要があります。以降の階層では，wsdl:service 要素以外の要素を組み合わせて，WSDL 定義をインポートできます。wsdl:service 要素が定義されていない場合の動作については，「20.1.16　wsdl:service 要素」を参照してください。

WSDL 定義の階層的なインポートの方法を，正しい場合の例および誤っている場合の例に分けて図に示します。

図 26–2　WSDL 定義の階層的なインポート（正しい場合の例）

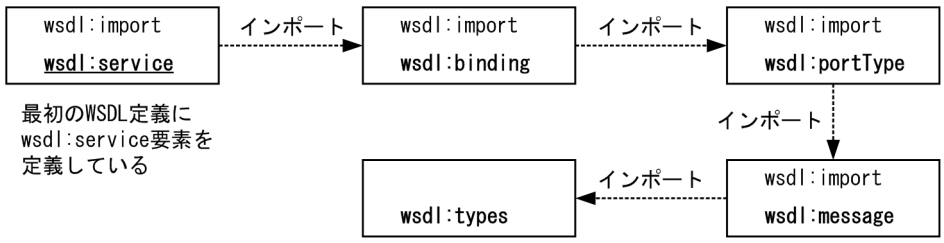
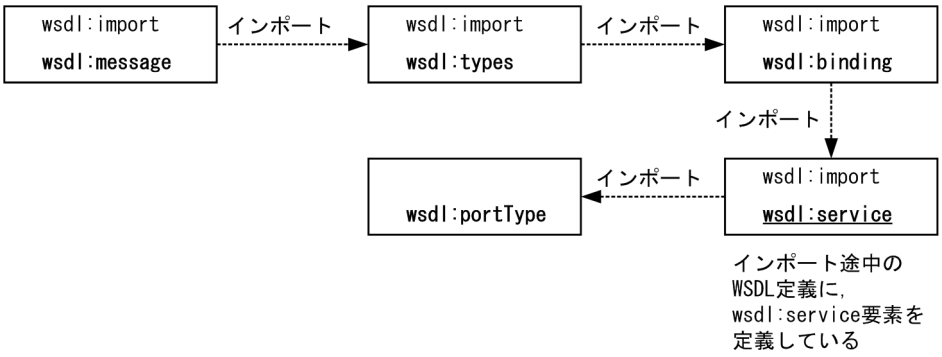


図 26-3　WSDL 定義の階層的なインポート（誤っている場合の例）

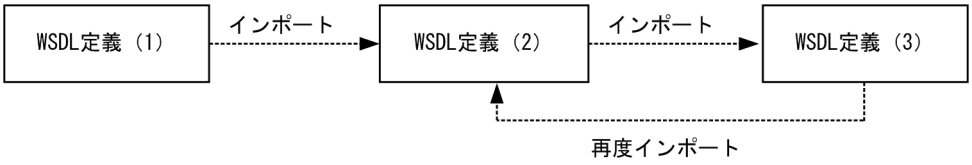


26.2.3　WSDL 定義の再帰的なインポート

WSDL インポート機能では，WSDL 定義を再帰的にインポートできません。再帰的にインポートしようとしても，WSDL 定義を読み込まないため，wsdl:import 要素が無視されます。

WSDL 定義の再帰的なインポートの例を次の図に示します。

図 26-4　WSDL 定義の再帰的なインポートの例



26.3 wsdl:import 要素の書式

26.3.1 書式と説明

wsdl:import 要素の書式例を次に示します。

```
<wsdl:import namespace="インポート対象のWSDL定義の名前空間名"
location="インポート対象のWSDL定義のロケーション"/>
```

wsdl:import 要素の属性について，それぞれ説明します。

(1) namespace 属性（wsdl:import 要素）

インポート対象の WSDL 定義の名前空間名を指定します。

インポート元の WSDL 定義に記述する wsdl:import 要素の namespace 属性には，インポート対象の WSDL 定義の名前空間名（wsdl:definitions 要素の targetNamespace 属性）と同じ名前空間名を指定してください。

インポート元の WSDL 定義の名前空間と，インポート対象の WSDL 定義の名前空間の関係を次の表に示します。

表 26-1 WSDL 定義の名前空間の関係（インポート元／インポート対象）

項番	インポート元の WSDL 定義	インポート対象の WSDL 定義	条件	実行時の動作
1	wsdl:import 要素の namespace 属性	wsdl:definitions 要素の targetNamespace 属性	一致	正常終了します。
2			不一致	標準エラー出力とログに警告メッセージが出力され，処理が続行されます（KDJW51191-W）。 インポート対象の WSDL 定義およびインポート元の WSDL 定義の要素は，それぞれの名前空間に属します。
3	wsdl:definitions 要素の targetNamespace 属性		一致	標準エラー出力とログに警告メッセージが出力され，処理が続行されます（KDJW51192-W）。インポート対象の WSDL 定義およびインポート元の WSDL 定義の要素は，同じ名前空間に属します。
4			不一致	正常終了します。

(2) location 属性（wsdl:import 要素）

インポート対象の WSDL 定義のロケーションを指定します。

location 属性に指定する文字列の条件を次に示します。

- URL の形式で相対パス，リモート，またはローカルにある WSDL ファイルを指定できます。
- WAR ファイルに含まれる WSDL ファイルは，メタデータとして発行されます（WSDL ファイルを新たに生成する場合は除きます）。ただし，メタデータを発行するときに注意事項があります。メタデータを発行するときの注意事項については，「10.6.6　WSDL 定義または XML Schema をインポート／インクルードしている場合の注意」を参照してください。
- RFC 2396 で規定される文字および xsd:anyURI を満たす文字を使用してください。また，RFC 2732（IPv6 アドレス）の規則に従った文字列も使用できます。
- 大文字と小文字は区別されません。
- 指定できる文字列の最大長の制限はありません。ただし，OS の制限を超えた場合は，エラーとなります。

location 属性の記述例を次の表に示します。

表 26-2　location 属性の指定例（WSDL インポート機能）

項番	指定内容	指定例
1	ローカルにある WSDL ファイルを相対パスで指定※1	./wsdl/input.wsdl
2	ローカルにある WSDL ファイルを URL（file://～）で表現する絶対パスで指定※1，※2	file:///C:/tmp/wsdl/input.wsdl
3	リモートにある WSDL ファイルを URL（http:～）で指定※3	http://example.com:8080/fromjava/test?wsdl

注※1

WSDL ファイルを相対パスまたは絶対パスで指定する場合は，正しいパスを指定してください。パスを間違えて WSDL ファイルが見つからない場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51197-E または KDJW51198-E）。

注※2

"C:/～"のようにドライブ指定で始まる絶対パスの文字列は指定できません。指定した場合，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51199-E）。

注※3

リモートの WSDL ファイルを URL で指定する場合は，正しい URL を指定してください。パスを間違えて，WSDL ファイルが見つからない場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW51197-E または KDJW51198-E）。

27 カタログ機能

この章では，カタログ機能について説明します。

27.1　カタログ機能とは

カタログ機能とは，XML Catalogs 1.1 仕様に基づき，マッピング（対応付け）を実行する機能です。マッピング（対応付け）には次の 2 通りがあります。

- xsd:import 要素の namespace 属性で指定する XML スキーマの名前空間 URI と，XML スキーマのロケーションを示す URI 参照とのマッピング
- WSDL または XML スキーマのロケーションを示す URI 参照と，別の WSDL または XML スキーマのロケーションを示す URI 参照とのマッピング

カタログ機能によるマッピングを実行するには，マッピング情報を定義したカタログファイルを配置する必要があります。カタログ機能は Web サービスクライアントの開発時および実行時に利用できます。ただし，メタデータ発行時におけるカタログ機能の利用はサポートしていません。

カタログファイルについては，「27.6　カタログファイル」を参照してください。

27.2　カタログ機能の利用（Web サービスクライアントの開発時）

スタブベースの Web サービスクライアント開発時にカタログ機能を利用するには，cjwsimport コマンドの-catalog オプションでカタログファイルを指定してください。-catalog オプションについては，「14.1.1(2)　オプション一覧」を参照してください。

cjwsimport コマンドは，引数に指定した WSDL およびその WSDL が参照している XML スキーマの名前空間やロケーションを，カタログファイルのマッピング情報に従って別の WSDL または XML スキーマのロケーションにマッピングします。cjwsimport コマンドは，マッピングされたロケーションにある WSDL および XML スキーマを読み込んで Java コードを生成します。

27.2.1　マッピング対象

cjwsimport コマンドでの，カタログ機能のマッピング対象を表 27-1 に，cjwsimport コマンドの引数に指定した WSDL の記述におけるマッピング対象を表 27-2 に示します。

表 27–1　cjwsimport コマンドでのマッピング対象

項番	マッピング対象	サポート
1	cjwsimport コマンドの引数に指定した WSDL のロケーション	×

（凡例）

×：サポートしていません。

表 27–2　cjwsimport の引数に指定した WSDL の記述におけるマッピング対象

項番	要素	属性	サポート
1	wsdl:import	namespace	×
2		location	○
3	xsd:import	namespace	○※
4		schemaLocation	○
5	xsd:include	schemaLocation	○

（凡例）

○：サポートしています。

×：サポートしていません。

注※

マッピングの対象になるのは，xsd:import 要素に namespace 属性だけ存在する場合です。xsd:import 要素に，namespace 属性と schemaLocation 属性の両方が含まれるときは，namespace 属性はマッピングの対象になりません。

27.2.2　注意事項

Web サービスクライアントの開発時のカタログ機能は，次に示すロケーションや要素の値を変更しません。

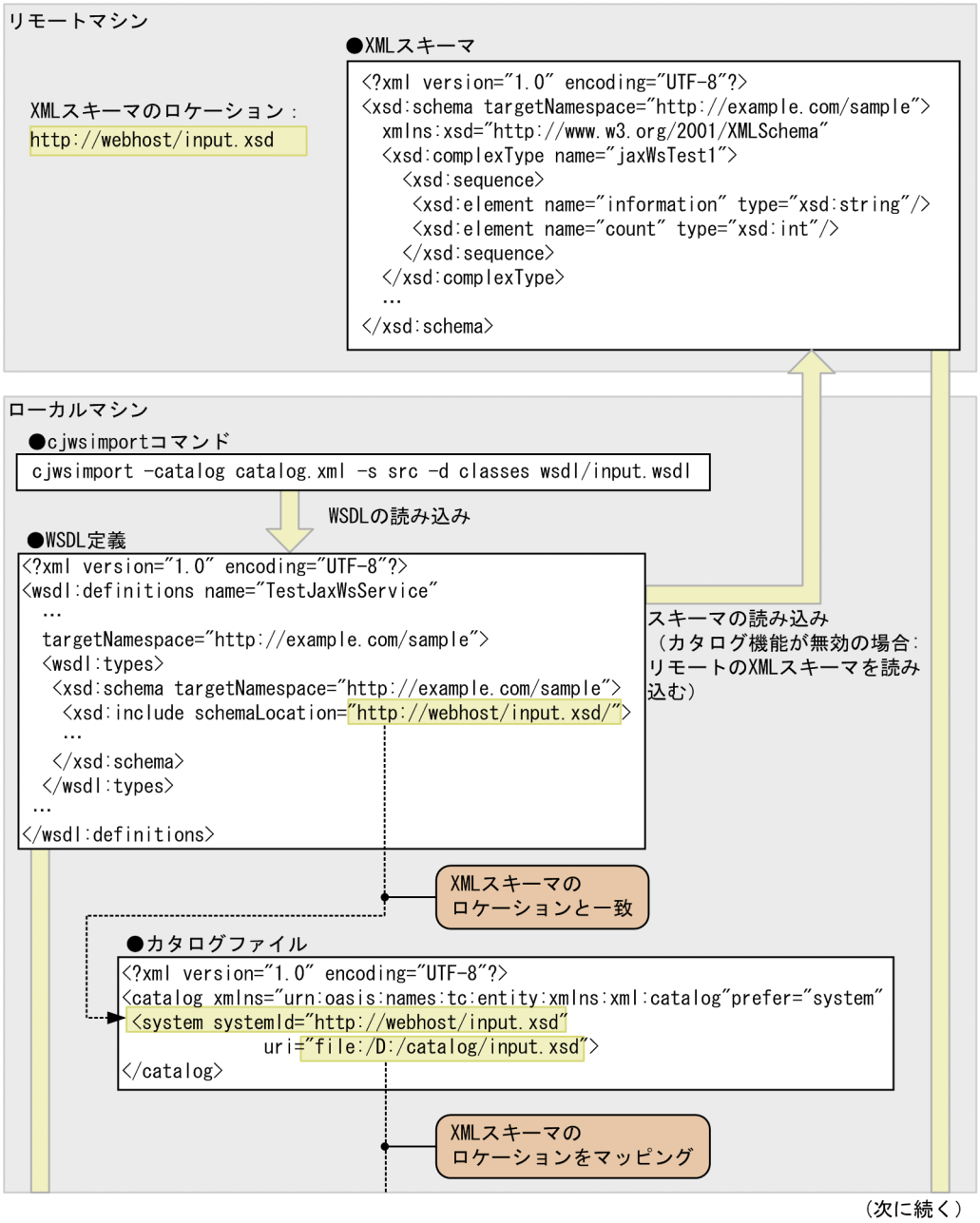
- cjwsimport コマンドが生成するサービスクラスのデフォルトの WSDL ロケーション（WSDL ロケーションを引数に持たないコンストラクタが使用する WSDL ロケーション）
- サービスクラスに付与する javax.xml.ws.WebServiceClient アノテーションにある wsdlLocation 要素

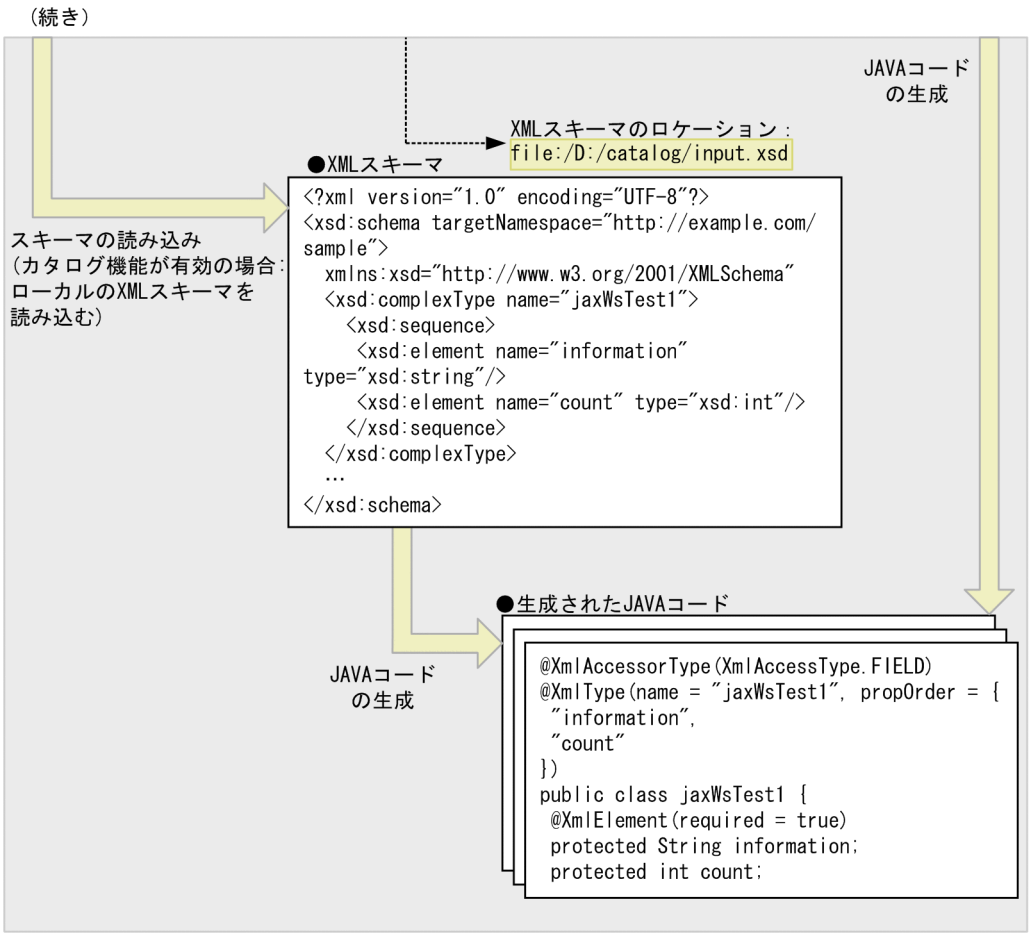
これらのロケーションや要素の値を変更するには，cjwsimport コマンドの-wsdllocation オプションを利用してください。詳細については，「14.1.1　cjwsimport コマンド」を参照してください。

27.2.3　マッピング例

Web サービスクライアントの開発時に，カタログ機能を利用して WSDL が参照する XML スキーマのロケーションをマッピングする例を次の図に示します。

図 27-1　WSDL が参照する XML スキーマのロケーションのマッピング例





この図の例では，WSDL にある xsd:include 要素の schemaLocation 属性に，リモートの XML スキーマのロケーションを指定しています。また，次のようにカタログ機能が無効な場合と有効な場合の流れを示しています。

カタログ機能が無効な場合

cjwsimport コマンドはリモートにある XML スキーマを読み込んで，Java コードを生成します。

カタログ機能が有効な場合

カタログファイルのマッピング情報に従って，xsd:include 要素の schemaLocation 属性に指定したリモートの XML スキーマのロケーションを，ローカルにある XML スキーマのロケーションにマッピングします。cjwsimport コマンドはローカルにある XML スキーマを読み込んで，Java コードを生成します。

27.3　カタログ機能の利用（Web サービスクライアントの実行時）

Web サービスクライアントの実行時では，カタログファイルを配置するだけでカタログ機能を利用できます。WSDL を変更したり，cjwsimport コマンドで-catalog オプションを指定して，開発済みの Web サービスクライアントを作成し直したりする必要はありません。カタログファイルの配置については，「27.6.2　カタログファイルの配置」を参照してください。

WSDL からサービスクラスにマッピングされた WSDL ロケーション，およびサービスクラスのコンストラクタの引数に指定した WSDL ロケーションをカタログファイルのマッピング情報に従って別の WSDL ロケーションにマッピングし，サービスクラスを生成します。

27.3.1　マッピング対象

Web サービスクライアントの実行時におけるカタログ機能のマッピング対象について表 27-3 に，Web サービスクライアントの実行時に利用する WSDL の記述におけるマッピング対象を表 27-4 に示します。

表 27-3　Web サービスクライアントの実行時におけるカタログ機能のマッピング対象

項番	マッピング対象	サポート
1	サービスクラスのデフォルトの WSDL ロケーション※1	○
2	サービスクラスのコンストラクタの引数に指定した WSDL ロケーション※1	○
3	javax.xml.ws.Service クラスの API の引数に指定した WSDL ロケーション※2	○
4	javax.xml.ws.WebServiceRef アノテーションの wsdlLocation 要素に指定した WSDL ロケーション※3	○

（凡例）

○：サポートしています。

注※1

サービスクラスの WSDL ロケーションについては，「15.1.9　サービスおよびポートからサービスクラスへのマッピング」の「表 15-17　サービスクラスが持つメソッド」を参照してください。

注※2

javax.xml.ws.Service クラス WSDL ロケーションについては，「19.2.2(4)　javax.xml.ws.Service クラス」を参照してください。

注※3

javax.xml.ws.WebServiceRef アノテーションについては，「19.3　アノテーションのサポート範囲」を参照してください。

表 27-4　Web サービスクライアントの実行時に利用する WSDL の記述におけるマッピング対象

項番	要素	属性	サポート
1	wsdl:import	namespace	×
2		location	○

（凡例）

○：サポートしています。

×：サポートしていません。

27.3.2 注意事項

Web サービスクライアントの実行時のカタログ機能は，サービスクラスの生成時に XML スキーマを必要としないので，次のマッピングを実行しません。

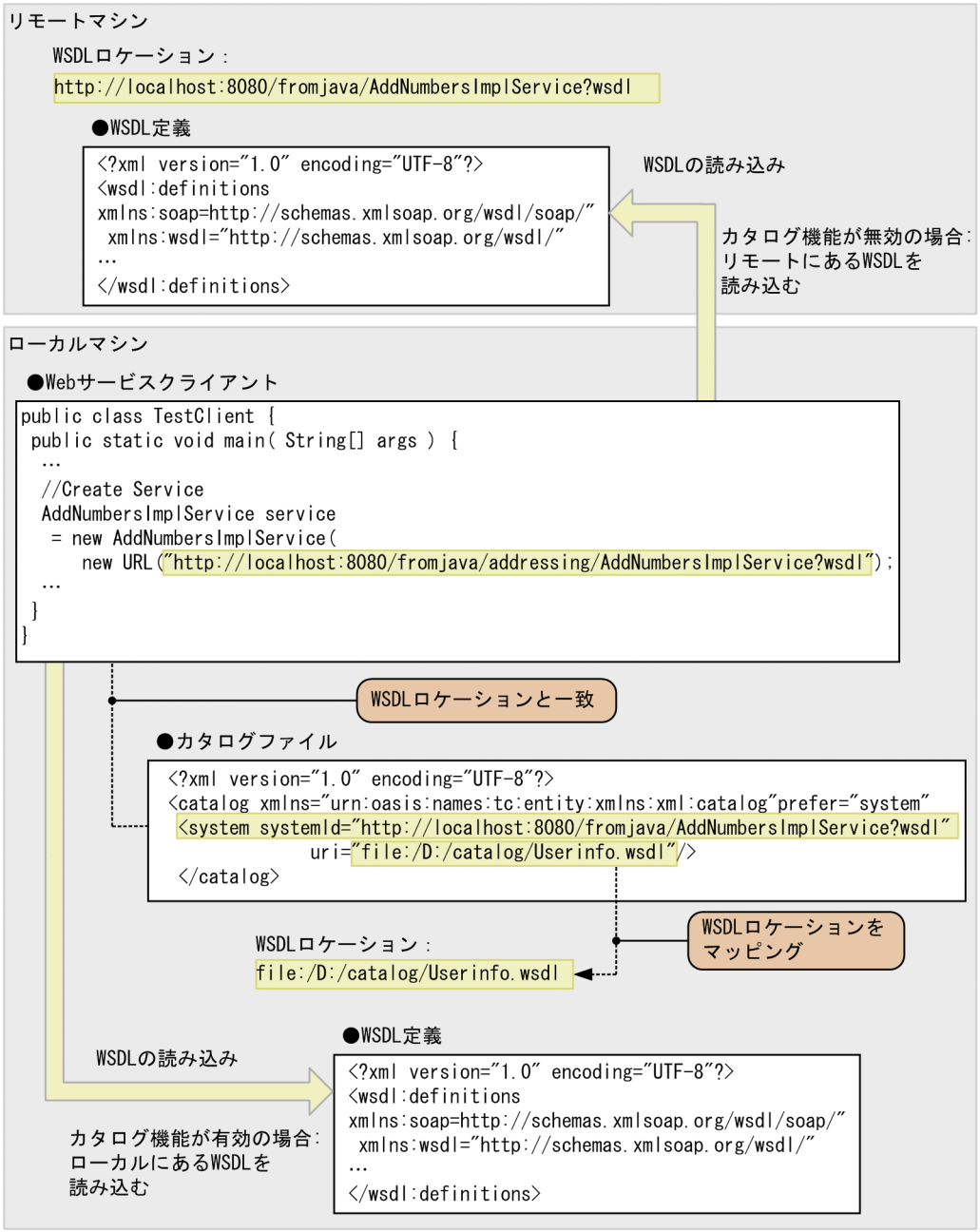
- XML スキーマの名前空間 URI のマッピング
- スキーマロケーションに関するマッピング

このため，カタログファイルには WSDL ロケーションに関するマッピング情報だけを記述してください。

27.3.3 マッピング例

Web サービスクライアントでサービスクラスを生成するときに，カタログ機能を利用して WSDL ロケーションをマッピングする例を次の図に示します。

図 27-2　WSDL ロケーションのマッピング例



この図の例では，サービスクラスのコンストラクタの引数に，リモートの WSDL ロケーションを指定しています。また，次のようにカタログ機能が無効な場合と有効な場合の流れを示しています。

カタログ機能が無効な場合

リモートにある WSDL を読み込み，サービスクラスを生成します。

カタログ機能が有効な場合

カタログファイルのマッピング情報に従って，引数に指定したリモートの WSDL ロケーションがローカルにある WSDL ロケーションにマッピングされ，ローカルにある WSDL を読み込み，サービスクラスを生成します。

27.4　カタログ機能の性能

カタログ機能利用時のオーバーヘッドの増加

カタログ機能は，Web サービスクライアントの実行時にサービスクラスのコンストラクタや javax.xml.ws.Service クラスの API を呼び出すときに動作します。このため，カタログ機能を利用しない場合に比べると，サービスクラスの生成や javax.xml.ws.Service クラスの API の実行に掛かる時間が，カタログファイルの読み込みに伴うオーバーヘッドの分，増加します。

カタログ機能利用時のオーバーヘッドの軽減

Web サービスクライアントの実行時のカタログ機能のオーバーヘッドを軽減する場合は，次の方法をお勧めします。

- 生成したサービスを再利用する

  詳細については，「3.6.1(3)　サービスクラスの生成およびポートの取得」を参照してください。

- javax.xml.ws.WebServiceRef アノテーションをフィールドなどにアノテートし，サービスクラスをインジェクションする

  詳細については，「19.3　アノテーションのサポート範囲」を参照してください。

27.5　カタログ機能利用時の注意事項

- カタログ機能とアドレッシング機能は併用できません。併用した場合の動作は保証されません。
- Web サービスクライアントの実行時に，マッピング対象の WSDL ロケーションを示す URI 参照を，リモートにある WSDL ロケーションを示す URI 参照にカタログ機能を利用してマッピングした場合，次に示すプロパティは適用されません。
  - com.cosminexus.jaxws.connect.timeout
  - com.cosminexus.jaxws.request.timeout
  - com.cosminexus.xml.ws.client.http.HostnameVerificationProperty
  - javax.xml.ws.security.auth.username
  - javax.xml.ws.security.auth.password
- Web サービス開発時にカタログ機能を利用する場合，カタログファイルは半角英数字（0～9，A～Z，a～z），空白，ピリオド（.），アンダースコア（_），コロン（:），スラッシュ（/），および¥を使用したディレクトリに配置してください。それ以外の文字が含まれるディレクトリに配置した場合，動作は保証されません。

27.6　カタログファイル

カタログファイルは，カタログ機能を利用する場合に配置するファイルです。このファイルにマッピング（対応付け）情報を定義します。カタログファイルのサポート範囲については，「21.　XML Catalogs 1.1 のサポート範囲」を参照してください。ここでは，カタログファイルの構文および配置について説明します。

27.6.1　カタログファイルの構文

カタログファイルの構文は，XML Catalogs 1.1 のスキーマ（W3C XML Schema for the XML Catalog），および Application Server の JAX-WS 機能でサポートされる範囲で記述します。

27.6.2　カタログファイルの配置

カタログファイルは OS のファイルシステムの制限と Java EE 6 仕様に従い，次の場所に配置します。

(1)　Web サービスクライアントの開発時の場合

cjwsimport コマンドの-catalog オプションの引数にカタログファイルのパスを指定します。カタログファイルのファイル名は任意です。詳細については，「14.1.1(2)　オプション一覧」の「-catalog オプション指定時の注意事項」を参照してください。カタログファイルの読み込みに失敗した場合は，標準エラー出力に警告メッセージ（KDJW51221-W）が出力され，カタログ機能が無効な状態で処理が続行されます。

(2)　Web サービスクライアントの実行時の場合

カタログファイルを，クラスパス上の META-INF ディレクトリの直下に"jax-ws-catalog.xml"という名称で配置します。カタログファイルの読み込みに失敗した場合は，ログファイルにメッセージ（KDJW30023-W）を出力し，カタログ機能が無効な状態で処理が続行されます。なお，複数のカタログファイルを配置した場合の動作は保証されません。例えば"jax-ws-catalog.xml"を格納した複数の JAR ファイルをクラスパスに追加したときなどの動作は保証されません。

27.6.3　カタログファイルの記述例

カタログ機能を利用して，リモートの WSDL ロケーションをローカルにある WSDL ロケーションにマッピングする場合の，カタログファイルの記述例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog" prefer="system">
    <system systemId="http://localhost:8080/fromjava/AddNumbersImplService?wsdl"
                uri="./wsdl/UserInfo.wsdl"/>
</catalog>
```

28 添付ファイル機能（wsi:swaRef 形式）

添付ファイル機能を使用すると，テキストデータだけでなく，画像データや音声データなどを Web サービスで扱えます。

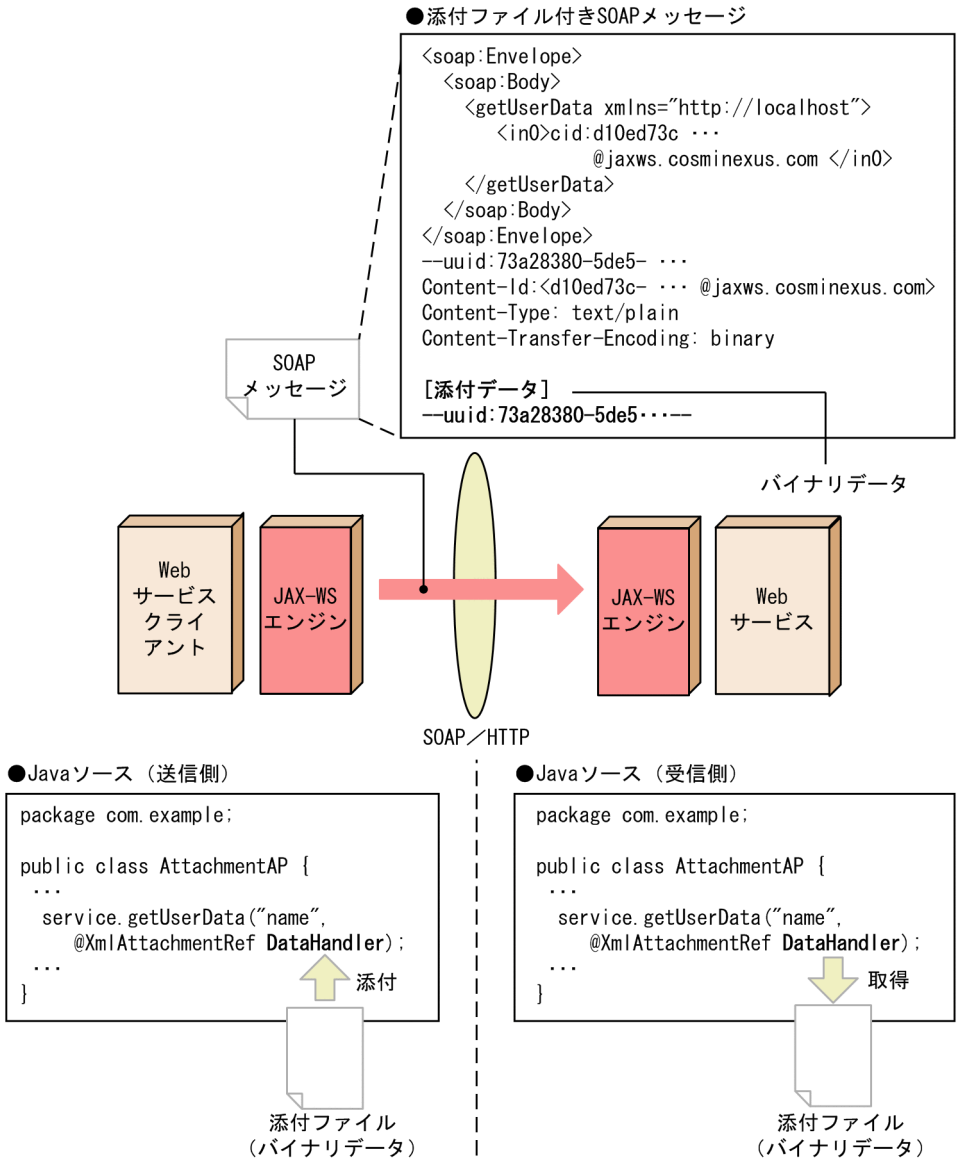
この章では，添付ファイル機能の概要，および添付ファイル機能を使用した場合のマッピング規則について説明します。

28.1　添付ファイル機能とは（wsi:swaRef 形式）

添付ファイル機能とは，SOAP メッセージにテキストや画像（バイナリデータ）などのファイルを添付した添付ファイル付き SOAP メッセージを送受信する機能です。

次に示す例を基に，添付ファイル機能の概要について説明します。

図 28-1　添付ファイル機能を使用したバイナリデータの送受信



SOAP メッセージにファイルを添付する場合，Web サービスを呼び出す Java メソッドの引数に，javax.activation.DataHandler クラスを使用します。ファイルを添付した Java ソースは，送信側の JAX-WS エンジンによって SOAP メッセージにマーシャルされ，受信側の JAX-WS エンジンによって Java ソースにアンマーシャルされます。受信側の JAX-WS エンジンは，アンマーシャルされた Java ソースのメソッドの引数から，添付ファイルを取得します。

なお，この機能は，WS-I Attachments Profile - Version 1.0 仕様に従っているため，wsi:swaRef 形式の WSDL を使用します。また，SOAP Messages with Attachments プロトコルを使用しているため，添付ファイル付き SOAP メッセージは MIME Multipart/Related 構造でエンコードされます。

SAAJ 1.3 仕様の API を使用して添付ファイル付き SOAP メッセージを送受信する場合は，「22.1　SAAJ 1.3 仕様のサポート範囲」を参照してください。

28.2　添付ファイルの Java インタフェース（wsi:swaRef 形式）

ここでは，Java インタフェース内で添付ファイルを指定する Java 型について説明します。

28.2.1　添付ファイルに使用できる Java 型

Java インタフェースでの添付ファイルの Java 型の使用可否を次の表に示します。

添付ファイルに使用できる Java 型には，javax.xml.bind.annotation.XmlAttachmentRef アノテーションを付加する必要があります。添付ファイルに使用できない型にこのアノテーションを付加した場合，動作は保証されません。

表 28–1　添付ファイルとしての Java 型の使用可否

項番	Java 型	使用可否
1	javax.activation.DataHandler	○
2	javax.xml.ws.Holder<DataHandler>	○
3	javax.activation.DataHandler の配列型	△
4	javax.xml.ws.Holder<DataHandler>の配列型	×
5	javax.activation.DataHandler を継承したデータ型	×

（凡例）

○：添付ファイルとして使用できます。

△：添付ファイルとして 1 次元配列だけ使用できます。多次元配列を使用した場合，動作は保証されません。

×：添付ファイルとして使用できません。

28.2.2　添付ファイルを指定できる個所

添付ファイルは，Java インタフェースのメソッド引数，メソッド戻り値，およびユーザ定義型のフィールドで指定できます。ユーザ定義例外では指定できません。

Java インタフェース内での添付ファイルの指定個所と，Java 型の指定可否を次の表に示します。

表 28–2　添付ファイルの Java 型の指定個所と指定可否

項番	Java インタフェースでの指定個所	添付ファイルの Java 型	指定可否
1	メソッド引数	javax.activation.DataHandler	○
2		javax.xml.ws.Holder<DataHandler>	○
3		javax.activation.DataHandler の配列型	△
4	メソッド戻り値	javax.activation.DataHandler	○

項番	Java インタフェースでの指定個所	添付ファイルの Java 型	指定可否
5		javax.xml.ws.Holder<DataHandler>	×
6		javax.activation.DataHandler の配列型	△
7	ユーザ定義型のフィールド	javax.activation.DataHandler	○
8		javax.xml.ws.Holder<DataHandler>	×
9		javax.activation.DataHandler の配列型	△
10	ユーザ定義例外のフィールド	javax.activation.DataHandler	×
11		javax.xml.ws.Holder<DataHandler>	×
12		javax.activation.DataHandler の配列型	×

（凡例）

○：指定できます。

△：1 次元配列を使用する場合だけ指定できます。多次元配列で指定した場合，動作は保証されません。

×：指定できません。

28.2.3　javax.activation.DataHandler 型に指定できる添付ファイル

javax.activation.DataHandler 型は，JavaBeans Activation Framework（JAF）の型であるため，任意の MIME タイプの添付ファイルを指定できます。添付ファイルの拡張子と，デフォルトで設定される MIME タイプのマッピングについては，「28.4.2(3)　添付ファイルの拡張子と MIME タイプのマッピング」を参照してください。

28.3　添付ファイルの WSDL（wsi:swaRef 形式）

添付ファイル付き SOAP メッセージを使用する場合，WSDL 上に添付ファイルを定義できます。ここでは，添付ファイルを使用する場合の WSDL の記述と，WSDL と Java 型のマッピングについて説明します。

28.3.1　添付ファイル使用時の WSDL の記述（wsi:swaRef 形式）

WSDL で添付ファイルを扱う場合，wsi:swaRef 形式で記述します。wsi:swaRef 形式は，WSDL で添付ファイルを扱う形式として WS-I Attachments Profile - Version 1.0 で規定されています。

WS-I Attachments Profile - Version 1.0 で規定された形式に基づいて，wsi:swaRef 形式で記述した WSDL の例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://localhost"
  xmlns:wsi="http://ws-i.org/profiles/basic/1.1/xsd" ...>
  <wsdl:types>
    <schema elementFormDefault="qualified" targetNamespace="http://localhost"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <import namespace="http://ws-i.org/profiles/basic/1.1/xsd"/>
      <element name="getUserData">
        <complexType>
          <sequence>
            <element name="in0" type="xsd:string"/>
            <element name="in1" type="wsi:swaRef"/>
          </sequence>
        </complexType>
      </element>
       ...
    </schema>
  </wsdl:types>
  <wsdl:message name="getUserDataRequest">
    <wsdl:part element="intf:getUserData" name="parameters"/>
  </wsdl:message>
  ...
  <wsdl:portType name="UserInfo">
    <wsdl:operation name="getUserData">
      <wsdl:input message="intf:getUserDataRequest" name="getUserDataRequest"/>
      ...
    </wsdl:operation>
  </wsdl:portType>
  ...
</wsdl:definitions>
```

この例では，getUserData メソッドの引数（スキーマ定義の element 要素の型）に，添付ファイルを表す wsi:swaRef 型を指定しています。

WSDL で添付ファイルを扱う場合の注意事項

- swaRef 型は，スキーマの要素の型（element 要素の type 属性）に指定できます。属性の型（attribute 要素の type 属性）に指定した場合，動作は保証されません。
- swaRef 型を指定している XML Schema では，WSDL の例で示すように名前空間 http://ws-i.org/profiles/basic/1.1/xsd をインポートする必要があります。インポートしていない場合，XML Processor のエラーが出力されます。

  名前空間 http://ws-i.org/profiles/basic/1.1/xsd をインポートする xsd:import 要素の schemaLocation 属性の形式と，使用可否を次の表に示します。

  表 28–3　xsd:import 要素の schemaLocation 属性の形式と使用可否（swaRef 型の使用時）

項番	schemaLocation 属性	使用可否
1	指定しない	使用できます。※1
2	http://wsi.org/profiles/basic/1.1/swaref.xsd	使用できます。※1
3	項番 1 および項番 2 以外	使用できません。※2

注※1

cjwsimport コマンドは，Application Server の JAX-WS 機能の内部で保持する名前空間 http://ws-i.org/profiles/basic/1.1/xsd のスキーマを参照します。

注※2

cjwsimport コマンドは，schemaLocation 属性で指定した場所にあるスキーマを参照します。ただし，Application Server の JAX-WS 機能で内容を保証できないスキーマが参照されるため，サポート対象外となります。

- ユーザ定義例外で添付ファイルは使用できないため，wsdl:fault 要素から参照するスキーマの要素の型（element 要素の type 属性）に swaRef 型は指定できません。wsdl:fault 要素から参照するスキーマの要素の型に swaRef 型を指定した場合，動作は保証されません。

28.3.2　添付ファイルの Java 型と WSDL のマッピング（wsi:swaRef 形式）

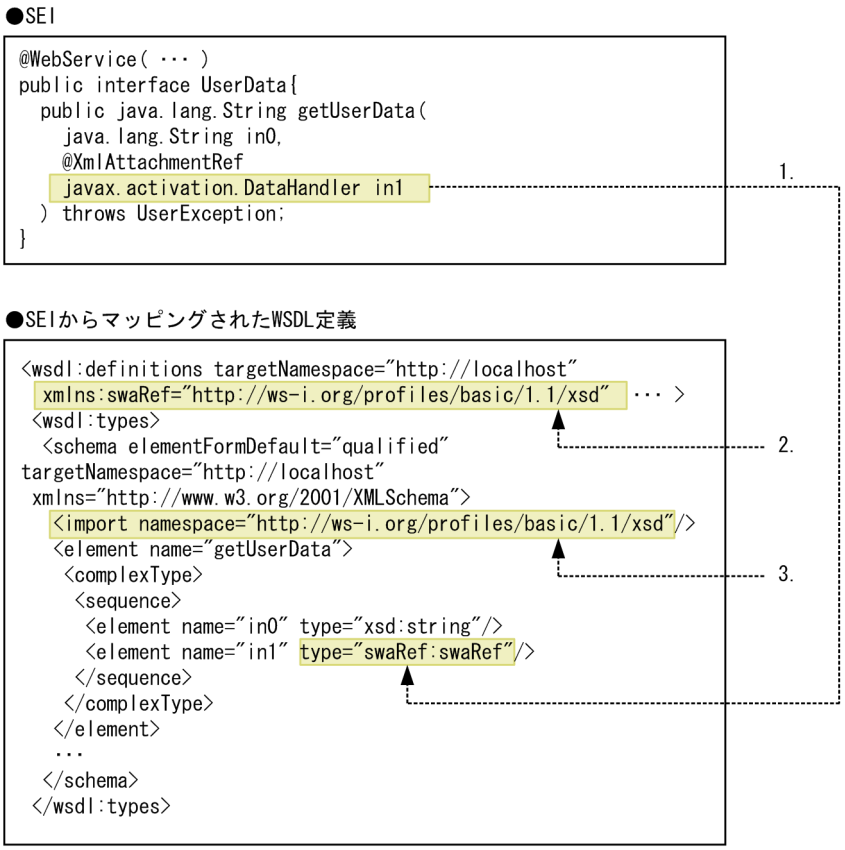
添付ファイルの Java 型から WSDL へのマッピング規則を次に示します。

添付ファイルの Java 型から WSDL へのマッピング規則

1. 添付ファイルの Java 型は，swaRef 型にマッピングされます。
2. wsdl:definitions 要素で，swaRef 型の名前空間が宣言されます。
3. swaRef 型を指定している XML Schema で，swaRef 型の名前空間がインポートされます。

添付ファイルの Java 型と WSDL のマッピング例を次の図に示します。上記の「添付ファイルの Java 型から WSDL へのマッピング規則」に示した番号と，次の図に示す番号は対応しています。

図 28–2　添付ファイルの Java 型から WSDL へのマッピング例



28.3.3　WSDL から添付ファイルの Java 型へのマッピング（wsi:swaRef 形式）

WSDL から添付ファイルの Java 型へのマッピング規則を次に示します。

WSDL から添付ファイルの Java 型へのマッピング規則

1. WSDL の swaRef 型は，添付ファイルの Java 型へマッピングされます。

WSDL と添付ファイルの Java 型のマッピング例を次の図に示します。上記の「WSDL から添付ファイルの Java 型へのマッピング規則」に示した番号と，次の図に示す番号は対応しています。

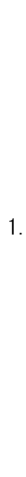
図 28-3　WSDL から添付ファイルの Java 型へのマッピング例

●WSDL定義

```
<wsdl:definitions targetNamespace="http://localhost"
  xmlns:swaRef="http://ws-i.org/profiles/basic/1.1/xsd" ···>
 <wsdl:types>
  <schema elementFormDefault="qualified"
    targetNamespace="http://localhost"
    xmlns="http://www.w3.org/2001/XMLSchema">
   <import namespace="http://ws-i.org/profiles/basic/1.1/xsd"/>
   <element name="getUserData">
    <complexType>
     <sequence>
      <element name="in0" type="xsd:string"/>
      <element name="in1" type="swaRef:swaRef"/>
     </sequence>
    </complexType>
   </element>
   ···
  </schema>
 </wsdl:types>
```

●WSDLからマッピングされたSEI

```
@WebService( ··· )
public interface UserData{
  @WebMethod
  public String getUserData(
    @WebParam(name = "in0", ··· )
    String in0,
    @WebParam(name = "in1", ··· )
    DataHandler in1
  ) throws UserException;
}
```

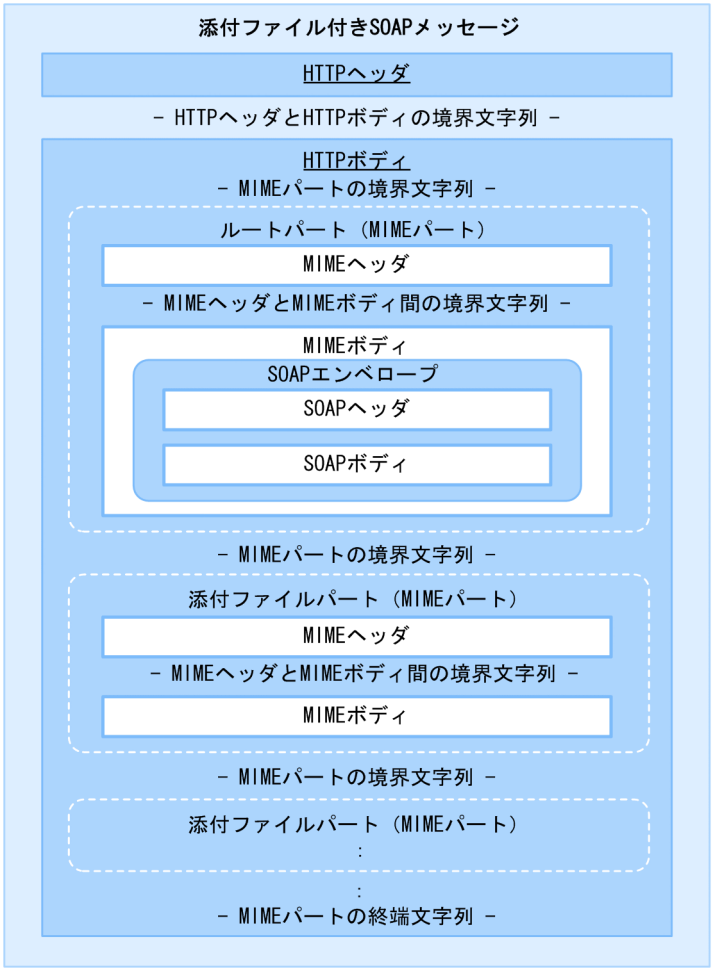


28.4　添付ファイル付き SOAP メッセージ（wsi:swaRef 形式）

添付ファイル付き SOAP メッセージは，SOAP Messages with Attachments プロトコルを使用し，MIME Multipart/Related 構造でエンコードされます。

添付ファイル付き SOAP メッセージの構造を次の図に示します。

図 28–4　添付ファイル付き SOAP メッセージの構造



添付ファイル付き SOAP メッセージの各部の説明を次の表に示します。

表 28–4　添付ファイル付き SOAP メッセージの各部の説明

各部の名称	説明
HTTP ヘッダ	HTTP プロトコルに依存するヘッダ情報です。
HTTP ヘッダと HTTP ボディの境界文字列	HTTP ヘッダと HTTP ボディの境界を示す文字列です。
HTTP ボディ	送信するメッセージを記述します。 ルートパートおよび添付ファイルパートから構成されます。
ト　MIME パートの境界文字列	各 MIME パートの境界を示す文字列です。

各部の名称				説明
├	ルートパート |			メッセージ本体を記述するパートです。 MIME ヘッダと MIME ボディから構成され，必ず 1 個定義します。
|	├	MIME ヘッダ		ルートパートのヘッダ情報です。
|	├	MIME ヘッダと MIME ボディ間の境界文字列		ルートパートの MIME ヘッダと MIME ボディ間の境界を示す文字列です。
|	└	MIME ボディ		メッセージ本体を記述します。
|		└	SOAP エンベロープ	SOAP エンベロープを記述します。
|			├ SOAP ヘッダ	SOAP メッセージのヘッダ情報を記述します。
|			└ SOAP ボディ	SOAP メッセージの本文（XML）を記述します。
├	MIME パートの境界文字列			各 MIME パートの境界を示す文字列です。
├	添付ファイルパート |			添付ファイルの内容を記述するパートです。 MIME ヘッダと MIME ボディから構成され，0 個以上定義します。
|	├	MIME ヘッダ		添付ファイルパートのヘッダ情報です。
|	├	MIME ヘッダと MIME ボディ間の境界文字列		添付ファイルパートの MIME ヘッダと MIME ボディ間の境界を示す文字列です。
|	└	MIME ボディ		添付ファイルの内容（バイナリデータ）を記述します。
└	MIME パートの終端文字列			MIME パートの終端を表す文字列です。

28.4.1 添付ファイルから SOAP メッセージへのマッピング（wsi:swaRef 形式）

添付ファイルから SOAP メッセージへマッピングするときの設定値について説明します。マッピング規則については，「28.4.2 添付ファイルから SOAP メッセージへのマッピングの注意事項（wsi:swaRef 形式）」と合わせて確認してください。

(1) HTTP ヘッダ

添付ファイル使用時に，HTTP ヘッダのフィールドおよびパラメタに設定される値を次の表に示します。

表 28-5　HTTP ヘッダのフィールドおよびパラメタの設定値

項番	フィールド名	パラメタ名	設定値
1	Content-Type	−	"multipart/related"が設定されます。
2		type	SOAP 1.1 仕様 "text/xml"が設定されます。 SOAP 1.2 仕様 "application/soap+xml"が設定されます。
3		boundary	MIME パートの境界文字列が設定されます。

（凡例）

−：Content-Type に直接設定されることを示します。

1 個以上の非ルート MIME パートを列挙している場合（添付ファイル付き SOAP メッセージを送受信する場合）は，Content-Type に multipart/related が設定されます。MIME パートがない場合，SOAP 1.1 仕様のときは"text/xml"が，SOAP 1.2 仕様のときは"application/soap+xml"が設定されます。

(2)　HTTP ボディ

HTTP ボディは，ルートパート，添付ファイルパート，および各パートの境界文字列で構成されます。添付ファイル使用時に，各パートおよび境界文字列に生成される内容，および設定される値を示します。

(a)　ルートパートの MIME ヘッダ

添付ファイル使用時に，ルートパートの Content-Type フィールドに設定される値を次の表に示します。

表 28-6　ルートパートのフィールドの設定値

項番	フィールド名	設定値
1	Content-Type	SOAP 1.1 仕様 "text/xml"が設定されます。 SOAP 1.2 仕様 "application/soap+xml"が設定されます。
2	Content-Id	"グローバルに一意な値"＋"@"＋"jaxws.cosminexus.com"が設定されます。

(b)　ルートパートの MIME ボディ

添付ファイルを使用する場合，ルートパートの MIME ボディには，SOAP エンベロープがそのまま格納されます。SOAP エンベロープ内の SOAP ボディから添付ファイルを参照する方法として，CID URL スキームが使用されます。

CID URL スキームの形式を次に示します。

```
"cid:"＋<添付ファイルパートのContent-Id>
```

(c)　ルートパートの MIME ヘッダと MIME ボディの境界文字列

ルートパートの MIME ヘッダと MIME ボディ間には，境界文字列として"CRLF"が設定されます。

(d)　添付ファイルパートの MIME ヘッダ

添付ファイル使用時に，添付ファイルパートの MIME ヘッダに設定される値を次の表に示します。

表 28–7　添付ファイルパートの MIME ヘッダの設定値

項番	フィールド名	パラメタ名	設定値
1	Content-Type	－	添付ファイルの種類に応じた MIME タイプ※1 が設定されます。
2		charset	DataHandler オブジェクト生成時に指定した文字コード※2 が設定されます。
3	Content-Transfer-Encoding	なし	"binary"が設定されます。
4	Content-Id	なし	"グローバルに一意な値"＋"@"＋"jaxws.cosminexus.com"が設定されます。

（凡例）

－：Content-Type に直接設定されることを示します。

注※1

Content-Type フィールドの設定値は，DataHandler オブジェクトの生成方法によって，次のように異なります。

- DataHandler(DataSource)コンストラクタで生成する場合

  FileDataSource の場合，入力となる添付ファイルの拡張子から，JAF によって決定された MIME タイプが Content-Type フィールド値として設定されます。添付ファイルの拡張子と MIME タイプのマッピングについては，「28.4.2(3)　添付ファイルの拡張子と MIME タイプのマッピング」を参照してください。

- DataHandler(Object, String)コンストラクタで生成する場合

  DataHandler オブジェクト生成時に，コンストラクタの第 2 引数に指定した内容（MIME タイプ）がそのまま Content-Type フィールド値として設定されます。

注※2

例えば，DataHandler オブジェクトの生成を次のように生成した場合，Content-Type の charset には，第 2 引数に指定した charset パラメタ値「Shift_JIS」が設定されます。

```
DataHandler dhandler = new DataHandler (”あいうえお”, ”text/plain; charset=Shift_JIS”);
```

添付ファイルの Java 型である javax.activation.DataHandler 型は，JavaBeans Activation Framework（JAF）の型であるため，任意の MIME タイプの添付ファイルを指定できます。FileDataSource を使用して DataHandler オブジェクトを生成する場合の添付ファイルの拡張子と，デフォルトで設定される MIME タイプの対応については，「28.4.2(3)　添付ファイルの拡張子と MIME タイプのマッピング」を参照してください。

(e) 添付ファイルパートの MIME ボディ

添付ファイルパートの MIME ボディには，添付ファイルの内容を表すバイナリデータが格納されます。

(f) 添付ファイルパートの MIME ヘッダと MIME ボディの境界文字列

添付ファイルパートの MIME ヘッダと MIME ボディの境界文字列は，ルートパートの場合と同様に，境界文字列として"CRLF"が設定されます。

(g) 各 MIME パート間の境界文字列

ルートパートと添付ファイルパート間，および添付ファイルパート間には，境界文字列として次の文字列が設定されます。

```
”CRLF”＋”--”＋”<HTTPヘッダのboundaryパラメタの値>”
```

(h) MIME パートの終端文字列

MIME パートの終端には，終端文字列として次の文字列が設定されます。

```
”CRLF”＋”--”＋”<HTTPヘッダのboundaryパラメタの値>”＋”--”
```

(3) HTTP ヘッダと HTTP ボディの境界文字列

HTTP ヘッダと HTTP ボディ間には，境界文字列として"CRLF"が設定されます。

28.4.2 添付ファイルから SOAP メッセージへのマッピングの注意事項（wsi:swaRef 形式）

添付ファイルから SOAP メッセージをマッピングするときの注意事項について説明します。

(1) MIME パートの記述順序

MIME パートは 1 個のルートパートと 0 個以上の添付ファイルパートで構成されます。

ルートパートは，MIME パートの先頭に記述されます。ルートパートのあとに，添付ファイルパートが記述されます。

添付ファイルパートでは，Java インタフェースで指定された引数や戻り値がマッピングされます。Java インタフェースの指定内容と添付ファイルパートでの記述順序の対応を次の表に示します。

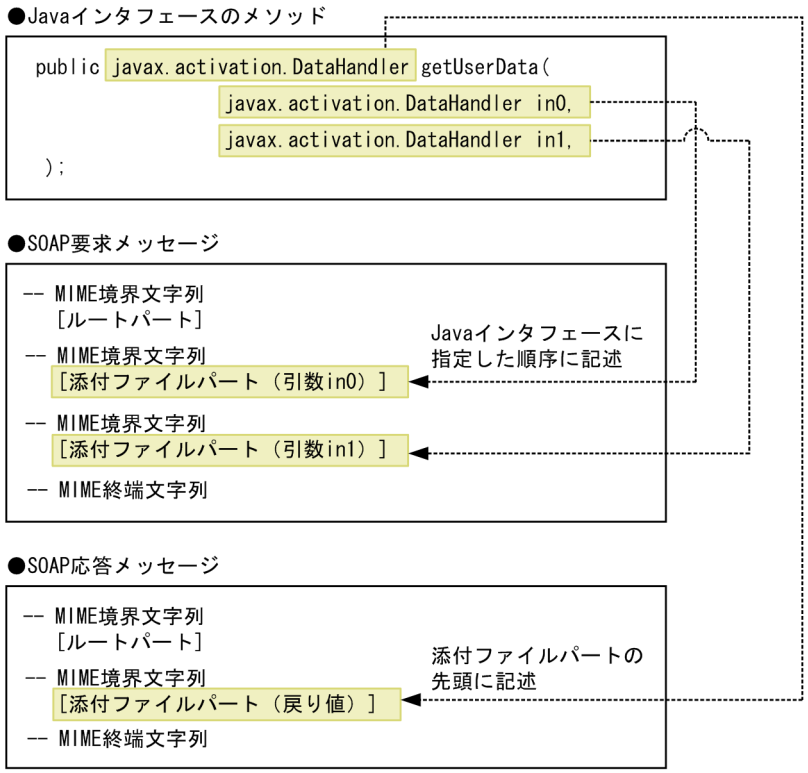
表 28-8 Java インタフェースの指定内容と添付ファイルパートの記述順序

項番	Java インタフェースでの指定個所	添付ファイルパートの記述順序
1	メソッド引数	メソッド引数に指定された順に記述されます。

項番	Java インタフェースでの指定個所	添付ファイルパートの記述順序
2	メソッド戻り値	添付ファイルパートの先頭に記述されます。
3	配列型	配列内の要素順に記述されます。
4	ユーザ定義型	ユーザ定義型内での指定順に記述されます。 ユーザ定義型の中で指定したユーザ定義型で，添付ファイルの Java 型を指定するときは，深さ優先順で添付ファイルパートが記述されます。

Java インタフェースと添付ファイルパートのマッピング例を示します。

図 28-5　Java インタフェースと添付ファイルパートのマッピング例



(2)　ルートパートから添付ファイルへのマッピング

ルートパートの SOAP ボディに記述される CID URL スキームには，対応する添付ファイルパートの Content-Id が記述されます。Content-Id によって，ルートパートから対応する添付ファイルパートが参照されます。

添付ファイル付き SOAP メッセージの一部を示します。背景色付きの太字部分が CID URL スキームと，対応する添付ファイルパートの Content-Id になります。

```
--uuid:73a28380-5de5-45e8-af15-c879e65d62a0
Content-Type: text/xml

<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
```

```
    <getUserData xmlns="http://localhost">
      <in0>cid:d10ed73c-e9b7-418e-8201-ba920cccb214@jaxws.cosminexus.com</in0>
    </getUserData>
  </S:Body>
</S:Envelope>

--uuid:73a28380-5de5-45e8-af15-c879e65d62a0
Content-Id:<d10ed73c-e9b7-418e-8201-ba920cccb214@jaxws.cosminexus.com>
Content-Type: text/plain
Content-Transfer-Encoding: binary

[添付データ]
--uuid:73a28380-5de5-45e8-af15-c879e65d62a0--
```

(3) 添付ファイルの拡張子と MIME タイプのマッピング

添付ファイルの拡張子と，デフォルトで設定される MIME タイプのマッピングを次の表に示します。

表 28-9 添付ファイルの拡張子と MIME タイプの対応表

項番	添付ファイルの拡張子	設定される MIME タイプ
1	html，htm	text/html
2	txt，text	text/plain
3	gif，GIF	image/gif
4	ief	image/ief
5	jpeg，jpg，jpe，JPG	image/jpeg
6	tiff，tif	image/tiff
7	xwd	image/x-xwindowdump
8	ai，eps，ps	application/postscript
9	rtf	application/rtf
10	tex	application/x-tex
11	texinfo，texi	application/x-texinfo
12	t，tr，roff	application/x-troff
13	au	audio/basic
14	midi，mid	audio/midi
15	aifc	audio/x-aifc
16	aif，aiff	audio/x-aiff
17	wav	audio/x-wav
18	mpeg，mpg，mpe	video/mpeg
19	qt，mov	video/quicktime

項番	添付ファイルの拡張子	設定される MIME タイプ
20	avi	video/x-msvideo

この表にない拡張子を指定した場合，MIME タイプは「application/octet-stream」が設定されます。

(4) 添付ファイルのデータサイズと添付ファイルパートへのマッピング

Java インタフェースに指定する添付ファイルのデータサイズが null の場合と，0 バイト以上のデータがある場合では，添付ファイルパートへのマッピング方法が異なります。それぞれの場合のマッピングについて説明します。

(a) 添付ファイルのデータサイズが null の場合

添付ファイルのデータサイズが null の場合，添付ファイルパートにマッピングされません。MIME パートを生成しないで，SOAP エンベロープだけが記述されます。また，SOAP ボディの該当する引数の要素には，空の要素が指定されます。

添付ファイルのデータサイズが null の場合のマッピング例を示します。

- Web サービス呼び出しプログラム

```
...
UserInfoService service = new UserInfoService();
UserInfo impl = service.getUserInfo();

result = impl.getUserData( null );
...
```

- SOAP メッセージ

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <getUserData xmlns="http://localhost">
    </getUserData>
  </S:Body>
</S:Envelope>
```

(b) 添付ファイルのデータサイズが 0 バイト以上の場合

添付ファイルのデータサイズが 0 バイト以上の場合，添付ファイルパートにマッピングされます。この場合，ルートパートと添付ファイルパートから構成されるマルチパートとなります。Java インタフェースで複数の添付ファイルを指定している場合は，添付ファイルパートは複数記述されます。ただし，0 バイトの場合は，添付ファイルパート内の MIME ボディは空となります。

28.4.3 SOAP メッセージから添付ファイルへのマッピング（wsi:swaRef 形式）

JAX-WS エンジンは，受信した添付ファイル付き SOAP メッセージが WS-I Attachments Profile - Version 1.0 仕様に従っている場合だけ，添付ファイル付きの SOAP メッセージから添付ファイルデータへのマッピングは保証されます。それ以外の SOAP メッセージを受信した場合の動作は保証されません。

28.5　添付ファイルの Java インスタンスの生成と取得（wsi:swaRef 形式）

添付ファイルをプログラムで扱うには，JAF の javax.activation パッケージを使用する必要があります。ここでは，javax.activation パッケージを使用して，添付ファイルのインスタンスを生成する方法，およびデータを取得する方法について説明します。

JAF の仕様については，JAF の API 仕様を参照してください。

28.5.1　添付ファイルのインスタンスを生成する方法（wsi:swaRef 形式）

添付ファイルのオブジェクトとして，javax.activation.DataHandler オブジェクトを生成します。javax.activation.DataHandler クラスのコンストラクタを次の表に示します。

表 28–10　javax.activation.DataHandler クラスのコンストラクタ

項番	利用ケース	コンストラクタ	引数の説明
1	ファイルを添付する	DataHandler(javax.activation.DataSource ds)	[第 1 引数] javax.activation.DataSource オブジェクトです。 javax.activation.FileDataSource クラスのオブジェクトを指定できます。
2	メモリ上のオブジェクトを添付する	DataHandler(java.lang.Object obj, java.lang.String mimeType)	[第 1 引数] Java オブジェクトです。 [第 2 引数] オブジェクトの MIME タイプです。 指定できる MIME タイプについては， 「28.4.2(3)　添付ファイルの拡張子と MIME タイプのマッピング」を参照してください。

添付ファイルとして送信するオブジェクトによって，javax.activation.DataHandler オブジェクトを生成する方法が異なります。送信するオブジェクトごとに，生成する方法を示します。

(1)　存在するファイルを添付ファイルとして送信する方法

すでに存在するファイルを添付して送信する場合の手順を示します。

1. javax.activation.FileDataSource オブジェクトを生成します。
   送信する添付ファイルのファイルパスを引数に指定して，javax.activation.FileDataSource オブジェクトを生成します。

```
javax.activation.FileDataSource fdSource =
 new javax.activation.FileDataSource("/tmp/sample.jpg");
```

2. javax.activation.DataHandler オブジェクトを生成します。

javax.activation.FileDataSource オブジェクトを引数に指定して，javax.activation.DataHandler オブジェクトを生成します。

```
javax.activation.DataHandler dhandler =
 new javax.activation.DataHandler(fdSource);
```

(2) Java オブジェクトを添付ファイルとして送信する方法

Java オブジェクトを添付ファイルとして送信する場合の手順を示します。

1. Java オブジェクトを生成します。

   ここでは，添付ファイルとして送信する java.awt.Image オブジェクトを生成します。

```
java.awt.Image attachments =
 Toolkit.getDefaultToolkit().createImage("sample.jpg");
```

2. javax.activation.DataHandler オブジェクトを生成します。

   Java オブジェクトと，Java オブジェクトに対応する MIME タイプを引数に指定し，javax.activation.DataHandler オブジェクトを生成します。

```
javax.activation.DataHandler dhandler = new
 javax.activation.DataHandler(attachments, "image/jpeg");
```

(3) java.lang.String オブジェクトを添付ファイルとして送信する方法

java.lang.String オブジェクトを添付ファイルとして送信する場合の手順を示します。

1. java.lang.String オブジェクトを生成します。

   添付ファイルとして送信する文字列の java.lang.String オブジェクトを生成します。

```
java.lang.String attachments = new java.lang.String("あいうえお");
```

2. javax.activation.DataHandler オブジェクトを生成します。

   java.lang.String オブジェクトと MIME タイプを引数に指定し，javax.activation.DataHandler オブジェクトを生成します。java.lang.String オブジェクトは，charset パラメタに指定された文字コードでエンコードされます。

```
javax.activation.DataHandler dhandler = new javax.activation.DataHandler(attachments, "te
xt/plain; charset=UTF-8");
```

(4) javax.activation.DataHandler オブジェクト生成時の注意事項

wsi:swaRef 形式の添付ファイルとして送信するオブジェクトが，テキストファイル，XML ファイル，または java.lang.String オブジェクト（文字列）の場合，javax.activation.DataHandler クラスの DataHandler（Object, String）コンストラクタを使用することで，オブジェクトに含まれる文字の文字コードを指定できます。

javax.activation.DataHandler オブジェクトの生成時に文字コードを指定しない場合，送信するオブジェクトはデフォルトの文字コード（US-ASCII）でエンコードされます。送信するオブジェクトにデフォルトの文字コード（US-ASCII）以外の文字が含まれている場合は，不正な添付ファイルを送信する扱いとなるため，特に日本語を含む場合には，適切な文字コードを指定してください。

送信するオブジェクトが XML ファイルの場合，XML 宣言に指定している文字コードは使用されません。デフォルトの文字コード（US-ASCII），または javax.activation.DataHandler オブジェクトの生成時に指定した文字コードでエンコードされて，XML ファイルが送信されます。

文字コードを指定するには，DataHandler（Object, String）コンストラクタの第 2 引数で指定するオブジェクトの MIME タイプに charset パラメタを付けます。DataHandler（Object, String）コンストラクタの第 2 引数の記述形式を次に示します。

```
送信するオブジェクトのMIMEタイプ＋";"＋"charset"＋"="＋文字コード※
```

注※

大文字または小文字を区別しません。また，JDK がサポートしていない文字コードおよび空文字を指定できません。

28.5.2　添付ファイルデータを取得する方法（wsi:swaRef 形式）

添付ファイルのデータを取得する方法について説明します。各例に示す"dhandler"は，受信した javax.activation.DataHandler オブジェクトを表します。

(1)　添付ファイルを java.io.InputStream オブジェクトとして取得する方法

受信した添付ファイルを java.io.InputStream オブジェクトとして取得する場合，受信した javax.activation.DataHandler オブジェクトから getInputStream メソッドで java.io.InputStream オブジェクトを取得します。

```
java.io.InputStream stream = dhandler.getInputStream();
```

(2)　添付ファイルを javax.activation.DataSource オブジェクトとして取得する方法

受信した添付ファイルを javax.activation.DataSource オブジェクトとして取得する場合，受信した javax.activation.DataHandler オブジェクトから getDataSource メソッドで関連づけられた javax.activation.DataSource オブジェクトを取得します。

```
javax.activation.DataSource datasource = dhandler.getDataSource();
```

(3) 添付ファイルを Java オブジェクトとして取得する方法

受信した添付ファイルを Java オブジェクトとして取得する場合の手順を示します。ここでは，java.awt.Image オブジェクトを取得する場合の例を示します。

1. 添付ファイルのデータをオブジェクトとして取得します。

   受信した javax.activation.DataHandler オブジェクトから getContent メソッドでオブジェクトを取得します。

```
java.lang.Object content = dhandler.getContent();
```

2. 添付ファイルの MIME タイプを取得します。

   受信した javax.activation.DataHandler オブジェクトに対して，getContentType メソッドを実行します。getContentType メソッドを実行することで，添付ファイルの MIME タイプを取得できます。

```
java.lang.String mimetype = dhandler.getContentType();

（取得したmimetypeの内容）image/jpeg
```

3. オブジェクトを適切な型にキャストします。

   添付ファイルの MIME タイプに応じて，オブジェクトを適切な型にキャストします。

```
java.awt.Image attachment = (java.awt.Image) content;
```

(4) 添付ファイルを java.lang.String オブジェクトとして取得する方法

受信した添付ファイルを Java オブジェクトとして取得する場合の手順を示します。

1. 添付ファイルのデータをオブジェクトとして取得します。

   受信した javax.activation.DataHandler オブジェクトから getContent メソッドでオブジェクトを取得します。

```
java.lang.Object content = dhandler.getContent();
```

2. 添付ファイルの MIME タイプを取得します。

   受信した javax.activation.DataHandler オブジェクトに対し，getContentType メソッドを実行します。getContentType メソッドを実行することで添付ファイルの MIME タイプおよび文字コードを取得できます。

```
java.lang.String mimetype = dhandler.getContentType();

（取得したmimetypeの内容）text/plain; charset=UTF-8
```

3. オブジェクトを適切な型にキャストします。

   添付ファイルの MIME タイプに応じて，オブジェクトを適切な型にキャストします。

```
java.lang.String attachment = (java.lang.String) content;
```

(5) javax.activation.DataHandler オブジェクト取得時の注意事項

wsi:swaRef 形式の添付ファイルを含む MIME Multipart/Related 構造の SOAP メッセージを受信すると，ストリーミングされた wsi:swaRef 形式の添付ファイルとして操作できます。wsi:swaRef 形式の添付ファイルの受信側では，javax.activation.DataHandler オブジェクトが持つ入力ストリームからすべてのデータを読み込まないと受信処理が完了しないため，送信側の送信処理は受信側の受信処理が完了するまで待ち状態が続きます。

この状態を解消するためには，javax.activation.DataHandler オブジェクトが持つ java.io.InputStream オブジェクトからすべてのデータを読み込むか，javax.activation.DataHandler クラスの writeTo(java.io.OutputStream)メソッドを使用して入力ストリームのデータを出力ストリームに書き込む必要があります。

28.6　MIME Multipart/Related 構造の SOAP メッセージの受信

1048576 バイト以上のサイズの MIME ボディを含む MIME Multipart/Related 構造の SOAP メッセージを受信した場合，JAX-WS エンジンは SOAP メッセージに含まれる MIME ボディを一時ファイルに出力します。

MIME Multipart/Related 構造の SOAP メッセージの受信の詳細については，「10.24　MIME Multipart/Related 構造の SOAP メッセージの受信」を参照してください。

29 SEI を起点とした開発の例（wsi:swaRef 形式の添付ファイル使用時）

この章では，添付ファイルを使用して，SEI を起点とした Web サービスを開発する場合の例を説明します。

29.1　開発例の構成（SEI 起点・wsi:swaRef 形式の添付ファイル）

この章で説明する開発例では，SEI を起点とした Web サービスを開発します。開発する Web サービスは，添付ファイルを使用します。

開発する Web サービスの概要および利用する情報について説明します。

開発例の概要

社員番号，顔写真，社員名，所属というユーザ情報を管理し，Web サービスクライアントからの入力に対して，処理結果を返す Web サービスを新規に開発します。

Web サービスクライアントからの要求情報およびサーバからの応答情報を次の表に示します。

表 29-1　Web サービスクライアントからの要求情報

情報名	Java データ型
社員番号	java.lang.String
顔写真	javax.activation.DataHandler

表 29-2　サーバからの応答情報

情報名	Java データ型
登録確認メッセージ	java.lang.String
名前	java.lang.String
所属	java.lang.String

サーバからの応答情報は，ユーザ定義型クラスの UserData クラスで保持されます。

Web サービス開発時のカレントディレクトリの構成を次の表に示します。

表 29-3　カレントディレクトリの構成（SEI 起点・添付ファイル）

ディレクトリ	説明
c:¥temp¥jaxws¥works¥attachments	カレントディレクトリです。
├server¥	Web サービスの開発で使用します。
│├META-INF¥	EAR ファイルの META-INF ディレクトリに対応します。
│└application.xml	「29.3.4　application.xml を作成する」で作成します。
│├src¥	Web サービスのソースファイル（*.java）を格納します。「29.3.1　Web サービス実装クラスを作成する」および「29.3.2　Web サービス実装クラスをコンパイルする」で使用します。
│├WEB-INF¥	WAR ファイルの WEB-INF ディレクトリに対応します。

ディレクトリ	説明
┃┃┃┃┃┃┃┃┃┣ web.xml	「29.3.3　web.xml を作成する」で作成します。
┃┃┃┃┃┃┃┃┃┗ classes¥	コンパイルしたクラスファイル（*.class）を格納します。「29.3.2　Web サービス実装クラスをコンパイルする」で使用します。
┃┃┣ attachments_dynamic_generate.ear	「29.3.5　EAR ファイルを作成する」で作成します。
┃┃┗ attachments_dynamic_generate.war	
┗ client¥	Web サービスクライアントの開発で使用します。
┣ src¥	Web サービスクライアントのソースファイル（*.java）を格納します。「29.5.1　サービスクラスを生成する」および「29.5.2　Web サービスクライアントの実装クラスを作成する」で使用します。
┣ classes¥	コンパイルしたクラスファイル（*.class）を格納します。「29.5.3　Web サービスクライアントの実装クラスをコンパイルする」で使用します。
┣ image.jpg	Web サービスクライアントで使用する JPEG ファイルで使用します。
┣ usrconf.cfg	「29.6.1　Java アプリケーション用オプション定義ファイルを作成する」で作成します。
┗ usrconf.properties	「29.6.2　Java アプリケーション用ユーザプロパティファイルを作成する」で作成します。

カレントディレクトリのパスは，開発する環境に合わせて変更してください。

なお，以降の説明では，この表に示すディレクトリおよびファイル名を使用します。コマンド実行例や Java ソースなどで背景色付きの太字になっている部分は，この例で使用する指定値や生成される値を示します。構築する環境に合わせて読み替えてください。

また，この章で説明する開発例では，Web サービスと Web サービスクライアントを同じ環境で開発しますが，別の環境で開発することもできます。別の環境で開発する場合は，それぞれの環境に合わせて，カレントディレクトリのパスを読み替えてください。

29.2　開発例の流れ（SEI 起点・wsi:swaRef 形式の添付ファイル）

この章で説明する開発例では，次の流れで開発および実行します。

Web サービスの開発

1. Web サービス実装クラスを作成する（29.3.1）
2. javac コマンドを実行し，Web サービス実装クラスをコンパイルする（29.3.2）
3. web.xml を作成する（29.3.3）
4. application.xml を作成する（29.3.4）
5. EAR ファイルを作成する（29.3.5）

デプロイと開始

1. EAR ファイルをデプロイする（29.4.1）
2. Web サービスを開始する（29.4.2）

Web サービスクライアントの開発

1. cjwsimport コマンドを実行し，サービスクラスを生成する（29.5.1）
2. Web サービスクライアントの実装クラスを作成する（29.5.2）
3. Web サービスクライアントの実装クラスをコンパイルする（29.5.3）

Web サービスの実行

1. Java アプリケーション用オプション定義ファイルを作成する（29.6.1）
2. Java アプリケーション用ユーザプロパティファイルを作成する（29.6.2）
3. Web サービスクライアントを実行する（29.6.3）

29.3　Web サービスの開発例（SEI 起点・wsi:swaRef 形式の添付ファイル）

ここでは，SEI を起点とした場合の Web サービス（添付ファイルを使用）の開発例を説明します。

29.3.1　Web サービス実装クラスを作成する

Web サービスの処理を記述した Web サービス実装クラスを作成します。ここでは，受け取った要求メッセージの内容を計算して，応答メッセージを返す Web サービス実装クラスを作成します。

Web サービス実装クラスの作成例を次に示します。

```
package com.sample;

import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.xml.bind.annotation.XmlAttachmentRef;
import javax.activation.DataHandler;

@javax.jws.WebService(serviceName="UserInfoService",targetNamespace="http://sample.com")
public class UserInfoImpl{

    public UserData getUserData( String in0,
        @XmlAttachmentRef javax.activation.DataHandler in1 )
    throws UserInfoException{
        //社員情報への顔写真の登録処理
        ・・・・

        UserData userdata = new UserData();
        //登録した社員の名前と所属を設定
        if ( in0.equals("1") )
        {
            userdata.setName("Sato Taro");
            userdata.setSection("The personnel section");
        } if ( ・・・・) {
            ・・・・・
        } ・・・・

        //登録確認メッセージを設定
        if ( in1 == null )
        {
            userdata.setMessage("Failure(no image).");
        } else {
            userdata.setMessage("Success.");
        }
        return userdata;
    }

}
```

作成した UserInfoImpl.java は，UTF-8 形式で
c:¥temp¥jaxws¥works¥attachments¥server¥src¥com¥sample¥ディレクトリに保存します。

また，com.sample.UserInfoImpl で使用しているユーザ定義型クラス com.sample.UserData を作成します。通常，ユーザ定義型クラスの作成は任意ですが，ここではユーザ定義型クラスを作成します。

ユーザ定義型クラスの作成例を次に示します。

```
package com.sample;

public class UserData{

    private java.lang.String message;
    private java.lang.String name;
    private java.lang.String section;

    public UserData(){
    }

    public java.lang.String getMessage() {
        return this.message;
    }

    public void setMessage(java.lang.String message) {
        this.message = message;
    }

    public java.lang.String getName() {
        return this.name;
    }

    public void setName(java.lang.String name) {
        this.name = name;
    }

    public java.lang.String getSection() {
        return this.section;
    }

    public void setSection(java.lang.String section) {
        this.section = section;
    }
}
```

作成した UserInfoImpl.java は，UTF-8 形式で，
c:¥temp¥jaxws¥works¥attachments¥server¥src¥com¥sample¥ディレクトリに保存します。

また com.sample.UserInfoImpl でスローしている例外クラス com.sample.UserInfoException を作成します。通常，例外クラスの作成は任意ですが，ここでは例外クラスを作成します。

例外クラスの作成例を次に示します。

```
package com.sample;

public class UserInfoException extends Exception {
    String detail;
```

```
    public UserInfoException (String message, String detail) {
        super (message);
        this.detail = detail;
    }

    public String getDetail () {
        return detail;
    }
}
```

作成した UserInfoException.java は，UTF-8 形式で c:¥temp¥jaxws¥works¥attachments¥server¥src¥com¥sample¥ディレクトリに保存します。

29.3.2　Web サービス実装クラスをコンパイルする

javac コマンドを実行して，Web サービス実装クラスをコンパイルします。javac コマンドについては，JDK のドキュメントを参照してください。

javac コマンドの実行例を次に示します。添付ファイル機能を使用した Java プログラムを javac コマンドでコンパイルする場合，javac コマンドの引数に"--add-modules=java.activation"を指定してください。

```
> cd c:¥temp¥jaxws¥works¥attachments¥server¥
> mkdir WEB-INF¥classes¥
> javac -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥javaee¥1100¥lib¥ja
vaee-api.jar;%COSMINEXUS_HOME%¥CC¥client¥lib¥HiEJBClientStatic.jar;%COSMINEXUS_HOME%¥jaxp¥li
b¥csmjaxb.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxp.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar"
 -d WEB-INF¥classes¥ -s src src¥com¥sample¥UserInfoImpl.java src¥com¥sample¥UserData.java sr
c¥com¥sample¥UserInfoException.java
```

javac コマンドが正常に終了すると，コンパイルしたクラスが，c:¥temp¥jaxws¥works¥attachments¥server¥WEB-INF¥classes¥com¥sample¥ディレクトリに出力されます。なお，コンパイルした Web サービス実装クラスに対して hwsgen コマンドを実行すると，事前にエラーチェックができます。hwsgen コマンドについては，「14.1.2　hwsgen コマンド」を，エラーチェックについては，「10.23.1　hwsgen コマンドによるエラーチェックについて」を参照してください。

29.3.3　web.xml を作成する

WAR ファイルの構成要素として必要な web.xml を作成します。

web.xml の作成例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/we
b-app_3_0.xsd">
  <description>Sample web service &quot;fromjava&quot;</description>
```

```
  <display-name>Sample_web_service_fromjava</display-name>
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/UserInfoService</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

バージョン 2.5 の web.xml を作成する場合は，web-app 要素の version 属性を"2.5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"にしてください。

作成した web.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥attachments¥server¥WEB-INF¥ディレクトリに保存します。web.xml の設定項目については，「3.4　web.xml の作成」を参照してください。

29.3.4　application.xml を作成する

EAR ファイルの構成要素として必要な application.xml を作成します。

application.xml の作成例を次に示します。なお，Web サービスとして application.xml に設定する項目はありません。

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/application_6.xsd">

  <description>Sample application &quot;fromjava&quot;</description>
  <display-name>Sample_application_fromjava</display-name>
  <module>
    <web>
      <web-uri>attachments_dynamic_generate.war</web-uri>
      <context-root>attachments_dynamic_generate</context-root>
    </web>
```

```
  </module>
</application>
```

バージョン 5 の application.xml を作成する場合は，application 要素の version 属性を"5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/application_5.xsd"にしてください。

作成した application.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥attachments¥server¥META-INF¥ディレクトリに保存します。application.xml を作成するときの注意事項については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「5.2.2　application.xml 編集時の注意事項」を参照してください。

29.3.5　EAR ファイルを作成する

jar コマンドを使用して，これまでに作成したファイルを含めた EAR ファイルを作成します。

EAR ファイルの作成例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥attachments¥server¥
> jar cvf attachments_dynamic_generate.war .¥WEB-INF
> jar cvf attachments_dynamic_generate.ear .¥attachments_dynamic_generate.war .¥META-INF¥app
lication.xml
```

正常に終了すると，c:¥temp¥jaxws¥works¥attachments¥server¥ディレクトリに attachments_dynamic_generate.ear が作成されます。

jar コマンドについては，JDK のドキュメントを参照してください。

29.4　デプロイと開始の例（SEI 起点・wsi:swaRef 形式の添付ファイル）

ここでは，添付ファイルを使用して，SEI を起点とした場合のデプロイと開始の例を説明します。

29.4.1　EAR ファイルをデプロイする

cjimportapp コマンドを使用して，作成した EAR ファイルを J2EE サーバにデプロイします。

デプロイの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥attachments¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjimportapp" jaxwsserver -nameserver corbaname::testserver
:900 -f attachments_dynamic_generate.ear
```

cjimportapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションをデプロイ（インポート）する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.3　J2EE アプリケーションのインポート」を参照してください。

29.4.2　Web サービスを開始する

cjstartapp コマンドを使用して，Web サービスを開始します。

Web サービスを開始する例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥attachments¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjstartapp" jaxwsserver -nameserver corbaname::testserver:
900 -name Sample_application_fromjava
```

cjstartapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartapp（J2EE アプリケーションの開始）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションを開始する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.1　J2EE アプリケーションの開始」を参照してください。

29.5　Web サービスクライアントの開発例（SEI 起点・wsi:swaRef 形式の添付ファイル）

ここでは，添付ファイルを使用して，SEI を起点とした場合の Web サービスクライアントの開発例を説明します。

29.5.1　サービスクラスを生成する

cjwsimport コマンドを実行すると，サービスクラスなど Web サービスクライアントの開発に必要な Java ソースが生成されます。cjwsimport コマンドについては，「14.1.1　cjwsimport コマンド」を参照してください。

cjwsimport コマンドの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥attachments¥client¥
> mkdir src¥
> mkdir classes¥
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" -s src -d classes http://webhost:8085/attachments_dynamic_generate/UserInfoService?wsdl
```

正常に終了すると，c:¥temp¥jaxws¥works¥attachments¥client¥src¥com¥sample¥ディレクトリに Java ソースが生成されます。

生成物の一覧を次に示します。

表 29-4　サービスクラス生成時の生成物（SEI 起点・添付ファイル）

ファイル名	説明
GetUserData.java	WSDL 定義の要求メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
GetUserDataResponse.java	WSDL 定義の応答メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
ObjectFactory.java	JAXB 2.2 仕様の ObjectFactory クラスです。
package-info.java	package-info.java ファイルです。
UserData.java	UserData に対応する JavaBean クラスです。
UserInfoImpl.java	WSDL 定義の「サービス」に対応する SEI です。
UserInfoService.java	サービスクラスです。
UserInfoException.java	UserInfoException に対応する JavaBean クラスです。
UserInfoException_Exception.java	フォルト bean のラッパ例外クラスです。

29.5.2　Web サービスクライアントの実装クラスを作成する

Web サービスを利用する Web サービスクライアントの実装クラスを作成します。

Web サービスに対して 1 回の呼び出しをする Web サービスクライアントの作成例を次に示します。

```
package com.sample.client;

import java.io.File;

import javax.activation.DataHandler;
import javax.activation.FileDataSource;

import com.sample.UserInfoImpl;
import com.sample.UserData;
import com.sample.UserInfoService;
import com.sample.UserInfoException_Exception;

public class TestClient {
    public static void main( String[] args ) {
        try {
            //DataHandlerオブジェクト生成
            File imagefile = new File("image.jpg");
            FileDataSource fdSource = new FileDataSource(imagefile);
            DataHandler dhandler = new DataHandler(fdSource);

            UserInfoService service = new UserInfoService();
            UserInfoImpl_port = service.getUserInfoImplPort();

            UserData userdata = port.getUserData( "1", dhandler );

            System.out.print( "[RESULT] " + userdata.getMessage() );
            System.out.println( " Name:" + userdata.getName()
                                + ", Section:" + userdata.getSection() );
        }
        catch( UserInfoException_Exception e ){
            e.printStackTrace();
        }
        catch( Exception e ){
            e.printStackTrace();
        }
    }

}
```

作成した TestClient.java は，UTF-8 形式で
c:¥temp¥jaxws¥works¥attachments¥client¥src¥com¥sample¥client¥ディレクトリに保存します。

29.5.3　Web サービスクライアントの実装クラスをコンパイルする

javac コマンドを使用して，作成した Web サービスクライアントをコンパイルします。

コンパイルの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥attachments¥client¥
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥javaee¥1100¥lib¥javaee-api.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;.¥classes" -d .¥classes src¥com¥sample¥client¥TestClient.java
```

正常に終了すると，c:¥temp¥jaxws¥works¥attachments¥client¥classes¥com¥sample¥client¥ディレクトリに，TestClient.class が生成されます。

javac コマンドについては，JDK のドキュメントを参照してください。

29.6　Web サービスの実行例（SEI 起点・wsi:swaRef 形式の添付ファイル）

ここでは，添付ファイルを使用して，SEI を起点とした場合の Web サービスクライアントの実行例を説明します。

29.6.1　Java アプリケーション用オプション定義ファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用オプション定義ファイル（usrconf.cfg）を作成します。

Java アプリケーション用オプション定義ファイルの作成例を次に示します。

```
add.class.path=<Application Serverのインストールディレクトリ>¥jaxws¥lib¥cjjaxws.jar
add.class.path=.¥classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home=<Application Serverのインストールディレクトリ>
```

<Application Server のインストールディレクトリ>の部分は，Application Server をインストールしているパスを絶対パスで指定します。

作成した Java アプリケーション用オプション定義ファイルは，c:¥temp¥jaxws¥works¥attachments¥client¥ディレクトリに保存します。Java アプリケーション用オプション定義ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.1　usrconf.cfg（Java アプリケーション用オプション定義ファイル）」を参照してください。

29.6.2　Java アプリケーション用ユーザプロパティファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用ユーザプロパティファイルを作成します。

ここでは特に設定を変更しないため，c:¥temp¥jaxws¥works¥attachments¥client¥ディレクトリに usrconf.properties という名称の空のファイルを作成します。Java アプリケーション用ユーザプロパティファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.2　usrconf.properties（Java アプリケーション用ユーザプロパティファイル）」を参照してください。

29.6.3　Web サービスクライアントを実行する

cjclstartap コマンドを使用して，Web サービスクライアントを実行します。

Web サービスクライアントの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥attachments¥client¥
> "%COSMINEXUS_HOME%¥CC¥client¥bin¥cjclstartap"  com.sample.client.TestClient
```

正常に終了すると，Web サービスクライアントの実行結果が表示されます。実行結果の表示例を次に示します。

```
KDJE40053-I The cjclstartap command will now start. (directory for the user definition file
= c:¥temp¥jaxws¥works¥attachments¥client, PID = 2636)
[RESULT] Success. Name:Sato Taro, Section:The personnel section
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

イタリック体になっている個所は，実行したタイミングや環境によって変わります。

cjclstartap コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjclstartap（Java アプリケーションの開始)」を参照してください。

30 添付ファイル機能（MTOM/XOP）

添付ファイル機能を使用することで，テキストデータだけでなく，画像データや音声データなどを Web サービスで扱えます。

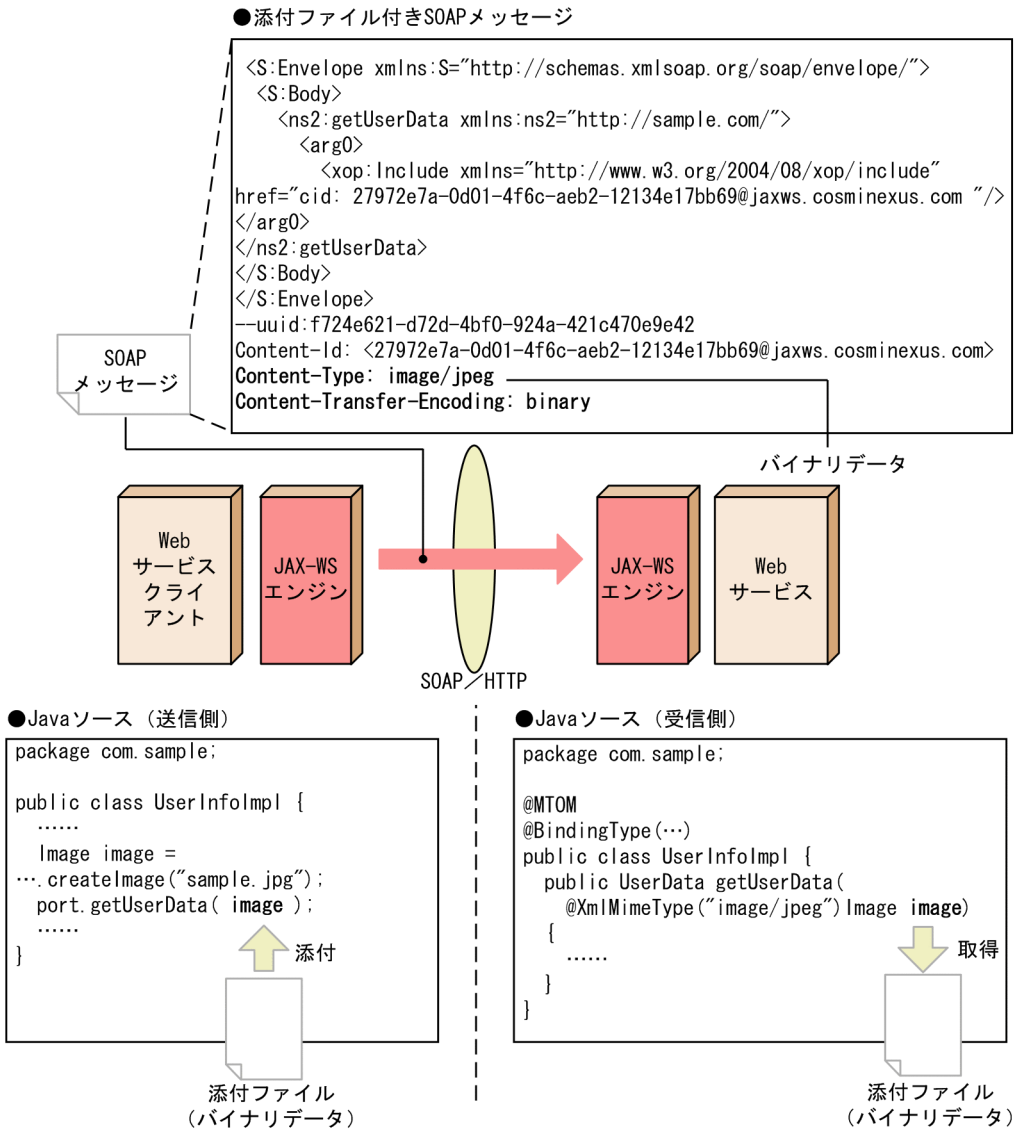
この章では，添付ファイル機能の概要，および添付ファイル機能を使用した場合のマッピング規則について説明します。

30.1　添付ファイル機能とは（MTOM/XOP）

MTOM/XOP 仕様形式の添付ファイル機能とは，SOAP Message Transmission Optimization Mechanism（MTOM）と XML-binary Optimized Packaging（XOP）の両機能を組み合わせ，SOAP メッセージ中にある Base64 形式のデータ（xsd:base64Binary 型の XML データ）をバイナリデータとして送受信する機能です。この機能によって送受信されるデータは，MIME Multipart/Related 構造の SOAP メッセージに添付されます。

次に示す例を基に，添付ファイル機能の概要について説明します。

図 30-1　添付ファイル機能を使用したバイナリデータの送受信



30.2　添付ファイルの Java インタフェース（MTOM/XOP）

ここでは，Java インタフェース内で添付ファイルを指定する Java 型について説明します。

30.2.1　MTOM/XOP 仕様形式の添付ファイルの対象

MTOM/XOP 仕様形式の添付ファイルは，次に示す個所の Base64 形式のデータにマッピングされる Java 型がすべて対象です。

- メソッド引数
- メソッド戻り値
- ユーザ定義型のフィールド

Base64 形式のデータにマッピングされる Java 型が複数ある場合，特定の Java 型だけを MTOM/XOP 仕様形式の添付ファイルの対象にすることはできません。

30.2.2　MTOM/XOP 仕様形式の添付ファイルで使用するアノテーション

MTOM/XOP 仕様形式の添付ファイルで使用するアノテーションは次のとおりです。

- javax.xml.ws.soap.MTOM
- javax.xml.bind.annotation.XmlMimeType

30.2.3　MTOM/XOP 仕様形式の添付ファイルの使用方法

Web サービスで MTOM/XOP 仕様形式の添付ファイルを使用するには，Web サービス実装クラスに javax.xml.ws.soap.MTOM アノテーションをアノテートします。MTOM/XOP 仕様形式の添付ファイルを使用した Web サービス実装クラスの例を次に示します。

```
package com.sample;

@MTOM
@BindingType(・・・)
public class UserInfoImpl implements UserInfo {

    public UserData setUserData(Image image)
        throws UserDefinedException {
        ・・・・・・・
    }
}
```

また，javax.xml.ws.soap.MTOM アノテーションの代わりに javax.xml.ws.soap.SOAPBinding インタフェースのフィールド値を javax.xml.ws.BindingType アノテーションに指定することで，MTOM/XOP 仕様形式の添付ファイルを使用できます。javax.xml.ws.soap.MTOM アノテーションと javax.xml.ws.BindingType アノテーションに指定する javax.xml.ws.soap.SOAPBinding インタフェースのフィールド値の関係を次の表に示します。

表 30–1　MTOM アノテーションと SOAPBinding インタフェースのフィールドの関係

項番	MTOM アノテーション		BindingType アノテーション		MTOM/XOP 仕様形式の添付ファイル
	アノテーション有無	enabled 要素値	SOAPBinding インタフェースのフィールド値	フィールド値の記述可否	
1	あり	true	SOAP11HTTP_BINDING	可	○
2			SOAP12HTTP_BINDING	可	○
3			SOAP11HTTP_MTOM_BINDING	可	○
4			SOAP12HTTP_MTOM_BINDING	可	○
5		false	SOAP11HTTP_BINDING	可	×
6			SOAP12HTTP_BINDING	可	×
7			SOAP11HTTP_MTOM_BINDING	否※	−
8			SOAP12HTTP_MTOM_BINDING	否※	−
9	なし	−	SOAP11HTTP_BINDING	可	×
10			SOAP12HTTP_BINDING	可	×
11			SOAP11HTTP_MTOM_BINDING	可	○
12			SOAP12HTTP_MTOM_BINDING	可	○

（凡例）

○：有効です。

×：無効です。

−：該当しません。

注※

Web サービス開始時に javax.xml.ws.WebServiceException が発生します。

javax.xml.ws.soap.MTOM アノテーションの代わりに javax.xml.ws.soap.SOAPBinding インタフェースのフィールドを使用した Web サービス実装クラスの例を次に示します。

```
package com.sample;

@BindingType(SOAPBinding.SOAP11HTTP_MTOM_BINDING)
public class UserInfoImpl implements UserInfo {

    public UserData setUserData(Image image)
        throws UserDefinedException {
        . . . . . .
```

```
    }
}
```

30.2.4 Document Bare スタイルでの MTOM/XOP 仕様形式の添付ファイル

Document Bare スタイルの Java クラスで，MTOM/XOP 仕様形式の添付ファイルを使用し，Java 型と MIME タイプを関連づける場合，Java 型をサービスメソッドの戻り値やパラメタに使用しないでください。Java 型と MIME タイプを関連づける場合は，ユーザ定義型のフィールドに Java 型を使用し，そのフィールドの getter メソッドに javax.xml.bind.annotation.XmlMimeType アノテーションをアノテートします。

30.3　添付ファイルの WSDL（MTOM/XOP）

WSDL に MTOM/XOP 仕様形式の添付ファイルが使用されていることを示す要素および属性はありません。そのため，WSDL から Web サービスが MTOM/XOP 仕様形式の添付ファイルを使用しているかどうかを判別することはできません。

30.3.1　non-wrapper スタイルでの MTOM/XOP 仕様形式の添付ファイル（MTOM/XOP）

non-wrapper スタイルの WSDL で，MTOM/XOP 仕様形式の添付ファイルを使用し，xsd:base64Binary 型に MIME タイプを指定する場合，wsdl:part 要素から参照される xsd:element 要素の type 属性に xsd:base64Binary 型を使用しないでください。xsd:base64Binary 型に MIME タイプを指定する場合は，wsdl:part 要素から参照される xsd:element 要素の type 属性に複合型を使用し，その複合型の xsd:element 要素の type 属性に xsd:base64Binary 型を指定し，xmime:expectedContentTypes 属性を付与します。

(1)　WSDL 内での xop:Include 要素の使用

xop:Include 要素は MTOM/XOP 仕様形式の添付ファイルの SOAP メッセージ中に出現し，ルートパートと添付ファイルパートを関連づける要素です。そのため，WSDL のスキーマ宣言に xop:Include 要素を使用できません。WSDL のスキーマ宣言に xop:Include 要素を使用した場合，動作は保証されません。

30.4　JAX-WS エンジンの動作

Web サービス側と Web サービスクライアント側の JAX-WS エンジンの動作について説明します。

30.4.1　Web サービス側 JAX-WS エンジンの動作

Web サービス側 JAX-WS エンジンで MTOM/XOP 仕様形式の添付ファイルを使用できるかどうかは，Web サービス実装クラスに指定されている javax.xml.ws.soap.MTOM アノテーション，または javax.xml.ws.BindingType アノテーションに指定する SOAPBinding インタフェースのフィールド値に依存します。MTOM/XOP 仕様形式の添付ファイルの使用可否とリクエストメッセージを受信したときの動作を次の表に示します。

表 30–2　MTOM/XOP 仕様形式の添付ファイルの使用可否とリクエストメッセージを受信したときの動作

項番	MTOM/XOP 仕様形式の添付ファイル	受信したリクエストメッセージに含まれるデータ	受信の成功／失敗	しきい値と送信する添付ファイルサイズの関係※	送信するレスポンスメッセージに含まれるデータ
1	使用する	バイナリデータ	成功	しきい値≦添付ファイルのサイズ	バイナリデータ
2		Base64 形式のデータ	成功	しきい値＞添付ファイルのサイズ	
3	使用しない	バイナリデータ	成功	なし	Base64 形式のデータ
4		Base64 形式のデータ	成功	なし	

注※

javax.activation.DataHandler を使用した場合はしきい値で判定されません。常にバイナリデータとして送信します。

Web サービス実装クラスに javax.xml.ws.soap.MTOM アノテーションを指定しない場合や javax.xml.ws.BindingType アノテーションに指定する SOAPBinding インタフェースのフィールド値に SOAPBinding.SOAP11HTTP_MTOM_BINDING および SOAPBinding.SOAP12HTTP_MTOM_BINDING を指定しない場合，MTOM/XOP 仕様形式の添付ファイルを使用しません。バイナリデータを含んだメッセージかどうかに関係なく，すべてのリクエストメッセージを受信します。また，レスポンスメッセージは Base64 形式のデータを含んだメッセージを送信します。

(1)　javax.xml.bind.annotation.XmlMimeType アノテーションによる変化

Java 型と MIME タイプを関連づける javax.xml.bind.annotation.XmlMimeType アノテーションを SEI などにアノテートしている場合とアノテートしていない場合では，MTOM/XOP 仕様形式の添付ファイルで送信されるメッセージの添付ファイルパートにある Content-Type フィールドの値が変化します。

javax.xml.bind.annotation.XmlMimeType アノテーションの可否と添付ファイルパートにある Content-Type フィールドの値の変化を次に示します。

- XmlMimeType アノテーションをアノテートしている
  添付ファイルパートにある Content-Type フィールドの値は XmlMimeType アノテーションの value 要素の値となります。
- XmlMimeType アノテーションをアノテートしていない
  添付ファイルパートにある Content-Type フィールドの値は使用している Java 型に対応した初期値となります。初期値を次の表に示します。

表 30–3　Java 型と Content-Type の初期値

項番	Java 型	Content-Type の初期値
1	java.awt.Image	"image/png"
2	javax.xml.transform.Source	"application/xml"
3	javax.activation.DataHandler	javax.activation.DataHandler オブジェクトの MIME タイプ※
4	java.awt.Image の配列型	"image/png"
5	javax.activation.DataHandler の配列型	javax.activation.DataHandler オブジェクトの MIME タイプ※
6	java.util.List<Image>	"image/png"
7	java.util.List<DataHandler>	javax.activation.DataHandler オブジェクトの MIME タイプ※
8	javax.xml.ws.Holder<Image>	"image/png"
9	javax.xml.ws.Holder<Source>	"application/xml"
10	javax.xml.ws.Holder<DataHandler>	javax.activation.DataHandler オブジェクトの MIME タイプ※
11	byte[]	"application/octet-stream"

注※

DataHandler を使用した場合，Content-Type フィールドの設定値は DataHandler オブジェクトの生成方法によって異なります。

- DataHandler（DataSource）コンストラクタで生成する場合
  FileDataSource の場合，入力となるファイルデータの拡張子から，JAF によって決定された MIME タイプが Content-Type フィールド値として設定されます。
- DataHandler（Object, String）コンストラクタで生成する場合
  DataHandler オブジェクト生成時に，コンストラクタの第 2 引数に指定した内容（MIME タイプ）がそのまま Content-Type フィールド値として設定されます。

30.4.2　Web サービスクライアント側 JAX-WS エンジンの動作

Web サービスクライアント側 JAX-WS エンジンで MTOM/XOP 仕様形式の添付ファイルを使用できるかどうかは，SEI を取得する際に使用する MTOMFeature クラスに依存します。MTOM/XOP 仕様形式の添付ファイルの使用可否とメッセージを送受信したときの動作を次の表に示します。

表 30–4　MTOM/XOP 仕様形式の添付ファイルの使用可否とメッセージを送受信したときの動作

項番	MTOM/XOP 仕様形式の添付ファイル	しきい値と送信する添付ファイルサイズの関係※	送信するリクエストメッセージに含まれるデータ	受信したレスポンスメッセージに含まれるデータ	受信の成功／失敗
1	使用する	しきい値≦添付ファイルのサイズ	バイナリデータ	バイナリデータ	成功
2				Base64 形式のデータ	成功
3		しきい値＞添付ファイルのサイズ	Base64 形式のデータ	バイナリデータ	成功
4				Base64 形式のデータ	成功
5	使用しない	－	Base64 形式のデータ	バイナリデータ	成功
6				Base64 形式のデータ	成功

（凡例）

－：該当しません。

注※

javax.activation.DataHandler を使用した場合はしきい値で判定されません。常にバイナリデータとして送信します。

MTOMFeature クラスはほかの Feature クラスと同時に使用できます。MTOMFeature クラスとほかの Feature クラスを同時に使用した例を次に示します。

```
package com.sample;

 . . . . . .

public class TestClient {

    public static void main(String[] args) {
        try {
            File portrait = new File(”portrait.png”);
            if (!portrait.exists() ) {
                throw new RuntimeException(”Cannot file ¥”portrait.png¥”.”);
            }
            BufferedImage image = ImageIO.read(portrait);

            AddressingFeature addressingFeature = new AddressingFeature();
            MTOMFeature mtomFeature = new MTOMFeature();

            UserInfoService service = new UserInfoService();
            UserInfoImpl port = service.getUserInfoImplPort(addressingFeature, mtomFeature);

            UserData userData = port.getUserData(image);
             . . . . . .
        } catch (Exception e) {
```

```
            e.printStackTrace();
        }
    }
}
```

MTOMFeature クラスの代わりに，javax.xml.ws.soap.SOAPBinding の setMTOMEnabled メソッドを使用することで，MTOM/XOP 仕様形式の添付ファイルを使用できるかどうかを設定できます。javax.xml.ws.soap.SOAPBinding の setMTOMEnabled メソッドを使用した，MTOM/XOP 仕様形式の添付ファイルの使用可否の設定例を次に示します。

```
package com.sample;

 . . . . . .

public class TestClient {

    public static void main(String[] args) {
        try {
            File portrait = new File(”portrait.png”);
            if (!portrait.exists() ) {
                throw new RuntimeException(”Cannot file ¥”portrait.png¥”.”);
            }
            BufferedImage image = ImageIO.read(portrait);

            UserInfoService service = new UserInfoService();
            UserInfoImpl port = service.getUserInfoImplPort();

            BindingProvider bindingProvider = (BindingProvider)port;
            Binding         binding         = bindingProvider.getBinding();
            SOAPBinding     soapBinding     = (SOAPBinding)binding;
            soapBinding.setMTOMEnabled(true);

            UserData userData = port.getUserData(image);
             . . . . . .
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

MTOM/XOP 仕様形式の添付ファイルの使用可否を javax.xml.ws.soap.SOAPBinding の setMTOMEnabled メソッドで設定する場合，先に MTOMFeature クラスや javax.xml.ws.soap.SOAPBinding の setMTOMEnabled メソッドで使用可否を設定していると，あとから setMTOMEnabled メソッドに設定した値は無効となり，使用可否の設定はできません。

SEI を取得する際に MTOMFeature クラスを使用しない場合や javax.xml.ws.soap.SOAPBinding の setMTOMEnabled メソッドによる MTOM/XOP 仕様形式の添付ファイルの使用可否を設定しない場合，MTOM/XOP 仕様形式の添付ファイルを使用しません。リクエストメッセージは Base64 形式のデータを含んだメッセージを送信します。また，バイナリデータを含んだメッセージかどうかに関係なく，すべてのレスポンスメッセージを受信します。

(1) xmime:expectedContentTypes 属性による変化

WSDL のスキーマ要素に xmime:expectedContentTypes 属性がある場合とない場合では，MTOM/XOP 仕様形式の添付ファイルで送信されるメッセージの添付ファイルパートにある Content-Type フィールドの値が変化します。xmime:expectedContentTypes 属性の有無と添付ファイルパートにある Content-Type フィールドの値の変化を次に示します。

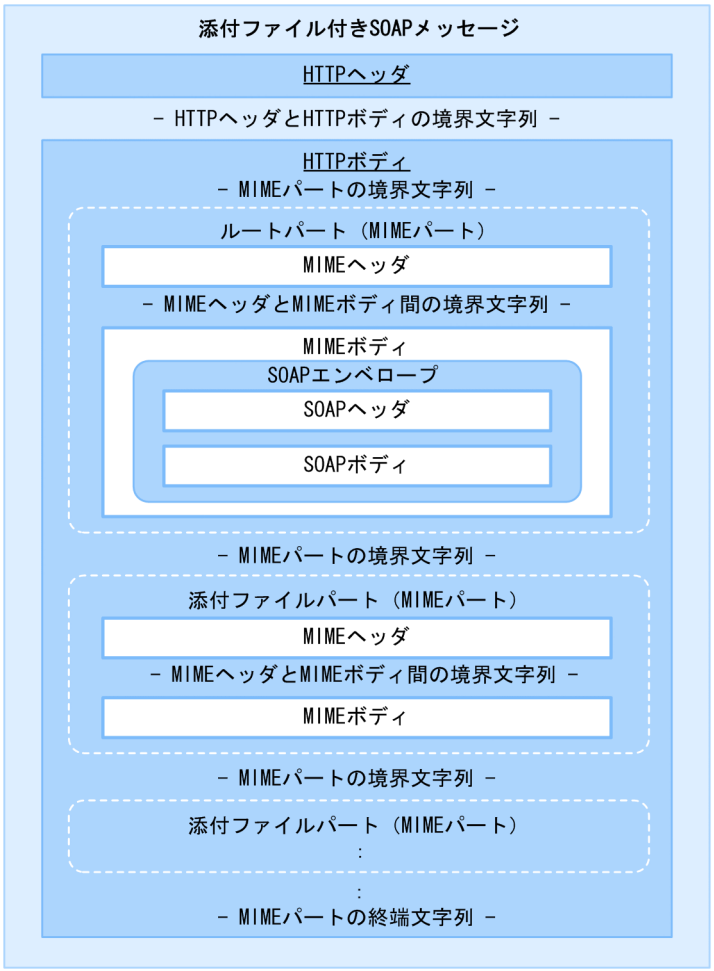
- xmime:expectedContentTypes 属性がある
  添付ファイルパートにある Content-Type フィールドの値は xmime:expectedContentTypes 属性に指定されている値となります。
- xmime:expectedContentTypes 属性がない
  添付ファイルパートにある Content-Type フィールドの値は"application/octet-stream"となります。

30.5　MTOM/XOP 仕様形式の添付ファイルの SOAP メッセージ

MTOM/XOP 仕様形式の添付ファイルを使用した場合の SOAP メッセージは，SOAP Messages with Attachments プロトコルを使用し生成された MIME multipart/related 構造です。ここでは，MTOM/XOP 仕様形式の添付ファイルの SOAP メッセージについて説明します。

MTOM/XOP 仕様形式の添付ファイル付き SOAP メッセージの構造を次の図に示します。

図 30-2　MTOM/XOP 仕様形式の添付ファイル付き SOAP メッセージの構造



MTOM/XOP 仕様形式の添付ファイル付き SOAP メッセージの各部の説明を次の表に示します。

表 30-5　MTOM/XOP 仕様形式の添付ファイル付き SOAP メッセージの各部の説明

各部の名称	説明
HTTP ヘッダ	HTTP プロトコルに依存するヘッダ情報です。
HTTP ヘッダと HTTP ボディの境界文字列	HTTP ヘッダと HTTP ボディの境界を示す文字列です。
HTTP ボディ	送信するメッセージを記述します。 ルートパートおよび添付ファイルパートから構成されます。
ト　MIME パートの境界文字列	各 MIME パートの境界を示す文字列です。

各部の名称				説明
├	ルートパート |			メッセージ本体を記述するパートです。 MIME ヘッダと MIME ボディから構成され，必ず 1 個定義します。
|	├	MIME ヘッダ		ルートパートのヘッダ情報です。
|	├	MIME ヘッダと MIME ボディ間の境界文字列		ルートパートの MIME ヘッダと MIME ボディ間の境界を示す文字列です。
|	└	MIME ボディ		メッセージ本体を記述します。
|		└	SOAP エンベロープ	SOAP エンベロープを記述します。
|			├ SOAP ヘッダ	SOAP メッセージのヘッダ情報を記述します。
|			└ SOAP ボディ	SOAP メッセージの本文（XML）を記述します。
├	MIME パートの境界文字列			各 MIME パートの境界を示す文字列です。
├	添付ファイルパート |			MTOM/XOP 仕様形式の添付ファイルの内容を記述するパートです。 MIME ヘッダと MIME ボディから構成され，0 個以上定義します。
|	├	MIME ヘッダ		添付ファイルパートのヘッダ情報です。
|	├	MIME ヘッダと MIME ボディ間の境界文字列		添付ファイルパートの MIME ヘッダと MIME ボディ間の境界を示す文字列です。
|	└	MIME ボディ		MTOM/XOP 仕様形式の添付ファイルの内容（バイナリデータ）を記述します。
└	MIME パートの終端文字列			MIME パートの終端を表す文字列です。

30.5.1　添付ファイルから SOAP メッセージへのマッピング（MTOM/XOP）

添付ファイルから SOAP メッセージへマッピングするときの設定値について説明します。マッピング規則については，「30.5.2　添付ファイルから SOAP メッセージへのマッピングの注意事項（MTOM/XOP）」と合わせて確認してください。

(1)　HTTP ヘッダ

MTOM/XOP 仕様形式の添付ファイル使用時に，HTTP ヘッダのフィールドおよびパラメタに設定される値を次の表に示します。

表 30–6　HTTP ヘッダのフィールドおよびパラメタの設定値

項番	フィールド名	パラメタ名	設定値
1	Content-Type	−	"multipart/related"が設定されます。※
2		start	ルートパートの Content-Id が設定されます。
3		type	"application/soap+xml"が設定されます。
4		boundary	MIME パートの境界文字列が設定されます。
5		start-info	SOAP 1.1 仕様 "text/xml"が設定されます。 SOAP 1.2 仕様 "application/soap+xml"が設定されます。

（凡例）

−：Content-Type に直接設定されることを示します。

注※

MIME パートがルートパートだけの場合や少なくとも 1 個の添付ファイル MIME パートを並べている（MTOM/XOP 仕様形式の添付ファイル付き SOAP メッセージである）場合でも，"multipart/related"を設定してください。

(2)　HTTP ボディ

HTTP ボディは，ルートパート，添付ファイルパート，および各パートの境界文字列で構成されます。添付ファイル使用時に，各パートに生成される内容，および設定される値を示します。

(a)　ルートパートの MIME ヘッダ

添付ファイル使用時に，ルートパートのフィールドに設定される値を次の表に示します。

表 30–7　ルートパートのフィールドおよびパラメタの設定値

項番	フィールド名	パラメタ名	設定値
1	Content-Type	−	"application/xop+xml"が設定されます。
2		charset	"utf-8"が設定されます。
3		type	SOAP 1.1 仕様 "text/xml"が設定されます。 SOAP 1.2 仕様 "application/soap+xml"が設定されます。
4	Content-Transfer-Encoding	なし	"binary"が設定されます。
5	Content-Id	なし	"rootpart*"＋"グローバルに一意な値"＋"@"＋"jaxws.cosminexus.com"が設定されます。

（凡例）

−：Content-Type に直接設定されることを示します。

(b) ルートパートの MIME ボディ

添付ファイルを使用する場合，ルートパートの MIME ボディには，SOAP エンベロープがそのまま格納されます。SOAP エンベロープ内の SOAP ボディから MTOM/XOP 仕様形式の添付ファイルを参照する方法として，XOP が使用されます。

(c) 添付ファイルパートの MIME ヘッダ

添付ファイル使用時に，添付ファイルパートの MIME ヘッダに設定される値を次の表に示します。

表 30-8 添付ファイルパートのフィールドおよびパラメタの設定値

項番	フィールド名	パラメタ名	設定値
1	Content-Type	－	XmlMimeType アノテーションの value 要素に指定した MIME タイプのテキスト表現が設定されます。※1，※2，※3
2		charset	MTOM/XOP 仕様形式の添付ファイルの対象が Source 型の場合，"UTF-8"が設定されます。※4 MTOM/XOP 仕様形式の添付ファイルの対象が DataHandler 型の場合，DataHandler オブジェクト生成時に指定した文字コードが設定されます。※5 MTOM/XOP 仕様形式の添付ファイルの対象が上記以外の Java 型の場合，パラメタは出現しません。
3	Content-Transfer-Encoding	なし	"binary"が設定されます。
4	Content-Id	なし	"グローバルに一意な値"＋"@"＋"jaxws.cosminexus.com"が設定されます。

（凡例）

－：Content-Type に直接設定されることを示します。

注※1

Content-Type フィールドに設定される MIME タイプで，XML を表す MIME タイプである"text/xml"と"application/xml"では，"application/xml"をお勧めします。

注※2

XmlMimeType アノテーションの value 要素にパラメタを持った MIME タイプを指定した場合，そのパラメタについても Content-Type フィールドに設定されます。

注※3

Content-Type フィールドの設定値は，DataHandler オブジェクトの生成方法によって，次のように異なります。

- DataHandler(DataSource)コンストラクタで生成する場合

  FileDataSource の場合，入力となる添付ファイルの拡張子から，JAF によって決定された MIME タイプが Content-Type フィールド値として設定されます。

- DataHandler(Object, String)コンストラクタで生成する場合

DataHandler オブジェクト生成時に，コンストラクタの第 2 引数に指定した内容（MIME タイプ）がそのまま Content-Type フィールド値として設定されます。

注※4

xmime:expectedContentTypes 属性または javax.xml.bind.annotation.XmlMimeType アノテーションで指定した MIME タイプが charset を含む場合，Content-Type フィールドにある charset の設定値は，指定した MIME タイプに含まれる charset の値になります。

注※5

次のように DataHandler オブジェクトを生成して，MTOM/XOP 仕様形式の添付ファイルとして送信した場合，Content-Type の charset には，第 2 引数に指定した charset パラメタ値「Shift_JIS」が設定されます。

```
DataHandler dhandler = new DataHandler (”あいうえお”, ”text/plain; charset=Shift_JIS”);
```

MTOM/XOP 仕様形式の添付ファイルの Java 型の javax.activation.DataHandler 型は，wsi:swaRef 形式の添付ファイルとして使用できる Java 型の javax.activation.DataHandler 型と同様に，JavaBeans Activation Framework(JAF)の型です。そのため，javax.activation.DataHandler 型には任意の MIME タイプの添付ファイルデータを指定できます。

30.5.2 添付ファイルから SOAP メッセージへのマッピングの注意事項（MTOM/XOP）

添付ファイルから SOAP メッセージをマッピングするときの注意事項について説明します。

(1) MIME パートの記述順序

MIME パートは 1 個のルートパートと 0 個以上の添付ファイルパートで構成されます。

ルートパートは，MIME パートの先頭に記述されます。ルートパートのあとに，添付ファイルパートが記述されます。

添付ファイルパートでは，Java インタフェースで指定された引数や戻り値がマッピングされます。Java インタフェースの指定内容と添付ファイルパートでの記述順序の対応を次の表に示します。

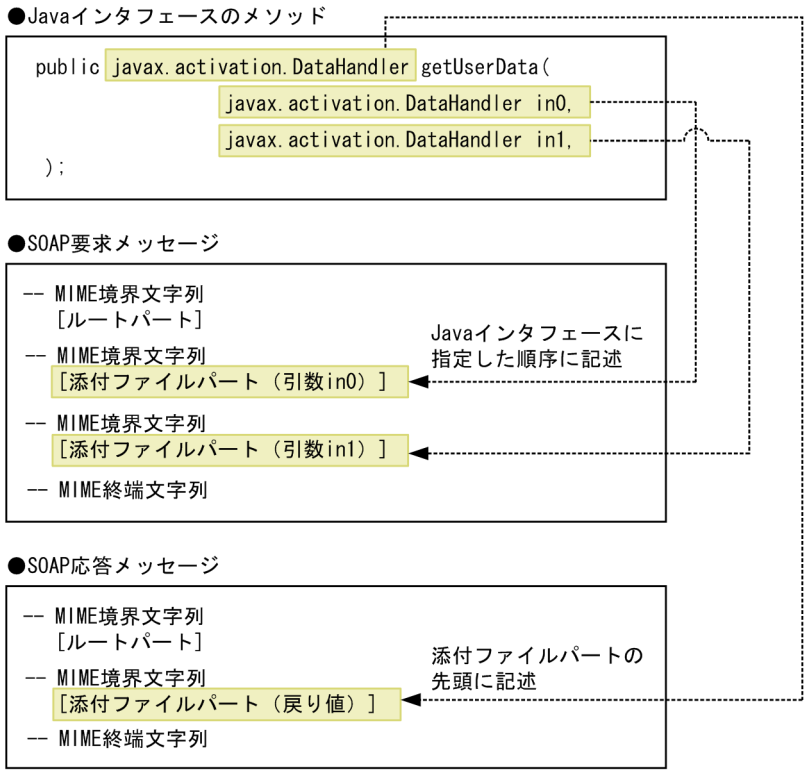
表 30-9 Java インタフェースの指定内容と添付ファイルパートの記述順序

項番	Java インタフェースでの指定内容	添付ファイルパートの記述順序
1	メソッド引数	メソッド引数に指定された順に記述されます。
2	メソッド戻り値	添付ファイルパートの先頭に記述されます。
3	配列型	配列内の要素順に記述されます。
4	ユーザ定義型	ユーザ定義型内での指定順に記述されます。

項番	Java インタフェースでの指定内容	添付ファイルパートの記述順序
		ユーザ定義型の中で指定したユーザ定義型で，添付ファイルの Java 型を指定するときは，深さ優先順で添付ファイルパートが記述されます。

Java インタフェースと添付ファイルパートのマッピング例を示します。

図 30-3　Java インタフェースと添付ファイルパートのマッピング例



(2)　ルートパートから添付ファイルへのマッピング

MTOM/XOP 仕様形式の添付ファイルの場合，ルートパートから添付ファイルパートへのマッピングには XOP 仕様で定められた XOP 情報セットが使用されます。XOP 情報セットがルートパートの SOAP ボディに記述され，対応する添付ファイルパートの Content-Id が設定されます。Content-Id によって，ルートパートから対応する添付ファイルパートを参照できます。ルートパートから添付ファイルパートの参照例を次に示します。背景色付きの太字部分が CID URL スキームと，対応する添付ファイルパートの Content-Id になります。

```
-uuid:e63fe7dc-ad8a-4fb1-8f56-ce7b5841a06f
Content-Type: text/xml

<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <getUserData xmlns="http://localhost">
      <in0>
        <xop:Include xmlns="http://www.w3.org/2004/08/xop/include" href="cid:39820675-44bb-4e28-9926-577bf27fa07c@jaxws.cosminexus.com"/>
      </in0>
```

```
    </getUserData>
  </S:Body>
</S:Envelope>

--uuid:e63fe7dc-ad8a-4fb1-8f56-ce7b5841a06f
Content-Id:<39820675-44bb-4e28-9926-577bf27fa07c@jaxws.cosminexus.com>
Content-Type: image/jpeg
Content-Transfer-Encoding: binary

[添付データ]
--e63fe7dc-ad8a-4fb1-8f56-ce7b5841a06f--
```

(a) XOP 情報セットの形式

ルートパートの SOAP ボディに記述される XOP 情報セットの形式を次に示します。

```
”<xop: Include href=” ＋ <CID URLスキーム> ＋ ”/>”
```

(b) MTOM/XOP 仕様形式の添付ファイルのデータサイズによるマッピング方法

MTOM/XOP 仕様形式の添付ファイルは，マッピング元である Java 型と Java インタフェースに指定する添付ファイルのデータサイズによって，添付ファイルパートへのマッピング方法が異なります。マッピング方法を次の表に示します。

表 30-10 MTOM/XOP 仕様形式の添付ファイルのデータサイズによる添付ファイルパートへのマッピング

項番	Java 型	添付ファイルのデータサイズ	添付ファイルパートへのマッピング
1	byte[]	null	マッピングしない※1
2		0 バイトのデータ	マッピングする※2
3		0 バイトより多いバイト数のデータ	マッピングする※3
4	java.awt.Image 型	null	マッピングしない※1
5		0 バイトのデータ	マッピングしない※1
6		0 バイトより多いバイト数のデータ	マッピングする※3
7	javax.xml.transform.Source 型	null	マッピングしない※1
8		0 バイトのデータ	マッピングする※2
9		0 バイトより多いバイト数のデータ	マッピングする※3
10	javax.activation.DataHandler 型	null	マッピングしない※1
11		0 バイトのデータ	マッピングする※2

項番	Java 型	添付ファイルのデータサイズ	添付ファイルパートへのマッピング
12		0 バイトより多いバイト数のデータ	マッピングする※3

注※1

wsi:swaRef 形式の添付ファイルと異なり，ルートパートだけで構成されるマルチパートとなります。ルートパートにある MIME ボディには SOAP エンベロープを格納します。SOAP ボディに該当する引数の要素は出現しません。添付ファイルデータが null である場合の XML へのマッピング例を次に示します。

- Web サービス呼び出しプログラム

```
...
UserInfoService service = new UserInfoService();
UserInfo impl = service.getUserInfo(new MTOMFeature());

result = impl.getUserData(  null  );
...
```

- SOAP メッセージ

```
--uuid:cbc0221b-8ee3-40a3-adc1-d5fa52a8d66e
Content-Id: <rootpart*cbc0221b-8ee3-40a3-adc1-d5fa52a8d66e@jaxws.cosminexus.com>
Content-Type: application/xop+xml;charset=utf-8;type="text/xml"
Content-Transfer-Encoding: binary

<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:getUserData xmlns:ns2="http://localhost">
    </ns2:getUserData>
  </S:Body>
</S:Envelope>
--uuid:cbc0221b-8ee3-40a3-adc1-d5fa52a8d66e--
```

注※2

ルートパートとデータがない（MIME ボディが空である）添付ファイルパートから構成されるマルチパートとなります。Java インタフェースで複数の添付ファイルデータを指定している場合は，添付ファイルパートは複数となります。

注※3

ルートパートと添付ファイルパートから構成されるマルチパートとなります。Java インタフェースで複数の添付ファイルデータを指定している場合は，添付ファイルパートは複数となります。

30.5.3 SOAP メッセージから添付ファイルへのマッピング（MTOM/XOP）

JAX-WS エンジンは，受信した添付ファイル付き SOAP メッセージが MTOM/XOP 仕様に従っている場合にだけ，添付ファイル付き SOAP メッセージから添付ファイルデータへのマッピングは保証されます。それ以外の SOAP メッセージを受信した場合の動作は保証されません。

30.6 注意事項

MTOM/XOP 形式の添付ファイルを使用するときの注意事項を次に示します。

- wsi:swaRef 形式の添付ファイルと同時に使用できません。同時に使用した場合の動作は保証されません。
- wsi:swaRef 形式の添付ファイルと，一つのオペレーションで同時に送受信することはできません。
- プロバイダ実装クラス（javax.xml.ws.provider インタフェースを実装するクラス）では使用できません。プロバイダ実装クラスで MTOM/XOP 仕様形式の添付ファイルを使用した場合，動作は保証されません。
- ハンドラフレームワークは併用できません。併用した場合の動作は保証されません。ハンドラフレームワークを使用する場合で添付ファイルを送受信したいときは，添付ファイル機能（wsi:swaRef 形式）を使用してください。
- WSS（Web Services-Security）は併用できません。併用した場合の動作は保証されません。

30.7　添付ファイルで使用できる Java 型と送受信できるデータ（MTOM/XOP 形式）

MTOM/XOP 形式の添付ファイルには，送受信するデータに応じて byte[]，およびそれ以外の Java 型を使用できます。MTOM/XOP 形式の添付ファイルで使用できる Java 型と送受信できるデータを次の表に示します。

表 30–11　MTOM/XOP 形式の添付ファイルで使用できる Java 型と送受信できるデータ

項番	Java 型	送受信できるデータ
1	byte[]	すべてのデータの送受信ができる。
2	javax.activation.DataHandler	
3	java.awt.Image	イメージデータの送受信ができる。
4	javax.xml.transform.Source	XML ファイルの送受信ができる。

30.7.1　送信するデータごとの Java オブジェクトの生成方法

MTOM/XOP 形式の添付ファイルとして送信するデータに応じた Java オブジェクトの生成方法を説明します。

(1)　存在するテキストファイルを送信する場合

すでに存在しているテキストファイルを送信する場合，byte[]を使用する方法と javax.activation.DataHandler を使用する方法があります。それぞれの手順を説明します。

(a)　byte[]を使用する方法

byte[]を使用して，すでに存在するテキストファイルを送信する場合の手順を次に示します。

1. java.io.FileInputStream オブジェクトを生成します。

   送信する添付ファイルデータのファイルパスを引数に指定して，java.io.FileInputStream オブジェクトを生成します。

```
java.io.FileInputStream fileInputStream =
    new java.io.FileInputStream(”sample.txt”);
```

2. byte[]を生成します。

   java.io.ByteArrayOutputStream オブジェクトに java.io.FileInputStream オブジェクトから読み込んだバイトデータを書き込みます。

   その後，java.io.ByteArrayOutputStream クラスの toByteArray()メソッドを使用して，byte[]を生成します。

```
java.io.ByteArrayOutputStream byteArrayOutputStream =
    new java.io.ByteArrayOutputStream();

int i = 0;
while ((i = fileInputStream.read()) != -1) {
    byteArrayOutputStream.write(i);
}

byte[] bytes = byteArrayOutputStream.toByteArray();
```

(b) javax.activation.DataHandler を使用する方法

javax.activation.DataHandler を使用して，すでに存在するファイルを添付して送信する場合の手順を次に示します。

1. javax.activation.DataSource オブジェクトを生成します。

   送信する添付ファイルデータのファイルパスを引数に指定し，javax.activation.FileDataSource オブジェクトを生成します。

```
javax.activation.FileDataSource fileDataSource =
    new javax.activation.FileDataSource("sample.txt");
```

2. javax.activation.DataHandler オブジェクトを生成します。

   javax.activation.FileDataSource オブジェクトを引数に指定し，javax.activation.DataHandler オブジェクトを生成します。

```
javax.activation.DataHandler dataHandler =
    new javax.activation.DataHandler(fileDataSource);
```

(2) 存在するイメージファイルを送信する場合

すでに存在しているイメージファイルを送信する場合，byte[]を使用する方法，javax.activation.DataHandler を使用する方法，および java.awt.Image を使用する方法があります。それぞれの手順を説明します。

(a) byte[]を使用する方法

byte[]を使用して，すでに存在するイメージファイルを送信する場合の手順を示します。

1. java.io.FileInputStream オブジェクトを生成します。

   送信する添付ファイルデータのファイルパスを引数に指定し，java.io.FileInputStream オブジェクトを生成します。

```
java.io.FileInputStream fileInputStream =
    new java.io.FileInputStream("sample.png");
```

2. byte[]を生成します。

java.io.ByteArrayOutputStream オブジェクトに java.io.FileInputStream オブジェクトから読み込んだバイトデータを書き込みます。

その後，java.io.ByteArrayOutputStream クラスの toByteArray()メソッドを使用して，byte[]を生成します。

```
java.io.ByteArrayOutputStream byteArrayOutputStream =
    new java.io.ByteArrayOutputStream();

int i = 0;
while ((i = fileInputStream.read()) != -1) {
    byteArrayOutputStream.write(i);
}

byte[] bytes = byteArrayOutputStream.toByteArray();
```

(b) javax.activation.DataHandler を使用する方法

javax.activation.DataHandler を使用して，すでに存在するイメージファイルを送信する場合の手順を示します。

1. javax.activation.DataSource オブジェクトを生成します。

   送信する添付ファイルデータのファイルパスを引数に指定し，javax.activation.FileDataSource オブジェクトを生成します。

```
javax.activation.FileDataSource fileDataSource =
    new javax.activation.FileDataSource("sample.png");
```

2. javax.activation.DataHandler オブジェクトを生成します。

   javax.activation.FileDataSource オブジェクトを引数に指定し，javax.activation.DataHandler オブジェクトを生成します。

```
javax.activation.DataHandler dataHandler =
    new javax.activation.DataHandler(fileDataSource);
```

(c) java.awt.Image を使用する方法

java.awt.Image をすでに存在するイメージファイルを送信する場合の手順を示します。

1. java.awt.Image オブジェクトを生成します。

   java.awt.Toolkit クラスにある createImage(String)メソッドの引数に送信する添付ファイルデータのファイルパスを指定し，java.awt.Image オブジェクトを生成します。

```
java.awt.Image image =
    Toolkit.getDefaultToolkit().createImage("sample.png");
```

(3) 存在する XML ファイルを送信する場合

すでに存在している XML ファイルを送信する場合，byte[]を使用する方法，javax.activation.DataHandler を使用する方法，および javax.xml.transform.Source を使用する方法があります。それぞれの手順を説明します。

(a) byte[]を使用する方法

byte[]を使用して，すでに存在する XML ファイルを送信する場合の手順を示します。

1. java.io.FileInputStream オブジェクトを生成します。

   送信する添付ファイルデータのファイルパスを引数に指定し，java.io.FileInputStream オブジェクトを生成する。

```
java.io.FileInputStream fileInputStream =
    new java.io.FileInputStream("sample.xml");
```

2. byte[]を生成します。

   java.io.ByteArrayOutputStream オブジェクトに java.io.FileInputStream オブジェクトから読み込んだバイトデータを書き込みます。

   その後，java.io.ByteArrayOutputStream クラスの toByteArray()メソッドを使用して，byte[]を生成します。

```
java.io.ByteArrayOutputStream byteArrayOutputStream =
    new java.io.ByteArrayOutputStream();

int i = 0;
while ((i = fileInputStream.read()) != -1) {
    byteArrayOutputStream.write(i);
}

byte[] bytes = byteArrayOutputStream.toByteArray();
```

(b) javax.activation.DataHandler を使用する方法

javax.activation.DataHandler を使用して，すでに存在する XML ファイルを送信する場合の手順を示します。

1. javax.activation.DataSource オブジェクトを生成します。

   送信する添付ファイルデータのファイルパスを引数に指定し，javax.activation.FileDataSource オブジェクトを生成します。

```
javax.activation.FileDataSource fileDataSource =
    new javax.activation.FileDataSource("sample.xml");
```

2. javax.activation.DataHandler オブジェクトを生成します。

   javax.activation.FileDataSource オブジェクトを引数に指定し，javax.activation.DataHandler オブジェクトを生成します。

```
javax.activation.DataHandler dataHandler =
    new javax.activation.DataHandler(fileDataSource);
```

(c) javax.xml.transform.Source を使用する方法

javax.xml.transform.Source を使用して，すでに存在する XML ファイルを送信する場合の手順を示します。

1. javax.xml.transform.Source オブジェクトを生成します。

   送信する添付ファイルデータのファイルパスを引数に指定し，
   javax.xml.transform.stream.StreamSource オブジェクトを生成します。

```
javax.xml.transform.stream.StreamSource streamSource =
    new javax.xml.transform.stream.StreamSource("sample.xml");
```

(4) java.lang.String オブジェクトを送信する場合

java.lang.String オブジェクトを送信する場合 byte[]を使用する方法，および
javax.activation.DataHandler を使用する方法があります。それぞれの手順を説明します。

(a) byte[]を使用する方法

byte[]を使用して，java.lang.String オブジェクトを送信する場合の手順を示します。

1. java.lang.String オブジェクトを生成します。

```
java.lang.String str =
    new java.lang.String("あいうえお");
```

2. byte[]を生成します。

   java.lang.String クラスの getBytes()メソッドを使用して，byte[]を生成します。

```
byte[] bytes =
    str.getBytes();
```

(b) javax.activation.DataHandler を使用する方法

javax.activation.DataHandler を使用して，java.lang.String オブジェクトを送信する場合の手順を示します。

1. java.lang.String オブジェクトを生成します。

```
java.lang.String str =
    new java.lang.String("あいうえお");
```

2. javax.activation.DataHandler オブジェクトを生成します。

   java.lang.String オブジェクトと MIME タイプを引数に指定し，javax.activation.DataHandler オブジェクトを生成します。

```
javax.activation.DataHandler dataHandler =
    new javax.activation.DataHandler(str, "text/plain; charset=UTF-8");
```

(5)　javax.activation.DataHandler オブジェクト生成時の注意事項

MTOM/XOP 仕様形式の添付ファイルで，テキストファイル，XML ファイル，または java.lang.String オブジェクト（文字列）を javax.activation.DataHandler オブジェクトとして送信する場合，wsi:swaRef 形式の添付ファイルと同様に javax.activation.DataHandler クラスの DataHandler（Object, String）コンストラクタを使用することで，オブジェクトに含まれる文字の文字コードを指定できます。

DataHandler（Object, String）コンストラクタを使用した文字コードの指定方法などの詳細は，「28.5.1(4)　javax.activation.DataHandler オブジェクト生成時の注意事項」を参照してください。

30.7.2　受信したデータの取得方法

MTOM/XOP 仕様形式の添付ファイルで受信したデータが javax.activation.DataHandler オブジェクト以外（byte[]，java.awt.Image オブジェクト，および javax.xml.transform.Source インスタンス）の場合，JAXB が受信したデータを適切な Java オブジェクトに変換するため，アプリケーションで変換する必要はありません。

受信したデータが javax.activation.DataHandler オブジェクトの場合，wsi:swaRef 形式の添付ファイルと同様の方法で，受信したデータを取得できます。

wsi:swaRef 形式の添付ファイルの取得方法については，「28.5.2　添付ファイルデータを取得する方法（wsi:swaRef 形式）」を参照してください。

(1)　javax.activation.DataHandler オブジェクト取得時の注意事項

MTOM/XOP 仕様形式の添付ファイルを含む MIME Multipart/Related 構造の SOAP メッセージを受信すると，ストリーミングされた MTOM/XOP 仕様形式の添付ファイルとして操作できます。MTOM/XOP 仕様形式の添付ファイルの受信側では，javax.activation.DataHandler オブジェクトが持つ入力ストリームからすべてのデータを読み込まないと受信処理が完了しないため，送信側の送信処理は受信側の受信処理が完了するまで待ち状態が続きます。

この状態を解消するためには，javax.activation.DataHandler オブジェクトが持つ java.io.InputStream オブジェクトからすべてのデータを読み込むか，javax.activation.DataHandler クラスの writeTo(java.io.OutputStream)メソッドを使用して入力ストリームのデータを出力ストリームに書き込む必要があります。

(2)　javax.xml.transform.Source オブジェクト取得時の注意事項

MTOM/XOP 仕様形式の添付ファイルで受信したデータが javax.xml.transform.Source オブジェクトの場合，ストリーミングされた MTOM/XOP 仕様形式の添付ファイルとして操作できます。

javax.xml.transform.Source オブジェクトは，JAXB が javax.xml.transform.stream.StreamSource オブジェクトに変換します。

MTOM/XOP 仕様形式の添付ファイルの受信側では，javax.xml.transform.stream.StreamSource オブジェクトが持つ入力ストリームからすべてのデータを読み込まないと受信処理が完了しないため，送信側の送信処理は受信側の受信処理が完了するまで待ち状態が続きます。

この状態を解消するためには，javax.xml.transform.stream.StreamSource オブジェクトが持つ java.io.Reader オブジェクトからすべてのデータを読み込む必要があります。

30.8　MIME Multipart/Related 構造の SOAP メッセージの受信

1048576 バイト以上のサイズの MIME ボディを含む MIME Multipart/Related 構造の SOAP メッセージを受信した場合，JAX-WS エンジンは SOAP メッセージに含まれる MIME ボディを一時ファイルに出力します。

MIME Multipart/Related 構造の SOAP メッセージの受信の詳細については，「10.24　MIME Multipart/Related 構造の SOAP メッセージの受信」を参照してください。

31 SEI を起点とした開発の例（MTOM/XOP 仕様形式の添付ファイル使用時）

この章では，添付ファイルを使用して，SEI を起点とした Web サービスを開発する場合の例を説明します。

31.1　開発例の構成（SEI 起点・MTOM/XOP 仕様形式の添付ファイル）

この章で説明する開発例では，SEI を起点とした Web サービスを開発します。開発する Web サービスは，MTOM/XOP 仕様形式の添付ファイルを使用します。

開発する Web サービスの概要および利用する情報について説明します。

開発例の概要

社員番号，顔写真，社員名，所属というユーザ情報を管理し，Web サービスクライアントからの入力に対して，処理結果を返す Web サービスを新規に開発します。

Web サービスクライアントからの要求情報およびサーバからの応答情報を次の表に示します。

表 31-1　Web サービスクライアントからの要求情報

情報名	Java データ型
社員番号	java.lang.String
顔写真	javax.activation.DataHandler

表 31-2　サーバからの応答情報

情報名	Java データ型
登録確認メッセージ	java.lang.String
社員名	java.lang.String
所属	java.lang.String

サーバからの応答情報は，ユーザ定義型クラスの UserData クラスで保持されます。

Web サービス開発時のカレントディレクトリの構成を次の表に示します。

表 31-3　カレントディレクトリの構成（SEI 起点・添付ファイル）

ディレクトリ	説明
c:¥temp¥jaxws¥works¥mtom¥	カレントディレクトリです。
┣server¥	Web サービスの開発で使用します。
┃┣META-INF¥	EAR ファイルの META-INF ディレクトリに対応します。
┃┃┗application.xml	「31.3.4　application.xml を作成する」で作成します。
┃┣src¥	Web サービスのソースファイル（*.java）を格納します。「31.3.1　Web サービス実装クラスを作成する」および「31.3.2　Web サービス実装クラスをコンパイルする」で使用します。
┃┣WEB-INF¥	WAR ファイルの WEB-INF ディレクトリに対応します。

ディレクトリ	説明
┃┃┃┃┃┃┃┃┃┣ web.xml	「31.3.3　web.xml を作成する」で作成します。
┃┃┃┃┃┃┃┃┃┗ classes¥	コンパイルしたクラスファイル（*.class）を格納します。「31.3.2　Web サービス実装クラスをコンパイルする」で使用します。
┃┃┣ mtom_dynamic_generate.ear	「31.3.5　EAR ファイルを作成する」で作成します。
┃┃┗ mtom_dynamic_generate.war	
┗ client¥	Web サービスクライアントの開発で使用します。
┃┃┣ src¥	Web サービスクライアントのソースファイル（*.java）を格納します。「31.5.1　サービスクラスを生成する」および「31.5.2　Web サービスクライアントの実装クラスを作成する」で使用します。
┃┃┣ classes¥	コンパイルしたクラスファイル（*.class）を格納します。「31.5.3　Web サービスクライアントの実装クラスをコンパイルする」で使用します。
┃┃┣ portrait.png	Web サービスクライアントで使用する PNG ファイルで使用します。
┃┃┣ usrconf.cfg	「31.6.1　Java アプリケーション用オプション定義ファイルを作成する」で作成します。
┃┃┗ usrconf.properties	「31.6.2　Java アプリケーション用ユーザプロパティファイルを作成する」で作成します。

カレントディレクトリのパスは，開発する環境に合わせて変更してください。

なお，以降の説明では，この表に示すディレクトリおよびファイル名を使用します。コマンド実行例や Java ソースなどで背景色付きの太字になっている部分は，この例で使用する指定値や生成される値を示します。構築する環境に合わせて読み替えてください。

また，この章で説明する開発例では，Web サービスと Web サービスクライアントを同じ環境で開発しますが，別の環境で開発することもできます。別の環境で開発する場合は，それぞれの環境に合わせて，カレントディレクトリのパスを読み替えてください。

31.2　開発例の流れ（SEI 起点・MTOM/XOP 仕様形式の添付ファイル）

この章で説明する開発例では，次の流れで開発および実行します。

Web サービスの開発

1. Web サービス実装クラスを作成する（31.3.1）
2. Web サービス実装クラスをコンパイルする（31.3.2）
3. web.xml を作成する（31.3.3）
4. application.xml を作成する（31.3.4）
5. EAR ファイルを作成する（31.3.5）

デプロイと開始

1. EAR ファイルをデプロイする（31.4.1）
2. Web サービスを開始する（31.4.2）

Web サービスクライアントの開発

1. cjwsimport コマンドを実行し，サービスクラスを生成する（31.5.1）
2. Web サービスクライアントの実装クラスを作成する（31.5.2）
3. Web サービスクライアントの実装クラスをコンパイルする（31.5.3）

Web サービスの実行

1. Java アプリケーション用オプション定義ファイルを作成する（31.6.1）
2. Java アプリケーション用ユーザプロパティファイルを作成する（31.6.2）
3. Web サービスクライアントを実行する（31.6.3）

31.3 Web サービスの開発例（SEI 起点・MTOM/XOP 仕様形式の添付ファイル）

ここでは，SEI を起点とした場合の Web サービス（添付ファイルを使用）の開発例を説明します。

31.3.1 Web サービス実装クラスを作成する

Web サービスの処理を記述した Web サービス実装クラスを作成します。ここでは，受け取った要求メッセージの内容を計算して，応答メッセージを返す Web サービス実装クラスを作成します。

Web サービス実装クラスの作成例を次に示します。

```
package com.sample;

import java.awt.Image;
import javax.jws.WebService;
import javax.xml.ws.soap.MTOM;
import javax.xml.bind.annotation.XmlMimeType;

@MTOM
@WebService(serviceName="UserInfoService",targetNamespace="http://sample.com")
public class UserInfoImpl {

    public UserData getUserData(String in0, @XmlMimeType("image/png")Image in1)
        throws UserInfoException {

        //社員情報への顔写真の登録処理
        ・・・・

        UserData userdata = new UserData();
        //登録した社員の名前と所属を設定
        if (in0.equals("1")) {
            userdata.setName("Sato Taro");
            userdata.setSection("The personnel section");
        } if ( ・・・・) {
            ・・・・・
        } ・・・・

        //登録確認メッセージを設定
        if (in1 == null) {
            userdata.setMessage("Failure(no image).");
        } else {
            userdata.setMessage("Success.");
        }
        return userdata;
}
}
```

作成した UserInfoImpl.java は，UTF-8 形式で
c:¥temp¥jaxws¥works¥mtom¥server¥src¥com¥sample¥ディレクトリに保存します。

また，com.sample.UserInfoImpl で使用しているユーザ定義型クラス com.sample.UserData を作成します。通常，ユーザ定義型クラスの作成は任意ですが，ここではユーザ定義型クラスを作成します。

ユーザ定義型クラスの作成例を次に示します。

```
package com.sample;

import java.lang.String;

public class UserData {

    private String message;
    private String name;
    private String section;

    public UserData() {
    }

    public String getMessage() {
        return this.message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSection() {
        return this.section;
    }

    public void setSection(String section) {
        this.section = section;
    }
}
```

作成した UserData.java は，UTF-8 形式で，
c:¥temp¥jaxws¥works¥mtom¥server¥src¥com¥sample¥ディレクトリに保存します。

また com.sample.UserInfoImpl でスローしている例外クラス com.sample.UserInfoException を作成します。通常，例外クラスの作成は任意ですが，ここでは例外クラスを作成します。

例外クラスの作成例を次に示します。

```
package com.sample;

import java.lang.Exception;
import java.lang.String;
```

```
public class UserInfoException extends Exception {

    String detail;

    public UserInfoException(String message, String detail) {
        super(message);
        this.detail = detail;
    }

    public String getDetail() {
        return detail;
    }
}
```

作成した UserInfoException.java は，UTF-8 形式で c:¥temp¥jaxws¥works¥mtom¥server¥src¥com¥sample¥ディレクトリに保存します。

31.3.2　Web サービス実装クラスをコンパイルする

javac コマンドを実行して，Web サービス実装クラスをコンパイルします。javac コマンドについては，JDK のドキュメントを参照してください。

javac コマンドの実行例を次に示します。添付ファイル機能を使用した Java プログラムを javac コマンドでコンパイルする場合，javac コマンドの引数に"--add-modules=java.activation"を指定してください。

```
> cd c:¥temp¥jaxws¥works¥mtom¥server¥
> mkdir WEB-INF¥classes¥
> javac -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥javaee¥1100¥lib¥javaee-api.jar;%COSMINEXUS_HOME%¥CC¥client¥lib¥HiEJBClientStatic.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxp.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar"
 -d WEB-INF¥classes¥ -s src src¥com¥sample¥UserInfoImpl.java src¥com¥sample¥UserData.java src¥com¥sample¥UserInfoException.java
```

javac コマンドが正常に終了すると，コンパイルしたクラスが，c:¥temp¥jaxws¥works¥mtom¥server¥WEB-INF¥classes¥com¥sample¥ディレクトリに出力されます。なお，コンパイルした Web サービス実装クラスに対して hwsgen コマンドを実行すると，事前にエラーチェックができます。hwsgen コマンドについては，「14.1.2　hwsgen コマンド」を，エラーチェックについては，「10.23.1　hwsgen コマンドによるエラーチェックについて」を参照してください。

31.3.3　web.xml を作成する

WAR ファイルの構成要素として必要な web.xml を作成します。

web.xml の作成例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/we
b-app_3_0.xsd">
  <description>Sample web service &amp;quot;mtom_dynamic_generate&amp;quot;</description>
  <display-name>Sample_web_service_mtom_dynamic_generate</display-name>
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/UserInfoService</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

バージョン 2.5 の web.xml を作成する場合は，web-app 要素の version 属性を"2.5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"にしてください。

作成した web.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥mtom¥server¥WEB-INF¥ディレクトリに保存します。web.xml の設定項目については，「3.4　web.xml の作成」を参照してください。

31.3.4　application.xml を作成する

EAR ファイルの構成要素として必要な application.xml を作成します。

application.xml の作成例を次に示します。なお，Web サービスとして application.xml に設定する項目はありません。

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/ap
plication_6.xsd">

  <description>Sample application &amp;quot;mtom_dynamic_generate&amp;quot;</description>
```

```
  <display-name>Sample_application_mtom_dynamic_generate</display-name>
  <module>
    <web>
      <web-uri>mtom_dynamic_generate.war</web-uri>
      <context-root>mtom_dynamic_generate</context-root>
    </web>
  </module>
</application>
```

バージョン 5 の application.xml を作成する場合は，application 要素の version 属性を"5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/application_5.xsd"にしてください。

作成した application.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥mtom¥server¥META-INF¥ディレクトリに保存します。application.xml を作成するときの注意事項については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「5.2.2　application.xml 編集時の注意事項」を参照してください。

31.3.5　EAR ファイルを作成する

jar コマンドを使用して，これまでに作成したファイルを含めた EAR ファイルを作成します。

EAR ファイルの作成例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥mtom¥server¥
> jar cvf mtom_dynamic_generate.war .¥WEB-INF
> jar cvf mtom_dynamic_generate.ear .¥mtom_dynamic_generate.war .¥META-INF¥application.xml
```

正常に終了すると，c:¥temp¥jaxws¥works¥mtom_dynamic_generate¥server¥ディレクトリに mtom.ear が作成されます。

jar コマンドについては，JDK のドキュメントを参照してください。

31.4　デプロイと開始の例（SEI 起点・MTOM/XOP 仕様形式の添付ファイル）

ここでは，添付ファイルを使用して，SEI を起点とした場合のデプロイと開始の例を説明します。

31.4.1　EAR ファイルをデプロイする

cjimportapp コマンドを使用して，作成した EAR ファイルを J2EE サーバにデプロイします。

デプロイの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥mtom¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjimportapp" jaxwsserver -nameserver corbaname::testserver
:900 -f mtom_dynamic_generate.ear
```

cjimportapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションをデプロイ（インポート）する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.3　J2EE アプリケーションのインポート」を参照してください。

31.4.2　Web サービスを開始する

cjstartapp コマンドを使用して，Web サービスを開始します。

Web サービスを開始する例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥mtom¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjstartapp" jaxwsserver -nameserver corbaname::testserver:
900 -name Sample_application_mtom_dynamic_generate
```

cjstartapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartapp（J2EE アプリケーションの開始）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションを開始する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.1　J2EE アプリケーションの開始」を参照してください。

31.5　Web サービスクライアントの開発例（SEI 起点・MTOM/XOP 仕様形式の添付ファイル）

ここでは，添付ファイルを使用して，SEI を起点とした場合の Web サービスクライアントの開発例を説明します。

31.5.1　サービスクラスを生成する

cjwsimport コマンドを実行すると，サービスクラスなど Web サービスクライアントの開発に必要な Java ソースが生成されます。cjwsimport コマンドについては，「14.1.1　cjwsimport コマンド」を参照してください。

cjwsimport コマンドの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥mtom¥client¥
> mkdir src¥
> mkdir classes¥
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" -s src -d classes http://webhost:8085/mtom_dynamic_generate/UserInfoService?wsdl
```

正常に終了すると，c:¥temp¥jaxws¥works¥mtom¥client¥src¥com¥sample¥ディレクトリに Java ソースが生成されます。

生成物の一覧を次の表に示します。

表 31-4　サービスクラス生成時の生成物（SEI 起点・添付ファイル）

ファイル名	説明
GetUserData.java	WSDL 定義の要求メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
GetUserDataResponse.java	WSDL 定義の応答メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
ObjectFactory.java	JAXB 2.2 仕様の ObjectFactory クラスです。
package-info.java	package-info.java ファイルです。
UserData.java	UserData に対応する JavaBean クラスです。
UserInfoImpl.java	WSDL 定義の「サービス」に対応する SEI です。
UserInfoService.java	サービスクラスです。
UserInfoException.java	UserInfoException に対応する JavaBean クラスです。
UserInfoException_Exception.java	フォルト bean のラッパ例外クラスです。

31.5.2　Web サービスクライアントの実装クラスを作成する

Web サービスを利用する Web サービスクライアントの実装クラスを作成します。

Web サービスに対して 1 回の呼び出しをする Web サービスクライアントの作成例を次に示します。

```
package com.sample.client;

import java.awt.Image;
import java.io.File;

import javax.imageio.ImageIO;
import javax.xml.ws.soap.MTOMFeature;

import com.sample.UserInfoImpl;
import com.sample.UserData;
import com.sample.UserInfoService;
import com.sample.UserInfoException_Exception;

public class TestClient {

    public static void main( String[] args ) {
        try {
            // Imageオブジェクト生成
            File imageFile = new File("portrait.png");
            if (!imageFile.exists()) {
                throw new RuntimeException("Cannot find the file ¥"portrait.png¥".");
            }
            Image image = ImageIO.read(imageFile);

            UserInfoService service = new UserInfoService();
            UserInfoImpl port = service.getUserInfoImplPort(new MTOMFeature());

            UserData userdata = port.getUserData("1", image);
            System.out.print("[RESULT] " + userdata.getMessage());
            System.out.println(" Name:" + userdata.getName()
                               + ", Section:" + userdata.getSection());
        } catch(UserInfoException_Exception e) {
            e.printStackTrace();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

作成した TestClient.java は，UTF-8 形式で
c:¥temp¥jaxws¥works¥mtom¥client¥src¥com¥sample¥client¥ディレクトリに保存します。

31.5.3　Web サービスクライアントの実装クラスをコンパイルする

javac コマンドを使用して，作成した Web サービスクライアントをコンパイルします。

コンパイルの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥mtom¥client¥
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥javaee¥1100¥lib¥javaee-api.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;.¥classes" -d .¥classes src¥com¥sample¥client¥TestClient.java
```

正常に終了すると，c:¥temp¥jaxws¥works¥mtom¥client¥classes¥com¥sample¥client¥ディレクトリに，TestClient.class が生成されます。

javac コマンドについては，JDK のドキュメントを参照してください。

31.6 Web サービスの実行例（SEI 起点・MTOM/XOP 仕様形式の添付ファイル）

ここでは，添付ファイルを使用して，SEI を起点とした場合の Web サービスクライアントの実行例を説明します。

31.6.1 Java アプリケーション用オプション定義ファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用オプション定義ファイル（usrconf.cfg）を作成します。

Java アプリケーション用オプション定義ファイルの作成例を次に示します。

```
add.class.path=<Application Serverのインストールディレクトリ>¥jaxws¥lib¥cjjaxws.jar
add.class.path=.¥classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home=<Application Serverのインストールディレクトリ>
```

<Application Server のインストールディレクトリ>の部分は，Application Server をインストールしているパスを絶対パスで指定します。

作成した Java アプリケーション用オプション定義ファイルは，c:¥temp¥jaxws¥works¥mtom¥client¥ディレクトリに保存します。Java アプリケーション用オプション定義ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.1　usrconf.cfg（Java アプリケーション用オプション定義ファイル)」を参照してください。

31.6.2 Java アプリケーション用ユーザプロパティファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用ユーザプロパティファイルを作成します。

ここでは特に設定を変更しないため，c:¥temp¥jaxws¥works¥mtom¥client¥ディレクトリに usrconf.properties という名称の空のファイルを作成します。Java アプリケーション用ユーザプロパティファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.2　usrconf.properties（Java アプリケーション用ユーザプロパティファイル)」を参照してください。

31.6.3 Web サービスクライアントを実行する

cjclstartap コマンドを使用して，Web サービスクライアントを実行します。

Web サービスクライアントの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥mtom¥client¥
> "%COSMINEXUS_HOME%¥CC¥client¥bin¥cjclstartap"  com.sample.client.TestClient
```

正常に終了すると，Web サービスクライアントの実行結果が表示されます。実行結果の表示例を次に示します。

```
KDJE40053-I The cjclstartap command will now start. (directory for the user definition file
= c:¥temp¥jaxws¥works¥mtom¥client, PID = 2636)
[RESULT] Success. Name:Sato Taro, Section:The personnel section
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

イタリック体になっている個所は，実行したタイミングや環境によって変わります。

cjclstartap コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjclstartap（Java アプリケーションの開始)」を参照してください。

32 ストリーミング

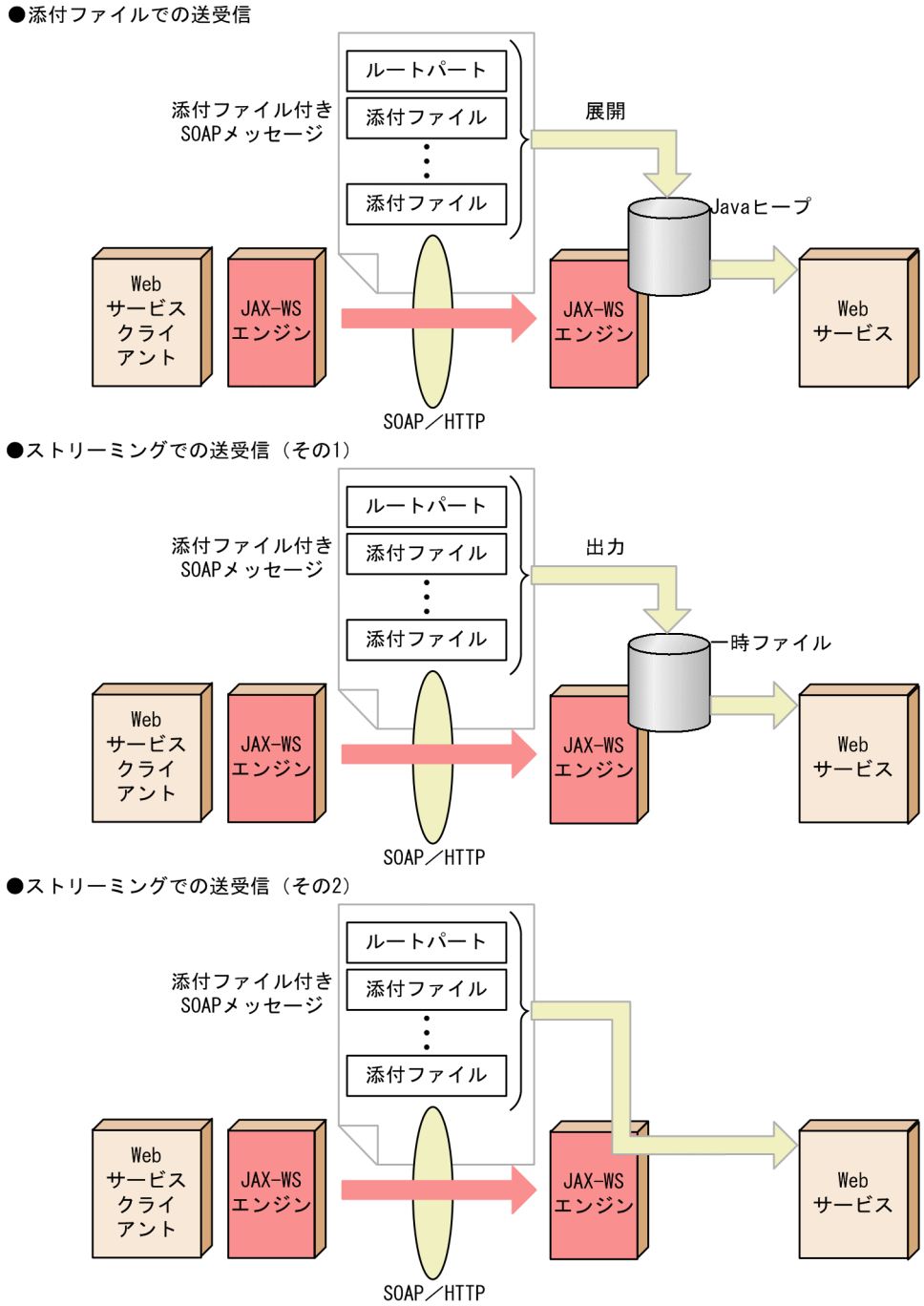
この章では，Application Server のストリーミングの概要，使用方法および使用時の設定について説明します。

32.1　ストリーミングとは

ストリーミングとは，添付ファイル機能で添付ファイルを含む MIME Multipart/Related 構造の SOAP メッセージを受信する際に，SOAP メッセージに含まれる大きいサイズの MIME ボディをメモリ上に展開しないで処理することで，Java ヒープサイズの制約を受けることなく大容量の添付ファイルを含む SOAP メッセージを受信する機能です。ストリーミングは，受信した添付ファイルを含む SOAP メッセージを javax.activation.DataHandler クラスにマッピングするときに使用できます。

ストリーミングを使用したバイナリデータの送受信を次の図に示します。

図 32–1　ストリーミングを使用したバイナリデータの送受信



ストリーミングは，MTOM/XOP 仕様形式の添付ファイルを使用する場合にだけ使用できます。
wsi:swaRef 形式の添付ファイルでは使用できません。

32.2　ストリーミングの使用方法

ストリーミングを使用するには，添付ファイルを含む MIME Multipart/Related 構造の SOAP メッセージを受信する側に設定する必要があります。ここでは Web サービス側および Web サービスクライアント側の使用方法を説明します。

32.2.1　Web サービス側

Web サービスでストリーミングを使用するには，Web サービス実装クラスに com.sun.xml.ws.developer.StreamingAttachment アノテーションをアノテートします。com.sun.xml.ws.developer.StreamingAttachment アノテーションについては，「16.2.2 com.sun.xml.ws.developer.StreamingAttachment アノテーション」を参照してください。

ストリーミングを使用した Web サービス実装クラスの例を次に示します。この例では，"C:/TMP"ディレクトリをストリーミングによる一時ファイルの出力先に指定し，添付ファイルを含む SOAP メッセージの詳細な解析をして，50,000 バイト以上の MIME ボディを一時ファイルとして出力します。

```
package com.sample;

 . . . . . .

@MTOM
@StreamingAttachment(dir="C:/TMP", parseEagerly=true, memoryThreshold=50000L)
@BindingType( . . . )
public class UserInfoImpl implements UserInfo {

    public DataHandler getUserInfo(DataHandler dataHandler)
        throws UserDefinedException {
        if (dataHandler instanceof StreamingDataHandler) {
            StreamingDataHandler sdh = null;
            try {
                sdh = (StreamingDataHandler)dataHandler;
                 . . . . . .
            } finally {
                try {
                    if (sdh != null) {
                        sdh.close();
                    }
                } catch(Exception ex) {
                     . . . . . .
                }
            }
        }
         . . . . . .
    }
}
```

Web サービス側でストリーミングを使用し，次の条件をすべて満たすリクエストメッセージを受信した場合，レスポンスメッセージに含まれるデータが Base64 形式のデータになります。

- リクエストメッセージの HTTP ヘッダにある Accept フィールドに application/xop+xml がない。
- リクエストメッセージのルートパートにある Content-Type に application/xop+xml がない。

32.2.2 Web サービスクライアント側

Web サービスクライアントでストリーミングを使用するには，SEI を取得する際に com.sun.xml.ws.developer.StreamingAttachmentFeature クラスを設定します。

com.sun.xml.ws.developer.StreamingAttachmentFeature クラスについては，「19.2.4(1) com.sun.xml.ws.developer.StreamingAttachmentFeature クラス」を参照してください。

ストリーミングを使用した Web サービスクライアントの例を次に示します。この例では，"C:/TMP"ディレクトリをストリーミングによる一時ファイルの出力先に指定し，添付ファイルを含む SOAP メッセージの詳細な解析をし，50,000 バイト以上の MIME ボディを一時ファイルとして出力します。

```
package com.sample;

. . . . . .

public class TestClient {

    public static void main(String[] args) {
        try {
            File portrait = new File(”portrait.png”);
            FileDataSource fileDataSource = new FileDataSource(portrait);
            DataHandler dataHandler = new DataHandler(fileDataSource);

            MTOMFeature mtomFeature = new MTOMFeature();
            StreamingAttachmentFeature streamingAttachmentFeature = new StreamingAttachmentF
eature(”C:/TMP”, true, 50000L);

            UserInfoService service = new UserInfoService();
            UserInfoImpl port = service.getUserInfoImplPort(mtomFeature, streamingAttachment
Feature);
            DataHandler userData = port.getUserInfo(dataHandler);
            if (userData instanceof StreamingDataHandler) {
                StreamingDataHandler sdh = null;
                try {
                    sdh = (StreamingDataHandler)userData;
                    sdh.moveTo(file);
                    . . . . . .
                } finally {
                    try {
                        if (sdh != null) {
                            sdh.close();
                        }
                    } catch(Exception ex) {
                        . . . . . .
                    }
                }
            }
        } catch (Exception e) {
```

```
            e.printStackTrace();
        }
    }
}
```

32.2.3　parseEagerly による変化

ストリーミングは，com.sun.xml.ws.developer.StreamingAttachment アノテーションの要素や com.sun.xml.ws.developer.StreamingAttachmentFeature クラスの引数にある parseEagerly に指定した値で，添付ファイルを含む SOAP メッセージを解析するタイミング，解析の結果，および SOAP メッセージに異常があるときに発生する例外が変わります。

parseEagerly の値と変化を次の表に示します。

表 32–1　parseEagerly の値と変化

項番	parseEagerly の値	SOAP メッセージを解析するタイミング	SOAP メッセージに異常があるときに発生する例外
1	true	添付ファイルを含む SOAP メッセージをアンマーシャルしたとき	org.jvnet.mimepull.MIMEParsingException
2	false	ストリーミングされた添付ファイルを操作したとき	java.io.IOException

parseEagerly の値は，添付ファイルを含む SOAP メッセージの異常をユーザアプリケーションではなく，アンマーシャル時に検知する場合には，true を指定します。

32.2.4　ストリーミングされた添付ファイルの操作

ストリーミングを使用しているときに添付ファイルを含む MIME Multipart/Related 構造の SOAP メッセージを受信すると，JAX-WS では受信した添付ファイルを javax.activation.DataHandler クラスではなく，com.sun.xml.ws.developer.StreamingDataHandler クラスにマッピングし，ストリーミングされた添付ファイルとして操作できます。

com.sun.xml.ws.developer.StreamingDataHandler クラスについては，「19.2.4(2) com.sun.xml.ws.developer.StreamingDataHandler クラス」を参照してください。

ストリーミングされた添付ファイルを操作する例を次に示します。この例では，ストリーミングされた添付ファイルを"C:/portrait.png"として別名で出力します。

```
package com.sample;

. . . . . .

@MTOM
```

```
@StreamingAttachment(dir="C:/TMP", parseEagerly=true, memoryThreshold=50000L)
@BindingType(・・・)
public class UserInfoImpl implements UserInfo {

    public DataHandler getUserInfo(DataHandler dataHandler)
        throws UserDefinedException {
        if (dataHandler instanceof StreamingDataHandler) {
            File file = new File("C:/portrait.png");
            StreamingDataHandler sdh = null;
            try {
                sdh = (StreamingDataHandler)dataHandler;
                sdh.moveTo(file);
                ・・・・・・
            } finally {
                try {
                    if (sdh != null) {
                        sdh.close();
                    }
                } catch(Exception ex) {
                    ・・・・・・
                }
            }
        }
    }
}
```

(1) ストリーミングされた添付ファイルの操作時の注意事項

ストリーミングされた添付ファイルを操作する場合の注意事項を次に示します。

- ストリーミングを使用する場合，instanceof 演算子を用いて com.sun.xml.ws.developer.StreamingDataHandler クラスを判別する必要があります。
- 一時ファイルに出力した添付ファイルは使用有無に関係なく，必ず com.sun.xml.ws.developer.StreamingDataHandler#close()メソッドでクローズする必要があります。クローズしない場合，JAX-WS は一時ファイルを消去しません。
- com.sun.xml.ws.developer.StreamingDataHandler#readOnce()メソッドおよび com.sun.xml.ws.developer.StreamingDataHandler#moveTo(File)メソッドを呼び出したあとは，com.sun.xml.ws.developer.StreamingDataHandler#close()メソッドだけ呼び出すことができます。com.sun.xml.ws.developer.StreamingDataHandler#close()メソッド以外のメソッドを呼び出した場合，動作は保証されません。
- com.sun.xml.ws.developer.StreamingDataHandler#readOnce()メソッドを呼び出した場合，添付ファイルが必要なくなった時点で取得した入力ストリームからデータをすべて読み読み込んだあとに入力ストリームをクローズし，com.sun.xml.ws.developer.StreamingDataHandler#close()メソッドを呼び出す必要があります。

  com.sun.xml.ws.developer.StreamingDataHandler#readOnce()メソッドを呼び出した場合の例を次に示します。

```
package com.sample;
...
```

```
@MTOM
@StreamingAttachment(...)
...
public class UserInfoImpl implements UserInfo {

    public @XmlMimeType("application/octet-stream")
           DataHandler getUserInfo(
               @XmlMimeType("application/octet-stream")
               DataHandler dataHandler)
        throws ... {
        if (dataHandler instanceof StreamingDataHandler) {
            StreamingDataHandler sdh = null;
            try {
                sdh = (StreamingDataHandler)dataHandler;
                ...
                InputStream inputStream = sdh.readOnce();
                ...
                // 入力ストリームから添付ファイルのデータをすべて
                // 読み込む
                while ((inputStream.read()) != -1);

                // 入力ストリームをクローズする
                inputStream.close();
            } finally {
                try {
                    if (sdh != null) {
                        // StreamingDataHandlerクラスのclose()
                        // メソッドを呼び出す
                        sdh.close();
                    }
                } catch(Exception ex) {
                    ...
                }
            }
        }
        ...
    }
}
```

- com.sun.xml.ws.developer.StreamingDataHandler#moveTo(File)メソッドを呼び出した場合，添付ファイルが必要なくなった時点で com.sun.xml.ws.developer.StreamingDataHandler#close()メソッドを呼び出す必要があります。
- com.sun.xml.ws.developer.StreamingDataHandler#moveTo(File)メソッドを呼び出したときに例外が発生した場合，com.sun.xml.ws.developer.StreamingDataHandler#readOnce()メソッドを呼び出して入力ストリームを取得し，その入力ストリームからデータをすべて読み込んだあとに入力ストリームのクローズを行い，com.sun.xml.ws.developer.StreamingDataHandler#close()メソッドを呼び出す必要があります。

  com.sun.xml.ws.developer.StreamingDataHandler#moveTo(File)メソッドを呼び出した場合の例を次に示します。

```
package com.sample;
...
```

```
@MTOM
@StreamingAttachment(...)
...
public class UserInfoImpl implements UserInfo {

    public @XmlMimeType("application/octet-stream")
           DataHandler getUserInfo(
               @XmlMimeType("application/octet-stream")
               DataHandler dataHandler)
        throws ... {
        if (dataHandler instanceof StreamingDataHandler) {
            StreamingDataHandler sdh = null;
            try {
                File file = new File(...);
                sdh = (StreamingDataHandler)dataHandler;
                ...
                sdh.moveTo(file);
            } catch(Exception e)
                try {
                    if (sdh != null) {
                        InputStream inputStream = sdh.readOnce();

                        // 入力ストリームから添付ファイルのデータをすべて
                        // 読み込む
                        while ((inputStream.read()) != -1);

                        // 入力ストリームをクローズする
                        inputStream.close();
                    }
                } catch(Exception ex) {
                    ...
                }
            } finally {
                try {
                    if (sdh != null) {
                        // StreamingDataHandlerクラスのclose()
                        // メソッドを呼び出す
                        sdh.close();
                    }
                } catch(Exception ex) {
                    ...
                }
            }
        }
        ...
    }
}
```

- com.sun.xml.ws.developer.StreamingDataHandler#readOnce()メソッドを呼び出していない，かつ com.sun.xml.ws.developer.StreamingDataHandler#moveTo(File)メソッドを呼び出していない場合，添付ファイルが必要なくなった時点で com.sun.xml.ws.developer.StreamingDataHandler#readOnce()メソッドを呼び出して入力ストリームを取得し，その入力ストリームからデータをすべて読み込んだあとに入力ストリームをクローズし，com.sun.xml.ws.developer.StreamingDataHandler#close()メソッドを呼び出す必要があります。

- com.sun.xml.ws.developer.StreamingDataHandler クラスは javax.activation.DataHandler クラスのメソッドのうち，getContentType()メソッドだけ呼び出すことができます。それ以外のメソッドを呼び出した場合，動作は保証されません。
- Web サービスクライアントまたは Web サービス実装クラスで，受信した javax.activation.DataHandler クラスまたは javax.xml.ws.Holder <DataHandler>クラスのストリーミングされた添付ファイルをそのまま使用して送信しないでください。送信する場合，新たに javax.activation.DataHandler オブジェクトを生成して送信してください。

32.3　一時ファイル（ストリーミング）

ストリーミングを使用しているときに添付ファイルを含む MIME Multipart/Related 構造の SOAP メッセージを受信する場合，SOAP メッセージに含まれる MIME ボディをメモリ上ではなく，一時ファイルに出力する場合があります。SOAP メッセージに含まれる MIME ボディをメモリ上に展開するか，一時ファイルに出力するかは，parseEagerly に指定した値，MIME パートの種類，および MIME ボディのサイズによって決まります。

SOAP メッセージに含まれる MIME ボディの出力先を次の表に示します。

表 32–2　SOAP メッセージに含まれる MIME ボディの出力先

項番	MIME パートの種類	parseEagerly に指定した値	しきい値※と MIME ボディサイズの関係	MIME ボディの出力先
1	ルートパート	－	しきい値※≦MIME ボディのサイズ	メモリ上に展開しないで，一時ファイルに出力します。
2			しきい値※＞ MIME ボディのサイズ	一時ファイルに出力しないで，メモリ上に展開します。
3	添付ファイルパート	true	しきい値※≦MIME ボディのサイズ	メモリ上に展開しないで，一時ファイルに出力します。
4			しきい値※＞ MIME ボディのサイズ	一時ファイルに出力しないで，メモリ上に展開します。
5		false	なし	一時ファイルにも出力しないで，メモリ上にも展開しません。

（凡例）

－：指定した値は MIME ボディの出力先に影響しません。

注※

しきい値とは，com.sun.xml.ws.developer.StreamingAttachment アノテーションの memoryThreshold 要素値または com.sun.xml.ws.developer.StreamingAttachmentFeature クラスの memoryThreshold 値です。

SOAP メッセージに含まれる MIME ボディを一時ファイルに出力するかどうかは，各 MIME パートにある MIME ボディについて判定して決定します。受信する添付ファイルを含む MIME Multipart/Related 構造の SOAP メッセージに，ルートパートが一つ，添付ファイルパートが二つ存在する場合，一時ファイルを出力するかどうかの判定は 3 回実行します。

32.3.1　命名規則

命名規則の詳細については，「10.24.1(1)　命名規則」を参照してください。

なお，一時ファイルの出力先を指定しない場合，異なるプロセスの cjclstartap コマンドでストリーミングを使用した Web サービスクライアントを実行する際に，一時ファイルのファイル名が重複することがあ

ります。そのため，com.sun.xml.ws.developer.StreamingAttachmentFeature クラスの dir 引数に一時ファイルの出力先を指定してください。

また，次の場合，一時ファイルの出力先として同じディレクトリを指定すると，一時ファイルのファイル名が重複することがあります。

- 複数の J2EE サーバにストリーミングを使用した Web サービスまたは Web サービスクライアントを配置する場合
- 異なるプロセスの cjclstartap コマンドでストリーミングを使用した Web サービスクライアントを実行する場合
- 上記を組み合わせる場合

そのため，複数の J2EE サーバにストリーミングを使用した Web サービスまたは Web サービスクライアントを配置する場合には J2EE サーバごとに，異なるプロセスの cjclstartap コマンドでストリーミングを使用した Web サービスクライアントを実行する場合にはプロセスごとに一時ファイルの出力先を変える必要があります。

32.3.2　出力と削除

出力と削除の詳細については，「10.24.1(2)　出力と削除」を参照してください。

32.3.3　見積もり方法

見積もり方法の詳細については，「10.24.1(3)　見積もり方法」を参照してください。

33 SEI を起点とした開発の例（ストリーミング使用時）

この章では，ストリーミングを使用して，SEI を起点とした Web サービスを開発する場合の例を説明します。

33.1　開発例の構成（SEI 起点・ストリーミング）

この章で説明する開発例では，SEI を起点とした Web サービスを開発します。開発する Web サービスは，ストリーミングを使用します。なお，ここでは JAX-WS 2.2 仕様 3.3 節で説明されている，暗黙の SEI の形式（明示的な SEI を作成しない形式）の Web サービスを開発します。

開発する Web サービスの概要および利用する情報について説明します。

開発例の概要

社員番号，顔写真，社員名，所属というユーザ情報を管理し，Web サービスクライアントからの入力に対して，処理結果を返す Web サービスを新規に開発します。

Web サービスクライアントからの要求情報およびサーバからの応答情報を次の表に示します。

表 33–1　Web サービスクライアントからの要求情報

情報名	Java データ型
社員番号	java.lang.String
顔写真	javax.activation.DataHandler

表 33–2　サーバからの応答情報

情報名	Java データ型
登録確認メッセージ	java.lang.String
社員名	java.lang.String
所属	java.lang.String

サーバからの応答情報は，ユーザ定義型クラスの UserData クラスで保持されます。

この章で説明する開発例では，次の表に示す構成の Web サービスを開発します。

表 33–3　Web サービスの構成（SEI 起点・ストリーミング）

項番	項目		値
1	デプロイする J2EE サーバの名称		jaxwsserver
2	Web サーバのホスト名とポート番号		webhost:8085
3	ネーミングサーバの URL		corbaname::testserver:900
4	コンテキストルート		streaming_dynamic_generate
5	スタイル		document/literal/wrapped
6	名前空間 URI		http://sample.com
7	ポートタイプ	個数	1
8		ローカル名	UserInfoImpl

項番	項目		値
9	オペレーション	個数	1
10		ローカル名	GetUserData
11	サービス	個数	1
12		ローカル名	UserInfoService
13	ポート	個数	1
14		ローカル名	UserInfoImplPort
15	Web サービス実装クラス		com.sample.UserInfoImpl
16	Web サービス実装クラスで公開するメソッド	個数	1
17		メソッド名	getUserData
18	Web サービス実装内のメソッドでスローする例外	個数	1
19		クラス名	com.sample.UserInfoException

Web サービス開発時のカレントディレクトリの構成を次の表に示します。

表 33–4　カレントディレクトリの構成（SEI 起点・ストリーミング）

ディレクトリ	説明
c:¥temp¥jaxws¥works¥streaming	カレントディレクトリです。
├server¥	Web サービスの開発で使用します。
│├META-INF¥	EAR ファイルの META-INF ディレクトリに対応します。
│└application.xml	「33.3.4　application.xml を作成する」で作成します。
│├src¥	Web サービスのソースファイル（*.java）を格納します。「33.3.1　Web サービス実装クラスを作成する」および「33.3.2　Web サービス実装クラスをコンパイルする」で使用します。
│├WEB-INF¥	WAR ファイルの WEB-INF ディレクトリに対応します。
││├web.xml	「33.3.3　web.xml を作成する」で作成します。
││└classes¥	コンパイルしたクラスファイル（*.class）を格納します。「33.3.2　Web サービス実装クラスをコンパイルする」で使用します。
│├streaming_dynamic_generate.ear	「33.3.5　EAR ファイルを作成する」で作成します。
│└streaming_dynamic_generate.war	

ディレクトリ			説明
└	client¥		Web サービスクライアントの開発で使用します。
	├	src¥	Web サービスクライアントのソースファイル（*.java）を格納します。「33.5.1　サービスクラスを生成する」および「33.5.2　Web サービスクライアントの実装クラスを作成する」で使用します。
	├	classes¥	コンパイルしたクラスファイル（*.class）を格納します。「33.5.3　Web サービスクライアントの実装クラスをコンパイルする」で使用します。
	├	portrait.png	Web サービスクライアントで使用する PNG ファイルで使用します。
	├	usrconf.cfg	「33.6.1　Java アプリケーション用オプション定義ファイルを作成する」で作成します。
	└	usrconf.properties	「33.6.2　Java アプリケーション用ユーザプロパティファイルを作成する」で作成します。

カレントディレクトリのパスは，開発する環境に合わせて変更してください。

なお，以降の説明では，この表に示すディレクトリおよびファイル名を使用します。コマンド実行例や Java ソースなどで背景色付きの太字になっている部分は，この例で使用する指定値や生成される値を示します。構築する環境に合わせて読み替えてください。

また，この章で説明する開発例では，Web サービスと Web サービスクライアントを同じ環境で開発しますが，別の環境で開発することもできます。別の環境で開発する場合は，それぞれの環境に合わせて，カレントディレクトリのパスを読み替えてください。

33.2　開発例の流れ（SEI 起点・ストリーミング）

この章で説明する開発例では，次の流れで開発および実行します。

Web サービスの開発

1. Web サービス実装クラスを作成する（33.3.1）
2. javac コマンドを実行し，Web サービス実装クラスをコンパイルする（33.3.2）
3. web.xml を作成する（33.3.3）
4. application.xml を作成する（33.3.4）
5. EAR ファイルを作成する（33.3.5）

デプロイと開始

1. EAR ファイルをデプロイする（33.4.1）
2. Web サービスを開始する（33.4.2）

Web サービスクライアントの開発

1. cjwsimport コマンドを実行し，サービスクラスを生成する（33.5.1）
2. Web サービスクライアントの実装クラスを作成する（33.5.2）
3. Web サービスクライアントの実装クラスをコンパイルする（33.5.3）

Web サービスの実行

1. Java アプリケーション用オプション定義ファイルを作成する（33.6.1）
2. Java アプリケーション用ユーザプロパティファイルを作成する（33.6.2）
3. Web サービスクライアントを実行する（33.6.3）

33.3　Web サービスの開発例（SEI 起点・ストリーミング）

ここでは，SEI を起点とした場合の Web サービス（ストリーミングを使用）の開発例を説明します。

33.3.1　Web サービス実装クラスを作成する

Web サービスの処理を記述した Web サービス実装クラスを作成します。ここでは，受け取った要求メッセージの内容を計算して，応答メッセージを返す Web サービス実装クラスを作成します。

Web サービス実装クラスの作成例を次に示します。

```
package com.sample;


import javax.activation.DataHandler;
import javax.jws.WebService;
import javax.xml.ws.soap.MTOM;
import javax.xml.bind.annotation.XmlMimeType;
import com.sun.xml.ws.developer.StreamingAttachment;
import com.sun.xml.ws.developer.StreamingDataHandler;

@MTOM
@StreamingAttachment(dir="C:/TMP", parseEagerly=true, memoryThreshold=50000L)
@WebService(serviceName="UserInfoService",targetNamespace="http://sample.com")
public class UserInfoImpl {

    public UserData getUserData(String in0, @XmlMimeType("application/octet-stream")DataHand
ler in1)
        throws UserInfoException {

        if (in1 != null) {
            if (in1 instanceof StreamingDataHandler) {
                StreamingDataHandler sdh = null;
                try {
                    //社員情報への顔写真の登録処理
                    sdh = (StreamingDataHandler)in1;
                     . . . .
                } catch(Exception e) {
                    throw new UserInfoException("Exception occurred.", e.getMessage());
                } finally {
                    try {
                        if (sdh != null) {
                            //データのクローズ
                            sdh.close();
                        }
                    } catch(Exception ex) {
                         . . . .
                    }
                }
            }
        }

        UserData userdata = new UserData();
```

```
        //登録した社員の名前と所属を設定
        if (in0.equals("1")) {
            userdata.setName("Sato Taro");
            userdata.setSection("The personnel section");
        } if ( ・・・・) {
             ・・・・・
        } ・・・・

        //登録確認メッセージを設定
        if (in1 == null) {
            userdata.setMessage("Failure(no image).");
        } else {
            userdata.setMessage("Success.");
        }
        return userdata;
    }
}
```

作成した UserInfoImpl.java は，UTF-8 形式で
c:¥temp¥jaxws¥works¥streaming¥server¥src¥com¥sample¥ディレクトリに保存します。

また，com.sample.UserInfoImpl で使用しているユーザ定義型クラス com.sample.UserData を作成します。通常，ユーザ定義型クラスの作成は任意ですが，ここではユーザ定義型クラスを作成します。

ユーザ定義型クラスの作成例を次に示します。

```
package com.sample;

import java.lang.String;

public class UserData {

    private String message;
    private String name;
    private String section;

    public UserData() {
    }

    public String getMessage() {
        return this.message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSection() {
        return this.section;
```

```
    }

    public void setSection(String section) {
        this.section = section;
    }
}
```

作成した UserData.java は，UTF-8 形式で，
c:¥temp¥jaxws¥works¥streaming¥server¥src¥com¥sample¥ディレクトリに保存します。

また com.sample.UserInfoImpl でスローしている例外クラス com.sample.UserInfoException を作成します。通常，例外クラスの作成は任意ですが，ここでは例外クラスを作成します。

例外クラスの作成例を次に示します。

```
package com.sample;

import java.lang.Exception;
import java.lang.String;

public class UserInfoException extends Exception {

    String detail;

    public UserInfoException(String message, String detail) {
        super(message);
        this.detail = detail;
    }

    public String getDetail() {
        return detail;
    }
}
```

作成した UserInfoException.java は，UTF-8 形式で
c:¥temp¥jaxws¥works¥streaming¥server¥src¥com¥sample¥ディレクトリに保存します。

33.3.2　Web サービス実装クラスをコンパイルする

javac コマンドを実行して，Web サービス実装クラスをコンパイルします。javac コマンドについては，JDK のドキュメントを参照してください。

javac コマンドの実行例を次に示します。添付ファイル機能を使用した Java プログラムを javac コマンドでコンパイルする場合，javac コマンドの引数に"--add-modules=java.activation"を指定してください。

```
> cd c:¥temp¥jaxws¥works¥streaming¥server¥
> mkdir WEB-INF¥classes¥
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥ja
vaee¥1100¥lib¥javaee-api.jar;%COSMINEXUS_HOME%¥CC¥client¥lib¥HiEJBClientStatic.jar;%COSMINEX
US_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxp.jar;%COSMINEXUS_HOME%¥jaxp¥
```

```
lib¥csmstax.jar" -d WEB-INF¥classes¥ -s src src¥com¥sample¥UserInfoImpl.java src¥com¥sample¥
UserData.java src¥com¥sample¥UserInfoException.java
```

javac コマンドが正常に終了すると，コンパイルしたクラスが，c:¥temp¥jaxws¥works¥streaming¥server¥WEB-INF¥classes¥com¥sample¥ディレクトリに出力されます。なお，コンパイルした Web サービス実装クラスに対して hwsgen コマンドを実行すると，事前にエラーチェックができます。hwsgen コマンドについては，「14.1.2　hwsgen コマンド」を，エラーチェックについては，「10.23.1　hwsgen コマンドによるエラーチェックについて」を参照してください。

33.3.3　web.xml を作成する

WAR ファイルの構成要素として必要な web.xml を作成します。

web.xml の作成例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/we
b-app_3_0.xsd">
  <description>Sample web service &quot;streaming_dynamic_generate&quot;</description>
  <display-name>Sample_web_service_streaming_dynamic_generate</display-name>
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/UserInfoService</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

バージョン 2.5 の web.xml を作成する場合は，web-app 要素の version 属性を"2.5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"にしてください。

作成した web.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥streaming¥server¥WEB-INF¥ディレクトリに保存します。web.xml の設定項目については，「3.4　web.xml の作成」を参照してください。

33.3.4　application.xml を作成する

EAR ファイルの構成要素として必要な application.xml を作成します。

application.xml の作成例を次に示します。なお，Web サービスとして application.xml に設定する項目はありません。

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/application_6.xsd">

  <description>Sample application &quot;streaming_dynamic_generate&quot;</description>
  <display-name>Sample_application_streaming_dynamic_generate</display-name>
  <module>
    <web>
      <web-uri>streaming_dynamic_generate.war</web-uri>
      <context-root>streaming_dynamic_generate</context-root>
    </web>
  </module>
</application>
```

バージョン 5 の application.xml を作成する場合は，application 要素の version 属性を"5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/application_5.xsd"にしてください。

作成した application.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥streaming¥server¥META-INF¥ディレクトリに保存します。application.xml を作成するときの注意事項については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「5.2.2　application.xml 編集時の注意事項」を参照してください。

33.3.5　EAR ファイルを作成する

jar コマンドを使用して，これまでに作成したファイルを含めた EAR ファイルを作成します。

EAR ファイルの作成例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥streaming¥server¥
> jar cvf streaming_dynamic_generate.war .¥WEB-INF
> jar cvf streaming_dynamic_generate.ear .¥streaming_dynamic_generate.war .¥META-INF¥application.xml
```

正常に終了すると，c:¥temp¥jaxws¥works¥streaming¥server¥ディレクトリに streaming_dynamic_generate.ear が作成されます。

jar コマンドについては，JDK のドキュメントを参照してください。

33.4　デプロイと開始の例（SEI 起点・ストリーミング）

ここでは，ストリーミングを使用して，SEI を起点とした場合のデプロイと開始の例を説明します。

33.4.1　EAR ファイルをデプロイする

cjimportapp コマンドを使用して，作成した EAR ファイルを J2EE サーバにデプロイします。

デプロイの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥streaming¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjimportapp" jaxwsserver -nameserver corbaname::testserver
:900 -f streaming_dynamic_generate.ear
```

cjimportapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションをデプロイ（インポート）する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.3　J2EE アプリケーションのインポート」を参照してください。

33.4.2　Web サービスを開始する

cjstartapp コマンドを使用して，Web サービスを開始します。

Web サービスを開始する例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥streaming¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjstartapp" jaxwsserver -nameserver corbaname::testserver:
900 -name Sample_application_streaming_dynamic_generate
```

cjstartapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartapp（J2EE アプリケーションの開始）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションを開始する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.1　J2EE アプリケーションの開始」を参照してください。

33.5　Web サービスクライアントの開発例（SEI 起点・ストリーミング）

ここでは，添付ファイルを使用して，SEI を起点とした場合の Web サービスクライアントの開発例を説明します。

33.5.1　サービスクラスを生成する

cjwsimport コマンドを実行すると，サービスクラスなど Web サービスクライアントの開発に必要な Java ソースが生成されます。cjwsimport コマンドについては，「14.1.1　cjwsimport コマンド」を参照してください。

cjwsimport コマンドの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥streaming¥client¥
> mkdir src¥
> mkdir classes¥
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" -s src -d classes http://webhost:8085/streaming_dynamic_generate/UserInfoService?wsdl
```

正常に終了すると，c:¥temp¥jaxws¥works¥streaming¥client¥src¥com¥sample¥ディレクトリに Java ソースが生成されます。

生成物の一覧を次に示します。

表 33-5　サービスクラス生成時の生成物（SEI 起点・添付ファイル）

ファイル名	説明
GetUserData.java	WSDL 定義の要求メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
GetUserDataResponse.java	WSDL 定義の応答メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
ObjectFactory.java	JAXB 2.2 仕様の ObjectFactory クラスです。
package-info.java	package-info.java ファイルです。
UserData.java	UserData に対応する JavaBean クラスです。
UserInfoImpl.java	WSDL 定義の「サービス」に対応する SEI です。
UserInfoService.java	サービスクラスです。
UserInfoException.java	UserInfoException に対応する JavaBean クラスです。
UserInfoException_Exception.java	フォルト bean のラッパ例外クラスです。

33.5.2　Web サービスクライアントの実装クラスを作成する

Web サービスを利用する Web サービスクライアントの実装クラスを作成します。

Web サービスに対して 1 回の呼び出しをする Web サービスクライアントの作成例を次に示します。

```
package com.sample.client;

import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import javax.xml.ws.soap.MTOMFeature;
import java.io.File;

import com.sample.UserInfoImpl;
import com.sample.UserData;
import com.sample.UserInfoService;
import com.sample.UserInfoException_Exception;

public class TestClient {

    public static void main( String[] args ) {
        try {
            //DataHandlerオブジェクト生成
            File imageFile = new File("portrait.png");
            if (!imageFile.exists()) {
                throw new RuntimeException("Cannot find the file ¥"portrait.png¥".");
            }
            FileDataSource fdSource = new FileDataSource(imageFile);
            DataHandler dhandler = new DataHandler(fdSource);


            UserInfoService service = new UserInfoService();
            UserInfoImpl port = service.getUserInfoImplPort(new MTOMFeature());

            UserData userdata = port.getUserData("1", dhandler);
            System.out.print("[RESULT] " + userdata.getMessage());
            System.out.println(" Name:" + userdata.getName()
                               + ", Section:" + userdata.getSection());
        } catch(UserInfoException_Exception e) {
            e.printStackTrace();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

作成した TestClient.java は，UTF-8 形式で
c:¥temp¥jaxws¥works¥streaming¥client¥src¥com¥sample¥client¥ディレクトリに保存します。

33.5.3　Web サービスクライアントの実装クラスをコンパイルする

javac コマンドを使用して，作成した Web サービスクライアントをコンパイルします。

コンパイルの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥streaming¥client¥
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥javaee¥1100¥lib¥javaee-api.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;.¥classes" -d .¥classes src¥com¥sample¥client¥TestClient.java
```

正常に終了すると，c:¥temp¥jaxws¥works¥streaming¥client¥classes¥com¥sample¥client¥ディレクトリに，TestClient.class が生成されます。

javac コマンドについては，JDK のドキュメントを参照してください。

33.6　Web サービスの実行例（SEI 起点・ストリーミング）

ここでは，ストリーミングを使用して，SEI を起点とした場合の Web サービスクライアントの実行例を説明します。

33.6.1　Java アプリケーション用オプション定義ファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用オプション定義ファイル（usrconf.cfg）を作成します。

Java アプリケーション用オプション定義ファイルの作成例を次に示します。

```
add.class.path=<Application Serverのインストールディレクトリ>¥jaxws¥lib¥cjjaxws.jar
add.class.path=.¥classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home=<Application Serverのインストールディレクトリ>
```

<Application Server のインストールディレクトリ>の部分は，Application Server をインストールしているパスを絶対パスで指定します。

作成した Java アプリケーション用オプション定義ファイルは，c:¥temp¥jaxws¥works¥streaming¥client¥ディレクトリに保存します。Java アプリケーション用オプション定義ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.1　usrconf.cfg（Java アプリケーション用オプション定義ファイル）」を参照してください。

33.6.2　Java アプリケーション用ユーザプロパティファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用ユーザプロパティファイルを作成します。

ここでは特に設定を変更しないため，c:¥temp¥jaxws¥works¥streaming¥client¥ディレクトリに usrconf.properties という名称の空のファイルを作成します。Java アプリケーション用ユーザプロパティファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.2　usrconf.properties（Java アプリケーション用ユーザプロパティファイル）」を参照してください。

33.6.3　Web サービスクライアントを実行する

cjclstartap コマンドを使用して，Web サービスクライアントを実行します。

Web サービスクライアントの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥streaming¥client¥
> "%COSMINEXUS_HOME%¥CC¥client¥bin¥cjclstartap"  com.sample.client.TestClient
```

正常に終了すると，Web サービスクライアントの実行結果が表示されます。実行結果の表示例を次に示します。

```
KDJE40053-I The cjclstartap command will now start. (directory for the user definition file
= c:¥temp¥jaxws¥works¥streaming¥client, PID = 2968)
[RESULT] Success. Name:Sato Taro, Section:The personnel section
KDJE40054-I The cjclstartap command was stopped. (PID = 2968, exit status = 0)
```

イタリック体になっている個所は，実行したタイミングや環境によって変わります。

cjclstartap コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjclstartap（Java アプリケーションの開始)」を参照してください。

34 WS-RM 1.2 機能

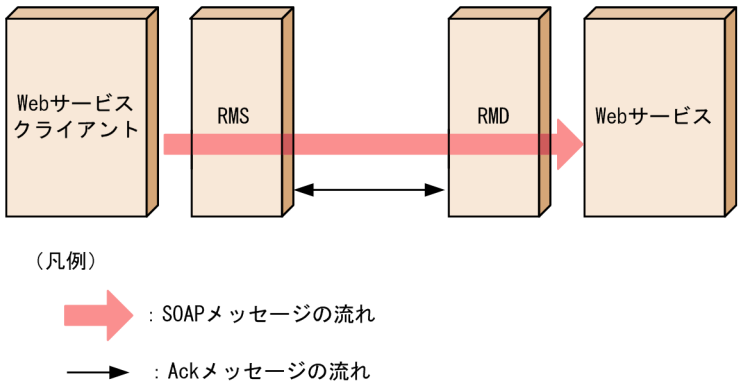
この章では，WS-RM 1.2 機能（Web サービス Reliable Messaging 機能）について説明します。

34.1　WS-RM 1.2 機能とは

WS-RM 1.2 機能とは，Web サービスと Web サービスクライアントの SOAP メッセージのやり取りに，RMD（Reliable Messaging Destination）および RMS（Reliable Messaging Source）を介することで，SOAP メッセージを確実に送受信するための機能です。RMS と RMD の間では，バックグラウンドでメッセージを受信したことを通知する Ack メッセージを送信して，メッセージが相手先に届いたかどうかを管理しています。必要に応じて WS-RM 1.2 機能が自動的にメッセージの再送や重複したメッセージの排除をして，通信の信頼性を高めています。

WS-RM 1.2 機能を使用した場合の SOAP メッセージの送受信の流れを次に示します。

図 34–1　WS-RM 1.2 機能を使用した場合の SOAP メッセージの送受信の流れ

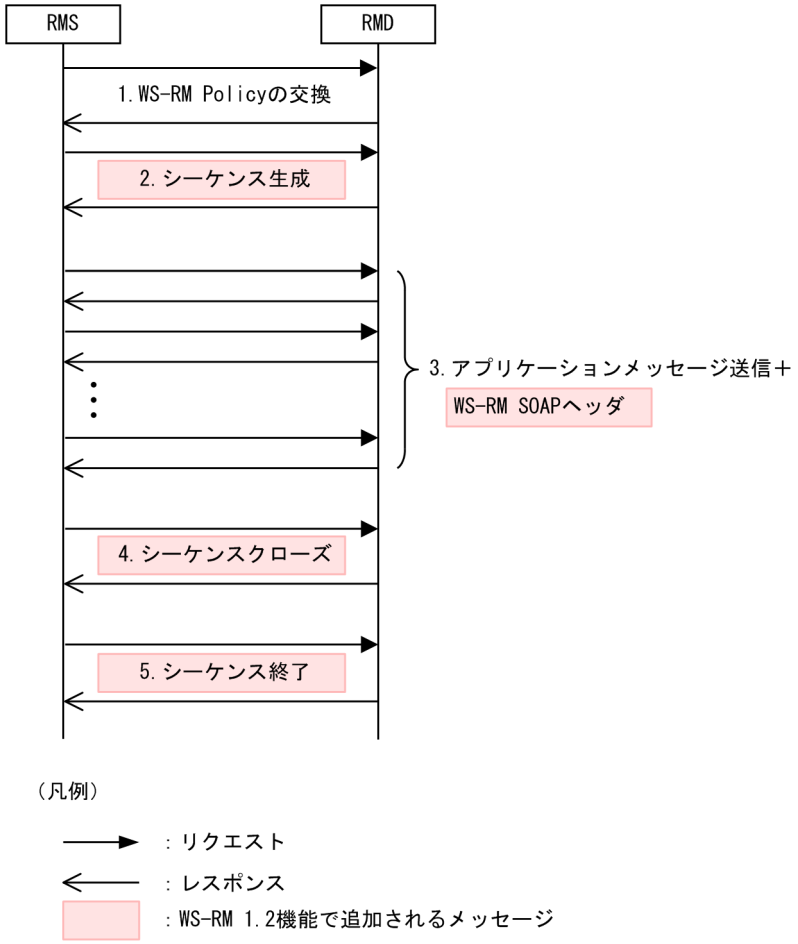


Dispatch インタフェースおよび Provider インタフェースは利用できません。

34.2　WS-RM 1.2 機能を使用したメッセージの流れ

WS-RM 1.2 機能を使用した場合の，メッセージの流れを次に示します。

図 34–2　WS-RM 1.2 機能を使用したメッセージの流れ



（凡例）

：リクエスト

：レスポンス

：WS-RM 1.2機能で追加されるメッセージ

1. WS-RM Policy の交換

メッセージの送信元と送信先の間で，WS-RM 1.2 機能の使用の有無や WS-RM 1.2 機能の WS-RM Policy を交換します。WSDL ファイルに，WS-RM Policy 仕様を記述することで WS-RM Policy を交換します。

2. シーケンス生成

アプリケーションメッセージを送信する前に，RMS と RMD はシーケンスを生成して，共有します。シーケンスとは，WS-RM 1.2 機能で送信される一連のアプリケーションメッセージの集合に関するコンテキストです。シーケンス生成処理は，ポートのオブジェクトに対する初回 Web サービスメソッド呼び出し時に自動で実行されます。

3. アプリケーションメッセージ送信

シーケンスを生成したあと，RMS は RMD にアプリケーションメッセージを送信します。送信するアプリケーションメッセージには，シーケンスの識別子とメッセージ番号が付与されます。

4. シーケンスクローズ

アプリケーションメッセージの送信が終了すると，RMS はシーケンスをクローズできます。シーケンスをクローズすると，新たにアプリケーションメッセージを送受信することはできません。シーケンスをクローズしても，関連するリソースは破棄しないで，保持されます。シーケンスのクローズ処理は，Web サービスクライアント側で close メソッドが呼び出されたときに実行されます。

5. シーケンス終了

アプリケーションメッセージの送信が終了すると，RMS はシーケンスを終了させます。シーケンスが終了すると，関連するリソースは破棄されます。再度シーケンスを生成するには，ポートのオブジェクトを取得し直し，Web サービスメソッドを呼び出します。

以降では，シーケンス生成に関するメッセージ，シーケンスクローズに関するメッセージ，およびシーケンス終了に関するメッセージをまとめて，シーケンスライフサイクルメッセージと呼びます。

34.3　WS-RM 1.2 機能の送達保証

送達保証とは，SOAP メッセージを確実に送受信するために，SOAP メッセージ時の再送，重複排除，および順序制御をする機能です。

送達保証の種類および Application Server の WS-RM 1.2 機能でのサポート範囲を次の表に示します。

表 34–1　送達保証の種類と Application Server の WS-RM 1.2 機能でのサポート範囲

項番	種類	動作	RMS の処理	RMD の処理	サポート
1	AtLeastOnce	少なくとも 1 回送達	再送	－	×
2	AtMostOnce	重複なく送達	－	重複排除	×
3	ExactlyOnce	1 回だけ送達	再送	重複排除	○
4	InOrder	順序どおりに送達	－	順序制御	×

（凡例）

○：使用できます。

×：使用できません。

－：該当しません。

34.3.1　再送

WS-RM 1.2 機能を使用している場合，アプリケーションメッセージが接続先に届かないときに，WS-RM Policy のアプリケーションメッセージの再送回数で設定した値を上限として，RMS から自動的にアプリケーションメッセージが再送されます。指定した回数再送されてもアプリケーションメッセージが届かない場合，クライアントのアプリケーションに javax.xml.ws.WebServiceException が返されます。

再送の条件を次に示します。

- JAX-WS の動作定義ファイルや要求コンテキストで設定したクライアントソケットの読み込みタイムアウト値または接続タイムアウト値を超えてもレスポンスがなく，タイムアウトした場合
- サーバとの接続が不意に切断され，レスポンスが何も受信できなかった場合
- アプリケーションメッセージまたは自動的に再送されたアプリケーションメッセージへのレスポンスとして，Body 要素が空の Ack メッセージを受信した場合
- アプリケーションメッセージまたは自動的に再送されたアプリケーションメッセージへのレスポンスとして，HTTP ステータスコードの 202，300 番台，400 番台，500 番台を受信した場合

34.3.2 重複排除

以前受信したメッセージと同じメッセージを受信した場合，メッセージ送信先にはメッセージを届けないで RMD で破棄されます。アプリケーションは呼び出されません。

重複メッセージを受信した場合，次のメッセージをクライアントに返します。

- すでにレスポンスを返しているリクエストと重複したリクエストを受信した場合
  一度返したレスポンスと同じ内容を返します。
- リクエスト処理中に，同じリクエストを重複して受信した場合
  クライアントに Body 要素が空の Ack メッセージを返します。ただし，返す Ack がないときは HTTP ステータスコード 202 を返します。

34.4　WS-RM Policy の追加方法

WS-RM 1.2 機能では，WSDL ファイルに WS-RM Policy を追加することで，WS-RM 1.2 機能を有効にします。追加する項目は次のとおりです。

1. WS-RM Policy で使用する名前空間プレフィクスを定義する
2. wsdl:definitions 要素の子要素としてポリシーを定義する
3. wsdl:binding 要素の子要素でポリシーを参照する

追加例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="TestJaxWsService"

<!-- 中略 -->

<!-- 1.WS-RM Policyで使用する名前空間プレフィクスを定義する -->
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:wsrmp="http://docs.oasis-open.org/ws-rx/wsrmp/200702"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.
xsd"
  xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl">

<!-- 2.wsdl:definitions要素の子要素としてポリシーを定義する -->
  <wsp:Policy wsu:Id="WSRM_policy">
    <wsp:ExactlyOne>
      <wsp:All>
        <wsrmp:RMAssertion>
          <wsp:Policy>
            <wsrmp:DeliveryAssurance>
              <wsp:Policy>
                <wsrmp:ExactlyOnce/>
              </wsp:Policy>
            </wsrmp:DeliveryAssurance>
          </wsp:Policy>
        </wsrmp:RMAssertion>
        <wsaw:UsingAddressing/>
        <!-- その他の設定 -->
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>

  <wsdl:types>

<!-- 中略 -->

<!-- 3.wsdl:binding要素の子要素でポリシーを参照する -->
  <wsdl:binding name="testJaxWsBinding" type="tns:TestJaxWs">
    <wsp:PolicyReference URI="#WSRM_policy"/>

<!-- 後略 -->
```

背景色付きの太字で示す個所が，名前空間プレフィクスの定義，WS-RM Policy の定義，または WS-RM Policy の参照個所です。

wsrmp:DeliveryAssurance 要素以下は省略できます。wsrmp:DeliveryAssurance 要素以下および「<!-- その他の設定 -->」以外は，そのまま記載します。「<!-- その他の設定 -->」は必要に応じて設定を追加します。「<!-- その他の設定 -->」として WSDL に追加するプロパティについては「23.4　WS-Policy による設定」を参照してください。

35 WSDL を起点とした開発の例（WS-RM 1.2 機能使用時）

この章では，WS-RM 1.2 機能を使用する場合に，WSDL を起点とした Web サービスを開発するときの例を説明します。

35.1　開発例の構成（WSDL 起点・WS-RM 1.2）

この章では，WSDL を起点とした Web サービスを開発します。

開発する Web サービスの構成を次の表に示します。

表 35–1　Web サービスの構成（WSDL 起点）

項番	項目		値
1	デプロイする J2EE サーバの名称		jaxwsserver
2	Web サーバのホスト名とポート番号		webhost:8085
3	ネーミングサーバの URL		corbaname::testserver:900
4	コンテキストルート		wsrm
5	スタイル		document/literal/wrapped
6	名前空間 URI		http://example.com/sample
7	ポートタイプ	個数	1
8		ローカル名	TestJaxWs
9	オペレーション	個数	1
10		ローカル名	jaxWsTest1
11	サービス	個数	1
12		ローカル名	TestJaxWsService
13	ポート	個数	1
14		ローカル名	testJaxWs
15	WSDL のファイル名		input.wsdl

Web サービス開発時のカレントディレクトリの構成を次の表に示します。

表 35–2　カレントディレクトリの構成（WSDL 起点）

ディレクトリ	説明
c:¥temp¥jaxws¥works¥wsrm	カレントディレクトリです。
├server¥	Web サービスの開発で使用します。
├├META-INF¥	EAR ファイルの META-INF ディレクトリに対応します。
├└application.xml	「35.3.7　application.xml を作成する」で作成します。
├src¥	Web サービスのソースファイル（*.java）を格納します。「35.3.3　SEI を生成する」および「35.3.5　Web サービス実装クラスをコンパイルする」で使用します。

ディレクトリ	説明
│	
├WEB-INF¥	WAR ファイルの WEB-INF ディレクトリに対応します。
│ ├web.xml	「35.3.6　web.xml を作成する」で作成します。
│ ├classes¥	コンパイルしたクラスファイル（*.class）を格納します。
│ └wsdl¥	作成した wsdl を格納します。
├temporary¥	Java で記述，変換したものを基に WSDL を作成する場合に一時ファイルを格納します。このディレクトリの作成は任意です。
│ ├src¥	
│ └classes¥	
├wsrm.ear	「35.3.8　EAR ファイルを作成する」で作成します。
└wsrm.war	
└client¥	Web サービスクライアントの開発で使用します。
├src¥	Web サービスクライアントのソースファイル（*.java）を格納します。「35.5.1　サービスクラスを生成する」および「35.5.2　Web サービスクライアントの実装クラスを作成する」で使用します。
├classes¥	コンパイルしたクラスファイル（*.class）を格納します。「35.5.4　Web サービスクライアントの実装クラスをコンパイルする」で使用します。
├usrconf.cfg	「35.6.1　Java アプリケーション用オプション定義ファイルを作成する」で作成します。
└usrconf.properties	「35.6.2　Java アプリケーション用ユーザプロパティファイルを作成する」で作成します。

カレントディレクトリのパスは，開発する環境に合わせて変更してください。

なお，以降の説明では，この表に示すディレクトリおよびファイル名を使用します。コマンド実行例や Java ソースなどで背景色付きの太字になっている部分は，この例で使用する指定値や生成される値を示します。構築する環境に合わせて読み替えてください。

また，この章で説明する開発例では，Web サービスと Web サービスクライアントを同じ環境で開発しますが，別の環境で開発することもできます。別の環境で開発する場合は，それぞれの環境に合わせて，カレントディレクトリのパスを読み替えてください。

35.2　開発例の流れ（WSDL 起点・WS-RM 1.2）

この章で説明する開発例では，次の流れで開発および実行します。

Web サービスの開発

1. WSDL ファイルを作成する（35.3.1）
2. WSDL ファイルに WS-RM Policy を追加する（35.3.2）
3. cjwsimport コマンドを実行し，SEI を生成する（35.3.3）
4. Web サービス実装クラスを作成する（35.3.4）
5. Web サービス実装クラスをコンパイルする（35.3.5）
6. web.xml を作成する（35.3.6）
7. application.xml を作成する（35.3.7）
8. EAR ファイルを作成する（35.3.8）

デプロイと開始

1. EAR ファイルをデプロイする（35.4.1）
2. Web サービスを開始する（35.4.2）

Web サービスクライアントの開発

1. cjwsimport コマンドを実行し，サービスクラスを生成する（35.5.1）
2. Web サービスクライアントの実装クラスを作成する（35.5.2）
3. Web サービスクライアントの実装クラスにシーケンス終了処理を追加する（35.5.3）
4. Web サービスクライアントの実装クラスをコンパイルする（35.5.4）

Web サービスの実行

1. Java アプリケーション用オプション定義ファイルを作成する（35.6.1）
2. Java アプリケーション用ユーザプロパティファイルを作成する（35.6.2）
3. Web サービスクライアントを実行する（35.6.3）

35.3　Web サービスの開発例（WSDL 起点・WS-RM 1.2）

ここでは，WSDL を起点とした場合の Web サービスの開発例を説明します。

35.3.1　WSDL ファイルを作成する

WSDL ファイルを作成し，Web サービスのメタ情報を定義します。WSDL 定義は，次の仕様のサポート範囲内で定義してください。

- WSDL 1.1 仕様
  サポート範囲については，「20.1　WSDL 1.1 仕様のサポート範囲」を参照してください。
- XML Schema 仕様
  サポート範囲については，マニュアル「XML Processor ユーザーズガイド」を参照してください。
- WS-I Basic Profile 1.1 仕様

WSDL ファイルの作成方法には，新規に作成する方法と，Java ソースを変換したものを利用して作成する方法の 2 種類があります。

(1)　新規に WSDL ファイルを作成する

ここでは，WSDL ファイル（input.wsdl）を作成します。作成した WSDL ファイルは，UTF-8 形式で c:¥temp¥jaxws¥works¥wsrm¥server¥WEB-INF¥wsdl¥ディレクトリに保存してください。

SOAP 1.1 の場合の作成例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="TestJaxWsService"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://example.com/sample"
  targetNamespace="http://example.com/sample">

  <wsdl:types>
    <xsd:schema targetNamespace="http://example.com/sample">
      <!-- 要求メッセージの wrapper要素 -->
      <xsd:element name="jaxWsTest1" type="tns:jaxWsTest1"/>

      <!-- 応答メッセージの wrapper要素 -->
      <xsd:element name="jaxWsTest1Response" type="tns:jaxWsTest1Response"/>

      <!-- フォルトメッセージの wrapper要素 -->
      <xsd:element name="UserDefinedFault" type="tns:UserDefinedFault"/>

      <!-- 要求メッセージの wrapper要素が参照する型 -->
      <xsd:complexType name="jaxWsTest1">
        <xsd:sequence>
```

```
        <xsd:element name="information" type="xsd:string"/>
        <xsd:element name="count" type="xsd:int"/>
      </xsd:sequence>
    </xsd:complexType>

    <!-- 応答メッセージの wrapper要素が参照する型 -->
    <xsd:complexType name="jaxWsTest1Response">
      <xsd:sequence>
        <xsd:element name="return" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>

    <!-- フォルトメッセージの wrapper要素が参照する型 -->
    <xsd:complexType name="UserDefinedFault">
      <xsd:sequence>
        <xsd:element name="additionalInfo" type="xsd:int"/>
        <xsd:element name="detail" type="xsd:string"/>
        <xsd:element name="message" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
</wsdl:types>

<!-- 要求メッセージ -->
<wsdl:message name="jaxWsTest1Request">
  <wsdl:part name="inputParameters" element="tns:jaxWsTest1"/>
</wsdl:message>

<!-- 応答メッセージ -->
<wsdl:message name="jaxWsTest1Response">
  <wsdl:part name="outputParameters" element="tns:jaxWsTest1Response"/>
</wsdl:message>

<!-- フォルトメッセージ -->
<wsdl:message name="UserDefinedException">
  <wsdl:part name="fault" element="tns:UserDefinedFault"/>
</wsdl:message>

<!-- ポートタイプ -->
<wsdl:portType name="TestJaxWs">
  <!-- オペレーション -->
  <wsdl:operation name="jaxWsTest1">
    <wsdl:input message="tns:jaxWsTest1Request"/>
    <wsdl:output message="tns:jaxWsTest1Response"/>
    <wsdl:fault name="UserDefinedFault"
      message="tns:UserDefinedException"/>
  </wsdl:operation>
</wsdl:portType>

<!-- バインディング(SOAP 1.1/HTTPバインディング) -->
<wsdl:binding name="testJaxWsBinding" type="tns:TestJaxWs">
  <!-- document/literal/wrapped -->
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <!-- オペレーション -->
  <wsdl:operation name="jaxWsTest1">
    <soap:operation/>
    <wsdl:input>
      <soap:body use="literal"/>
```

```
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="UserDefinedFault">
        <soap:fault name="UserDefinedFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>

  <!-- サービス -->
  <wsdl:service name="TestJaxWsService">
    <!-- ポート -->
    <wsdl:port name="testJaxWs" binding="tns:testJaxWsBinding">
      <soap:address location="http://webhost:8085/wsrm/TestJaxWsService"/>
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

SOAP 1.2 の場合の作成例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="TestJaxWsService"
  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://example.com/sample"
  targetNamespace="http://example.com/sample">

  <wsdl:types>
    <xsd:schema targetNamespace="http://example.com/sample">
      <!-- 要求メッセージの wrapper要素 -->
      <xsd:element name="jaxWsTest1" type="tns:jaxWsTest1"/>

      <!-- 応答メッセージの wrapper要素 -->
      <xsd:element name="jaxWsTest1Response" type="tns:jaxWsTest1Response"/>

      <!-- フォルトメッセージの wrapper要素 -->
      <xsd:element name="UserDefinedFault" type="tns:UserDefinedFault"/>

      <!-- 要求メッセージの wrapper要素が参照する型 -->
      <xsd:complexType name="jaxWsTest1">
        <xsd:sequence>
          <xsd:element name="information" type="xsd:string"/>
          <xsd:element name="count" type="xsd:int"/>
        </xsd:sequence>
      </xsd:complexType>

      <!-- 応答メッセージの wrapper要素が参照する型 -->
      <xsd:complexType name="jaxWsTest1Response">
        <xsd:sequence>
          <xsd:element name="return" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>

      <!-- フォルトメッセージの wrapper要素が参照する型 -->
```

```
      <xsd:complexType name="UserDefinedFault">
        <xsd:sequence>
          <xsd:element name="additionalInfo" type="xsd:int"/>
          <xsd:element name="detail" type="xsd:string"/>
          <xsd:element name="message" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </wsdl:types>

  <!-- 要求メッセージ -->
  <wsdl:message name="jaxWsTest1Request">
    <wsdl:part name="inputParameters" element="tns:jaxWsTest1"/>
  </wsdl:message>

  <!-- 応答メッセージ -->
  <wsdl:message name="jaxWsTest1Response">
    <wsdl:part name="outputParameters" element="tns:jaxWsTest1Response"/>
  </wsdl:message>

  <!-- フォルトメッセージ -->
  <wsdl:message name="UserDefinedException">
    <wsdl:part name="fault" element="tns:UserDefinedFault"/>
  </wsdl:message>

  <!-- ポートタイプ -->
  <wsdl:portType name="TestJaxWs">
    <!-- オペレーション -->
    <wsdl:operation name="jaxWsTest1">
      <wsdl:input message="tns:jaxWsTest1Request"/>
      <wsdl:output message="tns:jaxWsTest1Response"/>
      <wsdl:fault name="UserDefinedFault"
        message="tns:UserDefinedException"/>
    </wsdl:operation>
  </wsdl:portType>

<!-- バインディング(SOAP 1.2/HTTPバインディング) -->
<wsdl:binding name="testJaxWsBinding" type="tns:TestJaxWs">
  <!-- document/literal/wrapped -->
  <soap12:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <!-- オペレーション -->
    <wsdl:operation name="jaxWsTest1">
      <soap12:operation/>
      <wsdl:input>
        <soap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="UserDefinedFault">
        <soap12:fault name="UserDefinedFault" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>

  <!-- サービス -->
  <wsdl:service name="TestJaxWsService">
    <!-- ポート -->
```

```
    <wsdl:port name="testJaxWs" binding="tns:testJaxWsBinding">
      <soap12:address location="http://webhost:8085/wsrm/TestJaxWsService"/>
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

(2) Java ソースを変換したものを基に WSDL ファイルを作成する

ここでは，WSDL 変換用に仮実装の Web サービス実装クラスと例外クラスを作成し，hwsgen コマンドの WSDL 生成機能を実行して，コンパイル済みの Java ソースから WSDL ファイルを作成します。作成したクラスは，javax.jws.WebService アノテーションでアノテートします。SOAP 1.2 を使用する場合，さらに SOAP 1.2 を指定した javax.xml.ws.BindingType アノテーションでアノテートしてください。メソッドを実装する必要はありません。

SOAP 1.1 を使用する場合と SOAP 1.2 を使用する場合では，仮実装の Web サービス実装クラスと TestJaxWsService.wsdl でソースコードが異なります。

SOAP 1.1 の場合の仮実装の Web サービス実装クラスの例を次に示します。

```
package com.example.sample;

@javax.jws.WebService
public class TestJaxWsImpl {

    public String jaxWsTest1(String information, int count)
        throws UserDefinedException
    {
        // 実装不要
        return null;
    }

}
```

SOAP 1.2 の場合の仮実装の Web サービス実装クラスの例を次に示します。

```
package com.example.sample;

@javax.jws.WebService
@javax.xml.ws.BindingType(javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_BINDING)
public class TestJaxWsImpl {

    public String jaxWsTest1(String information, int count)
        throws UserDefinedException
    {
        // 実装不要
        return null;
    }

}
```

仮実装の例外クラスの例を次に示します。

```
package com.example.sample;

public class UserDefinedFault extends Exception{
    // 実装不要
    public int additionalInfo;
    public String detail;
    public String message;
}
```

作成した TestJaxWsImpl.java と UserDefinedFault.java を UTF-8 形式で c:¥temp¥jaxws¥works¥wsrm¥server¥temporary¥src¥com¥example¥sample¥ディレクトリに保存し，コンパイルします。コンパイルの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥wsrm¥server¥
> mkdir .¥temporary
> mkdir .¥temporary¥classes
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥ja
vaee¥1100¥lib¥javaee-api.jar" -d .¥temporary¥classes .¥temporary¥src¥com¥example¥sample¥Test
JaxWsImpl.java .¥temporary¥src¥com¥example¥sample¥UserDefinedFault.java
```

コンパイルが正常に終了すると，c:¥temp¥jaxws¥works¥wsrm¥server¥temporary¥classes¥com¥example¥sample¥ディレクトリに TestJaxWsImpl.class と UserDefinedFault.class が生成されます。これらのクラスファイルを利用して，hwsgen コマンドの WSDL 生成機能で WSDL ファイルを作成します。

hwsgen コマンドの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥wsrm¥server¥
> mkdir .¥WEB-INF¥wsdl¥
> "%COSMINEXUS_HOME%¥jaxws¥bin¥hwsgen.bat" -r .¥WEB-INF¥wsdl -d .¥temporary¥classes -cp .¥te
mporary¥classes com.example.sample.TestJaxWsImpl
```

hwsgen コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥wsrm¥WEB-INF¥wsdl¥ディレクトリに TestJaxWsService.wsdl と TestJaxWsService_schema1.xsd が生成されます。c:¥temp¥jaxws¥works¥wsrm¥temporary¥classes¥ディレクトリにあるクラスは削除してください。

生成された TestJaxWsService.wsdl と TestJaxWsService_schema1.xsd は，一部修正する必要があります。

SOAP 1.1 の場合の TestJaxWsService.wsdl の修正例を次に示します。イタリック体になっている個所が，修正した個所です。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions targetNamespace="http://example.com/sample" name="TestJaxWsImplService" xmlns:t
ns="http://example.com/sample" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http
://schemas.xmlsoap.org/wsdl/soap/" xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <xsd:schema targetNamespace="http://example.com/sample">
      <xsd:include schemaLocation="TestJaxWsImplService_schema1.xsd"/>
    </xsd:schema>
```

```
  </types>
  <message name="jaxWsTest1">
    <part name="parameters" element="tns:jaxWsTest1"/>
  </message>
  <message name="jaxWsTest1Response">
    <part name="parameters" element="tns:jaxWsTest1Response"/>
  </message>
  <message name="UserDefinedFault">
    <part name="fault" element="tns:UserDefinedFault"/>
  </message>
  <portType name="TestJaxWs">
    <operation name="jaxWsTest1">
      <input message="tns:jaxWsTest1"/>
      <output message="tns:jaxWsTest1Response"/>
      <fault message="tns:UserDefinedFault" name="UserDefinedFault"/>
    </operation>
  </portType>
  <binding name="testJaxWsBinding" type="tns:TestJaxWs">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="jaxWsTest1">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
      <fault name="UserDefinedFault">
        <soap:fault name="UserDefinedFault" use="literal"/>
      </fault>
    </operation>
  </binding>
  <service name="TestJaxWsService">
    <port name="testJaxWs" binding="tns:testJaxWsBinding">
      <soap:address location="http://webhost:8085/wsrm/TestJaxWsService"/>
    </port>
  </service>
</definitions>
```

SOAP 1.2 の場合の TestJaxWsService.wsdl の修正例を次に示します。イタリック体になっている個所が，修正した個所です。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions targetNamespace="http://example.com/sample" name="TestJaxWsImplService" xmlns:t
ns="http://example.com/sample" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap12="ht
tp://schemas.xmlsoap.org/wsdl/soap12/" xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <xsd:schema targetNamespace="http://example.com/sample">
      <xsd:include schemaLocation="TestJaxWsImplService_schema1.xsd"/>
    </xsd:schema>
  </types>
  <message name="jaxWsTest1">
    <part name="parameters" element="tns:jaxWsTest1"/>
  </message>
  <message name="jaxWsTest1Response">
    <part name="parameters" element="tns:jaxWsTest1Response"/>
```

```
  </message>
  <message name="UserDefinedFault">
    <part name="fault" element="tns:UserDefinedFault"/>
  </message>
  <portType name="TestJaxWs">
    <operation name="jaxWsTest1">
      <input message="tns:jaxWsTest1"/>
      <output message="tns:jaxWsTest1Response"/>
      <fault message="tns:UserDefinedFault" name="UserDefinedFault"/>
    </operation>
  </portType>
  <binding name="testJaxWsBinding" type="tns:TestJaxWs">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="jaxWsTest1">
      <soap12:operation soapAction=""/>
      <input>
        <soap12:body use="literal"/>
      </input>
      <output>
        <soap12:body use="literal"/>
      </output>
      <fault name="UserDefinedFault">
        <soap12:fault name="UserDefinedFault" use="literal"/>
      </fault>
    </operation>
  </binding>
  <service name="TestJaxWsService">
    <port name="testJaxWs" binding="tns:testJaxWsBinding">
      <soap12:address location="http://webhost:8085/wsrm/TestJaxWsService"/>
    </port>
  </service>
</definitions>
```

TestJaxWsService_schema1.xsd の修正例を次に示します。イタリック体になっている個所が，修正した個所です。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" targetNamespace="http://example.com/sample" xmlns:tns="http://examp
le.com/sample" xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="UserDefinedFault" type="tns:UserDefinedFault"/>

  <xs:element name="jaxWsTest1" type="tns:jaxWsTest1"/>

  <xs:element name="jaxWsTest1Response" type="tns:jaxWsTest1Response"/>

  <xs:complexType name="jaxWsTest1">
    <xs:sequence>
      <xs:element name="arg0" type="xs:string" minOccurs="0"/>
      <xs:element name="arg1" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="jaxWsTest1Response">
    <xs:sequence>
      <xs:element name="return" type="xs:string" minOccurs="0"/>
```

```
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="UserDefinedFault">
    <xs:sequence>
      <xs:element name="message" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

修正した TestJaxWsService.wsdl は，名前を input.wsdl に変更して，c:¥temp¥jaxws¥works¥wsrm¥server¥WEB-INF¥wsdl¥ディレクトリに保存してください。

35.3.2 WSDL ファイルに WS-RM Policy を追加する

新規に作成した WSDL ファイルに，WS-RM Policy を追加します。追加する項目は次のとおりです。

1. WS-RM Policy で使用する名前空間プレフィクスを定義する
2. wsdl:definitions 要素の子要素としてポリシーを定義する
3. wsdl:binding 要素の子要素でポリシーを参照する

追加例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="TestJaxWsService"

<!-- 中略 -->

<!-- 1.WS-RM Policyで使用する名前空間プレフィクスを定義する -->
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:wsrmp="http://docs.oasis-open.org/ws-rx/wsrmp/200702"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.
xsd"
  xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl">

<!-- 2.wsdl:definitions要素の子要素としてポリシーを定義する -->
  <wsp:Policy wsu:Id="WSRM_policy">
    <wsp:ExactlyOne>
      <wsp:All>
        <wsrmp:RMAssertion>
          <wsp:Policy>
        </wsrmp:RMAssertion>
        <wsaw:UsingAddressing/>
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>

  <wsdl:types>

<!-- 中略 -->
```

```
<!-- 3.wsdl:binding要素の子要素でポリシーを参照する -->
  <wsdl:binding name="testJaxWsBinding" type="tns:TestJaxWs">
    <wsp:PolicyReference URI="#WSRM_policy"/>
    <!-- document/literal/wrapped -->

<!-- 後略 -->
```

35.3.3 SEI を生成する

cjwsimport コマンドを実行すると，SEI など，Web サービスの開発に必要な Java ソースが生成されます。cjwsimport コマンドについては，「14.1.1 cjwsimport コマンド」を参照してください。

cjwsimport コマンドの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥wsrm¥server¥
> mkdir src¥
> mkdir WEB-INF¥classes¥
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" -generateService -s src -d WEB-INF¥classes WE
B-INF¥wsdl¥input.wsdl
```

cjwsimport コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥wsrm¥server¥src¥com¥example¥sample¥ディレクトリに，Java ソースが生成されます。com¥example¥sample¥（パッケージに対応するディレクトリパス）は，名前空間 URI の記述によって変わります。名前空間 URI とパッケージとのマッピングについては，「15.1.1 名前空間からパッケージ名へのマッピング」を参照してください。

生成物の一覧を次の表に示します。

表 35-3 SEI 生成時の生成物（WSDL 起点）

ファイル名	説明
JaxWsTest1.java	WSDL 定義の「要求メッセージの wrapper 要素が参照する型」に対応する JavaBean クラスです。
JaxWsTest1Response.java	WSDL 定義の「応答メッセージの wrapper 要素が参照する型」に対応する JavaBean クラスです。
ObjectFactory.java	JAXB 2.2 仕様の ObjectFactory クラスです。
package-info.java	package-info.java ファイルです。
TestJaxWs.java	TestJaxWs ポートタイプに対応する SEI です。
TestJaxWsImpl.java	TestJaxWs ポートタイプに対応するスケルトンクラスです。
UserDefinedFault.java	WSDL 定義の「フォルトメッセージの wrapper 要素が参照する型」に対応する JavaBean クラス（フォルト bean）です。
UserDefinedException.java	フォルト bean のラッパ例外クラスです。

ファイル名の JaxWsTest1，TestJaxWs，および TestJaxWsImpl は，オペレーションのローカル名，ポートタイプのローカル名，およびサービスのローカル名の記述によって変わります。オペレーションのローカル名，ポートタイプのローカル名，およびサービスのローカル名と Java ソースとのマッピングについては，「15.　WSDL から Java へのマッピング」を参照してください。

35.3.4　Web サービス実装クラスを作成する

スケルトンクラスに Web サービスの処理を追加し，Web サービス実装クラスを作成します。ここでは，受け取った要求メッセージの内容を，日付情報とともに応答メッセージとして返す処理を追加します。

Web サービス実装クラスの作成例を次に示します。

```
package com.example.sample;

import java.util.Calendar;
import javax.jws.WebService;


@WebService(endpointInterface = "com.example.sample.TestJaxWs", targetNamespace = "http://ex
ample.com/sample", serviceName = "TestJaxWsService", portName = "testJaxWs")
public class TestJaxWsImpl {

    public String jaxWsTest1(String information, int count)
        throws UserDefinedException
    {
        Calendar today = Calendar.getInstance();
        StringBuffer result = new StringBuffer( 256 );
        result.append( "We've got your #" );
        result.append( new Integer( count ) );
        result.append( " message ¥"" );
        result.append( information );
        result.append( "¥"! It's " );
        result.append( String.format( "%04d.%02d.%02d %02d:%02d:%02d", new Object[]{
            new Integer( today.get( Calendar.YEAR ) ),
            new Integer( today.get( Calendar.MONTH ) + 1 ),
            new Integer( today.get( Calendar.DAY_OF_MONTH ) ),
            new Integer( today.get( Calendar.HOUR_OF_DAY ) ),
            new Integer( today.get( Calendar.MINUTE ) ),
            new Integer( today.get( Calendar.SECOND ) ) } ) );
        result.append( " now. See ya!" );

        return result.toString();
    }

}
```

イタリック体になっている個所が，スケルトンに対して追加した実装です。

35.3.5　Web サービス実装クラスをコンパイルする

javac コマンドを使用して，作成した Web サービス実装クラスをコンパイルします。

コンパイルの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥wsrm¥server¥
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥javaee¥1100¥lib¥javaee-api.jar;.¥WEB-INF¥classes" -d .¥WEB-INF¥classes src¥com¥example¥sample¥TestJaxWsImpl.java
```

javac コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥wsrm¥server¥WEB-INF¥classes¥com¥example¥sample¥ディレクトリの TestJaxWsImpl.class が上書きされます。

javac コマンドについては，JDK のドキュメントを参照してください。

35.3.6　web.xml を作成する

WAR ファイルの構成要素として必要な web.xml を作成します。

web.xml の作成例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <description>Sample web service &quot;wsrm&quot;</description>
  <display-name>Sample_web_service_wsrm</display-name>
  <listener>
    <listener-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/TestJaxWsService</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

バージョン 2.5 の web.xml を作成する場合は，web-app 要素の version 属性を"2.5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"にしてください。

作成した web.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥wsrm¥server¥WEB-INF¥ディレクトリに保存します。web.xml の設定項目については，「3.4　web.xml の作成」を参照してください。

35.3.7　application.xml を作成する

EAR ファイルの構成要素として必要な application.xml を作成します。

application.xml の作成例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/application_6.xsd">

  <description>Sample application &quot;wsrm&quot;</description>
  <display-name>Sample_application_wsrm</display-name>
  <module>
    <web>
      <web-uri>wsrm.war</web-uri>
      <context-root>wsrm</context-root>
    </web>
  </module>
</application>
```

バージョン 5 の application.xml を作成する場合は，application 要素の version 属性を"5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/application_5.xsd"にしてください。

作成した application.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥wsrm¥server¥META-INF¥ディレクトリに保存します。application.xml を作成するときの注意事項については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「5.2.2　application.xml 編集時の注意事項」を参照してください。

35.3.8　EAR ファイルを作成する

jar コマンドを使用して，EAR ファイルを作成します。

EAR ファイルの作成例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥wsrm¥server¥
> jar cvf wsrm.war .¥WEB-INF
> jar cvf wsrm.ear .¥wsrm.war .¥META-INF¥application.xml
```

jar コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥wsrm¥server¥ディレクトリに wsrm.ear が作成されます。

jar コマンドについては，JDK のドキュメントを参照してください。

35.4　デプロイと開始の例（WSDL 起点・WS-RM 1.2）

ここでは，WSDL を起点とした場合のデプロイと開始の例を説明します。

35.4.1　EAR ファイルをデプロイする

cjimportapp コマンドを使用して，作成した EAR ファイルを J2EE サーバにデプロイします。

デプロイの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥wsrm¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjimportapp" jaxwsserver -nameserver corbaname::testserver
:900 -f wsrm.ear
```

cjimportapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションをデプロイ（インポート）する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.3　J2EE アプリケーションのインポート」を参照してください。

35.4.2　Web サービスを開始する

cjstartapp コマンドを使用して，Web サービスを開始します。

Web サービスを開始する例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥wsrm¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjstartapp" jaxwsserver -nameserver corbaname::testserver:
900 -name Sample_application_wsrm
```

cjstartapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartapp（J2EE アプリケーションの開始）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションを開始する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.1　J2EE アプリケーションの開始」を参照してください。

35.5　Web サービスクライアントの開発例（WSDL 起点・WS-RM 1.2）

ここでは，WSDL を起点とした場合の Web サービスクライアントの開発例を説明します。

35.5.1　サービスクラスを生成する

cjwsimport コマンドを実行すると，サービスクラスなど Web サービスクライアントの開発に必要な Java ソースが生成されます。cjwsimport コマンドについては，「14.1.1　cjwsimport コマンド」を参照してください。

Web サービスの開発と同じ環境で，Web サービスクライアントを開発する場合の実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥wsrm¥client¥
> mkdir src¥
> mkdir classes¥
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" -s src -d classes ..¥server¥WEB-INF¥wsdl¥input.wsdl
```

Web サービスの開発と異なる環境で，Web サービスクライアントを開発する場合の実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥wsrm¥client¥
> mkdir src¥
> mkdir classes¥
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" -s src -d classes http://webhost:8085/wsrm/TestJaxWsService?wsdl
```

正常に終了すると，c:¥temp¥jaxws¥works¥wsrm¥client¥src¥com¥example¥sample¥ディレクトリに Java ソースが生成されます。com¥example¥sample¥（パッケージに対応するディレクトリパス）は，名前空間 URI の記述によって変わります。名前空間 URI とパッケージとのマッピングについては，「15.1.1 名前空間からパッケージ名へのマッピング」を参照してください。

生成物の一覧を次の表に示します。

表 35-4　サービスクラス生成時の生成物（WSDL 起点）

ファイル名	説明
JaxWsTest1.java	WSDL 定義の「要求メッセージの wrapper 要素が参照する型」に対応する JavaBean クラスです。
JaxWsTest1Response.java	WSDL 定義の「応答メッセージの wrapper 要素が参照する型」に対応する JavaBean クラスです。
ObjectFactory.java	JAXB 2.2 仕様の ObjectFactory クラスです。
package-info.java	package-info.java ファイルです。
TestJaxWs.java	TestJaxWs ポートタイプに対応する SEI です。

ファイル名	説明
TestJaxWsService.java	サービスクラスです。
UserDefinedFault.java	WSDL 定義の「フォルトメッセージの wrapper 要素が参照する型」に対応する JavaBean クラスです。
UserDefinedException.java	フォルト bean のラッパ例外クラスです。

ファイル名の JaxWsTest1，TestJaxWs，および TestJaxWsService は，オペレーションのローカル名，ポートタイプのローカル名，およびサービスのローカル名の記述によって変わります。オペレーションのローカル名，ポートタイプのローカル名，およびサービスのローカル名と Java ソースとのマッピングについては，次の個所を参照してください。

- 「15.1.2　ポートタイプから SEI 名へのマッピング」
- 「15.1.3　オペレーションからメソッド名へのマッピング」
- 「15.1.4　メッセージのパートからパラメタおよび戻り値へのマッピング（wrapper スタイルの場合）」
- 「15.1.5　メッセージのパートからパラメタおよび戻り値へのマッピング（non-wrapper スタイルの場合）」

35.5.2　Web サービスクライアントの実装クラスを作成する

Web サービスを利用する Web サービスクライアントの実装クラスを作成します。

Web サービスに対して 1 回の呼び出しをする Web サービスクライアントの実装クラスの作成例を次に示します。

```
package com.example.sample.client;

import com.example.sample.TestJaxWs;
import com.example.sample.TestJaxWsService;
import com.example.sample.UserDefinedException;

public class TestClient {
    public static void main( String[] args ) {
        try {
            TestJaxWsService service = new TestJaxWsService();
            TestJaxWs port = service.getTestJaxWs();

            String returnValue = port.jaxWsTest1( "Invocation test.", 1003 );

            System.out.println( "[RESULT] " + returnValue );
        }
        catch( UserDefinedException e ){
            e.printStackTrace();
        }
    }

}
```

作成した TestClient.java は，UTF-8 形式で
c:¥temp¥jaxws¥works¥wsrm¥client¥src¥com¥example¥sample¥client¥ディレクトリに保存します。
com.example.sample，TestJaxWs，TestJaxWsService，TestJaxWs，および jaxWsTest1 は，生成された Java ソースのパッケージ名，クラス名，およびクラス内のメソッド名によって変わります。異なる構成の Web サービスを開発する場合には，パッケージ名，クラス名，およびクラス内のメソッド名の記述を見直す必要があります。

35.5.3 Web サービスクライアントの実装クラスにシーケンス終了処理を追加する

Web サービスクライアントの実装クラスに，シーケンス終了処理を追加します。シーケンス終了処理の追加例を次に示します。

```
package com.example.sample.client;

import com.example.sample.TestJaxWs;
import com.example.sample.TestJaxWsService;
import com.example.sample.UserDefinedException;
import com.sun.xml.ws.Closeable;

public class TestClient {
    public static void main( String[] args ) {
        TestJaxWsService service = null;
        TestJaxWs port = null;
        try {
            service = new TestJaxWsService();
            port = service.getTestJaxWs();

            String returnValue = port.jaxWsTest1( "Invocation test.", 1003 );

            System.out.println( "[RESULT] " + returnValue );
        }
        catch( UserDefinedException e ){
            e.printStackTrace();
        }
        finally {
            if( port != null ) {
                ((Closeable)port).close();
            }
        }
    }

}
```

35.5.4 Web サービスクライアントの実装クラスをコンパイルする

javac コマンドを使用して，作成した Web サービスクライアントの実装クラスをコンパイルします。

コンパイルの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥wsrm¥client¥
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥javaee¥1100¥lib¥javaee-api.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;.¥classes" -d .¥classes src¥com¥example¥sample¥client¥TestClient.java
```

javac コマンドが正常に終了すると，
c:¥temp¥jaxws¥works¥wsrm¥client¥classes¥com¥example¥sample¥client¥ディレクトリに，
TestClient.class が生成されます。

javac コマンドについては，JDK のドキュメントを参照してください。

35.6　Web サービスの実行例（WSDL 起点・WS-RM 1.2）

ここでは，WSDL を起点とした場合の Web サービスクライアントの実行例を説明します。

35.6.1　Java アプリケーション用オプション定義ファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用オプション定義ファイル（usrconf.cfg）を作成します。

Java アプリケーション用オプション定義ファイルの作成例を次に示します。

```
add.class.path=<Application Serverのインストールディレクトリ>¥jaxws¥lib¥cjjaxws.jar
add.class.path=.¥classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home=<Application Serverのインストールディレクトリ>
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

<Application Server のインストールディレクトリ>の部分は，Application Server をインストールしているパスを絶対パスで指定します。

作成した Java アプリケーション用オプション定義ファイルは，UTF-8 形式で c:¥temp¥jaxws¥works¥wsrm¥client¥ディレクトリに保存します。Java アプリケーション用オプション定義ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.1　usrconf.cfg（Java アプリケーション用オプション定義ファイル）」を参照してください。

35.6.2　Java アプリケーション用ユーザプロパティファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用ユーザプロパティファイルを作成します。

ここでは特に設定を変更しないため，c:¥temp¥jaxws¥works¥wsrm¥client¥ディレクトリに usrconf.properties という名称の空のファイルを作成します。Java アプリケーション用ユーザプロパティファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.2　usrconf.properties（Java アプリケーション用ユーザプロパティファイル）」を参照してください。

35.6.3　Web サービスクライアントを実行する

cjclstartap コマンドを使用して，Web サービスクライアントを実行します。

Web サービスクライアントの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥wsrm¥client¥
> "%COSMINEXUS_HOME%¥CC¥client¥bin¥cjclstartap"  com.example.sample.client.TestClient
```

cjclstartap コマンドが正常に終了すると，Web サービスクライアントの実行結果が表示されます。実行結果の表示例を次に示します。

```
KDJE40053-I The cjclstartap command will now start. (directory for the user definition file
= c:¥temp¥jaxws¥works¥wsrm¥client, PID = 2636)
--------------------
[RESULT] We've got your #1003 message "Invocation test."! It's 2007.11.28 14:50:50 now. See
ya!
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

イタリック体になっている個所は，実行したタイミングや環境によって変わります。

cjclstartap コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjclstartap（Java アプリケーションの開始）」を参照してください。

36 ハンドラフレームワーク

Application Server の JAX-WS 機能では，JAX-WS 2.2 仕様で規定されたハンドラフレームワークを使用して，Web サービスの機能を拡張できます。

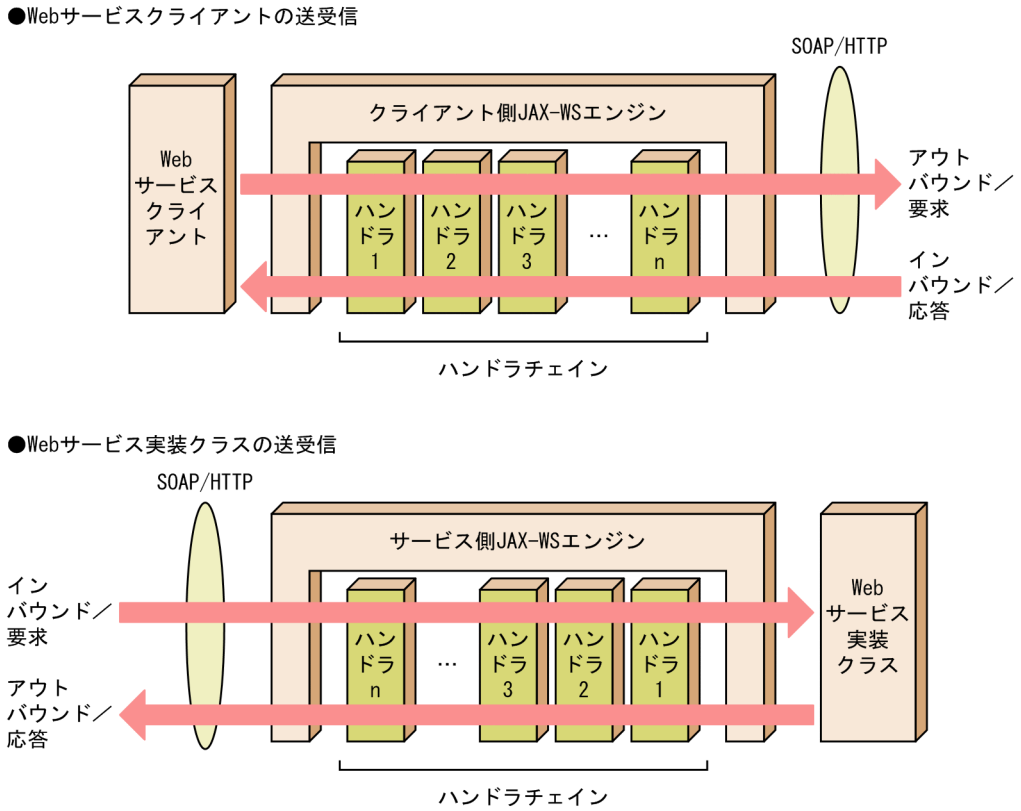
この章では，ハンドラフレームワークの概要，処理の流れ，および使用時の設定について説明します。

36.1　ハンドラフレームワークとは

ハンドラフレームワークとは，SOAP メッセージを送受信するときに，JAX-WS エンジン内で処理をインターセプトして，処理を追加する機能（フレームワーク）です。Web サービスクライアントと Web サービス実装クラスまたはプロバイダ実装クラスの間に複数のハンドラを追加し，Web サービスの機能を拡張できます。

ハンドラフレームワークの処理の流れを次の図に示します。

図 36-1　ハンドラフレームワークの処理の流れ



ハンドラの処理内容は，アウトバウンド／インバウンド，および要求／応答の組み合わせごとに異なります。組み合わせと処理内容を次の表に示します。

表 36-1　アウトバウンド／インバウンドおよび要求／応答の組み合わせと処理内容

アウトバウンド／インバウンド	要求／応答	処理内容
アウトバウンド	要求	Web サービスクライアントが要求メッセージを送信する動作を表します。 この処理で「メッセージを振り分ける」とは，要求メッセージを Web サービスに送信することを意味します。
	応答	Web サービス実装クラスまたはプロバイダ実装クラスが応答メッセージを送信する動作を表します。

アウトバウンド／インバウンド	要求／応答	処理内容
		この処理で「メッセージを振り分ける」とは，応答メッセージを Web サービスクライアントに送信することを意味します。ただし，one-way オペレーションでは応答メッセージを送信しないため，アウトバウンドは動作しません。
インバウンド	要求	Web サービス実装クラスまたはプロバイダ実装クラスが要求メッセージを受信する動作を表します。 この処理で「メッセージを振り分ける」とは，要求メッセージを Web サービス実装クラスまたはプロバイダ実装クラスに割り当てることを意味します。
	応答	Web サービスクライアントが応答メッセージを受信する動作を表します。 この処理で「メッセージを振り分ける」とは，応答メッセージを Web サービスクライアントに割り当てることを意味します。ただし，one-way オペレーションでは応答メッセージを受信しないため，インバウンドは動作しません。

ハンドラでは，メッセージコンテキストを取得できます。メッセージコンテキストについては，「19.2.5 メッセージコンテキストの使用」を参照してください。

ハンドラフレームワークのアーキテクチャについては，JAX-WS 2.2 仕様を参照してください。

36.2 Web サービスセキュリティ機能を使用する場合の注意事項

Web サービスクライアントに Web Services - Security（Web サービスセキュリティ機能）を適用する場合， Web サービスセキュリティハンドラとこの章で説明するハンドラフレームワークとを，併用できません。

Web Services - Security および Web サービスセキュリティハンドラについては，マニュアル「アプリケーションサーバ Web サービスセキュリティ構築ガイド」を参照してください。

36.3　EJB の Web サービスに適用する場合の注意事項

EJB の Web サービスにハンドラフレームワークを適用する場合，EJB の Web サービスに適用されたハンドラフレームワークが Web コンテナで動作するため，EJB コンテナが提供する機能を同時に適用できません。EJB コンテナが提供する機能については，「10.19　EJB の Web サービス呼び出し」を参照してください。

36.4 ハンドラの型

JAX-WS 2.2 仕様は，論理ハンドラとプロトコルハンドラの二つを定義しています。また，プロトコルハンドラは，SOAP ハンドラを定義しています。それぞれのハンドラの説明を次に示します。

論理ハンドラ

javax.xml.ws.handler.LogicalHandler インタフェースを実装するハンドラです。

プロトコルハンドラ

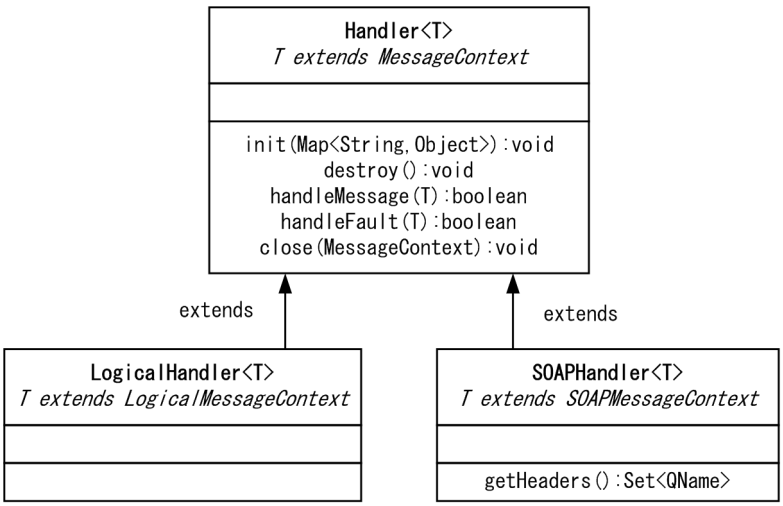
javax.xml.ws.handler.LogicalHandler インタフェースを除く，javax.xml.ws.handler.Handler のすべての継承インタフェースを実装するハンドラです。

SOAP ハンドラ

javax.xml.ws.handler.soap.SOAPHandler インタフェースを実装するハンドラです。

ハンドラの関係（クラス階層）を次の図に示します。

図 36–2 ハンドラのクラス階層



Application Server の JAX-WS 機能では，論理ハンドラと SOAP ハンドラを使用できます。

論理ハンドラでも SOAP ハンドラでもないハンドラをハンドラチェインに設定した場合，Web サービス側の JAX-WS エンジンでは，Web サービス初期化時にログおよび標準エラー出力にエラーメッセージが出力されます（KDJW00009-E）。Web サービスクライアント側の JAX-WS エンジンでは，ポートを取得しようとしたときに，javax.xml.ws.WebServiceException がスローされます。

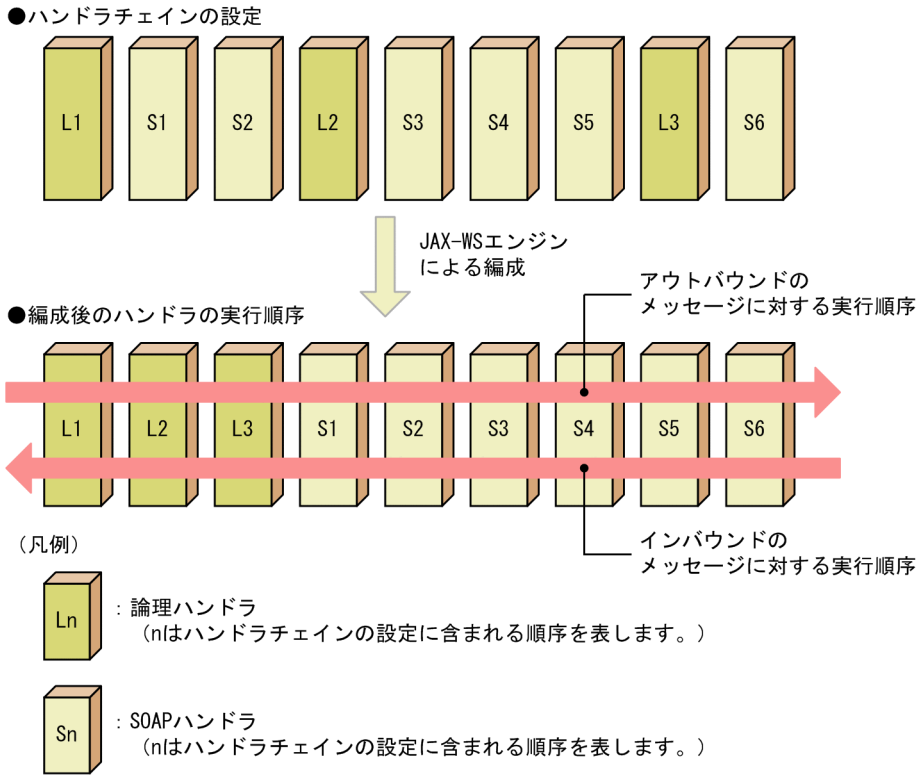
javax.xml.ws.handler.LogicalHandler インタフェースおよび
javax.xml.ws.handler.soap.SOAPHandler インタフェースの両方を実装するハンドラ（論理ハンドラでも SOAP ハンドラでもあるハンドラ）は論理ハンドラと見なされます。

36.5　ハンドラチェインの編成と実行順序

ハンドラチェインは，論理ハンドラが SOAP ハンドラに先行するように編成されます。論理ハンドラ同士，および SOAP ハンドラ同士の順序は，ハンドラチェインの設定に含まれる順序に従います。

ハンドラチェインの編成の例を次に示します。

図 36–3　ハンドラチェインの編成の例



JAX-WS エンジンでは，この図に示すように，次の順序でハンドラが実行されます。

アウトバウンドのメッセージの場合：

L1→L2→L3→S1→S2→S3→S4→S5→S6

インバウンドのメッセージの場合：

S6→S5→S4→S3→S2→S1→L3→L2→L1

アウトバウンドのメッセージに対しては，最初のハンドラからハンドラチェインの順序に，インバウンドのメッセージに対しては，最後のハンドラからハンドラチェインの順序とは逆順に実行されます。

ただし，ハンドラによって例外がスローされた場合，または handleMessage メソッドまたは handleFault メソッドによって false が返された場合は，ハンドラ処理の方向が変わります。

ここでは，handleMessage メソッド，handleFault メソッド，および close メソッドごとに処理についてそれぞれ説明します。

36.5.1　handleMessage メソッドの処理

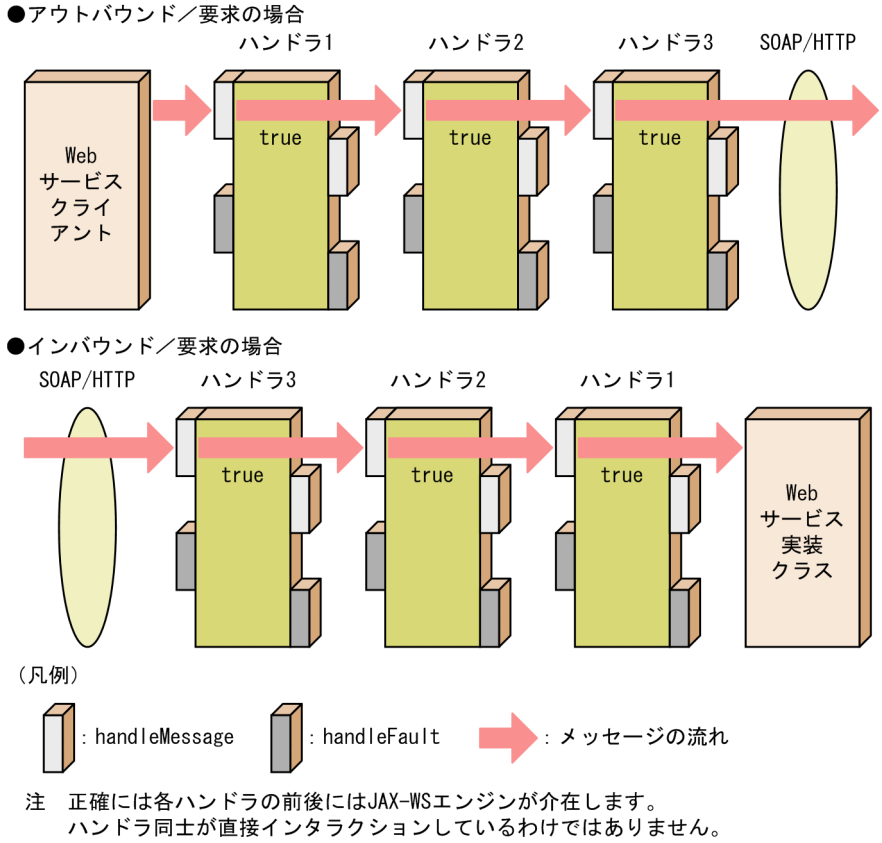
handleMessage メソッドについて，true を返す場合，false を返す場合，および例外をスローする場合の処理について説明します。

(1)　true を返す場合

handleMessage メソッドが true を返す場合，JAX-WS エンジンは現在実行中の方向のままハンドラを処理します。それ以上ハンドラがなければ，メッセージを振り分けます。

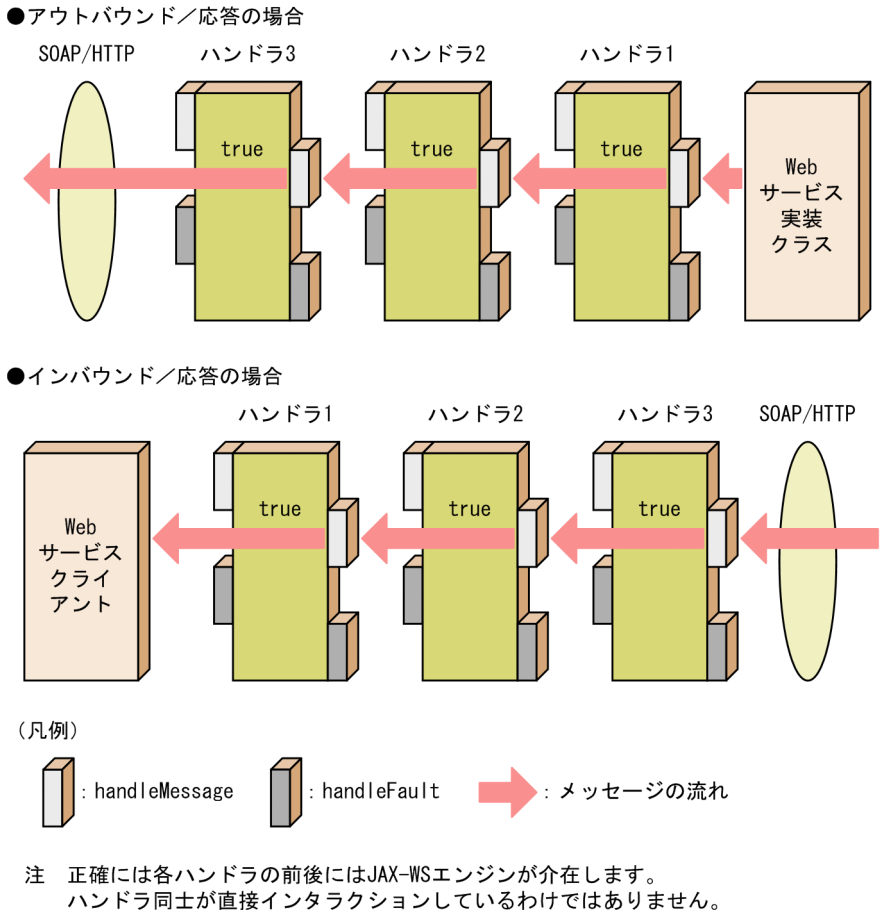
要求処理の流れを次の図に示します。

図 36-4　handleMessage で true を返す場合の処理（要求）



応答処理の流れを次の図に示します。

図 36-5　handleMessage で true を返す場合の処理（応答）



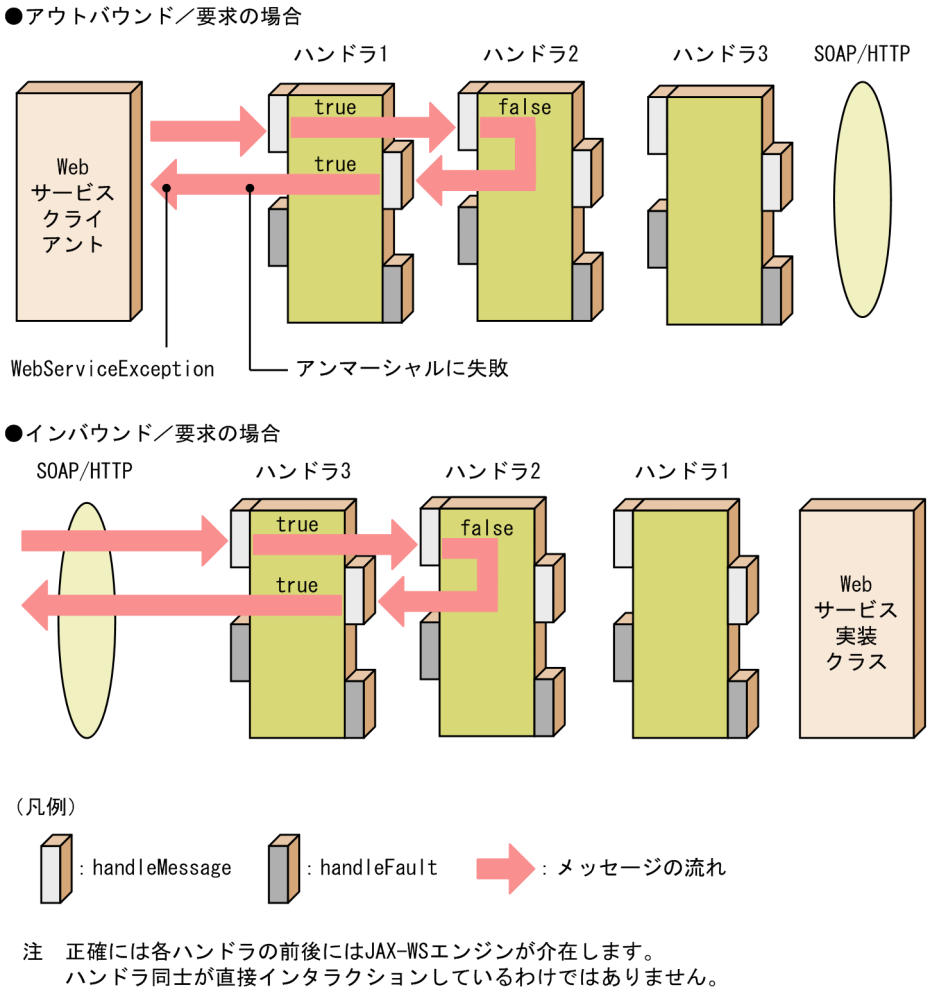
注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

(2)　false を返す場合

要求メッセージを処理している handleMessage メソッドが false を返す場合，JAX-WS エンジンはメッセージとハンドラ処理の方向を逆転させて，前のハンドラの handleMessage メソッドを呼び出します。それ以上ハンドラがなければ，メッセージを振り分けます。

要求処理でハンドラ 2 が false を返す場合の流れを次の図に示します。

図 36-6　handleMessage で false を返す場合の処理（要求）



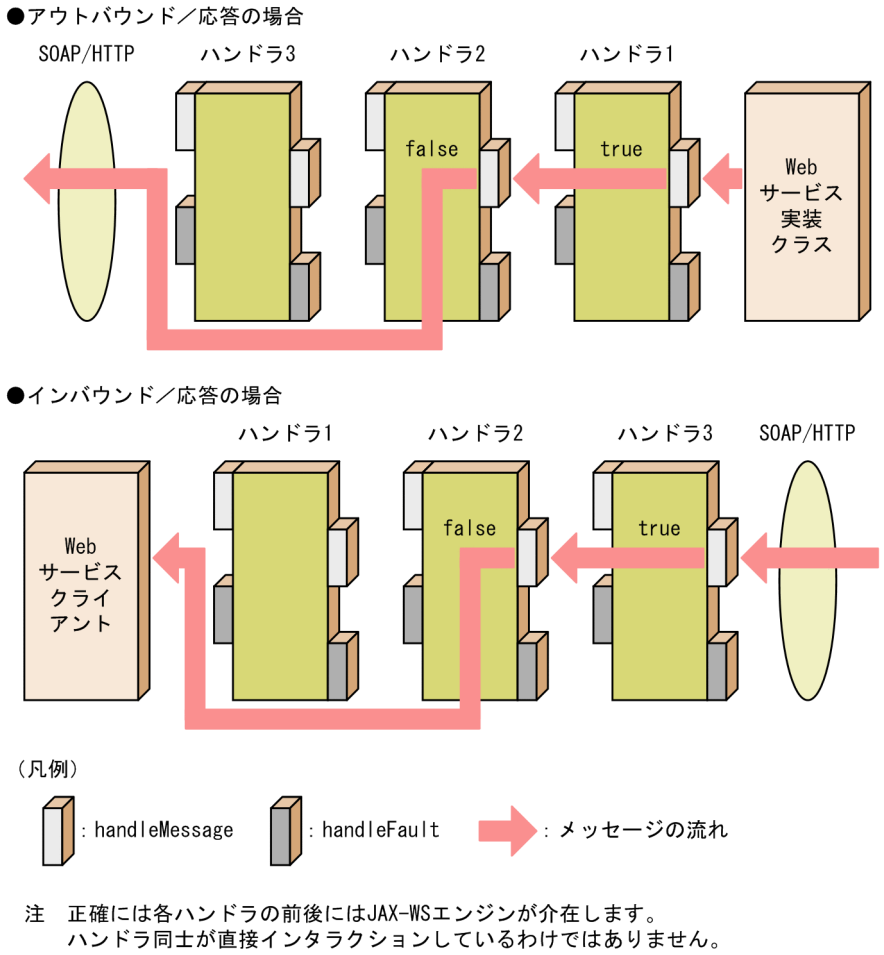
注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

JAX-WS エンジンは，メッセージコンテキストに設定されたメッセージを変更しません。したがって，要求メッセージはそのまま Web サービスクライアントに振り分けられます。意図した応答メッセージが振り分けられなかった Web サービスクライアントでは，javax.xml.ws.WebServiceException がスローされます。

応答メッセージを処理している handleMessage メソッドが false を返す場合，メッセージの方向は変えません。ただし，以降のハンドラをすべて省略して，メッセージを振り分けます。

応答処理でハンドラ 2 が false を返す場合の流れを次の図に示します。

図 36-7　handleMessage で false を返す場合の処理（応答）



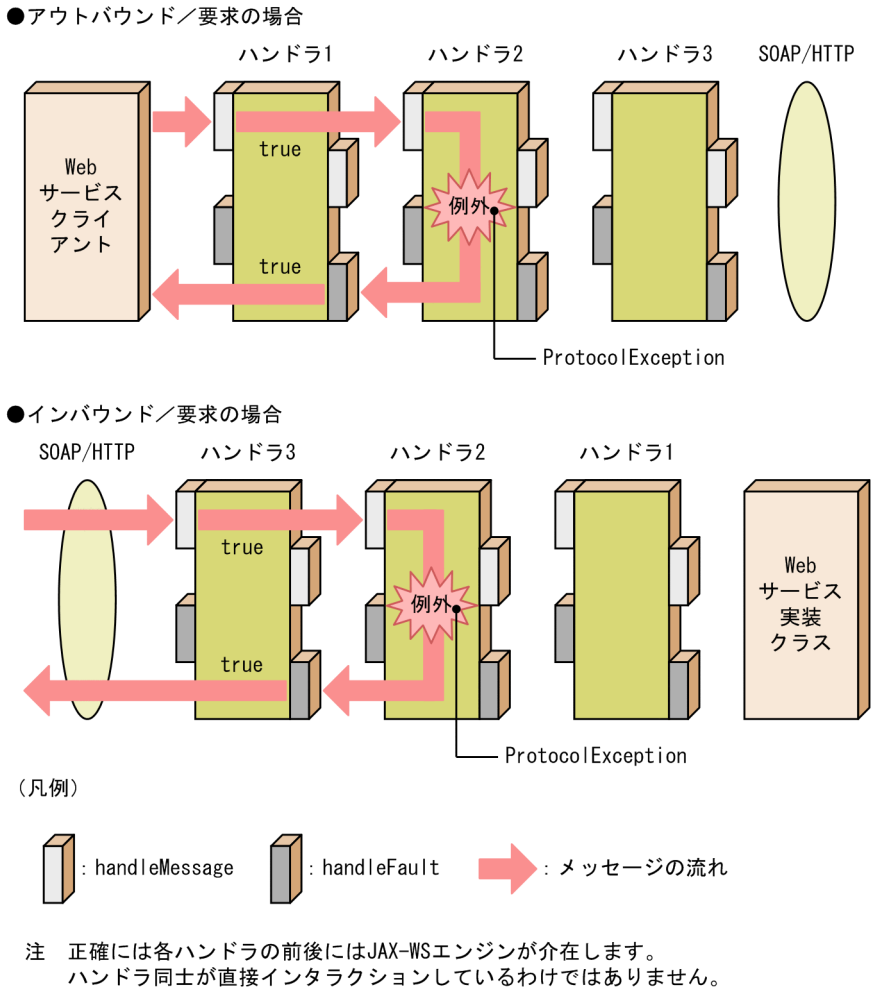
注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

(3)　ProtocolException またはそのサブクラスをスローする場合

要求メッセージを処理している handleMessage メソッドが ProtocolException またはそのサブクラスをスローする場合，JAX-WS エンジンはメッセージとハンドラ処理の方向を逆転させて，前のハンドラの handleFault メソッドを呼び出します。それ以上ハンドラがなければ，メッセージを振り分けます。

要求処理でハンドラ 2 が ProtocolException をスローする場合の流れを次の図に示します。

図 36-8　handleMessage で ProtocolException をスローする場合の処理（要求）



注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

JAX-WS エンジンは，フォルトメッセージを生成して伝搬させます。生成するメッセージは，SOAP のバージョンによって次の表のとおりに異なります。

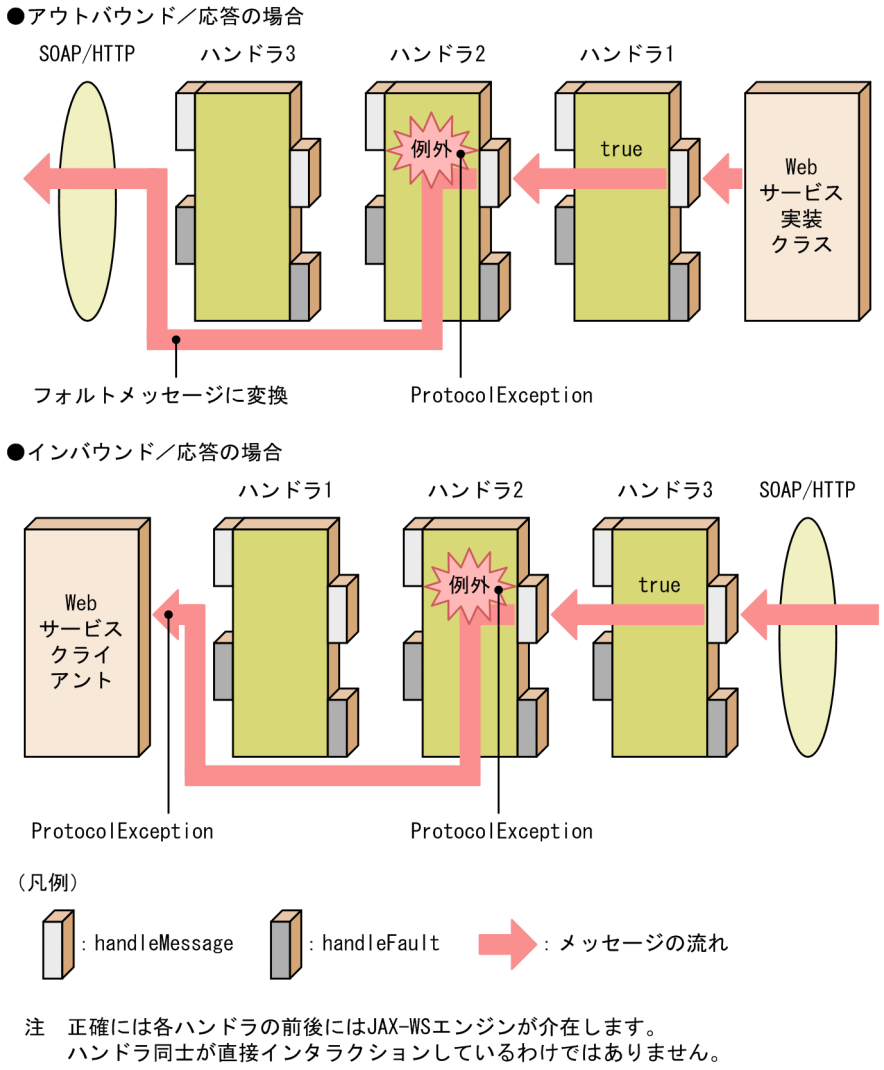
表 36-2　JAX-WS エンジンが生成するフォルトメッセージ（ProtocolException スロー・要求）

バージョン	フォルトメッセージ	
SOAP 1.1 仕様	Web サービスクライアント側	faultcode は QName soapenv:Client
	Web サービス側	faultcode は QName soapenv:Server
SOAP 1.2 仕様	Web サービスクライアント側	soapenv12:Code は QName soapenv12:Sender
	Web サービス側	soapenv12:Code は QName soapenv12:Receiver

応答メッセージを処理している handleMessage メソッドが ProtocolException またはそのサブクラスをスローする場合，メッセージの方向は変えません。ただし，以降のハンドラをすべて省略して，例外をそのままスローします。

応答処理でハンドラ 2 が ProtocolException をスローする場合の流れを次の図に示します。

図 36-9　handleMessage で ProtocolException をスローする場合の処理（応答）



注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

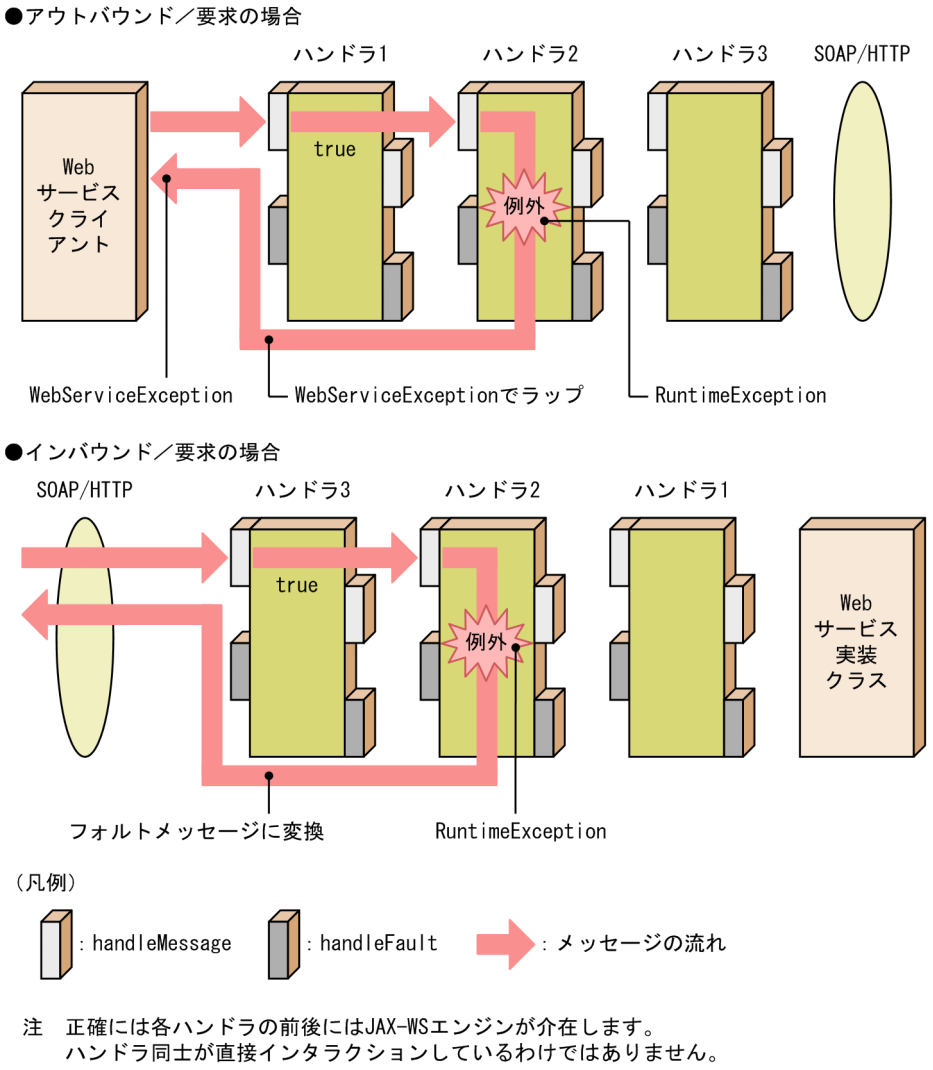
Web サービス側の場合，JAX-WS エンジンは例外を SOAP フォルトに変換して送信します。フォルトメッセージの変換については，「10.4.1(2)　ランタイム例外のバインディング」を参照してください。

(4)　ほかのランタイム例外をスローする場合

要求メッセージを処理している handleMessage メソッドがほかのランタイム例外をスローする場合，JAX-WS エンジンはメッセージとハンドラ処理の方向を逆転させます。また，以降のハンドラをすべて省略し，例外をそのままスローします。

要求処理でハンドラ 2 が RuntimeException をスローする場合の流れを次の図に示します。

図 36-10　handleMessage で RuntimeException をスローする場合の処理（要求）



注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

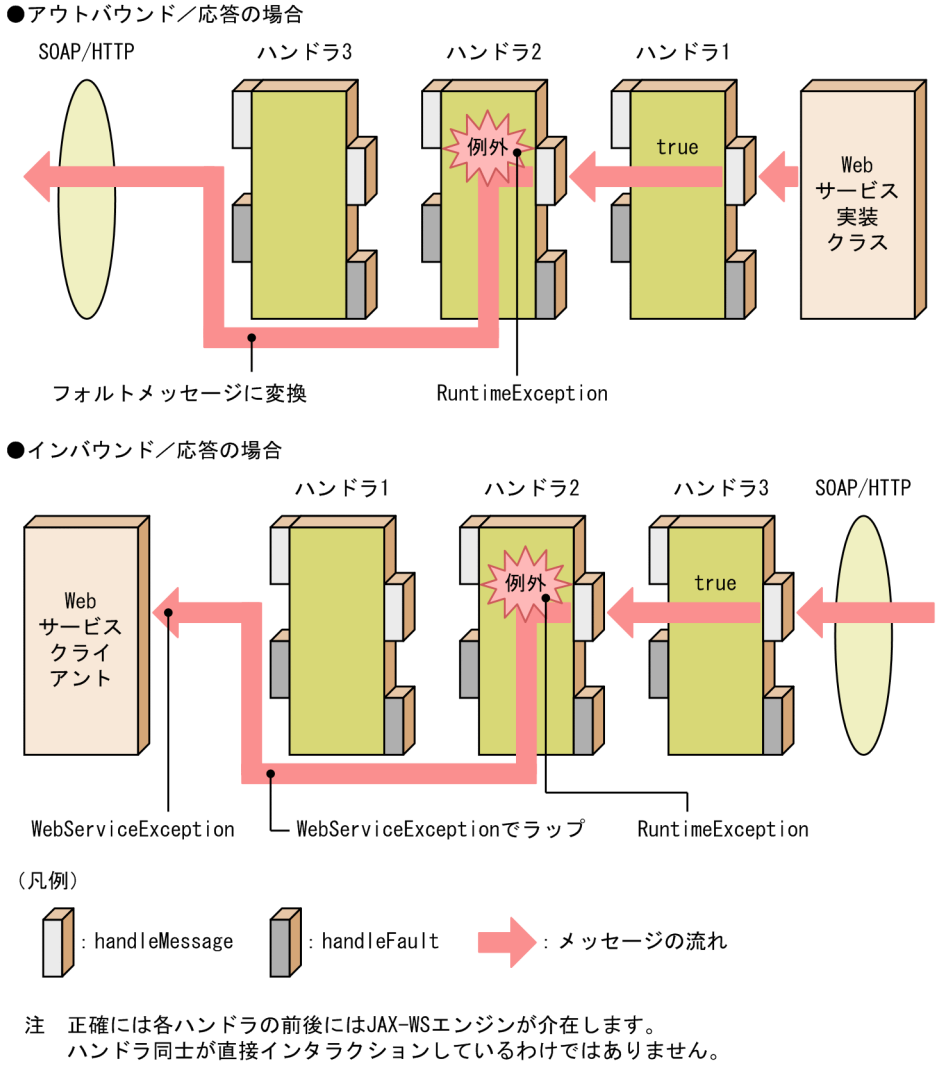
Web サービス側の場合，JAX-WS エンジンは例外を SOAP フォルトに変換して送信します。フォルトメッセージの変換については，「10.4.1(2)　ランタイム例外のバインディング」を参照してください。

Web サービスクライアント側の場合，JAX-WS エンジンは例外を javax.xml.ws.WebServiceException でラップして，WebServiceException をスローします。

応答メッセージを処理している handleMessage メソッドがランタイム例外をスローする場合，メッセージの方向は変えません。ただし，以降のハンドラをすべて省略して，メッセージを振り分けます。

応答処理でハンドラ 2 が RuntimeException をスローする場合の流れを次の図に示します。

図 36-11　handleMessage で RuntimeException をスローする場合の処理（応答）



注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
　　ハンドラ同士が直接インタラクションしているわけではありません。

Web サービス側の場合，JAX-WS エンジンは例外を SOAP フォルトに変換して送信します。フォルトメッセージの変換については，「10.4.1(2)　ランタイム例外のバインディング」を参照してください。

Web サービスクライアント側の場合，JAX-WS エンジンは例外を javax.xml.ws.WebServiceException でラップして，WebServiceException をスローします。

36.5.2　handleFault メソッドの処理

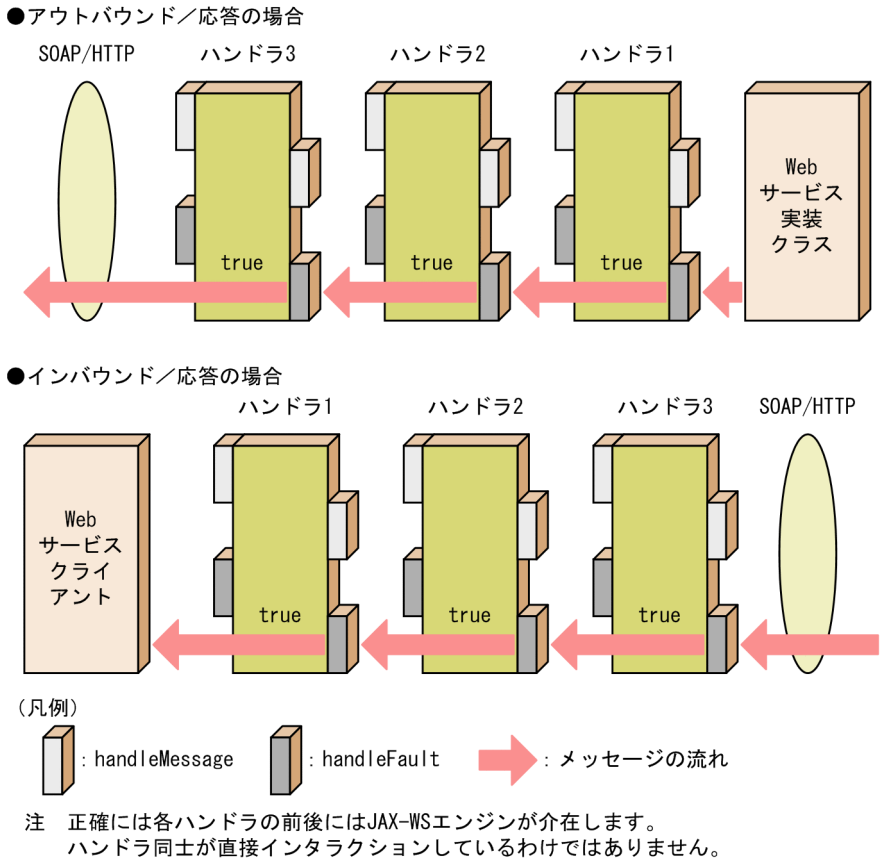
handleMessage メソッドについて，true を返す場合，false を返す場合，および例外をスローする場合の処理について説明します。

(1)　true を返す場合

handleFault メソッドが true を返す場合，JAX-WS エンジンは現在実行中の方向のままハンドラを処理します。それ以上ハンドラがなければ，メッセージを振り分けます。

応答処理の流れを次の図に示します。

図 36-12　handleFault で true を返す場合の処理（応答）



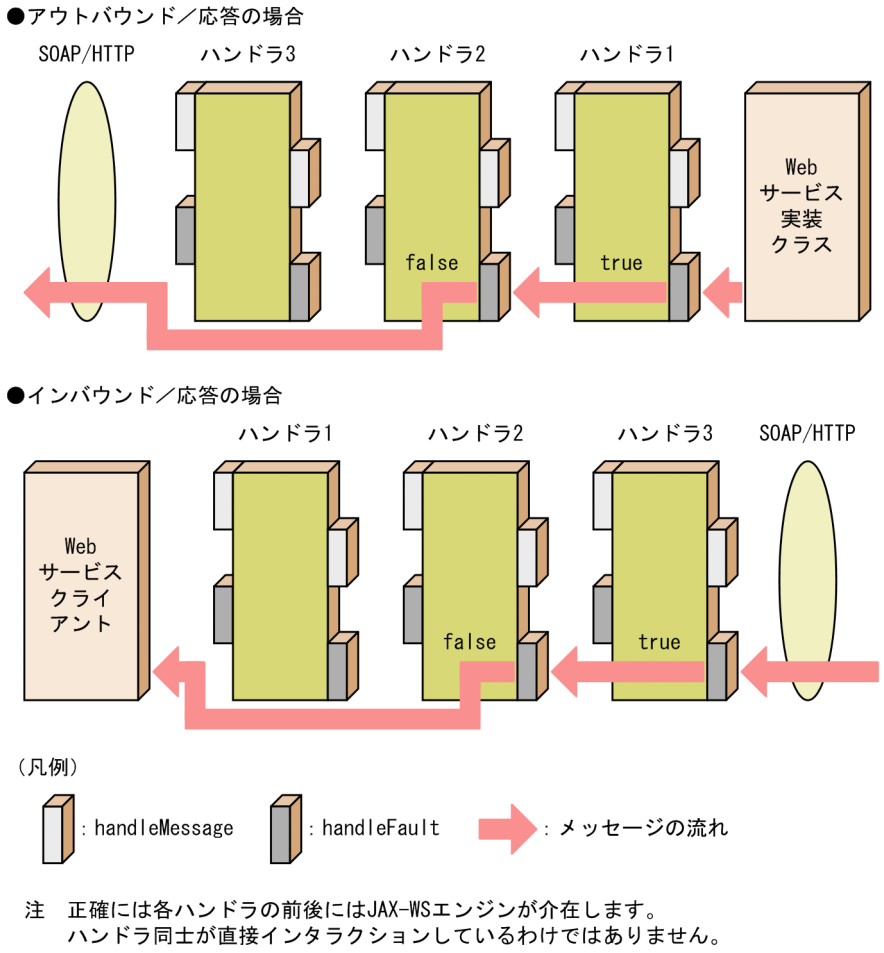
注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

(2)　false を返す場合

handleFault メソッドが false を返す場合，JAX-WS エンジンは現在実行中の方向のまま，以降のハンドラをすべて省略してメッセージを振り分けます。

応答処理でハンドラ 2 が false を返す場合の流れを次の図に示します。

図 36-13　handleFault で false を返す場合の処理（応答）



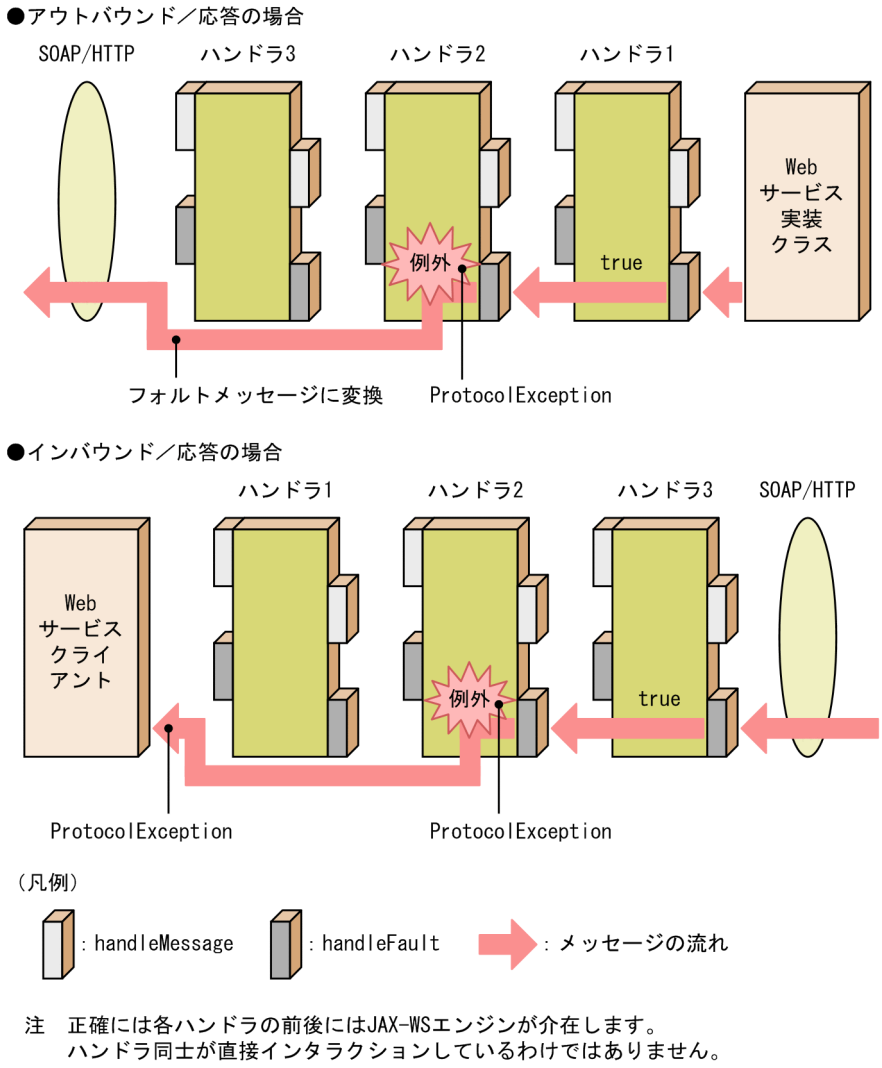
注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

(3)　ProtocolException またはそのサブクラスをスローする場合

handleFault メソッドが ProtocolException またはそのサブクラスをスローする場合，JAX-WS エンジンは，現在実行中の方向のまま，以降のハンドラをすべて省略して例外をスローします。

応答処理でハンドラ 2 が ProtocolException をスローする場合の流れを次の図に示します。

図 36-14　handleFault で ProtocolException を返す場合の処理（応答）



注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

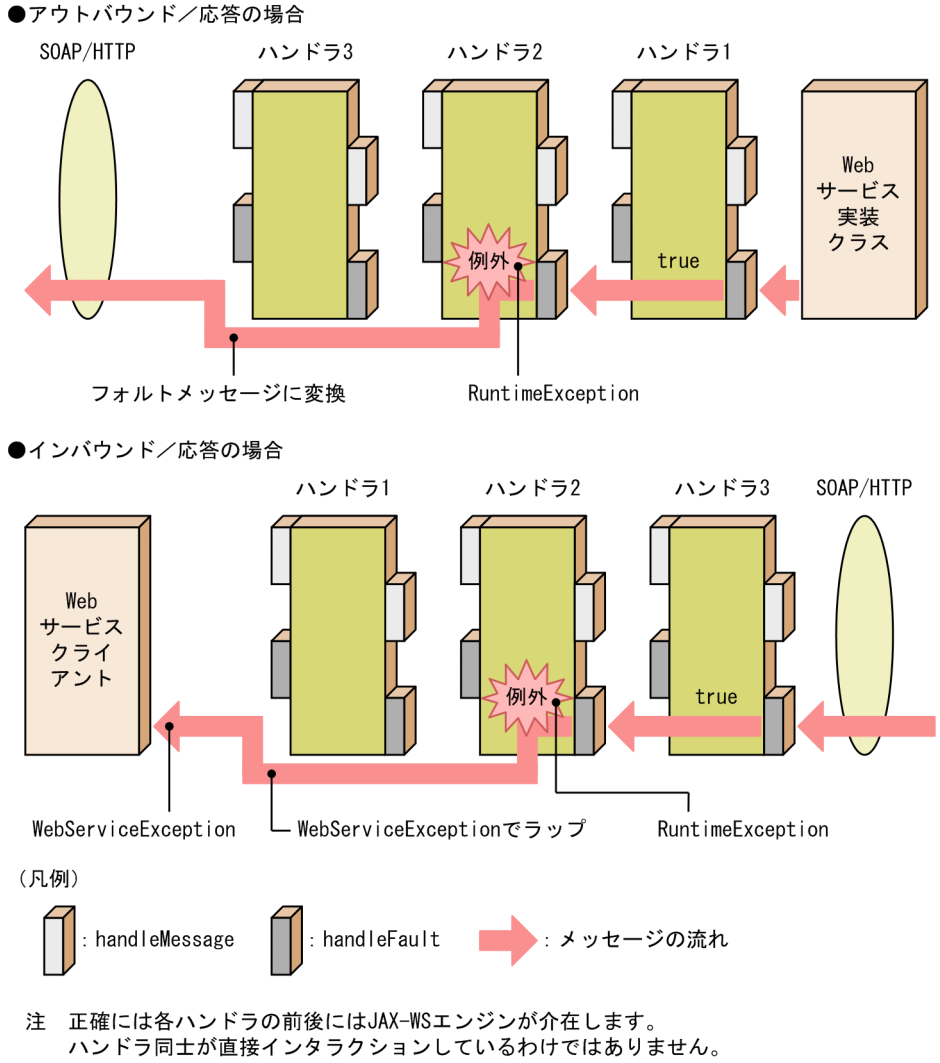
Web サービス側の場合，JAX-WS エンジンは例外を SOAP フォルトに変換し，もともと保持している SOAP フォルトを置き換えます。したがって，実際は新たに生成されたフォルトメッセージを送信します。フォルトメッセージの変換については，「10.4.1(2)　ランタイム例外のバインディング」を参照してください。

(4)　ほかのランタイム例外をスローする場合

handleFault メソッドがほかのランタイム例外をスローする場合，JAX-WS エンジンは，現在実行中の方向のまま，以降のハンドラをすべて省略してメッセージを振り分けます。

応答処理でハンドラ 2 が RuntimeException をスローする場合の流れを次の図に示します。

図 36-15　handleFault で RuntimeException を返す場合の処理（応答）



注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

Web サービス側の場合，JAX-WS エンジンは例外を SOAP フォルトに変換し，もともと保持している SOAP フォルトを置き換えます。したがって，実際には新たに生成されたフォルトメッセージを送信します。フォルトメッセージの変換については，「10.4.1(2)　ランタイム例外のバインディング」を参照してください。

Web サービスクライアント側の場合，JAX-WS エンジンは例外を javax.xml.ws.WebServiceException でラップして，WebServiceException をスローします。

36.5.3　close メソッドの処理

JAX-WS エンジンは，応答メッセージを Web サービスクライアントに振り分ける直前に，すでに呼び出したハンドラの close メソッドを呼び出します。

close メソッドは，呼び出しと逆順に呼び出されます。したがって，要求メッセージを処理しているときにハンドラの処理を逆転させた場合，呼び出さなかったハンドラの close メソッドは呼び出しません。しかし，要求メッセージを処理するときにすべてのハンドラを実行していれば，応答メッセージの処理でハ

ンドラの呼び出しを省略したとしても，すべてのハンドラの close メソッドを，要求メッセージを処理したときと逆順に呼び出します。

また，one-way オペレーションの場合は，Web サービス側と Web クライアント側で次のように呼び出す順序が異なります。

Web サービス側

Web サービス実装クラスまたはプロバイダ実装クラスの呼び出し後に，すでに呼び出したハンドラの close メソッドを，one-way メッセージを処理したときと逆順に呼び出します。

Web クライアント側

HTTP レスポンス受信後に，すでに呼び出したハンドラの close メソッドを，one-way メッセージを処理したときと逆順に呼び出します。

36.6　ハンドラの初期化と破棄

ハンドラを初期化するとき，およびハンドラを破棄するときの JAX-WS エンジンの動作について説明します。

Web サービス側

ハンドラが javax.annotation.PostConstruct アノテーションでアノテートされたメソッドを持つ場合，Web サービスを初期化するときに，そのメソッドが呼び出されます。

ハンドラが javax.annotation.PreDestory アノテーションでアノテートされたメソッドを持つ場合，Web サービスを破棄するときに，そのメソッドが呼び出されます。

Web サービスクライアント側

ハンドラが javax.annotation.PostConstruct アノテーションや javax.annotation.PreDestory アノテーションでアノテートされたメソッドを持っていても，それらのメソッドは呼び出されません。

36.7　SOAP ヘッダが含まれる場合のハンドラの動作と設定

ハンドラ，Web サービス実装クラス，およびスタブベースの Web サービスクライアントでは，SOAP メッセージに処理できる SOAP ヘッダを設定できます。プロバイダ実装クラスおよびディスパッチベースの Web サービスクライアントに SOAP ヘッダを設定しても，処理できないので注意してください。

ここでは，Web サービス側および Web サービスクライアント側に分けて，SOAP ヘッダを含む SOAP メッセージを受信した場合のハンドラの動作について説明します。また，SOAP ヘッダの設定方法について説明します。

36.7.1　SOAP ヘッダが含まれる場合のハンドラの動作（Web サービス側）

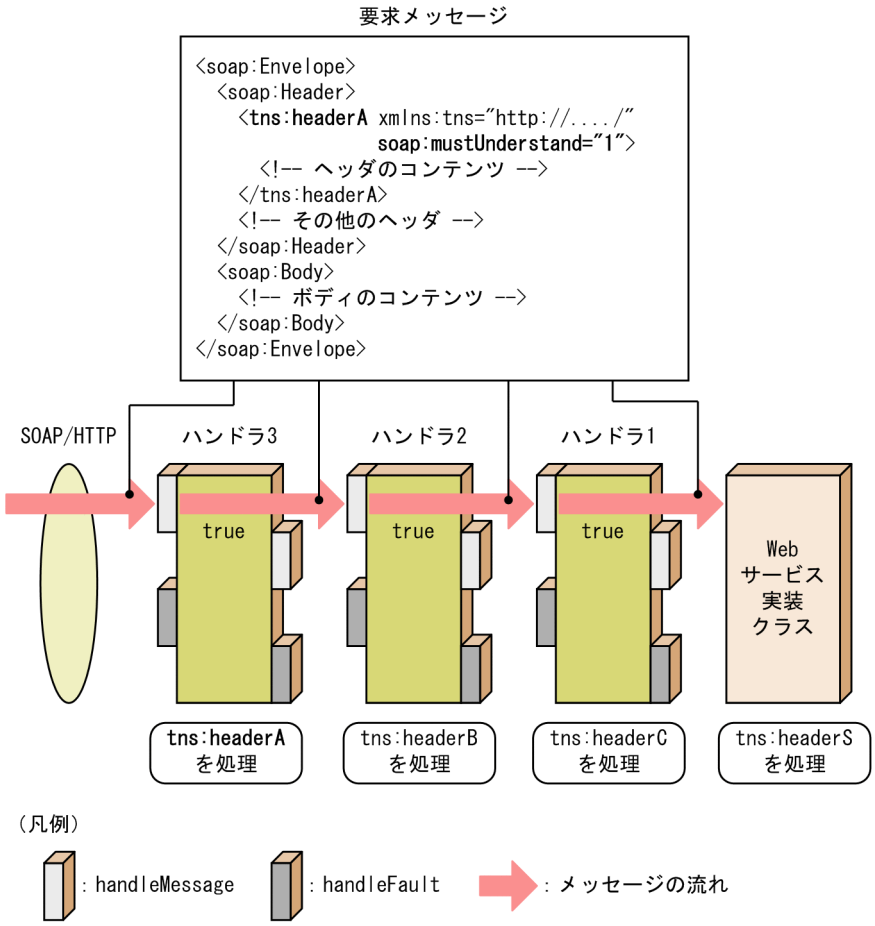
SOAP ヘッダが含まれる SOAP メッセージを受信した場合，SOAP ヘッダの soap:mustUnderstand 属性の設定値によって，ハンドラの処理が異なります。soap:mustUnderstand 属性が"1"の場合，および soap:mustUnderstand 属性が"1"以外の場合に分けて処理内容を示します。

(1)　soap:mustUnderstand 属性が"1"の場合

soap:mustUnderstand 属性が"1"の SOAP ヘッダを持つ SOAP メッセージを受信した場合，設定されたハンドラのどれか，または Web サービス実装クラス内で，その SOAP ヘッダを処理できれば，すべてのハンドラが呼び出されます。

ハンドラで SOAP ヘッダを処理できる場合のハンドラの処理を次の図に示します。

図 36-16　ハンドラで SOAP ヘッダを処理できる場合の処理（Web サービス側）

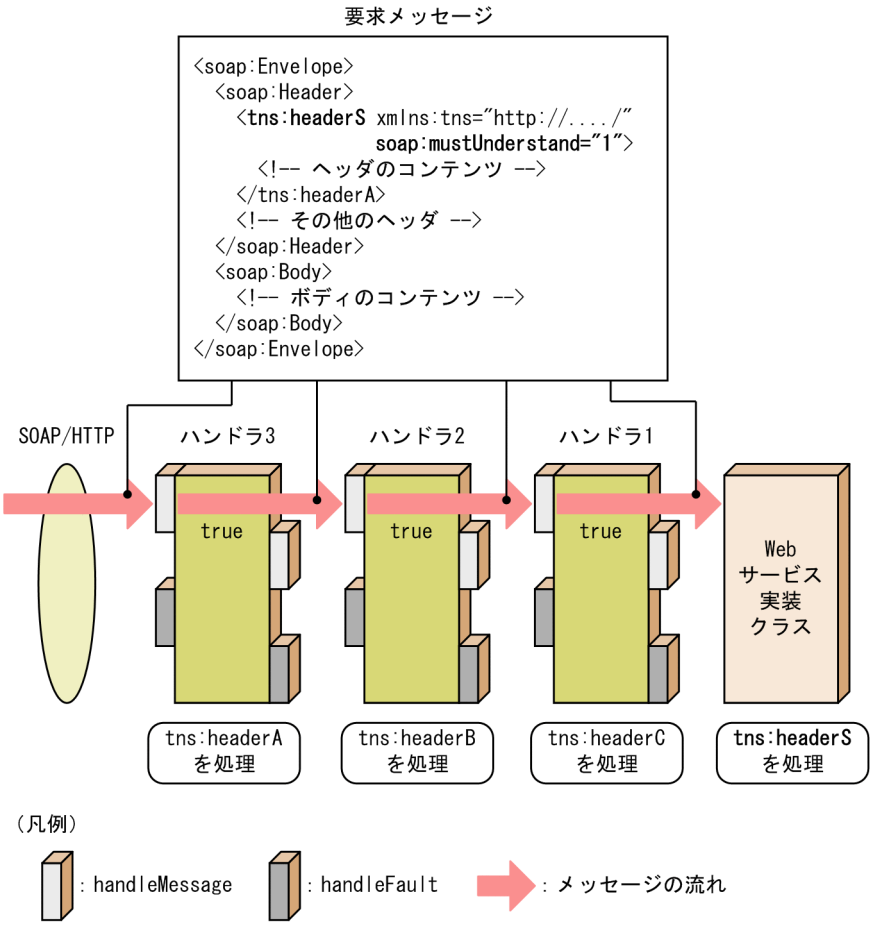


注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

この図の例では，要素名 tns:headerA を処理できるハンドラ 3 が設定されているとしています。このとき，tns:headerA の SOAP ヘッダを受信した場合，ハンドラ 3，ハンドラ 2，およびハンドラ 1 のすべてのハンドラが呼び出されます。

Web サービス実装クラスで SOAP ヘッダを処理できる場合のハンドラの処理を次の図に示します。

図 36-17　Web サービス実装クラスで SOAP ヘッダを処理できる場合の処理（Web サービス側）



注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

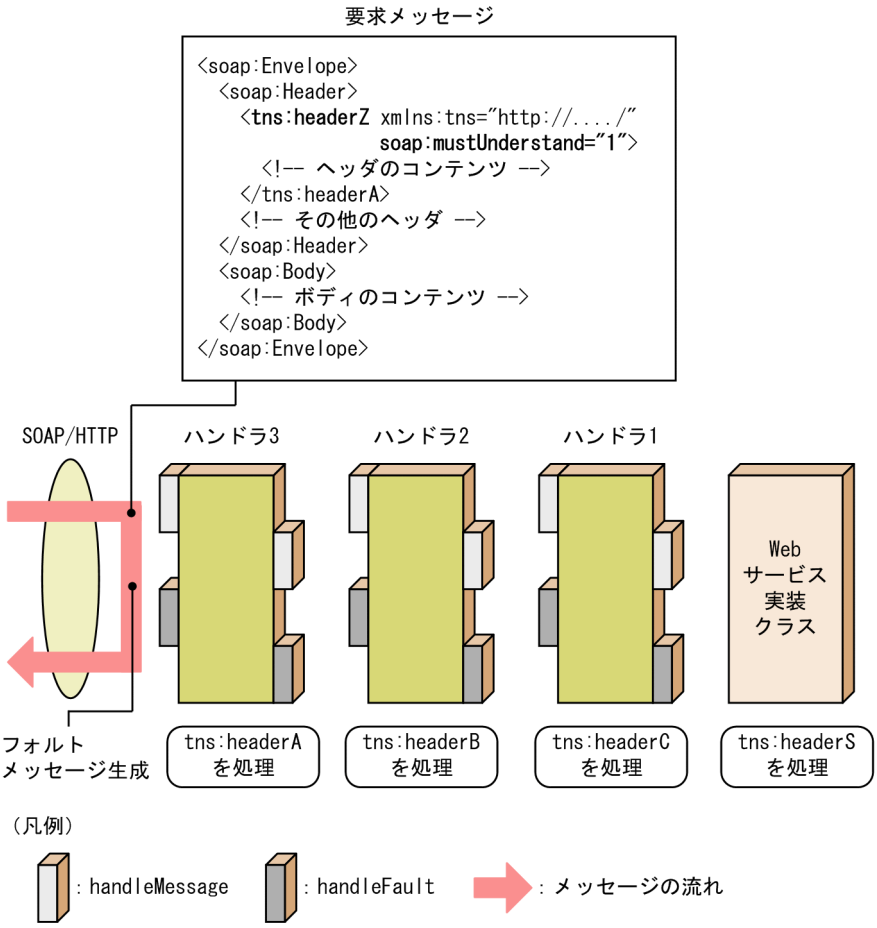
Web サービス実装クラスだけが要素名 tns:headerS を処理できるとした場合も，ハンドラで処理できる場合と同様にすべてのハンドラが呼び出されます。

soap:mustUnderstand 属性が"1"の SOAP ヘッダを持つ SOAP メッセージを受信した場合，次のようなときは faultcode に soap:MustUnderstand が設定された SOAP フォルトが返されます。

- 設定されたハンドラのどれかが SOAP ヘッダを処理できない
- Web サービス実装クラス内で SOAP ヘッダを処理できない
- プロバイダ実装クラスが使用されている

SOAP ヘッダを処理できない場合のハンドラの処理は，次の図のようになります。

図 36-18　SOAP ヘッダを処理できない場合のハンドラの処理（Web サービス側）



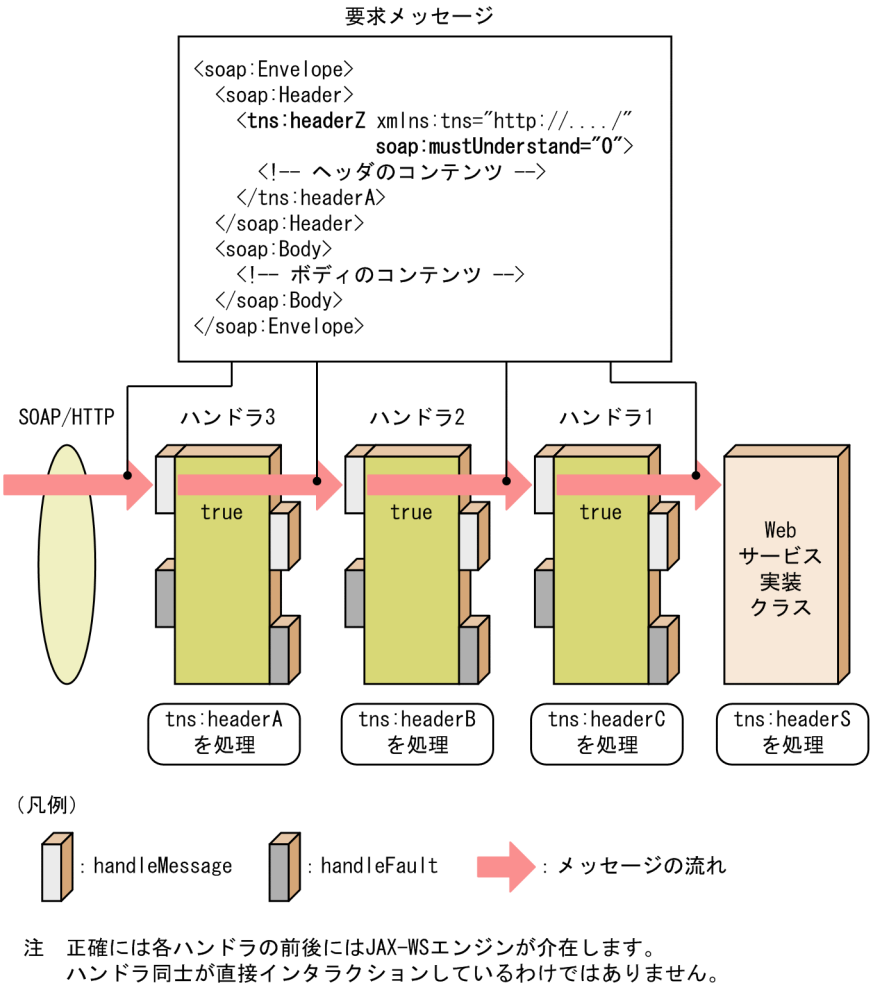
注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

(2)　soap:mustUnderstand 属性が"1"以外の場合

soap:mustUnderstand 属性が"1"以外の場合，設定されているすべてのハンドラが呼び出されます。

soap:mustUnderstand 属性が"1"以外の場合，ハンドラの処理は次の図のようになります。

図 36-19　soap:mustUnderstand 属性が"1"以外の場合のハンドラの処理（Web サービス側）



注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

この図の例では，ハンドラも Web サービス実装クラスも要素 tns:headerZ を処理できませんが，soap:mustUnderstand 属性が"1"以外の場合は SOAP フォルトにはならないで，すべてのハンドラが呼び出されます。

36.7.2　SOAP ヘッダが含まれる場合のハンドラの動作（Web サービスクライアント側）

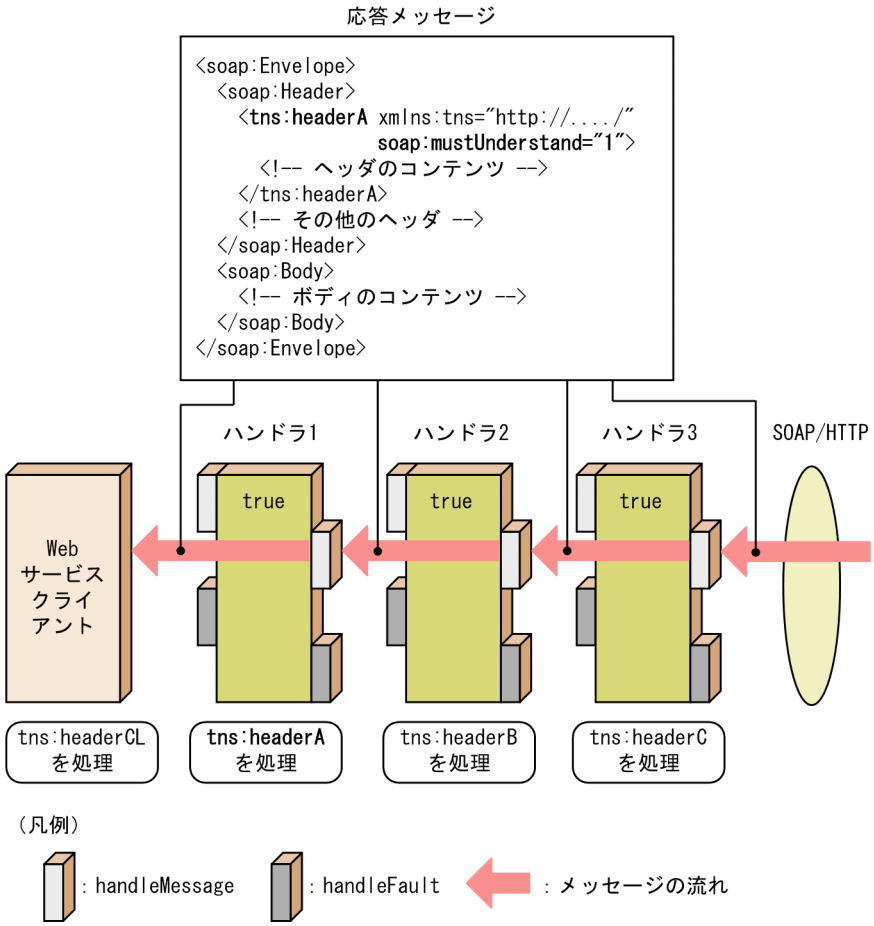
SOAP ヘッダが含まれる SOAP メッセージを受信した場合，SOAP ヘッダの soap:mustUnderstand 属性の設定値によって，ハンドラの処理が異なります。soap:mustUnderstand 属性が"1"の場合，および soap:mustUnderstand 属性が"1"以外の場合に分けて処理内容を示します。

(1)　soap:mustUnderstand 属性が"1"の場合

soap:mustUnderstand 属性が"1"の SOAP ヘッダを持つ SOAP メッセージを受信した場合，設定されたハンドラのどれか，または Web サービスクライアント内で，その SOAP ヘッダを処理できれば，すべてのハンドラが呼び出されます。

ハンドラで SOAP ヘッダを処理できる場合のハンドラの処理を次の図に示します。

図 36-20　ハンドラで SOAP ヘッダを処理できる場合の処理（Web サービスクライアント側）

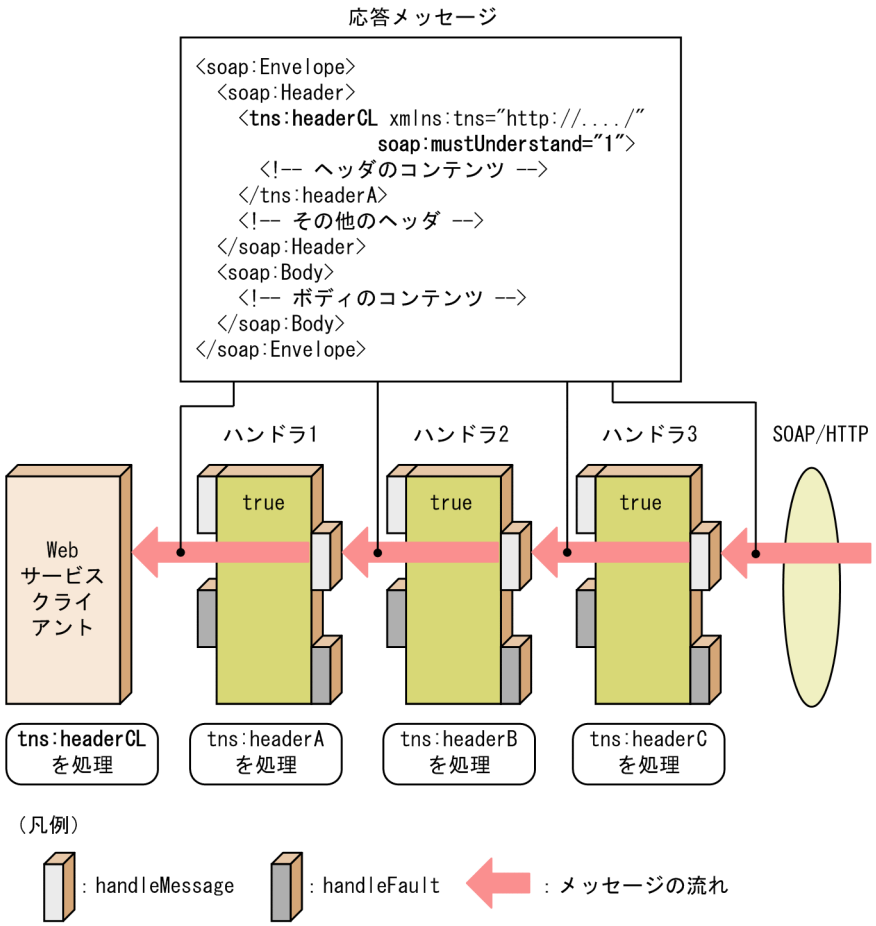


注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

この図の例では，要素名 tns:headerA を処理できるハンドラ 3 が設定されているとしています。このとき，tns:headerA の SOAP ヘッダを受信した場合，ハンドラ 3，ハンドラ 2，およびハンドラ 1 のすべてのハンドラが呼び出されます。

Web サービス実装クラスで SOAP ヘッダを処理できる場合のハンドラの処理を次の図に示します。

図 36-21　Web サービス実装クラスで SOAP ヘッダを処理できる場合の処理（Web サービスクライアント側）



注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

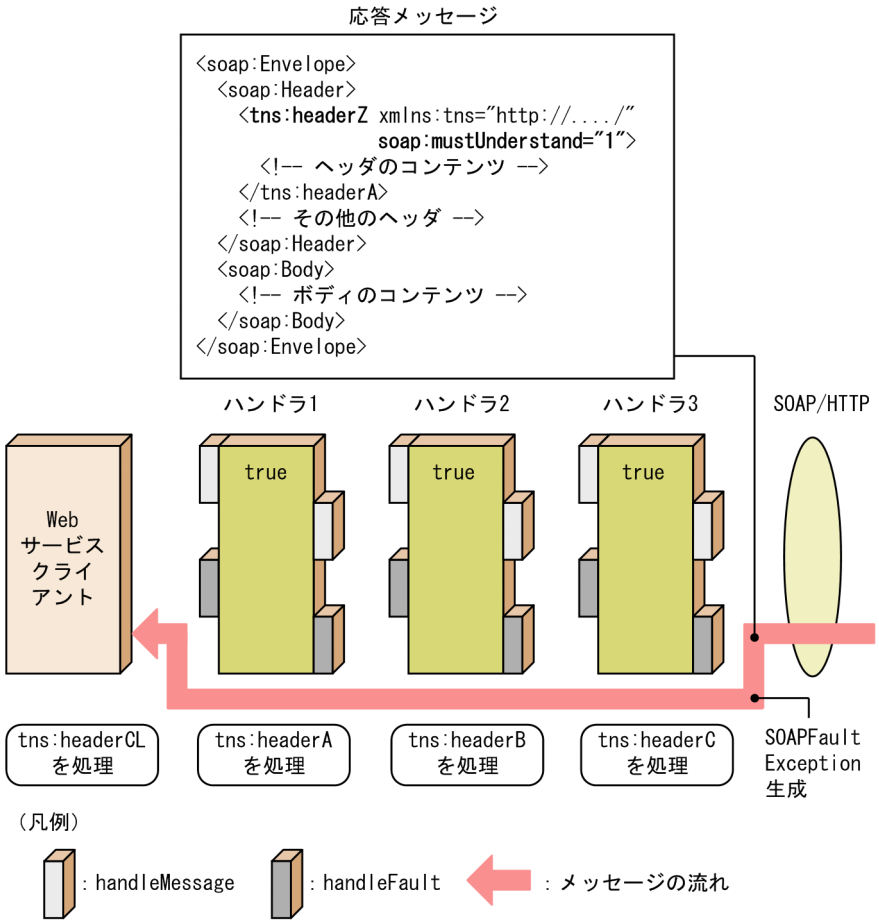
Web サービス実装クラスだけが要素名 tns:headerCL を処理できるとした場合も，ハンドラで処理できる場合と同様にすべてのハンドラが呼び出されます。

soap:mustUnderstand 属性が"1"の SOAP ヘッダを持つ SOAP メッセージを受信し，次のような場合は javax.xml.ws.soap.SOAPFaultException が返され，エラーとなります（KDJW10022-E）。

- 設定されたハンドラのどれかが SOAP ヘッダを処理できない
- Web サービスクライアント内で SOAP ヘッダを処理できない
- Web サービスクライアントがディスパッチベースで開発されている

SOAP ヘッダを処理できない場合のハンドラの処理は，次の図のようになります。

図 36-22　SOAP ヘッダを処理できない場合のハンドラの処理（Web サービスクライアント側）



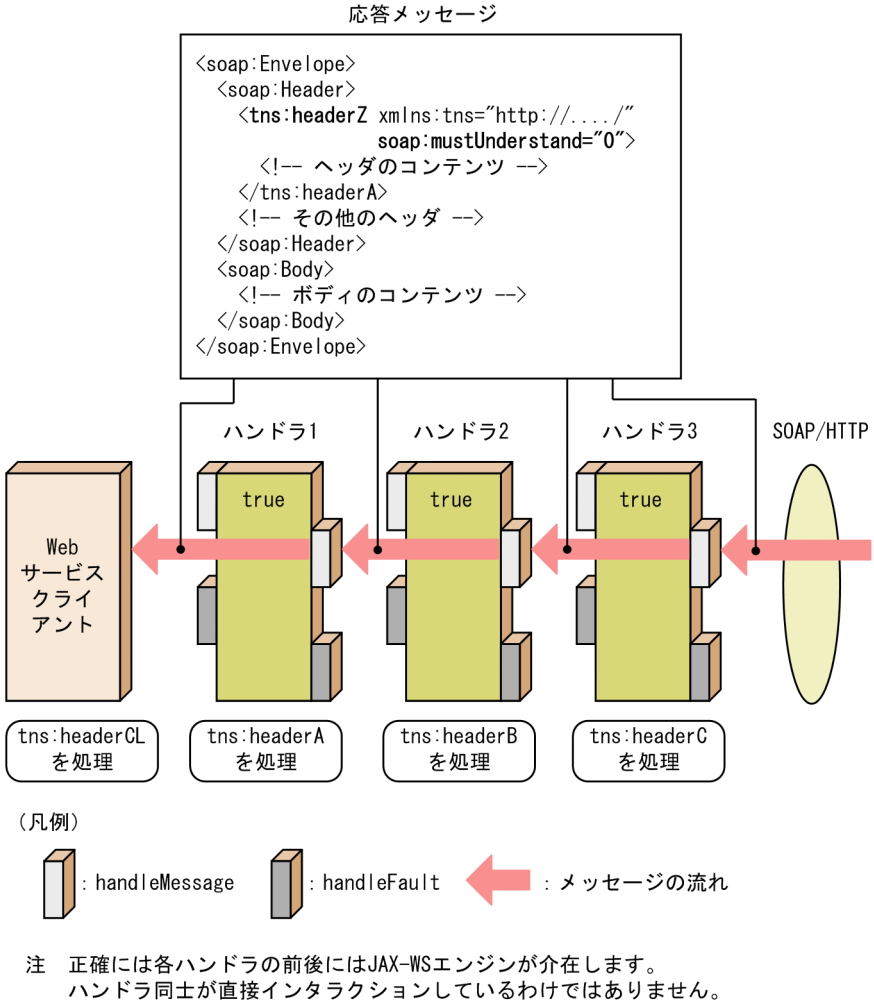
注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

(2)　soap:mustUnderstand 属性が"1"以外の場合

soap:mustUnderstand 属性が"1"以外の場合，設定されているすべてのハンドラが呼び出されます。

soap:mustUnderstand 属性が"1"以外の場合，ハンドラの処理は次の図のようになります。

図 36-23　soap:mustUnderstand 属性が"1"以外の場合のハンドラの処理（Web サービスクライアント側）



注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

この図の例では，ハンドラも Web サービス実装クラスも要素 tns:headerCL を処理できませんが，soap:mustUnderstand 属性が"1"以外の場合はエラーにはならないで，すべてのハンドラが呼び出されます。

36.7.3　処理できる SOAP ヘッダの設定方法

ハンドラ，Web サービス実装クラス，および Web サービスクライアントに分けて，処理できる SOAP ヘッダの設定方法を説明します。

(1)　SOAP ヘッダをハンドラで設定する場合

javax.xml.ws.handler.soap.SOAPHandler インタフェースを実装したハンドラクラスを作成し，getHeaders()メソッドで，処理できる SOAP ヘッダの QName を含む java.util.Set を返します。ハンドラでの設定例を次に示します。

```
public class MySOAPHandler implements SOAPHandler<SOAPMessageContext> {
  private final static Set<QName> headers;

  static {
    headers = new HashSet<QName>();
    headers.add(new QName("http://test.org/handler/", "headerA"));
  }

  public Set<QName> getHeaders(){
    return headers;
  }
   （以下，そのほかの実装は省略）
}
```

(2) SOAP ヘッダを Web サービス実装クラスで設定する場合

SEI を起点として開発する場合，Web サービス実装クラスに javax.jws.WebParam アノテーションでアノテートした引数を持つメソッドを定義します。javax.jws.WebParam アノテーションでは，次に示す設定を定義します。

- header 属性に true を設定する。
- mode 属性に javax.jws.WebParam.Mode.IN または javax.jws.WebParam.Mode.INOUT を設定する。

Web サービス実装クラスでの設定例を次に示します。

```
@WebService
@SOAPBinding (parameterStyle= javax.jws.soap.SOAPBinding.ParameterStyle.BARE)
@HandlerChain (file="handlerchainfile.xml")
public class MyWebService{
  @WebMethod(operationName = "webMethod")
  public String webMethod(
    @WebParam(targetNamespace="http://test.org/handler/", name="message")
    String message,
    @WebParam(targetNamespace="http://test.org/handler/", name="headerS", header = true, Web
Param.Mode.IN)
    String headerS
  ){
     （以下，メソッドの実装は省略）
  }
}
```

WSDL を起点に開発する場合は，WSDL 1.1 仕様および Application Server の JAX-WS 機能のサポート範囲内で，soap:header 要素を WSDL に記述します。Application Server の JAX-WS 機能での sopa:header 要素の記述については，「20.1.22　soap:header 要素」を参照してください。

(3) SOAP ヘッダを Web サービスクライアントで設定する場合

接続する Web サービスのメタデータである WSDL に，soap:header 要素が含まれる場合は，その soap:header 要素を処理できます。WSDL の例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:tns="http://test.org/handler/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             targetNamespace="http://test.org/handler/"
             name="HandlerTest01Service">

  <types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
               version="1.0"
               targetNamespace="http://test.org/handler/">
      <xs:element name="headerCL" nillable="true" type="xs:string"></xs:element>
      <xs:element name="message" nillable="true" type="xs:string"></xs:element>
      <xs:element name="testResponse" nillable="true" type="xs:string"></xs:element>
    </xs:schema>
  </types>

  <message name="test">
    <part name="message" element="tns:message"></part>
    <part name="headerCL" element="tns:headerCL"></part>
  </message>
  <message name="testResponse">
    <part name="testResponse" element="tns:testResponse"></part>
    <part name="headerCL" element="tns:headerCL"></part>
  </message>

  <portType name="HandlerTest01">
    <operation name="test" parameterOrder="message headerCL">
      <input message="tns:test"></input>
      <output message="tns:testResponse"></output>
    </operation>
  </portType>
  <binding name="HandlerTest01Binding" type="tns:HandlerTest01">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"></soap:b
inding>
    <operation name="test">
      <soap:operation soapAction=""></soap:operation>
      <input>
        <soap:body use="literal" parts="message"></soap:body>
        <soap:header message="tns:test" part="headerCL" use="literal"></soap:header>
      </input>
      <output>
        <soap:body use="literal" parts="testResponse"></soap:body>
        <soap:header message="tns:testResponse" part="headerCL" use="literal"></soap:header>
      </output>
    </operation>
  </binding>

  <service name="HandlerTest01Service">
    <port name="HandlerTest01Port" binding="tns:HandlerTest01Binding">
      <soap:address location="http://localhost:80/SOAPHeaderTest/HandlerTestService431"></so
ap:address>
    </port>
  </service>
</definitions>
```

この場合は，要素名{http://test.org/handler/}headerCL の SOAP ヘッダを処理できます。WSDL の soap:header 要素については「20.1.22　soap:header 要素」を参照してください。

36.8 ハンドラの配置

ハンドラの実装クラスの配置について説明します。

Web サービス側

ハンドラの実装クラスは，次に示す場所に配置します。

- POJO の Web サービスの場合：WAR ファイルのクラスパス上
- EJB の Web サービスの場合：EJB JAR ファイルのクラスパス上

WAR ファイルまたは EJB JAR ファイルのクラスパス上であれば，WAR ファイルや EJB JAR ファイルに含まれていなくてもかまいません。

Web サービスクライアント側

ハンドラの実装クラスは，Web サービスクライアントのクラスパス上に配置してください。

36.9　ハンドラチェインの設定

Web サービス実装クラスおよび Web サービスクライアントでは，動的なハンドラチェインの設定が利用できます。

ここでは，ハンドラチェインの設定方法と設定例について説明します。

36.9.1　Web サービス側のハンドラチェインの設定

Web サービス側でハンドラチェインを設定する場合，javax.jws.HandlerChain アノテーションで SEI，Web サービス実装クラス，またはプロバイダ実装クラスをアノテートし，javax.jws.HandlerChain アノテーションの file 要素でハンドラチェイン設定ファイルを指定します。

javax.jws.HandlerChain アノテーションで Web サービス実装クラスをアノテートした例を示します。

```
package com.sample;

@javax.jws.WebService
@javax.jws.HandlerChain(file="handlers.xml")
public class AddNumbersImpl{

    public int add( int number1, int number2 ) throws AddNumbersFault{
      ...
    }

}
```

この例では，Web サービス実装クラス com.sample.AddNumbersImpl にハンドラチェイン設定ファイル handlers.xml で定義されたハンドラチェインを関連づけています。

ハンドラチェイン設定ファイルは，Web サービスごとに必要になります。複数の Web サービスで共有してもかまいませんが，その場合は，同じ設定が適用されます。

javax.jws.HandlerChain アノテーションについては，「16.2.3　javax.jws.HandlerChain アノテーション」を参照してください。

(1)　ハンドラチェイン設定ファイル

ハンドラチェイン設定ファイルは，ハンドラを使用して Web サービスに処理を追加した場合に，ハンドラチェインの構成を定義するファイルです。javax.jws.HandlerChain アノテーションの file 要素で指定します。

ハンドラチェイン設定ファイルのサポート範囲については，「19.4　ハンドラチェイン設定ファイルのサポート範囲」を参照してください。ここでは，ハンドラチェイン設定ファイルの構文および配置について説明します。

(a) ハンドラチェイン設定ファイルの構文

ハンドラチェイン設定ファイルの構文は，Java EE 5 仕様の標準スキーマ（Java EE Web Services Metadata Handler Chain Schema），および Application Server の JAX-WS 機能でサポートされる範囲で記述します。

標準のスキーマは，Java EE 5 の名前空間 URI（http://java.sun.com/xml/ns/javaee/）にアクセスすることで参照できます。ハンドラチェイン設定ファイルの記述例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<handler-chains xmlns="http://java.sun.com/xml/ns/javaee">
  <handler-chain>
    <handler>
      <handler-class>org.test.handler.LoggingHandler</handler-class>
    </handler>
  </handler-chain>
</handler-chains>
```

(b) ハンドラチェイン設定ファイルの配置

ハンドラチェイン設定ファイルは，次に示す場所に格納します。

- POJO の Web サービスの場合：WAR ファイルの WEB-INF ディレクトリ以下
- EJB の Web サービスの場合：EJB JAR

OS のファイルシステムの制限と Java EE 5 仕様に従っていて，WAR ファイルまたは EJB JAR ファイルに格納できれば，ファイル名，WEB-INF ディレクトリ以下のディレクトリ名，EJB JAR のディレクトリ名，およびパスの長さに制限はありません。

(2) SOAP ロールとアクタの設定

SOAP ロールおよびアクタの設定を省略した場合，または空文字を指定した場合は，デフォルトの値が設定されます。デフォルトの値は，SOAP 1.1 仕様の SOAP メッセージの場合は"http://schemas.xmlsoap.org/soap/actor/next"，SOAP 1.2 仕様の SOAP メッセージの場合は"http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver"です。

(3) soap:mustUnderstand 属性の設定

soap:mustUnderstand 属性には，"0"，"1"，"true"，または"false"を設定してください。それ以外の値を設定した場合は，"0"または"false"が設定されているものと見なされます。

36.9.2 Web サービスクライアント側のハンドラチェインの設定

Web サービスクライアント側でハンドラチェインを設定する場合，JAX-WS API を使用します。使用できる JAX-WS API については，「19.2 API のサポート範囲」を参照してください。

(1) ハンドラチェインを設定するコードの追加

Web サービスクライアント側のハンドラチェインの設定には，次の二つの方法があります。

- サービスクラスへ設定する方法
- ポートへ設定する方法

それぞれの設定方法について説明します。

(a) サービスクラスへのハンドラチェインの設定

API を使用して，サービスクラスにハンドラチェインを設定するコードを追加した Web サービスクライアントの例を示します。

```
package com.example.sample.client;

import com.example.sample.TestJaxWs;
import com.example.sample.TestJaxWsService;
import com.example.sample.UserDefinedException;

public class TestClient {
    public static void main( String[] args ) {
        try {
            TestJaxWsService service = new TestJaxWsService();
            //ハンドラリゾルバを生成してサービスクラスに設定
            SampleHandlerResolver handlerResolver = new SampleHandlerResolver();
            service.setHandlerResolver( handlerResolver );
            TestJaxWs port = service.getTestJaxWs();

            String returnValue = port.jaxWsTest1( "Invocation test.", 1003 );

            System.out.println( "[RESULT] " + returnValue );
        }
        catch( UserDefinedException e ){
            e.printStackTrace();
        }
    }
}
```

ハンドラリゾルバの例を示します。

```
package com.example.sample.client;

import java.util.ArrayList;
import java.util.List;

import javax.xml.ws.handler.HandlerResolver;
import javax.xml.ws.handler.PortInfo;
import javax.xml.ws.soap.SOAPBinding;

public class SampleHandlerResolver implements HandlerResolver{
    //このハンドラリゾルバで保持するハンドラチェイン
    private List<Handler> handlerChain = new ArrayList<Handler>();
```

```
    //都合のよい方法でハンドラチェインにハンドラを追加しておく
    //ここではコンストラクタで追加する
    public SampleHandlerResolver(){
        this.handlerChain.add( new SomeHandler() );
    }

    //ハンドラチェインを返す
    public List<Handler> getHandlerChain( PortInfo portInfo ){
        //portInfoオブジェクトの内容を調べ
        //このハンドラリゾルバで処理可能であればハンドラチェインを返す
        if( portInfo.getBindingID().equals( SOAPBinding.SOAP11HTTP_BINDING )
            && portInfo.getPortName().equals( ... )
            && portInfo.getServiceName().equals( ... ) ){
            return this.handlerChain;
        }
        else{
            ...
        }
    }
}
```

javax.xml.ws.Service#setHandlerResolver メソッドで設定されたハンドラリゾルバへの変更は，それ以前に同じ Service オブジェクトから取得されたポートのハンドラチェインには影響しません。

(b) ポートへのハンドラチェインの設定

API を使用して，ポートにハンドラチェインを設定するコードを追加した Web サービスクライアントの例を示します。

```
package com.example.sample.client;

import java.util.ArrayList;
import java.util.List;

import javax.xml.ws.Binding;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.handler.Handler;

import com.example.sample.TestJaxWs;
import com.example.sample.TestJaxWsService;
import com.example.sample.UserDefinedException;

public class TestClient {
    public static void main( String[] args ) {
        try {
            TestJaxWsService service = new TestJaxWsService();
            TestJaxWs port = service.getTestJaxWs();

            //ポートに設定するハンドラチェインを生成
            List<Handler> handlerChain = new ArrayList<Handler>();
            //ハンドラチェインにハンドラを追加
            handlerChain.add( new SomeHandler() );

            //javax.xml.ws.Bindingを取得
```

```
            Binding binding = ( ( BindingProvider )port ).getBinding();

            //javax.xml.ws.Bindingを使用してポートにハンドラチェインを設定
            binding.setHandlerChain(handlerChain);

            String returnValue = port.jaxWsTest1( "Invocation test.", 1003 );
            System.out.println( "[RESULT] " + returnValue );
        }
        catch( UserDefinedException e ){
            e.printStackTrace();
        }
    }
}
```

サービスクラスとサービスクラスから取得したポートの両方にハンドラチェインを設定した場合は，ポートに設定されているハンドラチェインを使用します。また，javax.xml.ws.Binding#setHandlerChain メソッドで設定されたハンドラチェインは，ポートの取得元になった Service オブジェクトのハンドラチェインには影響しません。

(2) SOAP ロールとアクタの設定

SOAP ロールおよびアクタの設定を省略した場合，または空文字を指定した場合は，デフォルトの値が設定されます。デフォルトの値は，SOAP 1.1 仕様の SOAP メッセージの場合は"http://schemas.xmlsoap.org/soap/actor/next"，SOAP 1.2 仕様の SOAP メッセージの場合は"http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver"です。

(3) soap:mustUnderstand 属性の設定

soap:mustUnderstand 属性には，"0"，"1"，"true"，または"false"を設定してください。それ以外の値を設定した場合は，"0"または"false"が設定されているものと見なされます。

37 アドレッシング機能

この章では，Application Server のアドレッシング機能で実現できる通信方式，および使用時の設定について説明します。

37.1　アドレッシング機能とは

アドレッシング機能とは，一般的な転送プロトコルと通信システムによって与えられるサーバ名やポート番号などの情報を標準化し，特定のトランスポートまたは通信システムから，Web サービスやメッセージを独立させるための機能です。アドレッシング機能では，同期通信と非同期通信の 2 種類の通信方式をサポートしています。

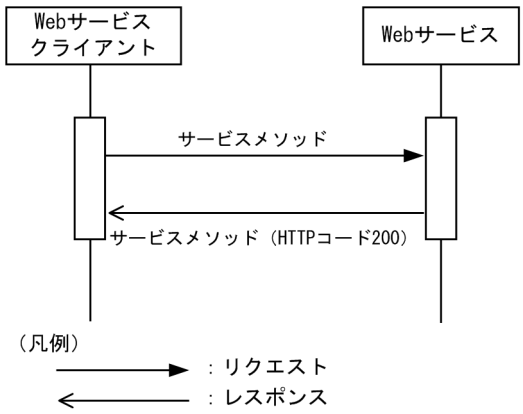
37.1.1　同期通信

(1)　同期通信の流れ

同期通信の場合，Web サービスにあるサービスメソッドの呼び出しのリクエストメッセージを Web サービスクライアントから送信し，Web サービスから結果のレスポンスメッセージを受信します。

同期通信の流れを次に示します。

図 37–1　同期通信の流れ



(2)　同期通信の設定方法

同期通信を使用するには，アドレッシング・ヘッダの wsa:ReplyTo/wsa:Address 要素に，WS-Addressing 1.0 仕様で規定されている匿名 URI である"http://www.w3.org/2005/08/addressing/anonymous"を設定します。

次の条件の場合に，アドレッシング・ヘッダに設定する内容の例を示します。

- Web サービスの URI
  http://localhost/addressing/AddNumbersImplService
- Web サービスを呼び出す際の Action 値
  http://sample.com/input
- 匿名 URI

http://www.w3.org/2005/08/addressing/anonymous

- このメッセージを表すユニークな ID

  uuid:4cfb4248-a552-4e33-a610-6af94f2aad07

```
<To xmlns="http://www.w3.org/2005/08/addressing">
  http://localhost/addressing/AddNumbersImplService
</To>
<Action xmlns=http://www.w3.org/2005/08/addressing>
  http://sample.com/input
</Action>
<ReplyTo xmlns="http://www.w3.org/2005/08/addressing">
  <Address>http://www.w3.org/2005/08/addressing/anonymous</Address>
</ReplyTo>
<MessageID xmlns="http://www.w3.org/2005/08/addressing">
  uuid:4cfb4248-a552-4e33-a610-6af94f2aad07
</MessageID>
```

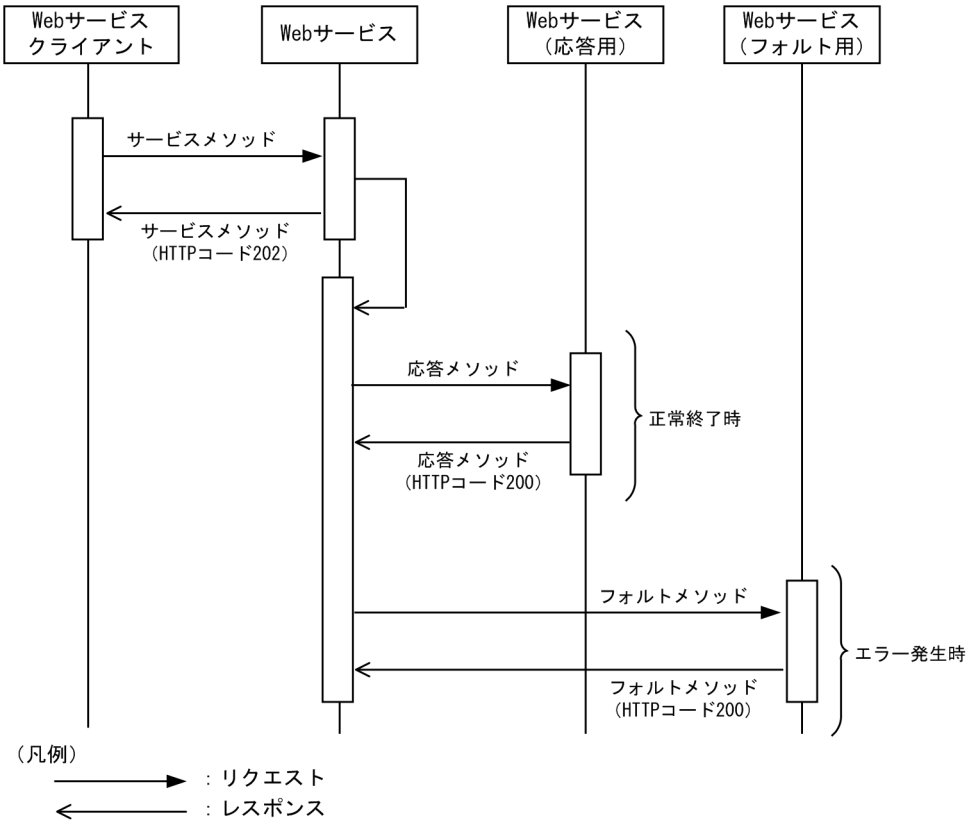
37.1.2　非同期通信

(1)　非同期通信の流れ

アドレッシング機能を使用した非同期通信の場合，Web サービスクライアントは Web サービスにあるサービスメソッドを呼び出すリクエストメッセージを送信したあと，Web サービスから結果のレスポンスメッセージを受信しないで処理を終了します。Web サービスは，レスポンスメッセージをほかの Web サービスに対して送信します。

非同期通信の流れを次に示します。

図 37-2　非同期通信の流れ



(2)　非同期通信の設定方法

非同期通信を使用するには，アドレッシング・ヘッダの wsa:ReplyTo/wsa:Address 要素や wsa:FaultTo/wsa:Address 要素に，レスポンスメッセージを送信する Web サービスの URL を設定します。

次の条件の場合に，アドレッシング・ヘッダに設定する内容の例を示します。

- Web サービスの URI
  http://localhost/addressing/AddNumbersImplService
- Web サービスを呼び出す際の Action 値
  http://sample.com/input
- Web サービスが正常終了した際の応答メッセージ送信先 URL
  http://localhost/responseserver/ResponseServerImplService
- Web サービスが異常終了した際の応答メッセージ送信先 URL
  http://localhost/responseserver/FaultServerImplService
- このメッセージを表すユニークな ID
  uuid:b19439fa-7a29-4045-93d9-56d6a2183afd

```
<To xmlns="http://www.w3.org/2005/08/addressing">
  http://localhost/addressing/AddNumbersImplService
</To>
```

```
<Action xmlns="http://www.w3.org/2005/08/addressing">
  http://sample.com/input
</Action>
<ReplyTo xmlns="http://www.w3.org/2005/08/addressing">
  <Address>
    http://localhost/responseserver/ResponseServerImplService
  </Address>
</ReplyTo>
<FaultTo xmlns="http://www.w3.org/2005/08/addressing">
  <Address>
    http://localhost/responseserver/FaultServerImplService
  </Address>
</FaultTo>
<MessageID xmlns="http://www.w3.org/2005/08/addressing">
  uuid:b19439fa-7a29-4045-93d9-56d6a2183afd
</MessageID>
```

(3) 非同期通信の注意事項

非同期通信を使用するときの注意事項について説明します。

- Web サービスがレスポンスメッセージの送信に失敗しても，Web サービスクライアントにはエラーは返りません。
- アドレッシング・ヘッダの wsa:ReplyTo/wsa:Address 要素や wsa:FaultTo/wsa:Address 要素に，wsa:To 要素に設定した Web サービスと同じ URL を設定した場合，レスポンスメッセージの送信元と送信先が同じになり，通信が無限ループになるおそれがあります。
- Web サービス側の JAX-WS エンジンは，リクエストメッセージとしてフォルトメッセージを受信した場合，サーバエラー（HTTP コード：500）として処理します。そのため，フォルトメッセージをリクエストメッセージとして受信する Web サービスは作成できません。
- プロキシを経由している場合，非同期通信は使用できません。非同期通信を使用する場合，レスポンスメッセージはプロキシを経由しないで，ほかの Web サービスに送信してください。

37.2　WSDL の拡張要素と拡張属性

WS-Addressing 1.0 仕様では，WSDL に記述する二つの拡張要素と一つの拡張属性が定義されています。ここでは，この WSDL 拡張要素と拡張属性について説明します。

37.2.1　WSDL の拡張要素

Application Server で使用できる WS-Addressing 1.0 仕様の WSDL 拡張要素を次に示します。

(1)　wsaw:UsingAddressing 要素

Web サービスでアドレッシング機能が有効かどうかを示します。

この要素は，wsdl:definitions/wsdl:binding 要素，または wsdl:definitions/wsdl:service/wsdl:port 要素の子要素として記述できます。この要素を記述した Web サービスでは，アドレッシング機能が有効になります。

- wsdl:required 属性

  リクエストメッセージにアドレッシング・ヘッダが必須かどうかを示します。

  この属性に"true"を指定した場合，アドレッシング・ヘッダは必須です。"false"を指定した場合，アドレッシング・ヘッダは任意です。

なお，wsaw:UsingAddressing 要素には，子要素または名前空間"http://www.w3.org/2006/05/addressing/wsdl"に属する属性は記述できません。記述した場合，cjwsimport コマンド実行時に標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます(KDJW51029-E)。

(2)　wsaw:Anonymous 要素

アドレッシング・ヘッダの応答エンドポイント（wsa:From/wsa:Address 要素，wsa:ReplyTo/wsa:Address 要素，および wsa:FaultTo/wsa:Address 要素）に匿名 URI を使用できるかどうかを示します。

この要素は，wsdl:definitions/wsdl:binding/wsdl:operation 要素の子要素として記述できます。指定できる値は次のとおりです。

- optional

  リクエストメッセージの応答エンドポイントに匿名 URI を使用するかどうかは任意です。

- required

  リクエストメッセージの応答エンドポイントとして，常に匿名 URI を使用する必要があります。

- prohibited

  リクエストメッセージの応答エンドポイントとして，匿名 URI を使用してはいけません。

wsaw:Anonymous 要素に上記以外の値は設定できません。上記以外の値を設定した場合，cjwsimport コマンド実行時に標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。また，wsaw:Anonymous 要素には子要素または名前空間"http://www.w3.org/2006/05/addressing/wsdl"に属する属性は記述できません。記述した場合，cjwsimport コマンド実行時に標準エラー出力とログにエラーメッセージが出力され，cjwsimport コマンドの処理が終了されます（KDJW51029-E）。

なお，Web サービス側の JAX-WS エンジンが発行した WSDL ファイルや，hwsgen コマンドで生成した WSDL ファイルには，この要素は記述されていません。匿名 URI が指定された場合の動作を制御する場合，この要素を記述した WSDL ファイルを用意してください。

(3) wsam:Addressing 要素

Web サービスでアドレッシング機能が有効かどうかを示します。JAX-WS 2.2 仕様（WS-Addressing 1.0 Metadata）では，wsaw:UsingAddressing 要素ではなくこの要素を使用します。wsaw:UsingAddressing 要素と同時に指定した場合は，どちらかの要素でアドレッシング・ヘッダを必須と指定することで，リクエストメッセージのアドレッシング・ヘッダが必須であることを示します。

この要素は，wsdl:definitions/wsdl:binding 要素または wsdl:definitions/wsdl:service/wsdl:port 要素の子要素として記述できます。この要素を記述した場合，この要素での設定が優先されます。

- wsp:Optional 属性

  リクエストメッセージにアドレッシング・ヘッダが必須であるかどうかを示します。

  この属性が true の場合，アドレッシング・ヘッダが任意であることを示します。false またはこの属性が省略された場合は，アドレッシング・ヘッダが必須であることを示します。

(4) wsam:AnonymousResponses 要素

リクエストメッセージにアドレッシング・ヘッダが含まれる場合，応答エンドポイント（wsa:From/wsa:Address 要素，wsa:ReplyTo/wsa:Address 要素，wsa:FaultTo/wsa:Address 要素）に，匿名 URI を指定する必要があることを示します。

この要素は，wsdl:definitions/wsdl:binding/wsam:Addressing/wsp:Policy または wsdl:definitions/wsdl:service/wsdl:port/wsam:Addressing/wsp:Policy の子要素として記述できます。

(5) wsam:NonAnonymousResponses 要素

リクエストメッセージにアドレッシング・ヘッダが含まれる場合，応答エンドポイント（wsa:From/wsa:Address 要素，wsa:ReplyTo/wsa:Address 要素，wsa:FaultTo/wsa:Address 要素）に，非匿名 URI を指定する必要があることを示します。

この要素は，wsdl:definitions/wsdl:binding/wsam:Addressing/wsp:Policy または wsdl:definitions/wsdl:service/wsdl:port/wsam:Addressing/wsp:Policy の子要素として記述できます。

(6) WSDL 拡張要素の注意事項

WSDL 拡張要素の内，次に示す要素を使用する場合の注意事項を説明します。

- wsaw:Anonymous 要素

  サービス側 JAX-WS エンジンが発行する WSDL ファイルや hwsgen コマンドが生成する WSDL ファイルには，この要素は現れません。この要素を使用して匿名 URI が指定された場合の動作を制御するためには，この要素を記述した WSDL ファイルを用意する必要があります。

37.2.2 WSDL の拡張属性

Application Server で使用できる WS-Addressing 1.0 仕様の WSDL 拡張属性を次に示します。

- wsaw:Action 属性

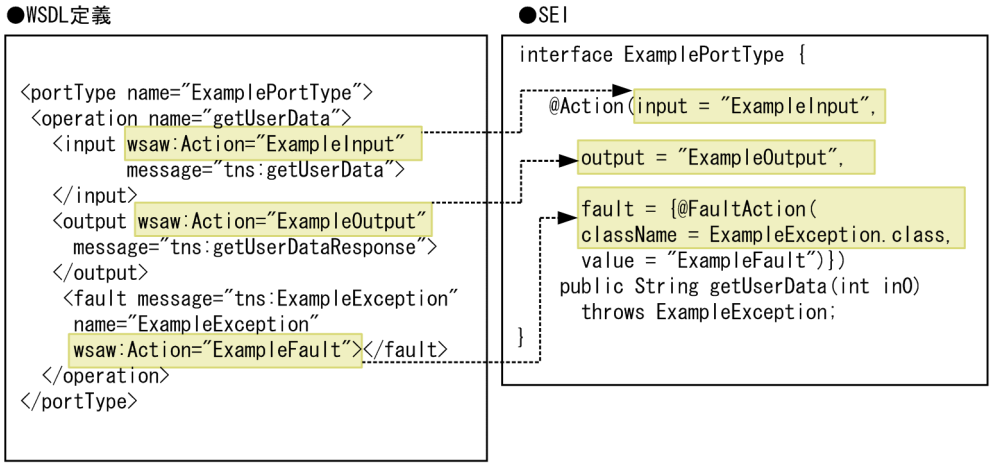
  wsdl:input 要素，wsdl:output 要素，および wsdl:fault 要素とアドレッシング・ヘッダの Action 値を結びつけるための拡張属性です。

  この属性は，wsdl:portType/wsdl:operation 要素の子要素である wsdl:input 要素，wsdl:output 要素，および wsdl:fault 要素の拡張属性として記述できます。それ以外の要素の拡張属性として記述した場合は，無視されます。この属性に空白を設定した場合，空白がそのまま Action 値として使用されます。また，空文字（""）を設定した場合，wsaw:Action 属性が無視されます。

  この属性が WSDL に記述されている場合，cjwsimport コマンドで生成される SEI に，javax.xml.ws.Action アノテーションや javax.xml.ws.FaultAction アノテーションが付与されます。javax.xml.ws.Action アノテーションについては「16.2.13　javax.xml.ws.Action アノテーション」を，javax.xml.ws.FaultAction アノテーションについては「16.2.15　javax.xml.ws.FaultAction アノテーション」を参照してください。

  wsaw:Action 属性を使用した場合のマッピング例を次に示します。

図 37–3　wsaw:Action 属性を使用した場合のマッピング例



37.3　アドレッシング機能で使用するアノテーションの注意事項

アドレッシング機能を使用するときの注意事項について説明します。

- javax.xml.ws.soap.Addressing アノテーションをサービス実装クラスに指定してください。
- cjwsimport コマンドを実行して生成されたサービス実装クラスのスケルトンクラスには，javax.xml.ws.soap.Addressing アノテーションがマッピングされません。そのため，サービス実装クラスのスケルトンクラスを使用する場合は，javax.xml.ws.soap.Addressing アノテーションを指定する必要があります。

アドレッシング機能で使用できるアノテーションについては，「16.2.1　アノテーション一覧」を参照してください。

37.4　フォルトメッセージ

ここでは，アドレッシング機能を使用した場合のフォルトメッセージについて説明します。

37.4.1　サポートしていないサブサブコード

WS-Addressing 1.0 仕様では，フォルトメッセージで使用されるサブサブコードがあらかじめ規定されています。あらかじめ規定されているサブサブコードのうち，Application Server でサポートしていないサブサブコードを次に示します。

- wsa:InvalidEPR
- wsa:DuplicateMessageID
- wsa:ActionMismatch※

注※

SOAP 1.1 仕様の場合，サポートしています。

37.4.2　フォルトメッセージの注意事項

アドレッシング機能を使用するときの注意事項について説明します。

- サブコードに wsa:MessageAddressingHeaderRequired を含むフォルトメッセージ
  Web サービス側の JAX-WS エンジンが，サブコードに wsa:MessageAddressingHeaderRequired を含むフォルトメッセージ※1 を送信する場合，フォルトメッセージのサブサブコードにサブコードと同じ値（wsa:MessageAddressingHeaderRequired）を設定します。
- サブコードに wsa:ActionNotSupported を含むフォルトメッセージ
  Web サービス側の JAX-WS エンジンが，サブコードに wsa:ActionNotSupported を含むフォルトメッセージ※2 を送信する場合，フォルトメッセージのサブサブコードに値を設定しません。

注※1

アドレッシング・ヘッダに，必須の要素がないフォルトメッセージ

注※2

アドレッシング・ヘッダの wsa:Action 要素の値が Web サービス側の wsa:Action 値と異なるフォルトメッセージ

37.5 Web サービス側の JAX-WS エンジンの動作（アドレッシング機能使用時）

ここでは，アドレッシング機能を使用した場合の，Web サービス側の JAX-WS エンジンの動作について説明します。

37.5.1 リクエストメッセージ受信時の動作

アドレッシング機能を使用する場合，サービス実装クラスに指定する javax.xml.ws.soap.Addressing アノテーションに設定されている要素の内容によって，リクエストメッセージを受信したときの動作が異なります。javax.xml.ws.soap.Addressing アノテーションと，リクエストメッセージを受信したときの動作の関係を次の表に示します。

表 37–1 javax.xml.ws.soap.Addressing アノテーションとリクエストメッセージ受信時の動作

項番	javax.xml.ws.soap.Addressing アノテーション		リクエストメッセージ受信時の動作		
			アドレッシング・ヘッダ		通信
	enabled 要素	required 要素	リクエストメッセージ	レスポンスメッセージ	
1	true	true	○	○	成功
2			×	× （フォルトメッセージ）	失敗
3		false	○	○	成功
4			×	×	成功
5	false	true	○	×	成功
6			×	×	成功
7		false	○	×	成功
8			×	×	成功

（凡例）

○：アドレッシング・ヘッダがあります。

×：アドレッシング・ヘッダはありません。

サービス実装クラスに javax.xml.ws.soap.Addressing アノテーションを指定しないと，アドレッシング機能は無効となります。その場合，アドレッシング・ヘッダの有無に関係なく，Web サービスはリクエストメッセージを受信します。また，アドレッシング機能が無効になっているため，受信したリクエストメッセージにアドレッシング・ヘッダが設定されていた場合も，Web サービスはアドレッシング・ヘッダなしのレスポンスメッセージを送信します。また，one-way オペレーションはレスポンスメッセージが存在しないため，one-way オペレーションを使用したアドレッシング機能はサポートしていません。one-way オペレーションでアドレッシング機能を使用した場合の動作は保証されません。

37.5.2　レスポンスメッセージ

ここでは，レスポンスメッセージについて説明します。

(1)　レスポンスメッセージの送信先

リクエストメッセージに含まれるアドレッシング・ヘッダの応答エンドポイントに匿名 URI 以外を使用している場合，wsa:From/wsa:Address 要素，wsa:ReplyTo/wsa:Address 要素，および wsa:FaultTo/wsa:Address 要素の有無によって，レスポンスメッセージの送信先が異なります。

各要素の有無とレスポンスメッセージの送信先の関係を次の表に示します。

表 37-2　レスポンスメッセージの送信先

項番	アドレッシング・ヘッダ			送信するレスポンスメッセージの種類	レスポンスメッセージの送信先
	wsa:From/wsa:Address 要素	wsa:ReplyTo/wsa:Address 要素	wsa:FaultTo/wsa:Address 要素		
1	存在しない	存在しない	存在しない	正常メッセージ	HTTP の送信元
2				異常メッセージ	HTTP の送信元
3			存在する	正常メッセージ	HTTP の送信元
4				異常メッセージ	wsa:FaultTo/wsa:Address 要素
5		存在する	存在しない	正常メッセージ	wsa:ReplyTo/wsa:Address 要素
6				異常メッセージ	wsa:ReplyTo/wsa:Address 要素
7			存在する	正常メッセージ	wsa:ReplyTo/wsa:Address 要素
8				異常メッセージ	wsa:FaultTo/wsa:Address 要素
9	存在する	存在しない	存在しない	正常メッセージ	HTTP の送信元
10				異常メッセージ	HTTP の送信元
11			存在する	正常メッセージ	HTTP の送信元
12				異常メッセージ	wsa:FaultTo/wsa:Address 要素
13		存在する	存在しない	正常メッセージ	wsa:ReplyTo/wsa:Address 要素
14				異常メッセージ	wsa:ReplyTo/wsa:Address 要素

項番	アドレッシング・ヘッダ			送信するレスポンスメッセージの種類	レスポンスメッセージの送信先
	wsa:From/ wsa:Address 要素	wsa:ReplyTo/ wsa:Address 要素	wsa:FaultTo/ wsa:Address 要素		
15			存在する	正常メッセージ	wsa:ReplyTo/ wsa:Address 要素
16				異常メッセージ	wsa:FaultTo/ wsa:Address 要素

(2) http://www.w3.org/2005/08/addressing/none 指定時の動作

http://www.w3.org/2005/08/addressing/none は，WS-Addressing 1.0 仕様でメッセージを送信しないことを示す URI です。この URI がアドレッシング・ヘッダの wsa:ReplyTo/wsa:Address 要素や wsa:FaultTo/wsa:Address 要素に設定されている場合，レスポンスメッセージは送信されません。

37.5.3 wsaw:Anonymous 要素指定時の動作

WSDL に wsaw:UsingAddressing 要素と wsaw:Anonymous 要素の両方が記述されている場合，受信したリクエストメッセージのアドレッシング・ヘッダにある応答エンドポイントの値によっては，リクエストメッセージの受信に失敗し，フォルトメッセージを返されることがあります。wsaw:Anonymous 要素と Web サービス側の JAX-WS エンジンの動作の関係を次の表に示します。

表 37–3 wsaw:Anonymous 要素とサービス側の JAX-WS エンジンの動作

項番	wsaw:Anonymous 要素の値	応答エンドポイントの値	Web サービス側の JAX-WS エンジンの動作
1	optional	匿名 URI	正常終了
2		非匿名 URI	正常終了
3	required	匿名 URI	正常終了
4		非匿名 URI	受信失敗（フォルトメッセージ）
5	prohibited	匿名 URI	受信失敗（フォルトメッセージ）
6		非匿名 URI	正常終了

wsaw:Anonymous 要素の指定値については，「37.2.1 WSDL の拡張要素」を参照してください。

37.5.4 Addressing アノテーション指定時の動作

サービス実装クラスに javax.xml.ws.soap.Addressing アノテーションを指定した場合，Web サービス側の JAX-WS エンジンが発行する WSDL ファイルや，hwsgen コマンドを実行して生成される WSDL ファイルは次のようになります。

- WSDL ファイルには，wsaw:UsingAddressing 要素だけでなく，wsdl:input 要素に wsaw:Action 属性も付与されます。付与される値は，WS-Addressing 1.0 仕様で決められているデフォルトの Action 値です。デフォルトの Action 値については，WS-Addressing 1.0 仕様を参照してください。
- javax.xml.ws.soap.Addressing アノテーションの required 要素に"false"を指定した場合，WSDL ファイルには wsaw:UsingAddressing 要素の wsdl:required 属性が付与されません。

37.5.5　Action アノテーション指定時の動作

SEI のメソッドに指定する javax.xml.ws.Action アノテーションの fault 要素に，メソッドの throws 節で宣言した例外クラス以外を示した javax.xml.ws.FaultAction アノテーションを指定した場合，javax.xml.ws.FaultAction アノテーションは無視されます。

37.5.6　wsa:Action 要素指定時の動作

ここでは，wsa:Action 要素の値，および wsa:Action 要素指定時の注意事項について説明します。

(1)　wsa:Action 要素の値

アドレッシング・ヘッダの wsa:Action 要素の値は，次の条件によって変化します。

- SEI のメソッドに javax.xml.ws.Action アノテーションが指定されているか
- WSDL に wsaw:Action 属性が記述されているか

javax.xml.ws.Action アノテーションおよび wsaw:Action 属性と，wsa:Action 要素の値の関係を次の表に示します。

表 37–4　wsa:Action 要素の値

項番	Action アノテーション	wsaw:Action 属性	wsa:Action 要素の値
1	指定あり	記述あり	Action アノテーションの値
2		記述なし	Action アノテーションの値
3	指定なし	記述あり	WSDL の wsaw:Action 属性の値
4		記述なし	WS-Addressing 1.0 仕様で決められているデフォルトの Action 値

非同期通信の応答用またはフォルト用の Web サービスが使用するアドレッシング・ヘッダの wsa:Action 要素の値も，上記のとおりに指定してください。

(2)　リクエストメッセージ受信時の注意事項

リクエストメッセージを受信するときの注意事項について説明します。

- wsa:Action 要素の値が，受信したリクエストメッセージのアドレッシング・ヘッダとサービス側の JAX-WS エンジンが使用するアドレッシング・ヘッダで異なる場合，Web サービスの呼び出しに失敗します。Web サービス側の JAX-WS エンジンでエラーが発生し，WS-Addressing 1.0 仕様で規定されているサブサブコードである wsa:ActionNotSupported を含んだフォルトメッセージが送信されます。
- 受信したリクエストメッセージのアドレッシング・ヘッダの wsa:Action 要素の値と SOAPAction の値が異なる場合，エラーが発生します。ただし，SOAPAction の値が空文字（""）の場合は，wsa:Action 要素の値が空文字以外でもエラーは発生しません。エラーが発生した場合は，Web サービス側の JAX-WS エンジンは，WS-Addressing 1.0 仕様で規定されているサブサブコードである wsa:ActionMismatch を含んだフォルトメッセージを送信します。

37.5.7 wsa:MessageID 要素を指定していない場合の動作

wsa:MessageID 要素を含まないアドレッシング・ヘッダは，正常ではないと見なされます。そのため，Web サービス側の JAX-WS エンジンでエラーが発生し，WS-Addressing 1.0 仕様で規定されているサブコードである wsa:MessageAddressingHeaderRequired を含んだフォルトメッセージを送信します。

37.6 Web サービスクライアント側の JAX-WS エンジンの動作（アドレッシング機能使用時）

ここでは，アドレッシング機能を使用した場合の，Web サービスクライアント側の JAX-WS エンジンの動作について説明します。

37.6.1 メッセージ送受信時の動作

アドレッシング機能は，取得した SEI にマッピングされる WSDL の設定内容に従います。ただし，AddressingFeature クラスを使用して SEI を取得した場合は，AddressingFeature クラスの設定内容が優先されるため，クラス生成時に設定した引数でメッセージを送受信したときの動作を指定できます。

AddressingFeature クラスと，メッセージを送信したときの動作の関係を次の表に示します。

表 37-5 AddressingFeature クラスとメッセージ送信時の動作

項番	AddressingFeature クラス			リクエストメッセージ送信時の動作
	enabled	required	responses	
1	true	True	Responses.ALL	○
			Responses.ANONYMOUS	
			Responses.NON_ANONYMOUS	
2		False	Responses.ALL	○
			Responses.ANONYMOUS	
			Responses.NON_ANONYMOUS	
3	false	True	Responses.ALL	×
			Responses.ANONYMOUS	
			Responses.NON_ANONYMOUS	
4		False	Responses.ALL	×
			Responses.ANONYMOUS	
			Responses.NON_ANONYMOUS	

（凡例）

○：メッセージにアドレッシング・ヘッダが付与されます。

×：メッセージにアドレッシング・ヘッダは付与されません。

AddressingFeature クラスと，メッセージを受信したときの動作の関係を次の表に示します。

表 37-6　AddressingFeature クラスとメッセージ受信時の動作

項番	AddressingFeature クラス			レスポンスメッセージ受信時の動作	
	enabled	required	responses	アドレッシング・ヘッダ	通信
1	true	true	Responses.ALL	○	成功
			Responses.ANONYMOUS		
			Responses.NON_ANONYMOUS		
2			Responses.ALL	×	失敗
			Responses.ANONYMOUS		
			Responses.NON_ANONYMOUS		
3		false	Responses.ALL	○	成功
			Responses.ANONYMOUS		
			Responses.NON_ANONYMOUS		
4			Responses.ALL	×	成功
			Responses.ANONYMOUS		
			Responses.NON_ANONYMOUS		
5	false	true	Responses.ALL	○	成功
			Responses.ANONYMOUS		
			Responses.NON_ANONYMOUS		
6			Responses.ALL	×	成功
			Responses.ANONYMOUS		
			Responses.NON_ANONYMOUS		
7		false	Responses.ALL	○	成功
			Responses.ANONYMOUS		
			Responses.NON_ANONYMOUS		
8			Responses.ALL	×	成功
			Responses.ANONYMOUS		
			Responses.NON_ANONYMOUS		

（凡例）

○：アドレッシング・ヘッダがあります。

×：アドレッシング・ヘッダはありません。

注

AddressingFeature クラスの responses 属性の指定は，クライアント側 JAX-WS エンジンの動作には影響しません。

37.6.2 AddressingFeature クラスと匿名 URI

AddressingFeature クラスを使用して SEI を取得した場合，Web サービスクライアント側の JAX-WS エンジンはアドレッシング・ヘッダの wsa:ReplyTo/wsa:Address 要素に匿名 URI を設定します。そのとき，次に示す Web サービスを呼び出すと，匿名 URI が使用できないため，サービスメソッドの呼び出し時に WebServiceException が発生します。

- WSDL の wsaw:Anonymous 要素に"prohibited"が指定されている Web サービス
- wsam:Addressing 要素の子要素に wsam:NonAnonymousResponses 要素が指定されている Web サービス

wsa:Anonymous 要素に"prohibited"が指定されている Web サービスや，wsam:Addressing 要素の子要素に wsam:NonAnonymousResponses 要素が設定されている Web サービスと通信する場合は，非匿名 URI を設定したアドレッシング・ヘッダを作成してください。

37.6.3 wsaw:Action 属性および wsam:Action 属性の注意事項

クライアント側 JAX-WS エンジンが送信するリクエストメッセージのアドレッシングヘッダにある wsa:Action 要素の値は，WSDL に記述する属性によって異なります。wsa:Action 要素の値には，次の値が設定されます。

wsaw:Action 属性を記述する場合

wsaw:Action 属性値が設定されます。

wsam:Action 属性を記述する場合

wsam:Action 属性値が設定されます。

wsam:Action 属性および wsaw:Action 属性を記述する場合

wsam:Action 属性値が設定されます。

記述しない場合

仕様で決められているデフォルト値の Action 値が設定されます。

デフォルトの Action 値については，WS-Addressing 1.0 Metadata 仕様を参照してください。

37.6.4 wsa:Action 要素の注意事項

Web サービスクライアント側の JAX-WS エンジンが受信したレスポンスメッセージで，アドレッシング・ヘッダの wsa:Action 要素の値と SOAPAction の値が異なる場合の動作は保証されません。

37.6.5　SEI の取得に関する注意事項

W3CEndpointReference クラスの getPort(Class<T>, WebServiceFeature...)メソッドを使用すると，WebServiceException が発生して SEI を取得できません。SEI を取得する場合は，Service クラスの getPort(EndpointReference, Class<T>, WebServiceFeature...)メソッドを使用してください。

38 SEI を起点とした開発の例（アドレッシング機能使用時）

この章では，アドレッシング機能を使用する場合に，SEI を起点とした Web サービスを開発するときの例を説明します。

38.1　開発例の構成（SEI 起点・アドレッシング）

この章で説明する開発例では，SEI を起点とした Web サービスを開発します。

開発する Web サービスの構成を次の表に示します。

表 38–1　Web サービスの構成（SEI 起点・アドレッシング）

項番	項目		値
1	デプロイする J2EE サーバの名称		jaxwsserver
2	Web サーバのホスト名とポート番号		webhost:8085
3	ネーミングサーバの URL		corbaname::testserver:900
4	コンテキストルート		addressing_dynamic_generate
5	スタイル		document/literal/wrapped
6	名前空間 URI		http://sample.com
7	ポートタイプ	個数	1
8		ローカル名	AddNumbersImpl
9	オペレーション	個数	3
10		ローカル名 1	add
11		ローカル名 2	add2
12		ローカル名 3	add3
13	サービス	個数	1
14		ローカル名	AddNumbersImplService
15	ポート	個数	1
16		ローカル名	AddNumbersImplPort
17	Web サービス実装クラス		com.sample.AddNumbersImpl
18	Web サービス実装クラスで公開するメソッド	個数	3
19		メソッド名 1	add
20		メソッド名 2	add2
21		メソッド名 3	add3
22	Web サービス実装クラスのメソッドでスローする例外	個数	1
23		クラス名	com.sample.AddNumbersFault

Web サービス開発時のカレントディレクトリの構成を次に示します。

表 38–2　カレントディレクトリの構成（SEI 起点・アドレッシング）

ディレクトリ	説明
c:¥temp¥jaxws¥works¥addressing	カレントディレクトリです。
├ server¥	Web サービスの開発で使用します。
│ ├ META-INF¥	EAR ファイルの META-INF ディレクトリに対応します。
│ │ └ application.xml	「38.3.4　application.xml を作成する」で作成します。
│ ├ src¥	Web サービスのソースファイル（*.java）を格納します。「38.3.1　Web サービス実装クラスを作成する」および「38.3.2　Web サービス実装クラスをコンパイルする」で使用します。
│ ├ WEB-INF¥	WAR ファイルの WEB-INF ディレクトリに対応します。
│ │ ├ web.xml	「38.3.3　web.xml を作成する」で作成します。
│ │ └ classes¥	コンパイルしたクラスファイル（*.class）を格納します。「38.3.2　Web サービス実装クラスをコンパイルする」で使用します。
│ ├ addressing_dynamic_generate.ear	「38.3.5　EAR ファイルを作成する」で作成します。
│ └ addressing_dynamic_generate.war	
└ client¥	Web サービスクライアントの開発で使用します。
├ src¥	Web サービスクライアントのソースファイル（*.java）を格納します。「38.5.1　サービスクラスを生成する」および「38.5.2　Web サービスクライアントの実装クラスを作成する」で使用します。
├ classes¥	コンパイルしたクラスファイル（*.class）を格納します。「38.5.3　Web サービスクライアントの実装クラスをコンパイルする」で使用します。
├ usrconf.cfg	「38.6.1　Java アプリケーション用オプション定義ファイルを作成する」で作成します。
└ usrconf.properties	「38.6.2　Java アプリケーション用ユーザプロパティファイルを作成する」で作成します。

カレントディレクトリのパスは，開発する環境に合わせて変更してください。

なお，以降の説明では，この表に示すディレクトリおよびファイル名を使用します。コマンド実行例や Java ソースなどで背景色付きの太字になっている部分は，この例で使用する指定値や生成される値を示します。構築する環境に合わせて読み替えてください。

また，この章で説明する開発例では，Web サービスと Web サービスクライアントを同じ環境で開発しますが，別の環境で開発することもできます。別の環境で開発する場合は，それぞれの環境に合わせて，カレントディレクトリのパスを読み替えてください。

38.2　開発例の流れ（SEI 起点・アドレッシング）

この章で説明する開発例では，次の流れで開発および実行します。

Web サービスの開発

1. Web サービス実装クラスを作成する（38.3.1）
2. javac コマンドを実行し，Web サービス実装クラスをコンパイルする（38.3.2）
3. web.xml を作成する（38.3.3）
4. application.xml を作成する（38.3.4）
5. EAR ファイルを作成する（38.3.5）

デプロイと開始

1. EAR ファイルをデプロイする（38.4.1）
2. Web サービスを開始する（38.4.2）

Web サービスクライアントの開発

1. cjwsimport コマンドを実行し，サービスクラスを生成する（38.5.1）
2. Web サービスクライアントの実装クラスを作成する（38.5.2）
3. Web サービスクライアントの実装クラスをコンパイルする（38.5.3）

Web サービスの実行

1. Java アプリケーション用オプション定義ファイルを作成する（38.6.1）
2. Java アプリケーション用ユーザプロパティファイルを作成する（38.6.2）
3. Web サービスクライアントを実行する（38.6.3）

38.3　Web サービスの開発例（SEI 起点・アドレッシング）

ここでは，SEI を起点とした場合の Web サービス（アドレッシング機能を使用）の開発例を説明します。

38.3.1　Web サービス実装クラスを作成する

Web サービスの処理を記述した Web サービス実装クラスを作成します。ここでは，受け取った要求メッセージの内容を計算して，応答メッセージを返す Web サービス実装クラスを作成します。

Web サービス実装クラスの作成例を次に示します。

```
package com.sample;

import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.xml.ws.Action;
import javax.xml.ws.FaultAction;

@WebService(name = "AddNumbers", targetNamespace = "http://sample.com/")
@SOAPBinding(style=SOAPBinding.Style.DOCUMENT, use=SOAPBinding.Use.LITERAL, parameterStyle=S
OAPBinding.ParameterStyle.WRAPPED)
interface AddNumbers {

    @Action(input = "http://sample.com/input",
            output = "http://sample.com/output")
    public int add(int number1, int number2) throws AddNumbersFault;

    public int add2(int number1, int number2) throws AddNumbersFault;

    @Action(input = "http://sample.com/input3",
            output = "http://sample.com/output3",
            fault = {@FaultAction(className = AddNumbersFault.class, value = "http://sample.
com/fault3")})
    public int add3(int number1, int number2) throws AddNumbersFault;
}
```

作成した AddNumbers.java は，UTF-8 形式で
c:¥temp¥jaxws¥works¥addressing¥server¥src¥com¥sample¥ディレクトリに保存します。

次に，SEI を実装した Web サービスの本体を作成します。ここでは，受け取った要求メッセージの内容を計算して応答メッセージとして返す Web サービス実装クラス com.sample.AddNumbersImpl を作成します。

Web サービスの本体の作成例を次に示します。

```
package com.sample;

import javax.jws.WebService;
import javax.xml.ws.soap.Addressing;
```

```
@Addressing
@WebService(endpointInterface = "com.sample.AddNumbers")
public class AddNumbersImpl implements AddNumbers {

    public int add(int number1, int number2) throws AddNumbersFault {
        return impl(number1, number2);
    }

    public int add2(int number1, int number2) throws AddNumbersFault {
        return impl(number1, number2);
    }

    public int add3(int number1, int number2) throws AddNumbersFault {
        return impl(number1, number2);
    }

    int impl(int number1, int number2) throws AddNumbersFault {
        if (number1 < 0 || number2 < 0) {
            throw new AddNumbersFault("Negative numbers can't be added!",
                                      "Numbers: " + number1 + ", " + number2);
        }
        return number1 + number2;
    }
}
```

作成した AddNumbersImpl.java は，UTF-8 形式で，
c:¥temp¥jaxws¥works¥addressing¥server¥src¥com¥sample¥ディレクトリに保存します。

また，com.sample.AddNumbersImpl クラスでスローしている例外クラス
com.sample.AddNumbersFault を作成します。

例外クラスの作成例を次に示します。

```
package com.sample;

public class AddNumbersFault extends Exception {

    String detail;

    public AddNumbersFault(String message, String detail) {
        super(message);
        this.detail = detail;
    }

    public String getDetail() {
        return detail;
    }
}
```

作成した AddNumbersFault.java は，UTF-8 形式で
c:¥temp¥jaxws¥works¥addressing¥server¥src¥com¥sample¥ディレクトリに保存します。

38.3.2　Web サービス実装クラスをコンパイルする

javac コマンドを実行して，Web サービス実装クラスをコンパイルします。javac コマンドについては，JDK のドキュメントを参照してください。

javac コマンドの実行例を次に示します。添付ファイル機能を使用した Java プログラムを javac コマンドでコンパイルする場合，javac コマンドの引数に"--add-modules=java.activation"を指定してください。

```
> cd c:¥temp¥jaxws¥works¥addressing¥server¥
> mkdir WEB-INF¥classes¥
> javac -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥javaee¥1100¥lib¥ja
vaee-api.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxp.jar;%
COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar" -d WEB-INF¥classes¥ -s src src¥com¥sample¥AddNumbers.
java src¥com¥sample¥AddNumbersImpl.java src¥com¥sample¥AddNumbersFault.java
```

javac コマンドが正常に終了すると，コンパイルしたクラスが，c:¥temp¥jaxws¥works¥addressing¥server¥WEB-INF¥classes¥com¥sample¥ディレクトリに出力されます。なお，コンパイルした Web サービス実装クラスに対して hwsgen コマンドを実行すると，事前にエラーチェックができます。hwsgen コマンドについては，「14.1.2　hwsgen コマンド」を，エラーチェックについては，「10.23.1　hwsgen コマンドによるエラーチェックについて」を参照してください。

38.3.3　web.xml を作成する

WAR ファイルの構成要素として必要な web.xml を作成します。

web.xml の作成例を次に示します。

```
<<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/we
b-app_3_0.xsd">
  <description>Sample web service &quot;addressing_dynamic_generate&quot;</description>
  <display-name>Sample_web_service_addressing_dynamic_generate</display-name>
  <listener>
    <listener-class>

com.cosminexus.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <description>Endpoint servlet for Cosminexus JAX-WS</description>
    <display-name>Endpoint_servlet_for_Cosminexus_JAX_WS</display-name>
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <servlet-class>
      com.cosminexus.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
```

```
    <servlet-name>CosminexusJaxwsServlet</servlet-name>
    <url-pattern>/AddNumbersImplService</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

バージョン 2.5 の web.xml を作成する場合は，web-app 要素の version 属性を"2.5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"にしてください。

作成した web.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥addressing¥server¥WEB-INF¥ディレクトリに保存します。web.xml の設定項目については，「3.4　web.xml の作成」を参照してください。

38.3.4　application.xml を作成する

EAR ファイルの構成要素として必要な application.xml を作成します。

application.xml の作成例を次に示します。なお，Web サービスとして application.xml に設定する項目はありません。

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/application_6.xsd">

  <description>Sample application &quot;addressing_dynamic_generate&quot;</description>
  <display-name>Sample_application_addressing_dynamic_generate</display-name>
  <module>
    <web>
      <web-uri>addressing_dynamic_generate.war</web-uri>
      <context-root>addressing_dynamic_generate</context-root>
    </web>
  </module>
</application>
```

バージョン 5 の application.xml を作成する場合は，application 要素の version 属性を"5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/application_5.xsd"にしてください。

作成した application.xml は，UTF-8 形式で c:¥temp¥jaxws¥works¥addressing¥server¥META-INF¥ディレクトリに保存します。application.xml を作成するときの注意事項については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」の「5.2.2　application.xml 編集時の注意事項」を参照してください。

38.3.5　EAR ファイルを作成する

jar コマンドを使用して，これまでに作成したファイルを含めた EAR ファイルを作成します。

EAR ファイルの作成例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥addressing¥server¥
> jar cvf addressing_dynamic_generate.war .¥WEB-INF
> jar cvf addressing_dynamic_generate.ear .¥addressing_dynamic_generate.war .¥META-INF¥appli
cation.xml
```

jar コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥addressing¥server ¥ディレクトリに addressing_dynamic_generate.ear が作成されます。

jar コマンドについては，JDK のドキュメントを参照してください。

38.4　デプロイと開始の例（SEI 起点・アドレッシング）

ここでは，アドレッシング機能を使用する場合に，SEI を起点としたときのデプロイと開始の例を説明します。

38.4.1　EAR ファイルをデプロイする

cjimportapp コマンドを使用して，作成した EAR ファイルを J2EE サーバにデプロイします。

デプロイの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥addressing¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjimportapp" jaxwsserver -nameserver corbaname::testserver
:900 -f addressing_dynamic_generate.ear
```

cjimportapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjimportapp（J2EE アプリケーションのインポート）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションをデプロイ（インポート）する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.3　J2EE アプリケーションのインポート」を参照してください。

38.4.2　Web サービスを開始する

cjstartapp コマンドを使用して，Web サービスを開始します。

Web サービスを開始する例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥addressing¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjstartapp" jaxwsserver -nameserver corbaname::testserver:
900 -name Sample_application_addressing_dynamic_generate
```

cjstartapp コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjstartapp（J2EE アプリケーションの開始）」を参照してください。

運用管理ポータルを使用して，J2EE アプリケーションを開始する方法については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「12.3.1　J2EE アプリケーションの開始」を参照してください。

38.5　Web サービスクライアントの開発例（SEI 起点・アドレッシング）

ここでは，SEI を起点とした場合の Web サービスクライアント（アドレッシング機能を使用）の開発例を説明します。

38.5.1　サービスクラスを生成する

cjwsimport コマンドを実行すると，サービスクラスなど Web サービスクライアントの開発に必要な Java ソースが生成されます。cjwsimport コマンドについては，「14.1.1　cjwsimport コマンド」を参照してください。

cjwsimport コマンドの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥addressing¥client¥
> mkdir src¥
> mkdir classes¥
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" -s src -d classes http://webhost:8085/addressing_dynamic_generate/AddNumbersImplService?wsdl
```

cjwsimport コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥addressing¥client¥src¥com¥sample¥ディレクトリに Java ソースが生成されます。

生成物の一覧を次の表に示します。

表 38–3　サービスクラス生成時の生成物（SEI 起点・アドレッシング）

ファイル名	説明
Add.java	WSDL 定義の「オペレーション」の要求メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
AddResponse.java	WSDL 定義の「オペレーション」の応答メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
Add2.java	WSDL 定義の「オペレーション」の要求メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
Add2Response.java	WSDL 定義の「オペレーション」の応答メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
Add3.java	WSDL 定義の「オペレーション」の要求メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
Add3Response.java	WSDL 定義の「オペレーション」の応答メッセージの wrapper 要素が参照する型に対応する JavaBean クラスです。
ObjectFactory.java	JAXB 2.2 仕様の ObjectFactory クラスです。
package-info.java	package-info.java ファイルです。

ファイル名	説明
AddNumbers.java	WSDL 定義の「サービス」に対応するサービスエンドポイントインタフェース（SEI）です。
AddNumbersImplService.java	サービスクラスです。
AddNumbersFault.java	WSDL 定義の AddNumbersFault に対応する JavaBean クラスです。
AddNumbersFault_Exception.java	フォルト bean のラッパ例外クラスです。

ファイル名の Add，Add2，Add3，AddNumbers，および AddNumbersImplService は，オペレーションのローカル名，ポートタイプのローカル名，およびサービスのローカル名の記述によって変わります。オペレーションのローカル名，ポートタイプのローカル名，およびサービスのローカル名のマッピングについては，次の個所を参照してください。

- 「15.1.2　ポートタイプから SEI 名へのマッピング」
- 「15.1.3　オペレーションからメソッド名へのマッピング」
- 「15.1.4　メッセージのパートからパラメタおよび戻り値へのマッピング（wrapper スタイルの場合）」
- 「15.1.5　メッセージのパートからパラメタおよび戻り値へのマッピング（non-wrapper スタイルの場合）」

38.5.2　Web サービスクライアントの実装クラスを作成する

Web サービスを利用する Web サービスクライアントの実装クラスを作成します。

Web サービスに対して 3 回の呼び出しをする Web サービスクライアントの作成例を次に示します。

```
package com.sample.client;

import javax.xml.namespace.QName;
import javax.xml.ws.soap.AddressingFeature;
import javax.xml.ws.wsaddressing.W3CEndpointReference;
import javax.xml.ws.wsaddressing.W3CEndpointReferenceBuilder;

import com.sample.AddNumbers;
import com.sample.AddNumbersImplService;
import com.sample.AddNumbersFault_Exception;

public class TestClient {

    int number1 = 10;
    int number2 = 10;
    int negativeNumber = -10;

    public static void main(String[] args) {
        TestClient client = new TestClient();

        client.existActionAnnotation1();
        client.existActionAnnotation2();
```

```
        client.notExistActionAnnotation();
        client.existFaultActionAnnotation();
        client.notExistFaultActionAnnotation();
    }

    public void existActionAnnotation1() {
        System.out.println("existActionAnnotation1");
        try {
            AddressingFeature feature = new AddressingFeature();

            AddNumbersImplService service = new AddNumbersImplService();
            AddNumbers stub = service.getAddNumbersImplPort(feature);
            int result = stub.add(number1, number2);
            assert result == 20;
        } catch (Exception ex) {
            ex.printStackTrace();
            assert false;
        }
    }

    public void existActionAnnotation2() {
        System.out.println("existActionAnnotation2");
        try {
            AddressingFeature feature = new AddressingFeature();
            W3CEndpointReferenceBuilder eprBuilder = new W3CEndpointReferenceBuilder();
            eprBuilder.address("http://webhost:8085/addressing_dynamic_generate/AddNumbersImplService");
            eprBuilder.serviceName(new QName("http://sample.com/", "AddNumbersImplService"));
            eprBuilder.endpointName(new QName("http://sample.com/", "AddNumbersImplPort"));
            W3CEndpointReference epr = eprBuilder.build();

            AddNumbersImplService service = new AddNumbersImplService();
            AddNumbers stub = service.getPort(epr, AddNumbers.class, feature);
            int result = stub.add(number1, number2);
            assert result == 20;
        } catch (Exception ex) {
            ex.printStackTrace();
            assert false;
        }
    }

    public void notExistActionAnnotation() {
        System.out.println("notExistActionAnnotation");
        try {
            AddressingFeature feature = new AddressingFeature();

            AddNumbersImplService service = new AddNumbersImplService();
            AddNumbers stub = service.getAddNumbersImplPort(feature);
            int result = stub.add2(number1, number2);
            assert result == 20;
        } catch (Exception ex) {
            ex.printStackTrace();
            assert false;
        }
    }

    public void existFaultActionAnnotation() {
```

```
        System.out.println("existFaultActionAnnotation");
        try {
            AddressingFeature feature = new AddressingFeature();

            AddNumbersImplService service = new AddNumbersImplService();
            AddNumbers stub = service.getAddNumbersImplPort(feature);
            stub.add3(negativeNumber, number2);
            assert false;
        } catch (AddNumbersFault_Exception e) {
            System.out.println("This is expected exception");
        } catch (Exception e) {
            e.printStackTrace();
            assert false;
        }
    }

    public void notExistFaultActionAnnotation() {
        System.out.println("notExistFaultActionAnnotation");
        try {
            AddressingFeature feature = new AddressingFeature();

            AddNumbersImplService service = new AddNumbersImplService();
            AddNumbers stub = service.getAddNumbersImplPort(feature);
            stub.add(negativeNumber, number2);
            assert false;
        } catch (AddNumbersFault_Exception ex) {
            System.out.println("This is expected exception");
        } catch (Exception e) {
            e.printStackTrace();
            assert false;
        }
    }
}
```

作成した TestClient.java は，UTF-8 形式で
c:¥temp¥jaxws¥works¥addressing¥client¥src¥com¥sample¥client¥ディレクトリに保存します。

なお，com.sample，AddNumbers，AddNumbersImplService，AddNumbersImplPort，add，add2，および add3 は，生成された Java ソースのパッケージ名，クラス名，およびクラス内のメソッド名によって変わります。異なる構成の Web サービスを開発する場合には，パッケージ名，クラス名，およびクラス内のメソッド名の記述を見直す必要があります。

38.5.3　Web サービスクライアントの実装クラスをコンパイルする

javac コマンドを使用して，作成した Web サービスクライアントをコンパイルします。

コンパイルの例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥addressing¥client¥
> javac -encoding UTF-8 -cp "%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥ja
```

```
vaee¥1100¥lib¥javaee-api.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;.¥classes” -d .¥classes
src¥com¥sample¥client¥TestClient.java
```

javac コマンドが正常に終了すると，
c:¥temp¥jaxws¥works¥addressing¥client¥classes¥com¥sample¥client¥ディレクトリに，
TestClient.class が生成されます。

javac コマンドについては，JDK のドキュメントを参照してください。

38.6　Web サービスの実行例（SEI 起点・アドレッシング）

ここでは，SEI を起点とした場合の Web サービスクライアント（アドレッシング機能使用時)の実行例を説明します。

38.6.1　Java アプリケーション用オプション定義ファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用オプション定義ファイル（usrconf.cfg）を作成します。

Java アプリケーション用オプション定義ファイルの作成例を次に示します。

```
add.class.path=<Application Serverのインストールディレクトリ>¥jaxws¥lib¥cjjaxws.jar
add.class.path=.¥classes
ejb.client.log.directory=logs
add.jvm.arg=-Dcosminexus.home=<Application Serverのインストールディレクトリ>
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

<Application Server のインストールディレクトリ>の部分は，Application Server をインストールしているパスを絶対パスで指定します。

作成した Java アプリケーション用オプション定義ファイルは，c:¥temp¥jaxws¥works¥addressing¥client¥ディレクトリに保存します。Java アプリケーション用オプション定義ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.1　usrconf.cfg（Java アプリケーション用オプション定義ファイル)」を参照してください。

38.6.2　Java アプリケーション用ユーザプロパティファイルを作成する

Web サービスを実行するときに必要な Java アプリケーション用ユーザプロパティファイルを作成します。

ここでは特に設定を変更しないため，c:¥temp¥jaxws¥works¥addressing¥client¥ディレクトリに usrconf.properties という名称の空のファイルを作成します。Java アプリケーション用ユーザプロパティファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「12.2.2　usrconf.properties（Java アプリケーション用ユーザプロパティファイル)」を参照してください。

38.6.3　Web サービスクライアントを実行する

cjclstartap コマンドを使用して，Web サービスクライアントを実行します。

Web サービスクライアントの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥addressing¥client¥
> "%COSMINEXUS_HOME%¥CC¥client¥bin¥cjclstartap"  com.sample.client.TestClient
```

cjclstartap コマンドが正常に終了すると，Web サービスクライアントの実行結果が表示されます。実行結果の表示例を次に示します。

```
KDJE40053-I The cjclstartap command will now start. (directory for the user definition file
= c:¥temp¥jaxws¥works¥addressing¥client, PID = 2636)
existActionAnnotation1
existActionAnnotation2
notExistActionAnnotation
existFaultActionAnnotation
This is expected exception
notExistFaultActionAnnotation
This is expected exception
KDJE40054-I The cjclstartap command was stopped. (PID = 2636, exit status = 0)
```

イタリック体になっている個所は，実行したタイミングや環境によって変わります。

cjclstartap コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」の「cjclstartap（Java アプリケーションの開始）」を参照してください。

39 障害対策

この章では，Web サービスまたは Web リソースの運用中に発生した障害の対策について説明します。

39.1　障害の種類と対策

Web サービスまたは Web リソースの運用中に発生する障害として，次に示す現象が考えられます。

Web サービスまたは Web リソースが正常に動作しない場合

- プログラムの実行中に異常終了する
- プログラムが意図したとおりに動作しない

Web サービスまたは Web リソースが正常に動作する場合

- 期待した性能が出ない

プログラムが正常に動作しない場合，ログやコンソールにエラーメッセージ，または警告メッセージが出力され，異常終了します。または正常に終了しても，意図したとおりに動作しない場合も考えられます。

また，プログラムが正常に動作している場合でも，プログラムまたはコンポーネントの処理能力が，期待した性能に届かない場合が考えられます。

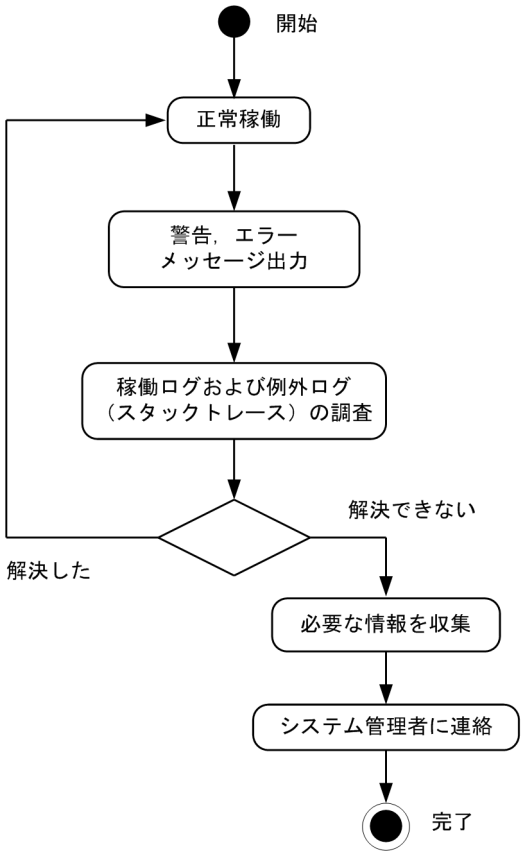
それぞれの場合について，対策方法を説明します。なお，EJB の Web サービス呼び出しをする際に，EJB コンテナ内で障害が発生する場合があります。EJB コンテナで発生した障害については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「8.9　EJB コンテナのトレース取得ポイント」を参照してください。

なお，Web サービスおよび Web リソースに関するメッセージは，JP1/IM 連携用システムログメッセージマッピングファイルに定義できません。このため，Web サービスおよび Web リソースに関するメッセージを JP1 イベントに変換して JP1 イベントを発行することはできません。JP1/IM 連携用システムログメッセージマッピングファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「8.3　JP1/IM 連携用システムログメッセージマッピングファイル」を参照してください。

39.1.1　プログラムの実行中に異常終了する場合

プログラムの実行中に異常終了した場合の対策の流れを次の図に示します。

図 39–1　警告，エラーメッセージ出力時の調査の流れ



作成したプログラムが正常に動作しないで，エラーメッセージまたは警告メッセージが出力される場合，ログに出力されたメッセージの内容から原因を調査してください。調査の対象となるログは，稼働ログおよび例外ログになります。各ログに出力された情報，および例外のスタックトレースの情報を基に，原因を調査してください。

なお，コマンド実行時に発生した警告メッセージおよびエラーメッセージは，コマンド稼働ログ，またはコマンド例外ログに出力されます。ログについては，「39.3　ログ」を参照してください。

発生した障害がプログラム以外に原因があると判断され，原因が解決できない場合は，障害情報を取得してシステム管理者に連絡してください。障害情報については，「39.2　障害発生時に取得する資料」を参照してください。

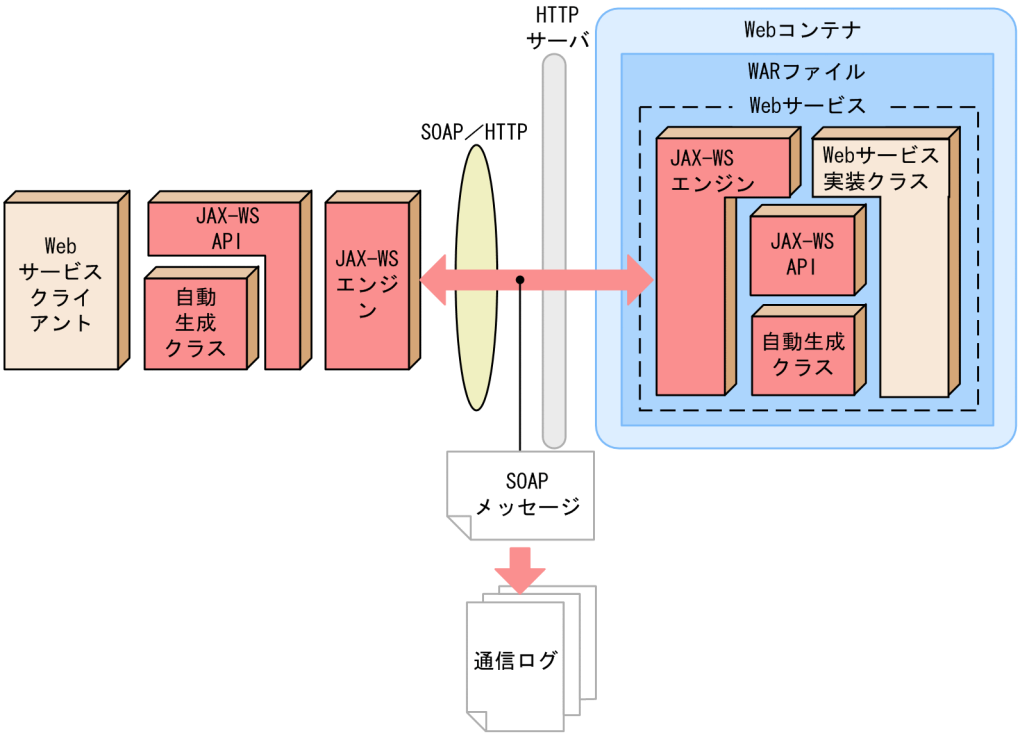
39.1.2　プログラムが意図したとおりに動作しない場合

エラーメッセージや警告メッセージが出力されない場合でも，不良などが原因で，作成したプログラムが意図したとおりに動作しないことがあります。この場合，プログラムから Web サービスまたは Web リソースの実装に対して，意図した SOAP メッセージ（Web サービスの場合）または HTTP メッセージ（Web リソースの場合）が送信されていないおそれがあります。プログラムが意図したとおりに動作しないときには，送受信されているメッセージの内容を解析して，意図したメッセージが送信されているかどうか確認してください。

通信ログの出力内容については，「39.3.5　ログのフォーマット」を参照してください。

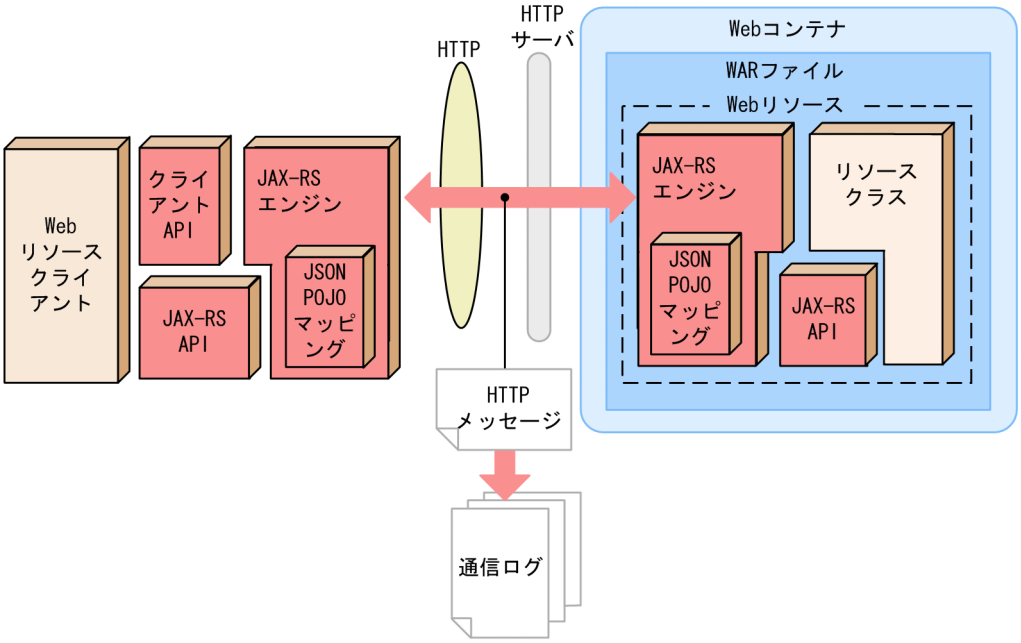
SOAP メッセージの通信の流れを次の図に示します。

図 39–2　通信ログの出力（Web サービスの場合）



HTTP メッセージの通信の流れを次の図に示します。

図 39–3　通信ログの出力（Web リソースの場合）



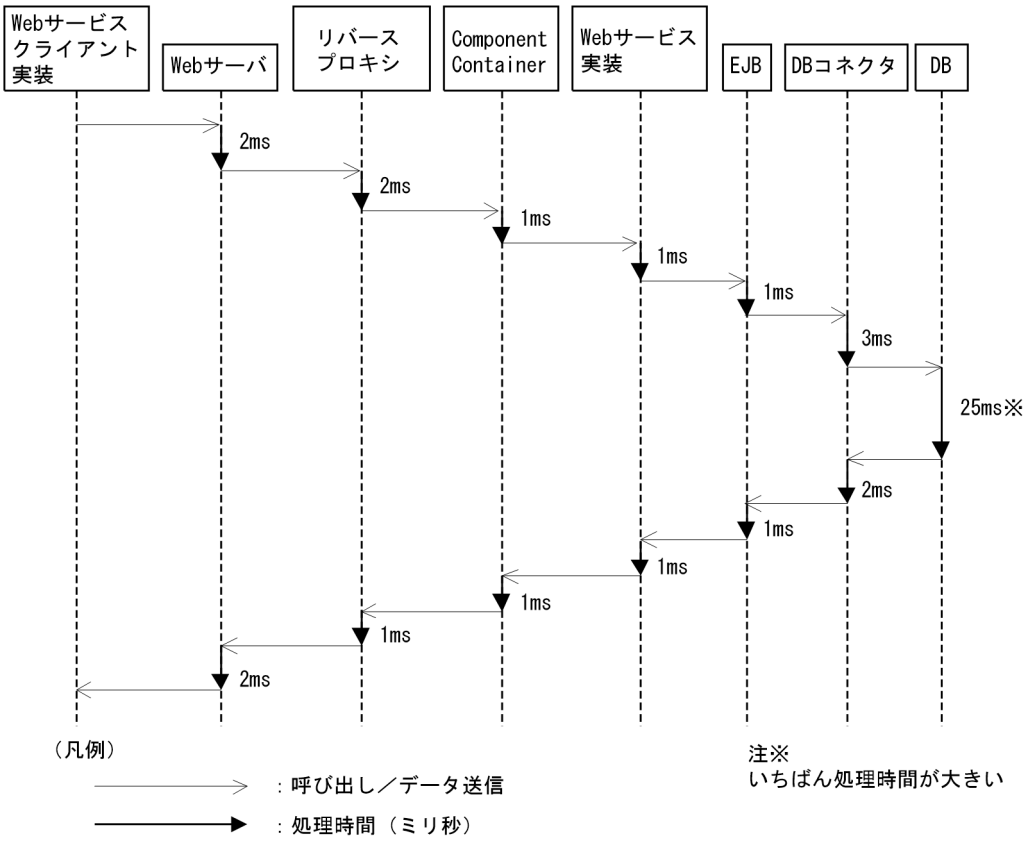
39.1.3　期待した性能が出ない場合

プログラムは正常に動作しているのに，実行時の性能が期待どおりにならない場合があります。原因として，プログラムの問題のほかに，Web サーバやデータベースなどのコンポーネントの問題が考えられます。コンポーネントに問題がある場合は，性能解析トレース（PRF）を使用した調査が有効です。性能解析トレースを使用すれば，各レイヤの処理時間や，リクエストの状況などが解析できます。リクエストの流れを追って，どこで遅延が発生しているかを確認してください。

性能解析トレースを使用した調査方法については，「39.4.3　性能解析トレースによる性能解析方法」を参照してください。

各機能レイヤの処理時間の例を次の図に示します。

図 39-4　各機能レイヤの処理時間の例



39.2　障害発生時に取得する資料

システム管理者へ連絡が必要になる障害，または対処方法がわからない障害が発生した場合は，次に示す障害情報を取得して，システム管理者に連絡してください。

なお，ファイル名の「?」は面数を表します。

- 稼働ログ（cjwmessage?.log または cjrmessage?.log）
- 例外ログ（cjwexception?_?.log または cjrexception?_?.log）
- 保守ログ（cjwmessage?.log または cjrmessage?.log）
- 通信ログ（cjwtransport?_?.log または cjrtransport?_?.log）
- コマンドの稼働ログ（cjwsimport?.log，cjwapt?.log）
- コマンドの例外ログ（cjwsimportex?_?.log，cjwaptex?_?.log）
- コマンドの保守ログ（cjwsimport?.log，cjwapt?.log）
- 性能解析トレース
- ユーザプログラム独自のログ（ある場合だけ）
- 共通定義ファイル（cjwconf.properties または cjrconf.properties）
- プロセス別の定義ファイル
- Web サービスの DD ファイル（web.xml，cosminexus-jaxws.xml）
- サーバおよびクライアントに設定したシステムプロパティとシステムクラスパス
- サーバおよびクライアントの標準出力と標準エラー出力
- Component Container および Web サーバのログ
- Component Container で規定する障害時取得情報
  - J2EE サーバのユーザ定義ファイル
  - J2EE サーバの保守情報
  - サーバ管理コマンドのユーザ定義ファイル
  - サーバ管理コマンドの保守情報
  - J2EE サーバの標準出力，標準エラー出力
  - J2EE サーバ側の TPBroker のログ
  - EJB クライアントアプリケーションに設定したシステムプロパティ，およびシステムクラスパス
  - EJB クライアントアプリケーションの標準出力，標準エラー出力

  Component Container で規定する障害時取得情報の取得方法については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「4.3.1　Component Container のログの取得」を参照してください。

39.3　ログ

Web サービスまたは Web リソースの開発時，および実行時に出力されるログについて説明します。

39.3.1　ログの種類

出力されるログの種類，および概要を次の表に示します。

表 39–1　ログの種類

項番	分類	概要
1	稼働ログ	稼働状態が出力されます。発生した問題の現象を確認して，問題を取り除くために使用します。
2	例外ログ	例外のスタックトレースが出力されます。発生した例外の詳細を確認するために使用します。
3	保守ログ	保守情報が出力されます。
4	通信ログ※	送受信したメッセージが出力されます。アプリケーション開発時や障害調査時に，送受信した内容を確認するために使用します。

注※

JAX-WS 機能の場合，通信ログを有効にするには，動作定義ファイルの com.cosminexus.jaxws.logger.runtime.transport.client_dump プロパティ（通信ログの出力（Web サービスクライアント側））， および com.cosminexus.jaxws.logger.runtime.transport.server_dump プロパティ（通信ログの出力（Web サービス側））に値を設定してください。

プロパティについては，「10.1.2　共通定義ファイルの設定項目」を参照してください。

JAX-RS 機能の場合，通信ログを有効にするには，動作定義ファイルの com.cosminexus.jaxrs.logger.runtime.transport.client.level プロパティ（通信ログの出力レベル（Web リソースクライアント側））， および com.cosminexus.jaxrs.logger.runtime.transport.server.level プロパティ（通信ログの出力レベル（Web リソース側））に値を設定してください。

プロパティについては，「13.1.2　共通定義ファイルの設定項目」を参照してください。

ログが出力される機能と出力されるログの対応を次の表に示します。

表 39–2　機能と出力されるログの対応

項番	機能	出力されるログ			
		稼働ログ	例外ログ	保守ログ	通信ログ
1	Web サービスおよび Web リソース	○	○	○	○
2	Web サービスクライアントおよび Web リソースクライアント	○	○	○	○
3	cjwsimport コマンド	○	○	○	－

（凡例）

○：ログが出力されます。

－：ログが出力されません。

39.3.2　ログファイルのローテーション

Application Server の JAX-WS 機能と JAX-RS 機能では，ログを複数のファイルへローテーションして出力します。ここでは，ログファイルのローテーションについて説明します。

(1)　ローテーションの方式

Application Server の JAX-WS 機能と JAX-RS 機能では，Web サービスまたは Web リソースが J2EE サーバ上で動作する場合に，出力するログの種類ごとにローテーションの切り替え方式を設定できます。ログの種類とローテーションの切り替えの設定可否を次の表に示します。

表 39–3　ログの種類とローテーションの切り替えの設定可否

項番	ログの種類	ローテーションの切り替えの設定可否
1	稼働ログ	○
2	例外ログ	×
3	保守ログ	○
4	通信ログ	×

（凡例）

○：ローテーションの切り替え方式を設定できます。

×：ローテーションの切り替え方式を設定できません。

注

デフォルトでは，出力先ファイルが指定したファイルサイズになると，出力先を次のファイルに切り替えます（ラップアラウンドモード）。詳細は，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「3. トラブルシューティングのための準備」を参照してください。

Java アプリケーションの開始コマンド（cjclstartap コマンド）で起動する Web サービスクライアントのログ，Web リソースクライアントのログ，またはコマンドラインインタフェースを実行する場合のログでは，ローテーションの切り替え方式を設定できません。

39.3.3　ログの出力先

利用する場面ごとに，ログの出力先について説明します。

(1)　J2EE サーバ上の Web サービス，および Web サービスクライアントを利用する場合

J2EE サーバ上で動作する Web サービス，および Web サービスクライアント（サーブレットや EJB など）のログの出力先を次の表に示します。

表 39–4　ログの出力先（J2EE サーバ上で動作する Web サービスや Web サービスクライアントの場合）

項番	ログの種類	出力先ディレクトリ	ファイル名
1	稼働ログ	<ejb.server.log.directory>/CJW	ラウンドアップ方式の場合 cjwmessage?.log シフト方式の場合 カレントログファイル：cjwmessage.log バックアップファイル：cjwmessage?.log
2	例外ログ	<ejb.server.log.directory>/CJW	cjwexception?_?.log
3	保守ログ	<ejb.server.log.directory>/CJW/maintenance	ラウンドアップ方式の場合 cjwmessage?.log シフト方式の場合 カレントログファイル：cjwmessage.log バックアップファイル：cjwmessage?.log
4	通信ログ	<ejb.server.log.directory>/CJW	cjwtransport?_?.log

注意事項

- <ejb.server.log.directory>は，J2EE サーバのログの出力先になります。詳細については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「3.3.6　J2EE サーバのログ取得の設定」を参照してください。
- ファイル名の「?」は面数を表します。
- ログ初期化処理前に問題が発生した場合は，Component Container の稼働ログ（cjmessage?.log）にログが出力されます。
- カレントログファイルとは，出力先のログファイルを示します。
- バックアップファイルとは，ローテーションによってバックアップされるログファイルを示します。

(2)　コマンドラインインタフェースで Web サービスクライアントを利用する場合

Java アプリケーションの開始コマンド（cjclstartap）で，Web サービスクライアントを起動する場合のログの出力先を次の表に示します。

表 39–5　ログの出力先（コマンドラインインタフェースで Web サービスクライアントを利用する場合）

項番	ログの種類	出力先ディレクトリ	ファイル名
1	稼働ログ	<ejbserver.client.log.directory>/ <ejbserver.client.ejb.log>/ <ejbserver.client.log.appid>/CJW	cjwmessage?.log

項番	ログの種類	出力先ディレクトリ	ファイル名
2	例外ログ	<ejbserver.client.log.directory>/ <ejbserver.client.ejb.log>/ <ejbserver.client.log.appid>/CJW	cjwexception?_?.log
3	保守ログ	<ejbserver.client.log.directory>/ <ejbserver.client.ejb.log>/ <ejbserver.client.log.appid>/CJW/ maintenance	cjwmessage?.log
4	通信ログ	<ejbserver.client.log.directory>/ <ejbserver.client.ejb.log>/ <ejbserver.client.log.appid>/CJW	cjwtransport?_?.log

注意事項

- <ejbserver.client.log.directory>，<ejbserver.client.ejb.log>，および<ejbserver.client.log.appid>は，EJB クライアントアプリケーションのシステムログの出力先になります。詳細については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「4.5.2　EJB クライアントアプリケーションのシステムログの出力先」を参照してください。
- ファイル名の「?」は面数を表します。
- ログ初期化処理前に問題が発生した場合は，Component Container の稼働ログ（cjclmessage?.log）にログが出力されます。

(3)　コマンドを実行する場合

コマンドを実行する場合のログの出力先を次の表に示します。

表 39-6　ログの出力先（コマンドを実行する場合）

項番	ログの種類	出力先ディレクトリ	ファイル名
1	稼働ログ	<コマンドログ出力先ディレクトリ（com.cosminexus.jaxws.tool.log.directory）>	cjwsimport コマンドの場合： cjwsimport?.log
2	例外ログ	<コマンドログ出力先ディレクトリ（com.cosminexus.jaxws.tool.log.directory）>	cjwsimport コマンドの場合：cjwsimportex?_?.log
3	保守ログ	<コマンドログ出力先ディレクトリ（com.cosminexus.jaxws.tool.log.directory）>/ maintenance	cjwsimport コマンドの場合： cjwsimport?.log

注意事項

- <コマンドログ出力先ディレクトリ（com.cosminexus.jaxws.tool.log.directory）>は動作定義ファイルのプロパティで設定します。デフォルトの出力先は，<Application Server のインストールディレクトリ>/jaxws/logs になります。

  プロパティについては，「10.1.2　共通定義ファイルの設定項目」を参照してください。

- ファイル名の「?」は面数を表します。

(4) J2EE サーバ上の Web リソース，および Web リソースクライアントを利用する場合

J2EE サーバ上で動作する Web リソース，および Web リソースクライアント（サーブレットや EJB など）のログの出力先を次の表に示します。

表 39-7 ログの出力先（J2EE サーバ上で動作する Web リソースや Web リソースクライアントの場合）

項番	ログの種類	出力先ディレクトリ	ファイル名
1	稼働ログ	<ejb.server.log.directory>/CJR	ラウンドアップ方式の場合 cjrmessage?.log シフト方式の場合 カレントログファイル： cjrmessage.log バックアップファイル： cjrmessage?.log
2	例外ログ	<ejb.server.log.directory>/CJR	cjrexception?_?.log
3	保守ログ	<ejb.server.log.directory>/CJR/maintenance	ラウンドアップ方式の場合 cjrmessage?.log シフト方式の場合 カレントログファイル： cjrmessage.log バックアップファイル： cjrmessage?.log
4	通信ログ	<ejb.server.log.directory>/CJR	cjrtransport?_?.log

注意事項

- <ejb.server.log.directory>は，J2EE サーバのログの出力先になります。詳細については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「3.3.6　J2EE サーバのログ取得の設定」を参照してください。
- ファイル名の「?」は面数を表します。
- ログ初期化処理前に問題が発生した場合は，Component Container の稼働ログ（cjrmessage?.log）にログが出力されます。
- カレントログファイルとは，出力先のログファイルを示します。
- バックアップファイルとは，ローテーションによってバックアップされるログファイルを示します。

(5) コマンドラインインタフェースで Web リソースクライアントを利用する場合

Java アプリケーションの開始コマンド（cjclstartap）で，Web リソースクライアントを起動する場合のログの出力先を次の表に示します。

表 39–8 ログの出力先（コマンドラインインタフェースで Web リソースクライアントを利用する場合）

項番	ログの種類	出力先ディレクトリ	ファイル名
1	稼働ログ	<ejbserver.client.log.directory>/ <ejbserver.client.ejb.log>/ <ejbserver.client.log.appid>/CJR	cjrmessage?.log
2	例外ログ	<ejbserver.client.log.directory>/ <ejbserver.client.ejb.log>/ <ejbserver.client.log.appid>/CJR	cjrexception?_?.log
3	保守ログ	<ejbserver.client.log.directory>/ <ejbserver.client.ejb.log>/ <ejbserver.client.log.appid>/CJR/ maintenance	cjrmessage?.log
4	通信ログ	<ejbserver.client.log.directory>/ <ejbserver.client.ejb.log>/ <ejbserver.client.log.appid>/CJR	cjrtransport?_?.log

注意事項

- <ejbserver.client.log.directory>，<ejbserver.client.ejb.log>，および<ejbserver.client.log.appid>は，EJB クライアントアプリケーションのシステムログの出力先になります。詳細については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「4.5.2　EJB クライアントアプリケーションのシステムログの出力先」を参照してください。
- ファイル名の「?」は面数を表します。
- ログ初期化処理前に問題が発生した場合は，Component Container の稼働ログ（cjclmessage?.log）にログが出力されます。

39.3.4 ログの重要度と出力条件

ログの重要度，および出力条件について説明します。

(1) ログの重要度

ログの重要度，およびメッセージの出力内容を次の表に示します。

表 39–9　重要度の意味と出力内容

項番	重要度	メッセージの意味	メッセージ出力内容
1	ERROR	重大な障害を示すメッセージレベルです。 問題が発生して，処理が継続できなかった場合に出力されます。 処理を継続するには，対処が必要です。	メッセージ ID のインジケータから，-E のメッセージが出力されます。メッセージには，発生した例外などの障害情報および対処方法が出力されます。
2	WARN	潜在的な問題を示すメッセージレベルです。 問題が発生していても，処理が継続できる場合に出力されます。 早急な対処は必要ありませんが，対処することが望ましい潜在的な問題があります。	メッセージ ID のインジケータから，-W のメッセージが出力されます。メッセージには，発生した例外などの障害情報および対処の内容が出力されます。 （例） 定義が誤りなので，デフォルトの値を使用した。
3	INFO	情報を提供するメッセージレベルです。 対処は必要ありませんが，通知する必要がある情報が出力されます。	メッセージ ID のインジケータが，-I のメッセージを出力します。メッセージには通知する内容が出力されます。 （例） コマンドの処理が開始した。
4	DEBUG	DEBUG レベルのメッセージです。 障害の要因調査で使用する情報です。	メッセージ ID が KDJW99999-I（Web サービスの場合）または KDJJ99999-I（Web リソースの場合）のメッセージが出力されます。 （例） 実行時の環境，メソッドトレースなど。

設定ファイルにログレベルの定義をした場合，出力されるログの重要度については，「39.3.6　ログの設定方法」を参照してください。

(2)　稼働ログ／例外ログ／保守ログの出力条件

稼働ログ，例外ログ，および保守ログは，動作定義ファイルに設定したログの定義によって出力内容が変更されます。

稼働ログ，例外ログ，および保守ログの出力条件を次の表に示します。

表 39–10　稼働ログ／例外ログ／保守ログの出力条件（イベントの重要度）

ログの種類	ログの定義	発生したイベントの重要度			
		ERROR	WARN	INFO	DEBUG
稼働ログ 例外ログ※	NONE	－	－	－	－
	ERROR	○	－	－	－
	WARN	○	○	－	－
	INFO	○	○	○	－

ログの種類	ログの定義	発生したイベントの重要度			
		ERROR	WARN	INFO	DEBUG
	DEBUG	○	○	○	○
保守ログ	NONE	－	－	－	－
	ALL	○	○	○	○

（凡例）

○：ログが出力されます。

－：ログが出力されません。

注※

例外ログは例外情報がない場合は出力されません。なお，例外情報とは，イベントを発生する原因となった例外のスタックトレースを指します。

ログの定義のデフォルト設定は，稼働ログと例外ログの出力レベルが「INFO」，保守ログの出力レベルが「ALL」です。つまり，デフォルト設定の状態で重要度が「ERROR」のイベントが発生した場合，稼働ログ，保守ログ，および例外ログのすべてにログが出力されます。

出力される稼働ログおよび例外ログの量は，ログの定義を NONE，ERROR，WARN，INFO，DEBUG へと変更するごとに増加する可能性が高くなります。より多くのログが出力されるようにログの定義を変更することを，出力レベルを上げると呼びます。ログの量が増加すると，全体の処理速度が低下するおそれがあるので，出力レベルを上げる場合は，処理速度や面数の見積もりに注意してください。

(3) 通信ログの出力条件

通信ログは，動作定義ファイルに設定したログの定義によって出力内容が変更されます。ただし，Web サービスクライアント，Web サービス，Web リソースクライアント，または Web リソースによって，設定するプロパティが異なります。設定するプロパティを次に示します。

- com.cosminexus.jaxws.logger.runtime.transport.client_dump（Web サービスクライアント側）
- com.cosminexus.jaxws.logger.runtime.transport.server_dump（Web サービス側）
- com.cosminexus.jaxrs.logger.runtime.transport.client.level（Web リソースクライアント側）
- com.cosminexus.jaxrs.logger.runtime.transport.server.level（Web リソース側）

通信ログの定義，および出力内容を次の表に示します。

表 39-11 通信ログの出力条件（Web サービスクライアント側または Web リソースクライアント側）

ログの定義	出力内容
ALL	送受信メッセージが常に出力されます。
HEADER	受信メッセージの HTTP ヘッダが常に出力されます。
ERROR_HEADER	受信メッセージの HTTP ヘッダがエラーのときだけ出力されます。

ログの定義	出力内容
NONE	送受信メッセージは出力されません。

表 39–12　通信ログの出力条件（Web サービス側または Web リソース側）

ログの定義	出力内容
ALL	Web サービスまたは Web リソースの送受信メッセージが常に出力されます。受信時は，HTTP のリクエスト情報も出力されます。
HEADER	受信メッセージの HTTP ヘッダおよび HTTP のリクエスト情報が常に出力されます。
ERROR_HEADER	受信メッセージの HTTP ヘッダおよび HTTP のリクエスト情報がエラーのときだけ出力されます（Web サービスの場合に設定できます）。
NONE	Web サービスまたは Web リソースの送受信メッセージは出力されません。

通信ログの量は，ログの定義が NONE，ERROR_HEADER，HEADER，ALL の順に増加します。ログの量が増加すると，全体の処理速度が低下するおそれがあるので，ログの定義のデフォルト設定を変更する場合は，処理速度や面数の見積もりに注意してください。

39.3.5　ログのフォーマット

稼働ログ，保守ログ，例外ログおよび通信ログのフォーマットについて説明します。

(1)　稼働ログと保守ログのフォーマット

稼働ログおよび保守ログの出力項目と出力内容を次の表に示します。

表 39–13　稼働ログと保守ログのフォーマット

項目	出力内容
番号	トレースコードの通番（4 桁）です。0000 から始まり，9999 まで行くと，0000 に戻ります。
日付	出力時の日付（yyyy/mm/dd 形式）
時刻	出力時の時刻（hh:mm:ss.nnn 形式）
アプリケーション名	• Web サービスおよび Web サービスクライアントの場合：「cjw」 • cjwsimport コマンドの場合：「cjwsimport」 • WS-RM 1.2 機能の場合：「wsrm」 • Web リソースおよび Web リソースクライアントの場合：「cjr」
プロセス識別子	プロセス識別子（16 進数）
スレッド識別子	スレッド識別子（16 進数）
メッセージ ID	メッセージ ID
メッセージ種別	メッセージ種別

項目	出力内容
	• ER：エラーメッセージを表示したことを表します。 • EC：例外をキャッチしたことを表します。 • なし：上記以外のトレース情報を表します。
メッセージテキスト	メッセージの本体
CRLF	レコード終端記号

(2) 例外ログと通信ログのフォーマット

例外ログと通信ログの出力項目と出力内容を次の表に示します。

表 39-14 例外ログと通信ログのフォーマット

項目	出力内容
日付	出力時の日付（yyyy/mm/dd 形式）
時刻	出力時の時刻（hh:mm:ss 形式）
Source クラス名	ログを発行したクラス名
レベル	ログの重要度
メッセージ	メッセージの本体

JAX-WS 機能の通信ログの出力例を次に示します。

```
2008/10/14 13:09:44 com.cosminexus.xml.ws.transport.http.client.HttpTransportPipe process
情報: KDJW30011-I http client message
---[HTTP request]---
SOAPAction: ""
Content-Type: text/xml;charset="utf-8"
X-hitachi-rootAp: MTgxNDczMTYyLzE2ODgvMC84MDI=
X-hitachi-clientAp: MTgxNDczMTYyLzE2ODgvMC84MDI=
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Bod
y><ns2:jaxWsTest1 xmlns:ns2="http://example.com/sample"><information>Invocation test.</infor
mation><count>1003</count></ns2:jaxWsTest1></S:Body></S:Envelope>
2008/10/14 13:09:45 com.cosminexus.xml.ws.transport.http.client.HttpTransportPipe process
情報: KDJW30012-I http client message
---[HTTP response 200]---
HTTP/1.1 200 OK
Keep-alive: timeout=3, max=100
Date: Tue, 14 Oct 2008 04:09:44 GMT
Content-type: text/xml;charset=utf-8
Connection: Keep-Alive
Transfer-encoding: chunked
Server: Cosminexus HTTP Server
<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Bod
y><ns2:jaxWsTest1Response xmlns:ns2="http://example.com/sample"><return>We've got your #1003
 message &quot;Invocation test.&quot;! It's 2008.10.14 13:09:45 now. See ya!</return></ns2:j
axWsTest1Response></S:Body></S:Envelope>
```

JAX-RS 機能の通信ログの出力例を次に示します。

- サーバ側

```
Sep 10, 2011 7:57:48 PM com.cosminexus.jersey.api.container.filter.LoggingFilter filter
INFO: KDJJ30013-I 1 * Server in-bound request
1 > POST http://sample.com/example/root/path%20value;matrix=matrix%20value?query=query%20
value
1 > user-agent: Java/1.5.0_11
1 > host: sample.com
1 > accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
1 > connection: keep-alive
1 > content-type: application/x-www-form-urlencoded
1 > content-length: 19
1 >
form=form%2520value

Sep 10, 2011 7:57:48 PM com.cosminexus.jersey.api.container.filter.LoggingFilter$Adapter
finish
INFO: KDJJ30015-I 1 * Server out-bound response
1 < 200
1 < Content-Type: text/html
1 <
matrix%20value path%20value form%20value query%20value
```

- クライアント側

```
Aug 27, 2012 10:30:06 AM com.cosminexus.jersey.api.client.filter.LoggingFilter printReque
st
INFO: KDJJ30026-I 1 * Client out-bound request
1 > POST http://sample.com/example
1 > Content-Type: text/plain
1 >
requestEntity

Aug 27, 2012 10:30:06 AM com.cosminexus.jersey.api.client.filter.LoggingFilter printRespo
nse
INFO: KDJJ30027-I 1 * Client in-bound response
1 < 200
1 < Content-Type: text/html
1 <
responseEntity
```

JAX-RS 機能の場合，動作定義ファイルにプロパティを設定していなくても，次に示すサーブレット初期化パラメタと値を web.xml に含めることで，Web リソースを含む Web アプリケーションごとに，通信ログを出力するかどうかを設定できます。

項番	サーブレット初期化パラメタ	値	動作
1	com.sun.jersey.spi.container.ContainerRequestFilters	com.cosminexus.jersey.api.container.filter.LoggingFilter	HTTP リクエストの情報が通信ログに出力されます。
2	com.sun.jersey.spi.container.ContainerResponseFilters	com.cosminexus.jersey.api.container.filter.LoggingFilter	HTTP レスポンスの情報が通信ログに出力されます。

注

この値以外を指定した場合，設定は無視されます。

動作定義ファイルにプロパティを設定している場合，Web リソースを含む Web アプリケーションごとに，通信ログを出力するかどうかを設定することはできません。

エンティティの情報の出力を抑止するかどうかも，Web リソースを含む Web アプリケーションごとに設定できます。サーブレット初期化パラメタと値を次に示します。

項番	サーブレット初期化パラメタ	値	動作
1	com.sun.jersey.config.feature.logging.DisableEntitylogging	true	エンティティの情報が通信ログに出力されません。
		false	エンティティの情報が通信ログに出力されます。

注

true または false 以外の値を指定した場合，共通定義ファイル（cjrconf.properties）で取得したプロパティの値が使用されて，エンティティの情報が通信ログに出力されます。なお，web.xml で指定した値は無視されます。

(3) 通信ログの文字エンコーディング

SOAP メッセージ（Web サービスの場合）や HTTP メッセージ（Web リソースの場合）の文字エンコーディングとは関係なく，通信ログには次に示す動作定義ファイルのプロパティで指定した文字エンコーディングが適用されます。デフォルトは，プラットフォーム依存のエンコーディングです。

- com.cosminexus.jaxws.logger.runtime.transport.encoding プロパティ（Web サービスの場合）
- com.cosminexus.jaxrs.logger.runtime.transport.encoding プロパティ（Web リソースの場合）

メッセージの文字エンコーディングと通信ログの文字エンコーディングが異なる場合，組み合わせによっては通信ログ内の一部の文字が不正になるおそれがあります。例えば，メッセージの文字エンコーディングが UTF-8，通信ログの文字エンコーディングが MS932 の場合，メッセージ内に MS932 に存在しない文字が含まれていると，その文字は通信ログ内で不正となります。そのため，Web サービスまたは Web リソース開発時には，使用できる文字コードまたは文字セットを規定して，Web サービスクライアントまたは Web リソースクライアントの開発者に通知することをお勧めします。

また，使用する機能によっても次の注意事項があります。

- 添付ファイル（wsi:swaRef 形式，および MTOM/XOP 仕様形式）を送受信する場合，添付ファイルにはバイナリデータが含まれたり，異なる文字エンコーディング形式の文字列が含まれたりするため，ルートパートの SOAP エンベロープも含めて，通信ログの文字が不正になるおそれがあります。

(4) 通信ログに出力される HTTP ヘッダ名

通信ログに出力される HTTP ヘッダ名は，実際に送受信される HTTP メッセージ（SOAP メッセージを含む HTTP メッセージ）に関係なく，常に先頭文字だけが大文字になります。

例）Content-type

39.3.6　ログの設定方法

出力レベル，サイズ，および面数を動作定義ファイルで指定します。

JAX-WS 機能の動作定義ファイルの設定例を次に示します。

```
com.cosminexus.jaxws.logger.runtime.message.level=INFO
com.cosminexus.jaxws.logger.runtime.message.file_num=2
com.cosminexus.jaxws.logger.runtime.message.file_size=2097152

com.cosminexus.jaxws.logger.runtime.maintenance.level=ALL
com.cosminexus.jaxws.logger.runtime.maintenance.file_num=2
com.cosminexus.jaxws.logger.runtime.maintenance.file_size=16777216

com.cosminexus.jaxws.logger.runtime.exception.level=INFO
com.cosminexus.jaxws.logger.runtime.exception.file_num=2
com.cosminexus.jaxws.logger.runtime.exception.file_size=16777216

com.cosminexus.jaxws.logger.runtime.transport.client_dump=NONE
com.cosminexus.jaxws.logger.runtime.transport.server_dump=NONE
com.cosminexus.jaxws.logger.runtime.transport.file_num=2
com.cosminexus.jaxws.logger.runtime.transport.file_size=16777216
com.cosminexus.jaxws.logger.runtime.transport.encoding=DEFAULT

com.cosminexus.jaxws.logger.cjwsimport.message.level=INFO
com.cosminexus.jaxws.logger.cjwsimport.message.file_num=2
com.cosminexus.jaxws.logger.cjwsimport.message.file_size=2097152

com.cosminexus.jaxws.logger.cjwsimport.exception.level=INFO
com.cosminexus.jaxws.logger.cjwsimport.exception.file_num=2
com.cosminexus.jaxws.logger.cjwsimport.exception.file_size=16777216

com.cosminexus.jaxws.logger.cjwsimport.maintenance.level=ALL
com.cosminexus.jaxws.logger.cjwsimport.maintenance.file_num=2
com.cosminexus.jaxws.logger.cjwsimport.maintenance.file_size=16777216
```

JAX-RS 機能の動作定義ファイルの設定例を次に示します。

```
com.cosminexus.jaxrs.logger.runtime.message.level=INFO
com.cosminexus.jaxrs.logger.runtime.message.file_num=2
com.cosminexus.jaxrs.logger.runtime.message.file_size=2097152

com.cosminexus.jaxrs.logger.runtime.maintenance.level=ALL
com.cosminexus.jaxrs.logger.runtime.maintenance.file_num=2
com.cosminexus.jaxrs.logger.runtime.maintenance.file_size=16777216

com.cosminexus.jaxrs.logger.runtime.exception.level=INFO
com.cosminexus.jaxrs.logger.runtime.exception.file_num=2
com.cosminexus.jaxrs.logger.runtime.exception.file_size=16777216

com.cosminexus.jaxrs.logger.runtime.transport.server.level=NONE
com.cosminexus.jaxrs.logger.runtime.transport.client.level=NONE
com.cosminexus.jaxrs.logger.runtime.transport.file_num=2
com.cosminexus.jaxrs.logger.runtime.transport.file_size=16777216
com.cosminexus.jaxrs.logger.runtime.transport.encoding=DEFAULT
```

39.3.7　ログの見積もり

各ログファイルの見積もり方法について説明します。

(1)　使用するモデルおよびログの設定

ログの見積もり方法は，次に示すモデル 1 およびモデル 2 を例に説明します。

モデル 1：正常系

Web サービスクライアントまたは Web リソースクライアントからリクエストを送信し，Web サービスまたは Web リソースが正常にレスポンスを返す場合のモデルです。

モデル 2：異常系

Web サービスまたは Web リソースで RuntimeException が発生する場合のモデルです。Web リソースには RuntimeException を処理できる例外マッピングプロバイダが存在しないものとします。

Web サービスクライアントまたは Web リソースクライアントでは，オブジェクトの再利用を前提にしています。オブジェクトの再利用については，「3.6.4　注意事項」または「11.4.5　注意事項」を参照してください。

JAX-WS 機能の場合，Web サービスクライアントや Web サービスを実装する形態によってさまざまです。必要に応じてモデルを追加してください。主な観点を次に示します。

- Web サービス実装クラスやプロバイダ実装クラスおよびスタブベースの Web サービスクライアントやディスパッチベースの Web サービスクライアントのバリエーション。
- SOAP 1.1 仕様および SOAP 1.2 仕様の SOAP メッセージのバリエーション。
- wsi:swaRef 形式および MTOM/XOP 仕様形式の添付ファイルの SOAP メッセージのバリエーション。

JAX-RS 機能の場合，JAX-RS 仕様の性質上 HTTP リクエストや HTTP レスポンスを送受信する方法は，Web リソースクライアントや Web リソースを実装する形態によってさまざまです。必要に応じてモデルを追加してください。主な観点を次に示します。

- リソースメソッドのエンティティパラメタ，その他のパラメタ，および戻り値の型や要求メソッド識別子のバリエーション。
- 例外マッピングプロバイダの有無および例外マッピングプロバイダが処理できる例外のバリエーション。
- Path アノテーションの使用方法によるバリエーション。

このモデルで使用するログの設定を示します。

- 稼働ログ，例外ログ，および保守ログの設定は，デフォルト値のまま変更しません。
- 通信ログの設定は，次のプロパティにそれぞれ「ALL」を指定します。

  JAX-WS 機能の場合

  - com.cosminexus.jaxws.logger.runtime.transport.client_dump

- com.cosminexus.jaxws.logger.runtime.transport.server_dump

JAX-RS 機能の場合

- com.cosminexus.jaxrs.logger.runtime.transport.client.level
- com.cosminexus.jaxrs.logger.runtime.transport.server.level

ログの設定値およびデフォルト値については，「10.1.2　共通定義ファイルの設定項目」または「13.1.2 共通定義ファイルの設定項目」を参照してください。

(2)　JAX-WS 機能のログの見積もり方法

モデル 1 およびモデル 2 の各ログの出力結果を次の表に示します。

次の表にある「起動時」とはアプリケーション開始時，「終了時」とはアプリケーション停止時，「1 リクエスト処理時」とは Web サービスによる 1 リクエストの処理を表します。出力量はこれらの期間に出力されるログの量を示します。

表 39–15　起動時，1 リクエスト処理時，停止時の各ログの出力例（モデル 1／モデル 2）

項番	ログの種類	出力量		
		起動時	1 リクエスト処理時	終了時
1	稼働ログ	0.7KB	• モデル 1：0.0KB • モデル 2：0.3KB	0.3KB
2	例外ログ	0KB	• モデル 1：0KB • モデル 2※：5KB	0KB
3	通信ログ	0KB	0.3KB ＋ HTTP リクエスト＋ HTTP レスポンス量	0KB
4	保守ログ	2.4KB	• モデル 1：3.2KB • モデル 2：3.6KB	0.4KB

注※

例外のスタックトレースの量になります。

各ログの見積もり式を次に示します。この結果を基に，各ログファイル一つのファイルサイズ，およびファイル面数を求めます。

```
［単位時間当たりのログの出力量］＝
　［初期化時の出力量］
　＋［モデル1の1リクエスト処理時の出力量］×総リクエスト数×モデル1のリクエストの割合
　＋［モデル2の1リクエスト処理時の出力量］×総リクエスト数×モデル2のリクエストの割合
　　：
　＋［モデルnの1リクエスト処理時の出力量］×総リクエスト数×モデルnのリクエストの割合
　＋［終了時の出力量］
```

この式を基に，例を使用して各ファイルの見積もり方法を説明します。

想定する例として，1 分間に平均 100 件のリクエストがあるようなシステムで，ログを 3 時間ラップアラウンドされないようにログファイルのサイズと個数を見積もるものとします。全リクエスト中，モデル 1 のリクエストが 80%，モデル 2 のリクエストが 20%を占めるとすると，3 時間では計 18,000 リクエスト（100 リクエスト／分×180 分）が到着します。

この場合，稼働ログの出力量を計算すると，次のとおりになります。

```
0.7KB（初期化時の出力量）
＋{0KB×18000}（モデル1の全リクエスト処理時の出力量）×0.8（モデル1のリクエストの割合）
＋{0.3KB×18000}（モデル2の全リクエスト処理時の出力量）×0.2（モデル2のリクエストの割合）
＋0.3KB（終了時の出力量）
＝1081KB
```

計算の結果，ファイルサイズとファイル面数はデフォルト値で間に合います。

同様に，保守ログの出力量を計算すると，次のとおりになります。

```
2.4KB（初期化時の出力量）
＋{3.2KB×18000}（モデル1の全リクエスト処理時の出力量）×0.8（モデル1のリクエストの割合）
＋{3.6KB×18000}（モデル2の全リクエスト処理時の出力量）×0.2（モデル2のリクエストの割合）
＋0.4KB（終了時の出力量）
＝59043KB≒57.6MB
```

計算の結果，ログファイル一つ当たり 15MB とすると，面数は四つとなります。

注意事項

通信ログでは，HTTP リクエストや HTTP レスポンスの内容によって，出力量が変化します。添付ファイルを使用して通信する場合は，通信ログにも添付ファイルの内容が記録されるため，特に大きな添付ファイルを使用しているときは，通信ログファイルのサイズや面数を考慮する必要があります。

(3) JAX-RS 機能のログの見積もり方法

モデル 1 およびモデル 2 の各ログの出力結果を次の表に示します。

次の表にある「起動時」とはアプリケーション開始時，「終了時」とはアプリケーション停止時，「1 リクエスト処理時」とは Web リソースによる 1 リクエストの処理を表します。出力量はこれらの期間に出力されるログの量を示します。

表 39-16　起動時，1 リクエスト処理時，停止時の各ログの出力例（モデル 1／モデル 2）

項番	ログの種類	出力量		
		起動時	1 リクエスト処理時	終了時
1	稼働ログ	1KB	• モデル 1：0KB • モデル 2：0.3KB	0KB
2	例外ログ	0KB	• モデル 1：0KB • モデル 2※：1.1KB	0KB

項番	ログの種類	出力量		
		起動時	1 リクエスト処理時	終了時
3	通信ログ	0KB	0.3KB ＋ HTTP リクエスト＋ HTTP レスポンス量	0KB
4	保守ログ	23KB	• モデル 1：0KB • モデル 2：0.3KB	0KB

注※

例外のスタックトレースの量になります。

各ログの見積もり式を次に示します。この結果を基に，各ログファイル一つのファイルサイズ，およびファイル面数を求めます。

```
[単位時間当たりのログの出力量] ＝
   [起動時の出力量]
   ＋[モデル1の1リクエスト処理時の出力量]×総リクエスト数×モデル1のリクエストの割合
   ＋[モデル2の1リクエスト処理時の出力量]×総リクエスト数×モデル2のリクエストの割合
         :
   ＋[モデルnの1リクエスト処理時の出力量]×総リクエスト数×モデルnのリクエストの割合
   ＋[終了時の出力量]
```

この式を基に，例を使用して各ファイルの見積もり方法を説明します。

想定する例として，1 分間に平均 100 件のリクエストがあるようなシステムで，ログを 3 時間ラップアラウンドされないようにログファイルのサイズと個数を見積もるものとします。全リクエスト中，モデル 1 のリクエストが 80%，モデル 2 のリクエストが 20%を占めるとすると，3 時間では計 18,000 リクエスト（100 リクエスト／分×180 分）が到着します。

この場合，稼働ログの出力量を計算すると，次のとおりになります。

```
1KB（初期化時の出力量）
＋{0KB×18000}（モデル1の1リクエスト処理時の出力量）×0.8（モデル1のリクエストの割合）
＋{0.3KB×18000}（モデル2の全リクエスト処理時の出力量）×0.2（モデル2のリクエストの割合）
＋0KB（終了時の出力量）
＝1081KB
```

計算の結果，ファイルサイズとファイル面数はデフォルト値で間に合います。

同様に，保守ログの出力量を計算すると，次のとおりになります。

```
23KB(初期化時の出力量)
＋{0KB×18000}（モデル1の1リクエスト処理時の出力量）×0.8（モデル1のリクエストの割合）
＋{0.3KB×18000}（モデル2の全リクエスト処理時の出力量）×0.2（モデル2のリクエストの割合）
＋0KB（終了時の出力量）
＝1103KB
```

計算の結果，ファイルサイズとファイル面数はデフォルト値で間に合います。

注意事項

通信ログでは，HTTP リクエストや HTTP レスポンスの内容によって，出力量が変化します。JSON 形式のデータを使用して通信する場合は，通信ログにも JSON 形式のデータが記録されるため，特に大きな JSON 形式のデータを使用しているときは，通信ログファイルのサイズや面数を考慮する必要があります。

39.4　性能解析トレース（PRF）

性能解析トレースとは，性能解析や障害を調査するためのトレース情報を指します。性能解析トレースを参照することで，システムの性能ボトルネックや，障害が発生した場合の，リクエスト処理の到達度合いを解析できます。

アプリケーションサーバが提供する性能解析トレースについては，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「4.6　性能解析トレース」を参照してください。

39.4.1　性能解析トレースの取得レベル

Application Server の JAX-WS 機能または JAX-RS 機能で使用できる性能解析トレースの取得レベルを次の表に示します。

表 39–17　性能解析トレースの取得レベル一覧

項番	取得レベル	目的
1	標準レベル （デフォルト）	Application Server の JAX-WS 機能または JAX-RS 機能とその他の機能との境界（入口と出口）が識別できるトレース情報が出力されます。
2	詳細レベル	標準レベルの出力内容に加えて，Application Server の JAX-WS 機能または JAX-RS 機能の内部処理のトレース情報が出力されます。
3	保守レベル	障害発生時などに保守情報を取得するためのレベルで，通常は使用しません。

39.4.2　性能解析トレースのトレース出力情報

Application Server の JAX-WS 機能または JAX-RS 機能で出力する性能解析トレースのトレース出力情報を次の表に示します。

表 39–18　性能解析トレースのトレース出力情報

項番	ファイル項目	内容
1	イベント ID	トレース取得ポイントのイベント ID です。
2	クライアントアプリケーション情報	ルートアプリケーション情報の有効範囲内の Web サービスクライアント，Web サービス間で一意の情報です。ただし，WS-RM 1.2 機能の場合，一意ではありません。オプション情報を参照してください。
3	ルートアプリケーション情報	一連のリクエストに対する処理で，一意の情報です。ただし，WS-RM 1.2 機能の場合，一意ではありません。オプション情報を参照してください。
4	リターンコード	トレース取得ポイントの種別（正常終了または異常終了）です。
5	インタフェース名	トレース取得ポイントのクラス名です。

項番	ファイル項目	内容
6	オペレーション名	トレース取得ポイントのメソッド名です。
7	オプション情報	オプション情報です。

ここでは，クライアントアプリケーション情報とルートアプリケーション情報の有効範囲，およびトレース取得ポイントについて説明します。

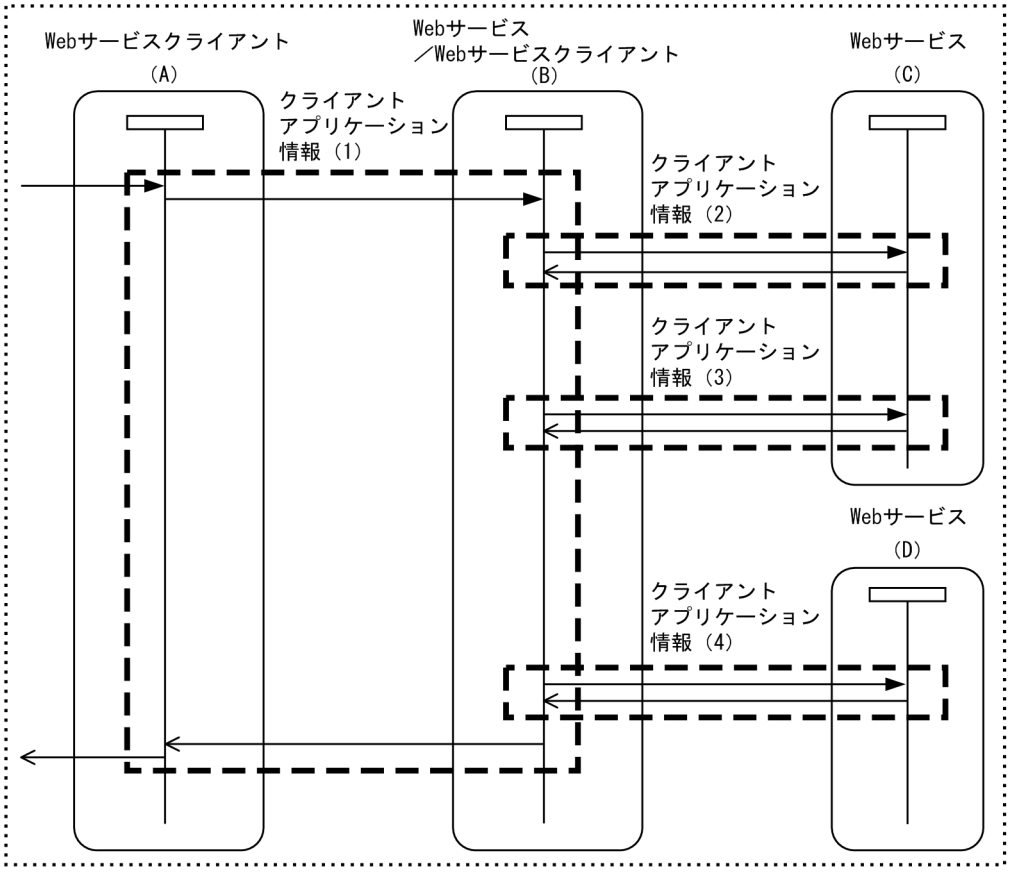
(1) クライアントアプリケーション情報とルートアプリケーション情報の有効範囲

クライアントアプリケーション情報とルートアプリケーション情報の有効範囲を，Web サービスの場合および Web リソースの場合に分けて説明します。

(a) Web サービスの場合

次の図を基に，Web サービスの場合のクライアントアプリケーション情報とルートアプリケーション情報の有効範囲について説明します。

図 39-5　クライアントアプリケーション情報とルートアプリケーション情報の有効範囲（Web サービスの場合）



（凡例）

：ルートアプリケーション情報の有効範囲

：クライアントアプリケーション情報の有効範囲

：リクエスト

：レスポンス

ルートアプリケーション情報（通信番号，プロセス ID，および IP アドレスから構成されます）は，一連のリクエストに対する処理で一意となります。

Java アプリケーションの開始コマンド（cjclstartap）で開始した Web サービスクライアント（A）から Web サービス（B）にリクエストが送信されると，Web サービス（B）が Web サービスクライアントになります。さらに，Web サービス（B）が Web サービスクライアントになって，ほかの Web サービス（C）および（D）にリクエストが送信されると，A から D までが一意の値となります。

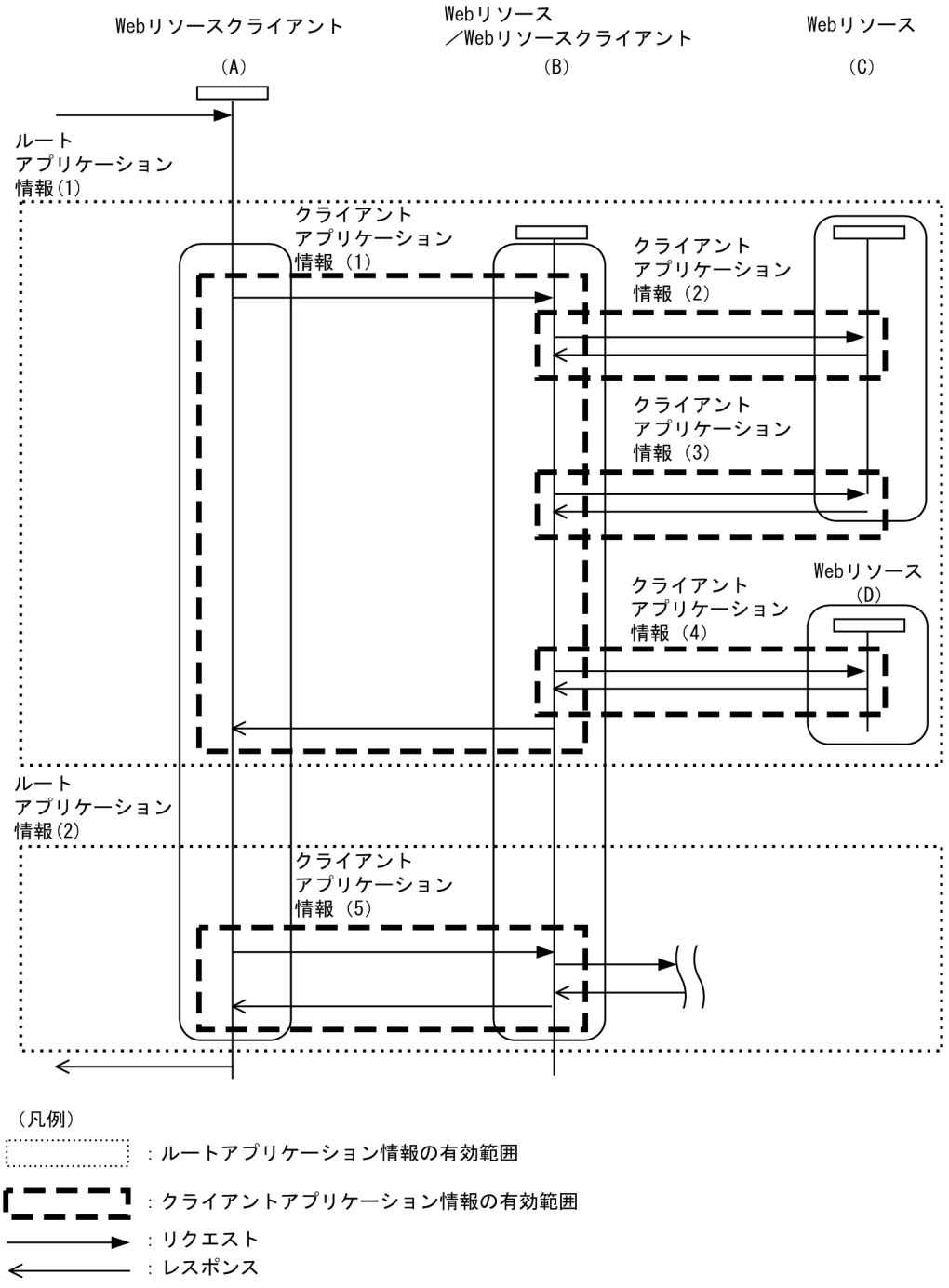
クライアントアプリケーション情報（通信番号，プロセス ID，および IP アドレスから構成されます）は，ルートアプリケーション情報の有効範囲内の Web サービスクライアント，および Web サービスの一連のリクエストに対する処理で一意となります。

クライアントアプリケーション情報（1）～（4）は，異なるクライアントアプリケーション情報になります。

(b) Web リソースの場合

次の図を基に，Web リソースの場合のクライアントアプリケーション情報とルートアプリケーション情報の有効範囲について説明します。

図 39-6 クライアントアプリケーション情報とルートアプリケーション情報の有効範囲（Web リソースの場合）



この図では，Web リソースクライアント（A）と Web リソース／Web リソースクライアント（B）は RESTful Web サービス用クライアント API を使用しています。Web リソース／Web リソースクライア

ント（B），Web リソース（C）および Web リソース（D）は JAX-RS エンジンで動作している Web リソースです。

ルートアプリケーション情報は，クライアントが RESTful Web サービス用クライアント API を呼び出してから，呼び出し結果の HTTP レスポンスを受けるまでの一連のリクエストに対する処理で一意となります。

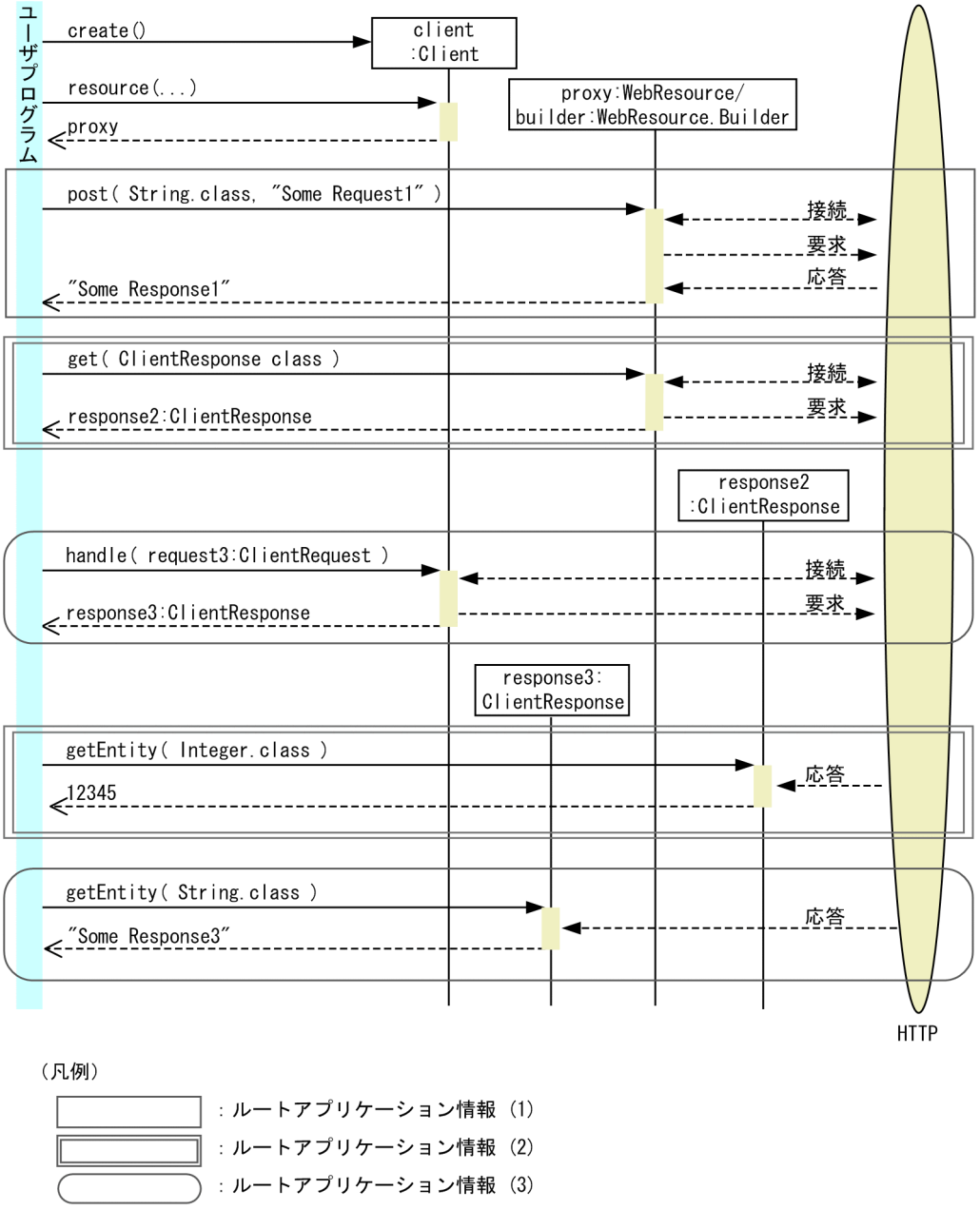
一方，クライアントアプリケーション情報はそれぞれの呼び出しにおいて一意となります。図中のクライアントアプリケーション情報（1）～（5）は，それぞれ異なる一意の値を持っています。

クライアントが，次のどちらかの操作によって返される ClientResponse オブジェクトから，それぞれ操作とは連続しないで HTTP レスポンスのエンティティを取り出した場合でも，ルートアプリケーション情報およびクライアントアプリケーション情報は，一連の呼び出しにおいて一意となります。

- Client クラスの handle メソッド
- WebResource クラスまたは WebResource.Builder クラスの method メソッド，または HTTP メソッドに対応する各メソッド

イメージを次の図で説明します。なお，図中ではクライアントアプリケーション情報の範囲がルートアプリケーション情報の範囲と同じなので，クライアントアプリケーション情報は省略しています。

図 39-7　連続しない呼び出しとアプリケーション情報



この場合，WebResource クラスの get(Class)メソッドと ClientResponse クラスの getEntity(Class)メソッドの連続しない呼び出しでのルートアプリケーション情報（図中の「ルートアプリケーション情報（2）」）は，HTTP 通信の接続から要求・応答まで一度に実行される WebResource クラスの post(Class, Object)メソッドのルートアプリケーション情報（図中の「ルートアプリケーション情報（1）」）と同様に一意です。

また，Client クラスの handle(Class)メソッドと ClientResponse クラスの getEntity(Class)メソッドの連続しない呼び出しでのルートアプリケーション情報（図中の「ルートアプリケーション情報（3）」）も一意です。

(2)　性能解析トレースのトレース取得ポイント

Application Server の JAX-WS 機能で出力する性能解析トレースのトレース取得ポイントを次の図に示します。なお，WS-RM 1.2 機能を使用する場合は，ここに記載する以外の性能解析トレースも出力されます。詳細については，「39.4.2(2)(c)　WS-RM 1.2 機能使用時のトレース取得ポイント」を参照してください。

(a)　POJO の Web サービス呼び出し時のトレース取得ポイント

POJO の Web サービス呼び出し時のトレース取得ポイントについて，request-response オペレーションの場合を図 39-8 に，one-way オペレーションの場合を図 39-9 に示します。ただし，one-way オペレーションでプロバイダ実装クラスが null を返すときの，トレース取得ポイントについては，図 39-8 を参照してください。

図 39-8　POJO の Web サービス呼び出し時のトレース取得ポイント（request-response オペレーション）

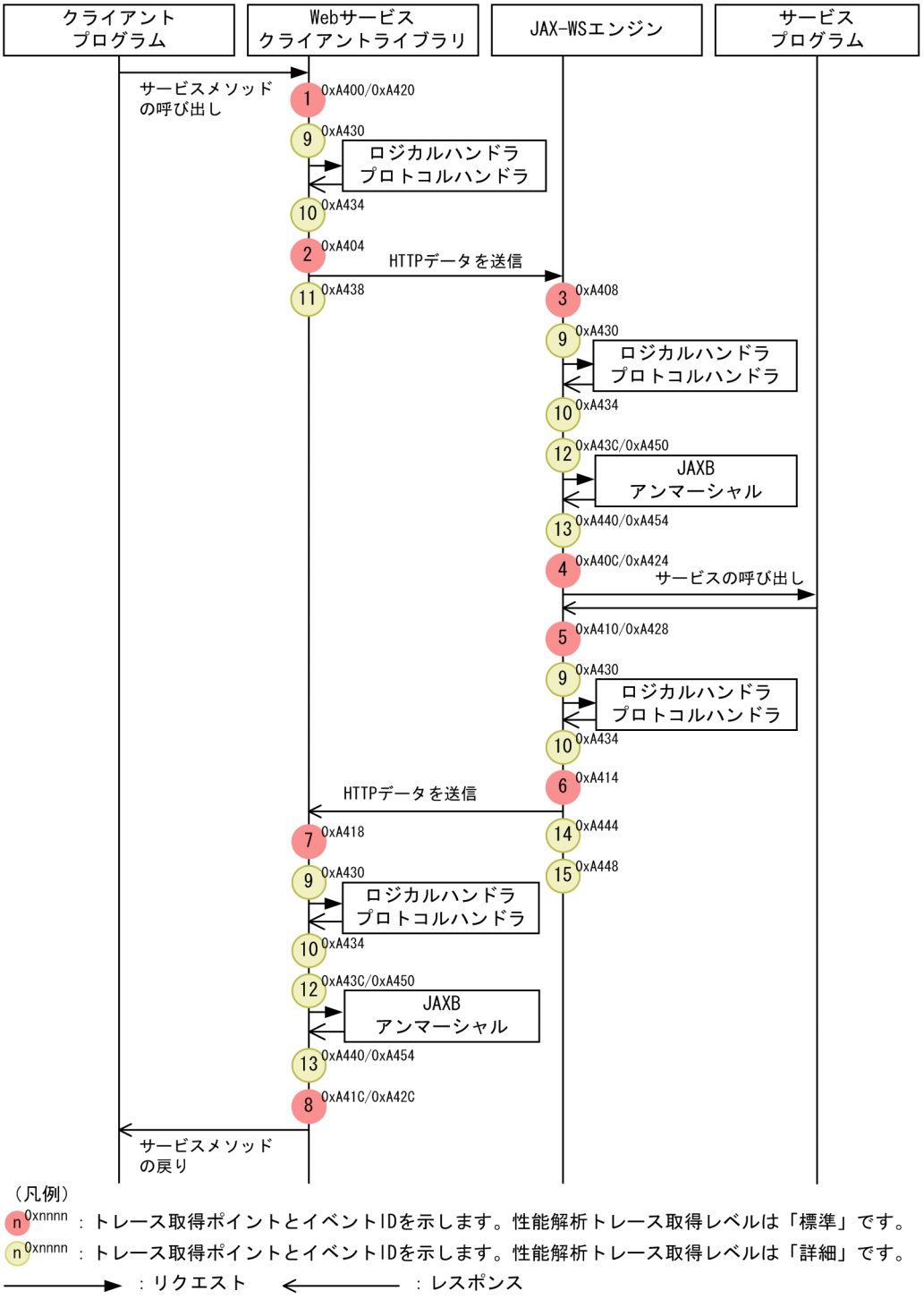
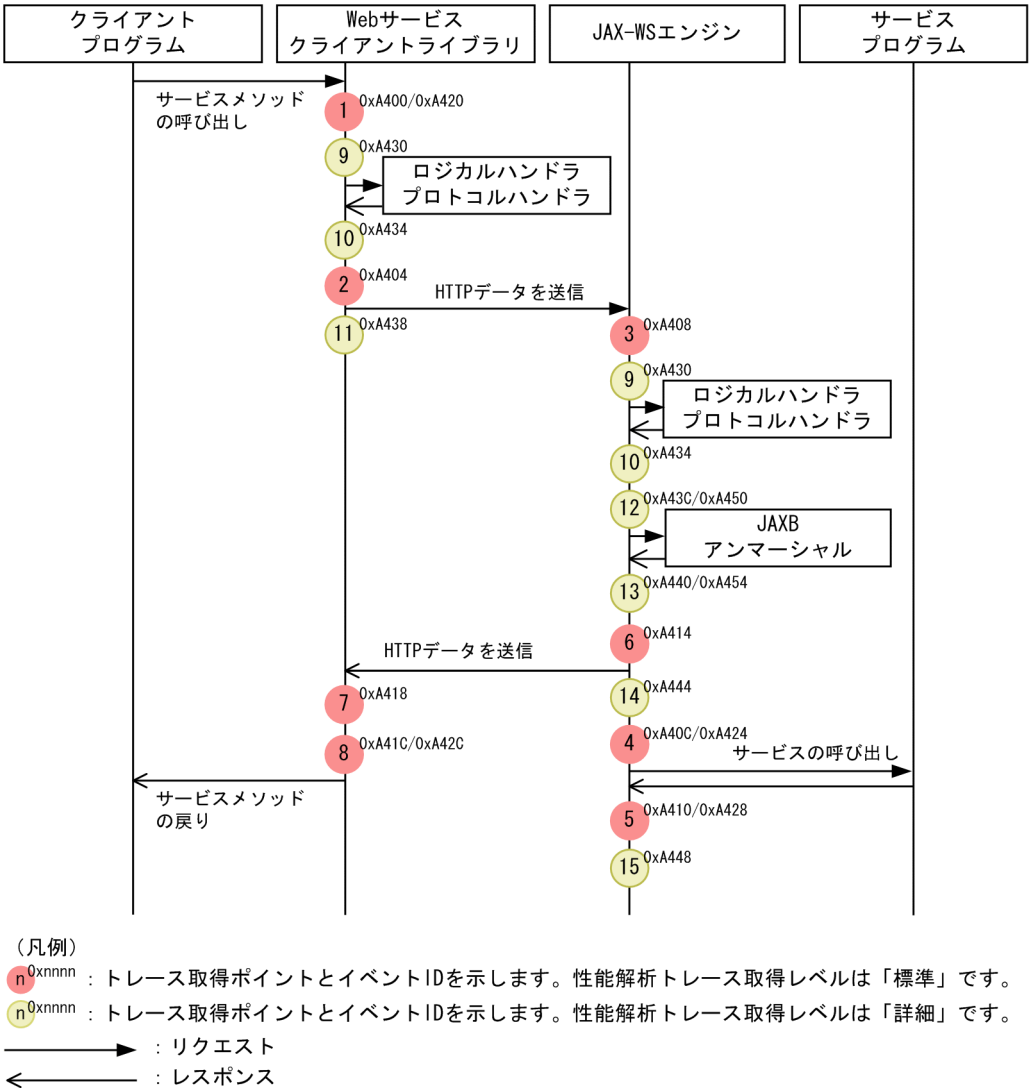


図 39-9　POJO の Web サービス呼び出し時のトレース取得ポイント（one-way オペレーション）



各トレース取得ポイントとイベント ID の対応を次の表に示します。

表 39-19　標準レベルでのトレース取得ポイントと出力される情報（JAX-WS 機能・POJO の Web サービス）

図中の番号	取得ポイント		出力される情報	
			イベント ID	オプション情報
1	Web サービスクライアントライブラリの開始時	スタブベースの場合	0xA400	－
		ディスパッチベースの場合	0xA420	
2	Web サービスクライアントライブラリの HTTP メッセージ送信前		0xA404	エンドポイント URL
3	JAX-WS エンジンのサーブレット開始時		0xA408	コンテキストルート
4	Web サービス呼び出し前	Web サービス実装クラスの場合	0xA40C	サービスメソッド名

図中の番号	取得ポイント		出力される情報	
			イベント ID	オプション情報
		プロバイダ実装クラスの場合	0xA424	−
5	Web サービス呼び出し後	Web サービス実装クラスの場合	0xA410	異常終了の場合は例外名
		プロバイダ実装クラスの場合	0xA428	
6	JAX-WS エンジンの HTTP メッセージ送信前		0xA414※	−
7	Web サービスクライアントライブラリの HTTP メッセージ受信後		0xA418	異常終了の場合は例外名
8	Web サービスクライアントライブラリ終了時	スタブベースの場合	0xA41C	異常終了の場合は例外名
		ディスパッチベースの場合	0xA42C	

（凡例）

−：オプション情報はありません。

注※

Web サービス呼び出し後にプロバイダ実装クラスが null を返した場合，トレースを取得しません。

表 39–20　詳細レベルでのトレース取得ポイントと出力される情報（JAX-WS 機能・POJO の Web サービス）

図中の番号	取得ポイント		出力される情報	
			イベント ID	オプション情報
9	ハンドラ呼び出し前		0xA430※	ハンドラの位置とハンドラクラス名
10	ハンドラ呼び出し後		0xA434※	異常終了の場合は例外名
11	Web サービスクライアントライブラリの HTTP メッセージ送信後		0xA438	異常終了の場合は例外名
12	アンマーシャル前	スタブベース，および Web サービス実装クラスの場合	0xA43C	−
		ディスパッチベース，およびプロバイダ実装クラスの場合	0xA450※	
13	アンマーシャル後	スタブベース，および Web サービス実装クラスの場合	0xA440	−
		ディスパッチベース，およびプロバイダ実装クラスの場合	0xA454※	
14	JAX-WS エンジンの HTTP メッセージ送信後		0xA444※	異常終了の場合は例外名
15	JAX-WS エンジンのサーブレット終了時		0xA448	異常終了の場合は例外名

（凡例）

−：オプション情報はありません。

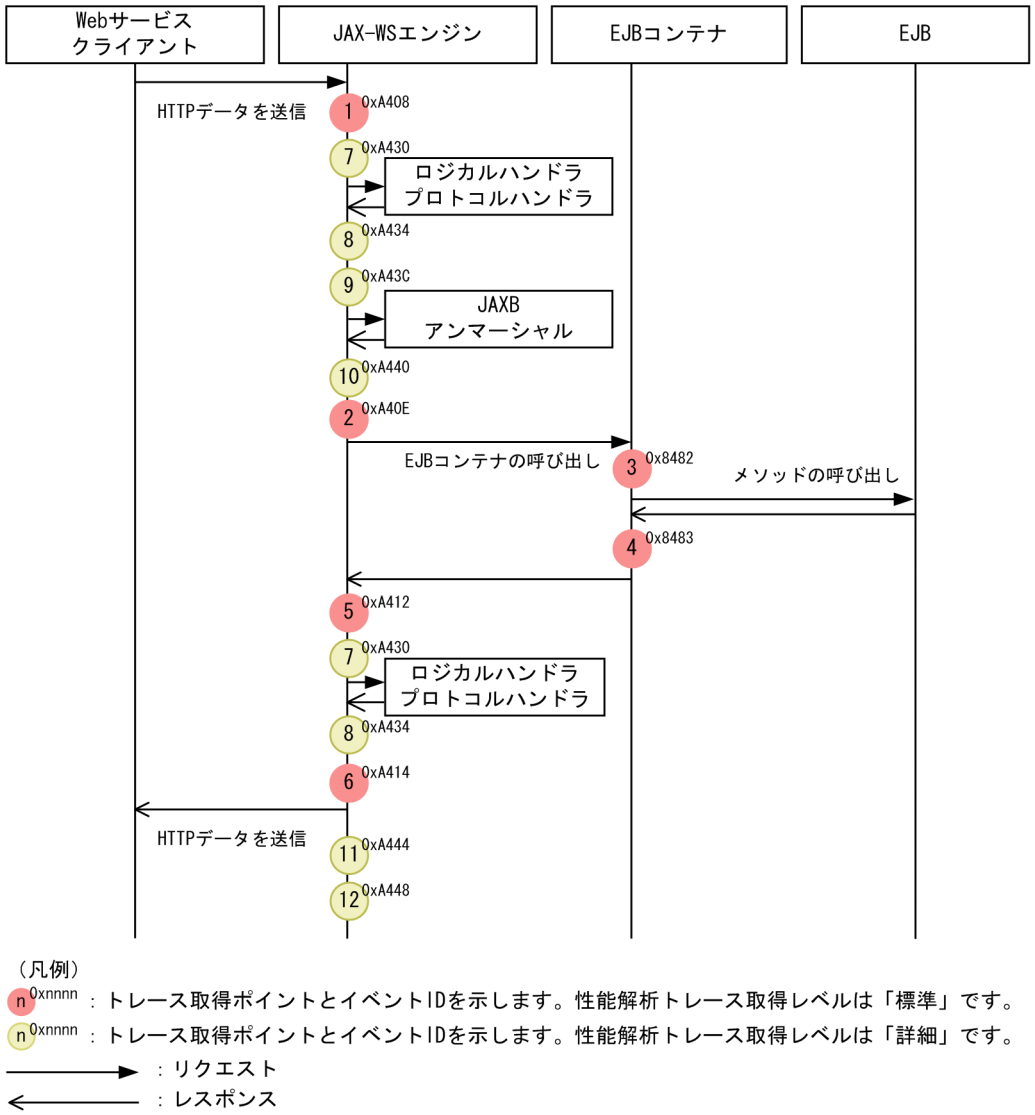
注※

Web サービス呼び出し後にプロバイダ実装クラスが null を返した場合，トレースを取得しません。

(b) EJB の Web サービス呼び出し時のトレース取得ポイント

request-response オペレーションの場合の，EJB の Web サービス呼び出し時のトレース取得ポイントを次の図に示します。なお，Web サービスクライアントのトレース取得ポイントは，POJO の Web サービス呼び出し時と同じです。また，one-way オペレーションの場合の，Web サービスにおけるトレース取得ポイントについては，「図 39-9 POJO の Web サービス呼び出し時のトレース取得ポイント（one-way オペレーション）」の「サービスの呼び出し」を「EJB コンテナの呼び出し」に読み替えてください。

図 39-10 EJB の Web サービス呼び出し時のトレース取得ポイント（request-response オペレーション）



各トレース取得ポイントとイベント ID の対応を次の表に示します。

表 39–21　標準レベルでのトレース取得ポイントと出力される情報（JAX-WS 機能・EJB の Web サービス）

図中の番号	取得ポイント		出力される情報	
			イベント ID	オプション情報
1	JAX-WS エンジンのサーブレット開始時		0xA408	コンテキストルート
2	EJB コンテナ呼び出し前	Web サービス実装クラス	0xA40E	サービスメソッド名
3	EJB のメソッド呼び出し前		0x8482	－
4	EJB のメソッド呼び出し後		0x8483	－
5	EJB コンテナ呼び出し後	Web サービス実装クラス	0xA412	異常終了の場合は例外名
6	JAX-WS エンジンの HTTP メッセージ送信前		0xA414	異常終了の場合は例外名

（凡例）

－：オプション情報はありません。

表 39–22　詳細レベルでのトレース取得ポイントと出力される情報（JAX-WS 機能・EJB の Web サービス）

図中の番号	取得ポイント		出力される情報	
			イベント ID	オプション情報
7	ハンドラ呼び出し前		0xA430	ハンドラの位置とハンドラクラス名
8	ハンドラ呼び出し後		0xA434	異常終了の場合は例外名
9	アンマーシャル前	Web サービス実装クラス	0xA43C	－
10	アンマーシャル後	Web サービス実装クラス	0xA440	－
11	JAX-WS エンジンの HTTP メッセージ送信後		0xA444	異常終了の場合は例外名
12	JAX-WS エンジンのサーブレット終了時		0xA448	異常終了の場合は例外名

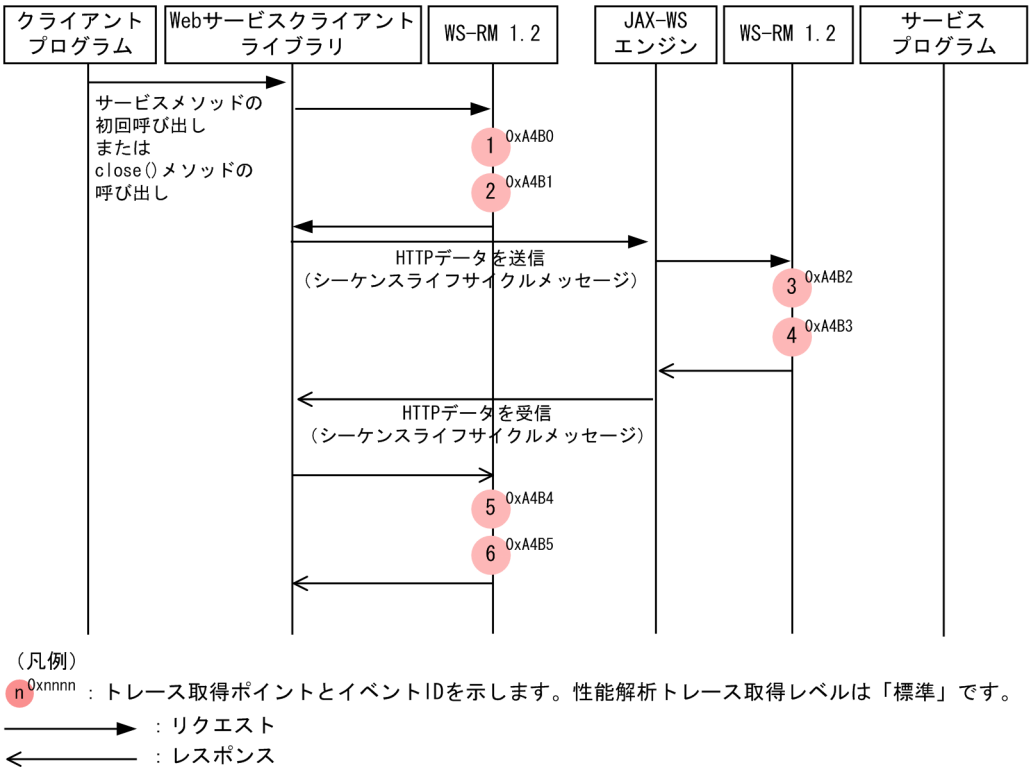
（凡例）

－：オプション情報はありません。

(c)　WS-RM 1.2 機能使用時のトレース取得ポイント

WS-RM 1.2 機能でシーケンスライフサイクルメッセージを送受信するときに出力される性能解析トレースのトレース取得ポイントを次の図に示します。

図 39-11　シーケンスライフサイクルメッセージ送受信時のトレース取得ポイント



各トレース取得ポイントとイベント ID の対応を次の表に示します。

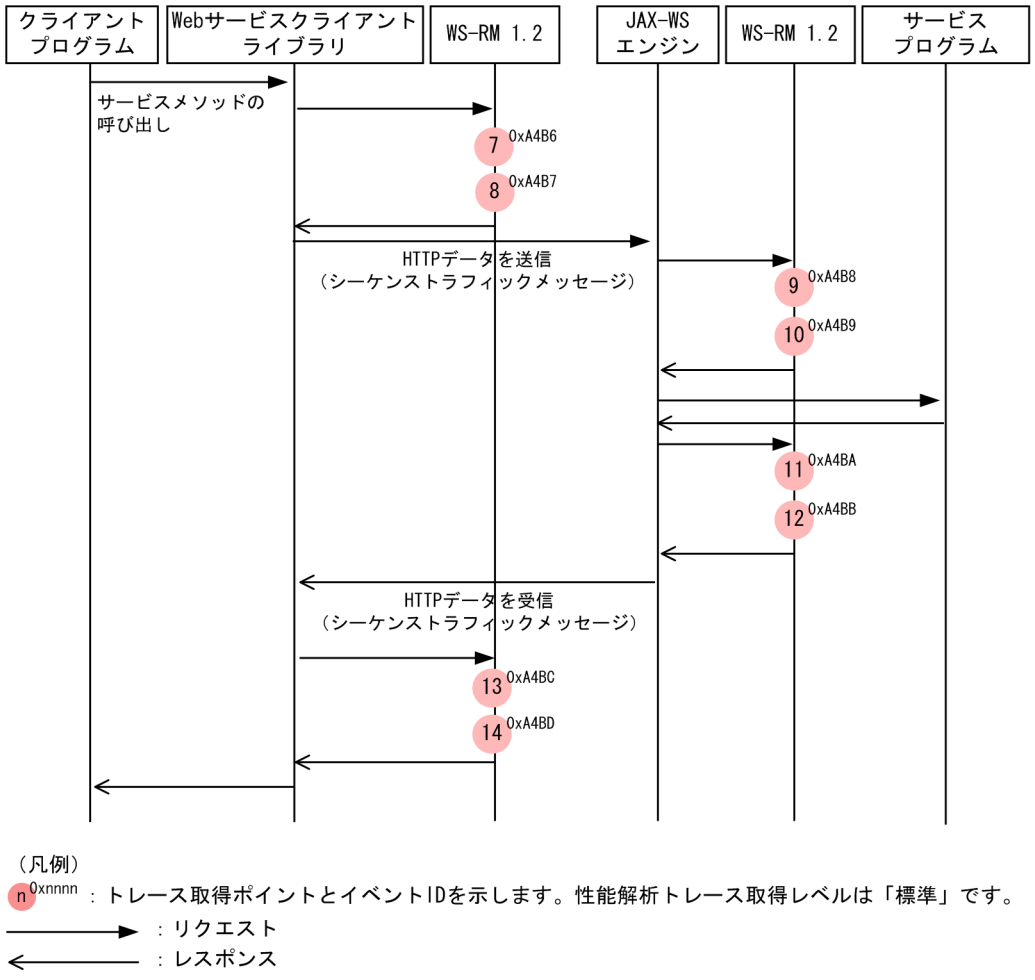
表 39-23　標準レベルでのトレース取得ポイントと出力される情報（WS-RM 1.2 機能のライフサイクルメッセージ）

図中の番号	取得ポイント	出力される情報	
		イベント ID	オプション情報
1	Web サービスクライアント側 WS-RM 1.2 機能開始時	0xA4B0	次のどれかが出力されます。 • CreateSequence • CloseSequence,[SequenceID] • TerminateSequence,[SequenceID]
2	Web サービスクライアント側シーケンスライフサイクルメッセージ送信前	0xA4B1	次のどれかが出力されます。 • CreateSequence • CloseSequence,[SequenceID] • TerminateSequence,[SequenceID]
3	Web サービス側 WS-RM 1.2 機能開始時	0xA4B2	次のどれかが出力されます。 • CreateSequence • CloseSequence,[SequenceID] • TerminateSequence,[SequenceID]
4	Web サービス側 WS-RM 1.2 機能終了時	0xA4B3	次のどれかが出力されます。 • CreateSequence,[SequenceID] • CloseSequence,[SequenceID]

図中の番号	取得ポイント	出力される情報	
		イベント ID	オプション情報
			• TerminateSequence,[SequenceID] • 異常終了の場合は例外名
5	Web サービスクライアント側シーケンスライフサイクルメッセージ受信後	0xA4B4	次のどれかが出力されます。 • CreateSequence • CloseSequence,[SequenceID] • TerminateSequence,[SequenceID] • 異常終了の場合は例外名
6	Web サービスクライアント側 WS-RM 1.2 機能終了時	0xA4B5	次のどれかが出力されます。 • CreateSequence,[SequenceID] • CloseSequence,[SequenceID] • TerminateSequence,[SequenceID] • 異常終了の場合は例外名

WS-RM 1.2 機能のシーケンストラフィックメッセージの性能解析トレースのトレース取得ポイントを次の図に示します。

図 39-12　シーケンストラフィックメッセージのトレース取得ポイント



（凡例）

n 0xnnnn ：トレース取得ポイントとイベントIDを示します。性能解析トレース取得レベルは「標準」です。

→ ：リクエスト

← ：レスポンス

各トレース取得ポイントとイベント ID の対応を次の表に示します。

表 39–24　標準レベルでのトレース取得ポイントと出力される情報（WS-RM 1.2 機能のシーケンストラフィックメッセージ）

図中の番号	取得ポイント	出力される情報	
		イベント ID	オプション情報
7	リクエストでの Web サービスクライアント側 WS-RM 1.2 機能開始時	0xA4B6	[SequenceID],[MessageNumber]
8	リクエストでの Web サービスクライアント側 WS-RM 1.2 機能終了時	0xA4B7	次のどちらかが出力されます。 • [SequenceID],[MessageNumber] • 異常終了の場合は例外名
9	リクエストでの Web サービス側 WS-RM 1.2 機能開始時	0xA4B8	[SequenceID],[MessageNumber]
10	リクエストでの Web サービス側 WS-RM 1.2 機能終了時	0xA4B9	次のどちらかが出力されます。 • [SequenceID],[MessageNumber] • 異常終了の場合は例外名
11	レスポンスでの Web サービス側 WS-RM 1.2 機能開始時※1	0xA4BA	[SequenceID],[MessageNumber]
12	レスポンスでの Web サービス側 WS-RM 1.2 機能終了時※2	0xA4BB	次のどちらかが出力されます。 • [SequenceID],[MessageNumber] • 異常終了の場合は例外名
13	レスポンスでの Web サービスクライアント側 WS-RM 1.2 機能開始時	0xA4BC	[SequenceID],[MessageNumber]
14	レスポンスでの Web サービスクライアント側 WS-RM 1.2 機能終了時	0xA4BD	次のどちらかが出力されます。 • [SequenceID] • 異常終了の場合は例外名

注※1

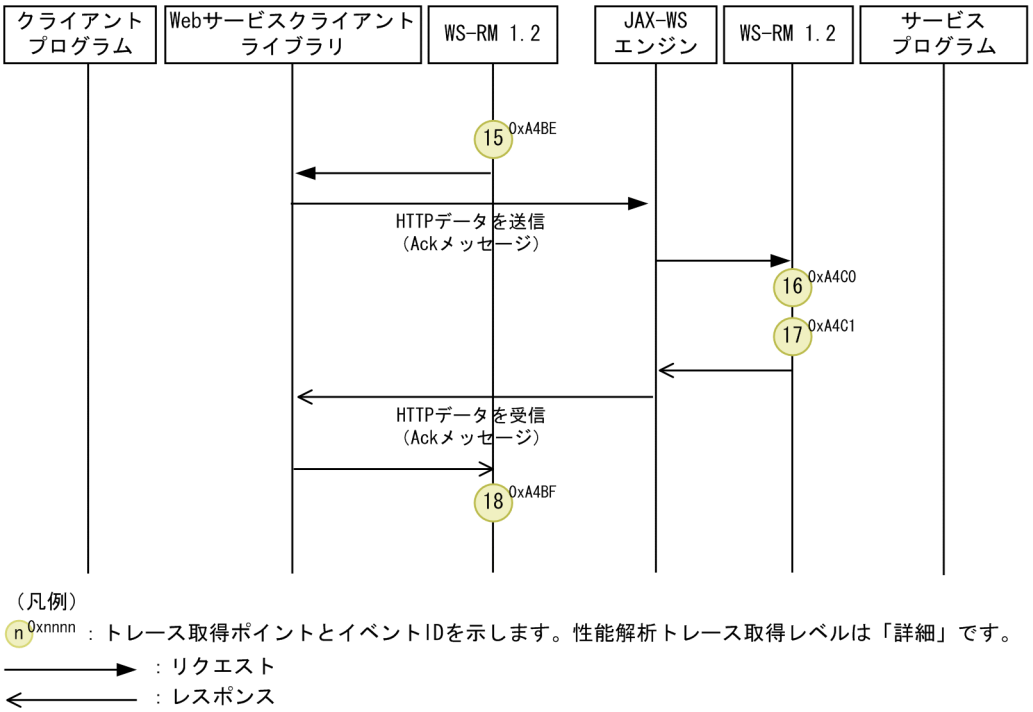
重複したメッセージを受信した場合は出力されません。

注※2

重複したメッセージを受信した場合に，HTTP ステータスコード 202 を返すときは出力されません。

WS-RM 1.2 機能の Ack メッセージの性能解析トレースのトレース取得ポイントを次の図に示します。

図 39–13　Ack メッセージのトレース取得ポイント



各トレース取得ポイントとイベント ID の対応を次の表に示します。

表 39–25　詳細レベルでのトレース取得ポイントと出力される情報（WS-RM 1.2 機能の Ack メッセージ）

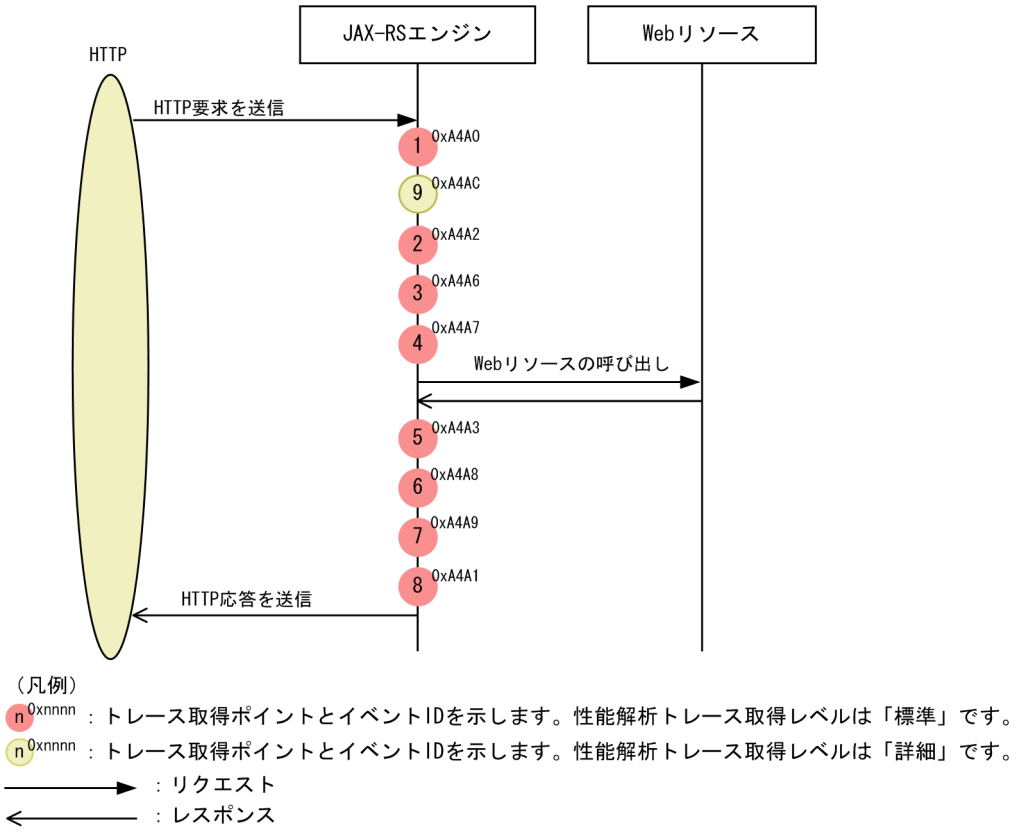
図中の番号	取得ポイント	出力される情報	
		イベント ID	オプション情報
15	Web サービスクライアント側 WS-RM 1.2 機能開始時	0xA4BE	[SequenceID]
16	Web サービス側 WS-RM 1.2 機能開始時	0xA4C0	[SequenceID]
17	Web サービス側 WS-RM 1.2 機能終了時	0xA4C1	次のどちらかが出力されます。 • [SequenceID] • 異常終了の場合は例外名
18	Web サービスクライアント側 WS-RM 1.2 機能終了時	0xA4BF	次のどちらかが出力されます。 • [SequenceID] • 異常終了の場合は例外名

(d)　RESTful Web サービス呼び出し時のトレース取得ポイント

RESTful Web サービス（Web サービス）呼び出し時のトレース取得ポイントを，リソース側，クライアント側に分けてそれぞれ次の図に示します。

リソース側

図 39-14　RESTful Web サービス呼び出し時のトレース取得ポイント（リソース側）



各トレース取得ポイントとイベント ID の対応を次の表に示します。

表 39-26　標準レベルでのトレース取得ポイントと出力される情報（JAX-RS 機能・RESTful Web サービス）

図中の番号	取得ポイント	出力される情報	
		イベント ID	オプション情報
1	JAX-RS エンジンのサーブレット開始時	0xA4A0	−
2	Web リソース呼び出し前	0xA4A2	次のどれかが出力されます。 • JResponseOutInvoker • ObjectOutInvoker • ResponseOutInvoker • TypeOutInvoker • VoidOutInvoker • VoidVoidMethodInvoker • HttpReqResDispatcher
3	アンマーシャル前	0xA4A6	−
4	アンマーシャル後	0xA4A7	例外発生の場合は例外名
5	Web リソース呼び出し後	0xA4A3	次のどれかが出力されます。 • JResponseOutInvoker

図中の番号	取得ポイント	出力される情報	
		イベント ID	オプション情報
			• ObjectOutInvoker • ResponseOutInvoker • TypeOutInvoker • VoidOutInvoker • VoidVoidMethodInvoker • HttpReqResDispatcher
6	マーシャル前	0xA4A8	－
7	マーシャル後	0xA4A9	例外発生の場合は例外名
8	JAX-RS エンジンの HTTP メッセージ送信前	0xA4A1	例外発生の場合は例外名

（凡例）

－：オプション情報はありません。

表 39–27　詳細レベルでのトレース取得ポイントと出力される情報（JAX-RS 機能・RESTful Web サービス）

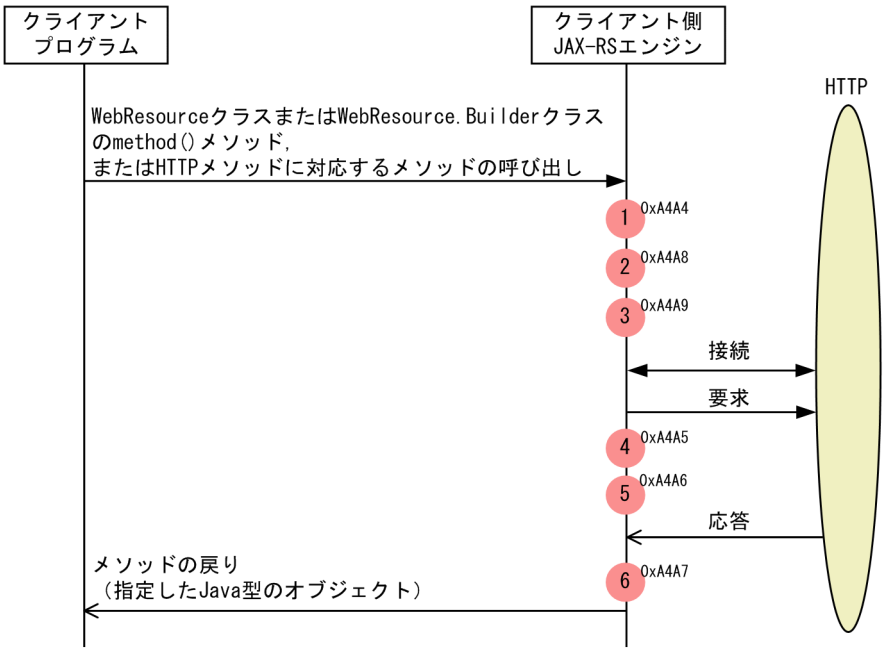
図中の番号	取得ポイント	出力される情報	
		イベント ID	オプション情報
9	HTTP リクエストとリソースのマッチング処理実行時	0xA4AC	次のどれかが出力されます。 • accept root resource classes • forwarding view to JSP page • accept implicit view • match path • accept sub-resource methods • accept resource methods • matched sub-resource method • matched resource method • accept resource message in ResourceClassRule • accept resource message in ResourceOjectRule • accept right hand path redirect • accept right hand path • accept sub-resource locator • accept termination • matched message body reader • matched exception mapper • mapped exception to response • matched message body writer

クライアント側

「11.4.1　Web リソースクライアントのユースケース」で説明したユースケースごとに説明します。

図 39-15　RESTful Web サービス呼び出し時のトレース取得ポイント（クライアント側）

●Java型を指定してHTTPリクエストおよびHTTPレスポンスを送受信する場合



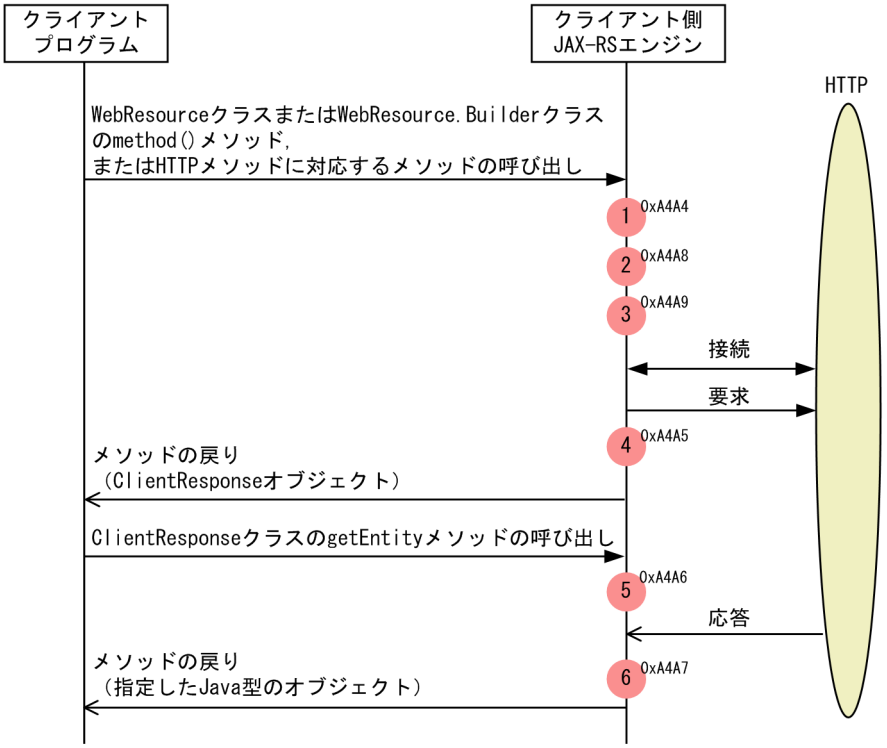
（凡例）

n 0xnnnn ：トレース取得ポイントとイベントIDを示します。性能解析トレース取得レベルは「標準」です。

：リクエスト

：レスポンス

●Java型を指定してHTTPリクエストを送信し，汎用型（ClientResponse）でHTTPレスポンスを受信する場合



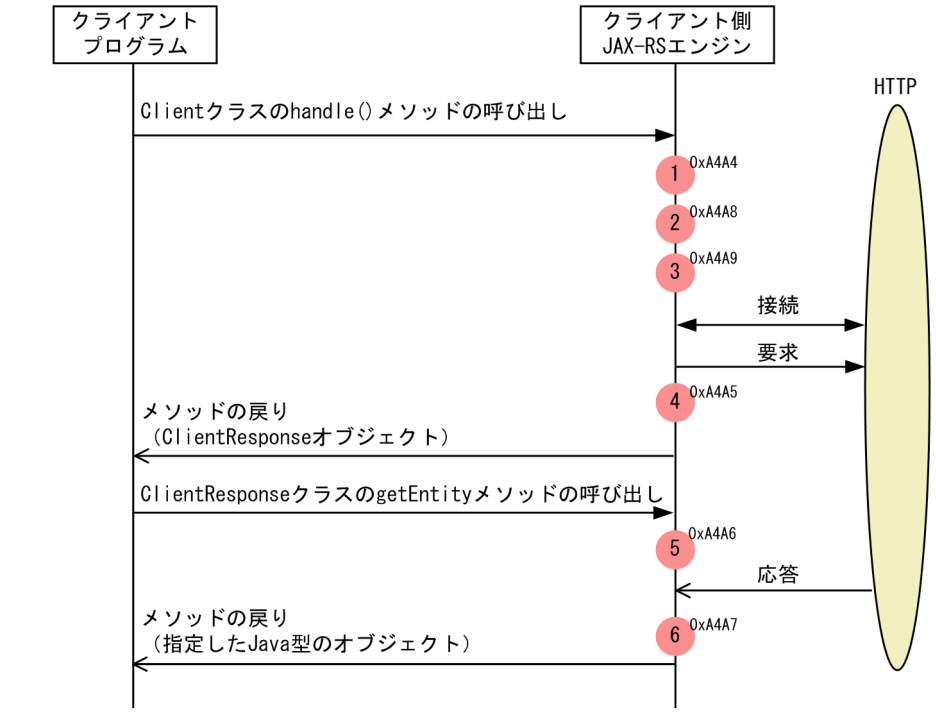
（凡例）

n 0xnnnn ：トレース取得ポイントとイベントIDを示します。性能解析トレース取得レベルは「標準」です。

——→ ：リクエスト

←—— ：レスポンス

●汎用型でHTTPリクエストおよびHTTPレスポンスを送受信する場合



（凡例）

n 0xnnnn ：トレース取得ポイントとイベントIDを示します。性能解析トレース取得レベルは「標準」です。

→ ：リクエスト

← ：レスポンス

各トレース取得ポイントとイベント ID の対応を次の表に示します。

表 39-28　標準レベルでのトレース取得ポイントと出力される情報（JAX-RS 機能・RESTful Web サービス用クライアント API）

図中の番号	取得ポイント	出力される情報	
		イベント ID	オプション情報
1	HTTP メソッド呼び出しの開始時	0xA4A4	エンドポイント URL
2	マーシャル前	0xA4A8	－
3	マーシャル後	0xA4A9	－
4	HTTP メソッド呼び出しの終了時	0xA4A5	例外発生の場合は例外名
5	アンマーシャル前	0xA4A6	－
6	アンマーシャル後	0xA4A7	例外発生の場合は例外名

（凡例）

－：オプション情報はありません。

39.4.3　性能解析トレースによる性能解析方法

性能解析トレースファイルとは，性能解析トレースを CSV 形式で出力したテキストファイルを指します。

(1)　Web サービスの場合

性能解析トレースファイルを使用して，JAX-WS エンジンを含む Web サービス全体のレスポンスタイムと，Web サービス（UP）のレスポンスタイムを解析する方法を説明します。なお，ここでは Web サービス実装クラスで開発した Web サービスと，スタブベースの Web サービスクライアントの例を説明します。そのほかの形態の Web サービスおよび Web サービスクライアントの場合はイベント ID が異なることがあるので，必要に応じて読み替えてください。

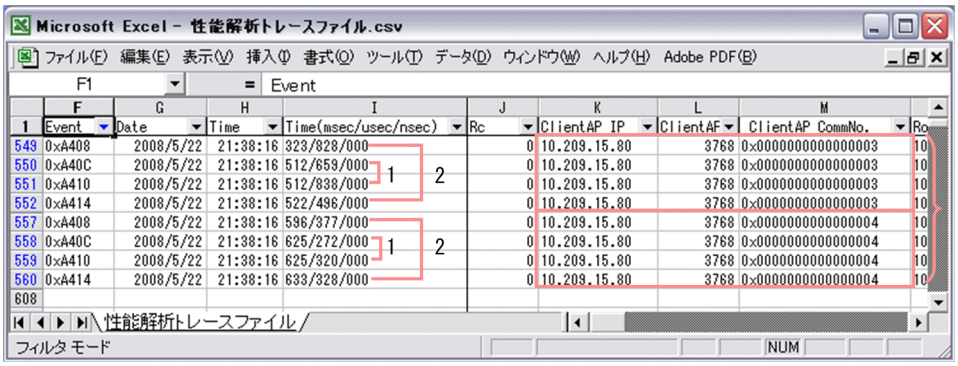
イベント ID「0xA408」,「0xA40C」,「0xA410」，および「0xA414」をキーにして，性能解析トレースファイルをフィルタリングします。各イベント ID に対するトレース取得ポイントを次の表に示します。

表 39-29　フィルタリングするイベント ID に対応するトレース取得ポイント

項番	イベント ID	トレース取得ポイント
1	0xA408	JAX-WS エンジンのサーブレット開始時
2	0xA40C	Web サービス呼び出し前
3	0xA410	Web サービス呼び出し後
4	0xA414	JAX-WS エンジンの SOAP メッセージ送信前

リクエストを 2 回送信した場合に，イベント ID「0xA408」,「0xA40C」,「0xA410」および「0xA414」をキーにして，性能解析トレースファイルをフィルタリングした例を次の図に示します。

図 39-16　性能解析トレースをフィルタリングした例



1. 0xA40Cと0xA410の時間差
   作成したプログラムのレスポンスタイム
2. 0xA408と0xA414の時間差
   Webサービス全体のレスポンスタイム
3. クライアントAP情報が同一（同じリクエストが処理中）

クライアントアプリケーション情報から，リクエストの処理状況を確認できます。同じリクエストを処理している場合，同じクライアントアプリケーション情報が出力されます。

同一クライアントアプリケーション情報の中のイベント ID「0xA408」および「0xA414」のトレース取得時刻から，JAX-WS エンジンを含めた Web サービス全体のレスポンスタイムを解析できます。また，「0xA40C」および「0xA410」のトレース取得時刻から，Web サービス（UP）のレスポンスタイムを解析できます。

(2) Web リソースの場合

性能解析トレースファイルを使用して，JAX-RS エンジンを含む Web リソース全体のレスポンスタイムと，Web リソース（UP）のレスポンスタイムを解析する方法を説明します。

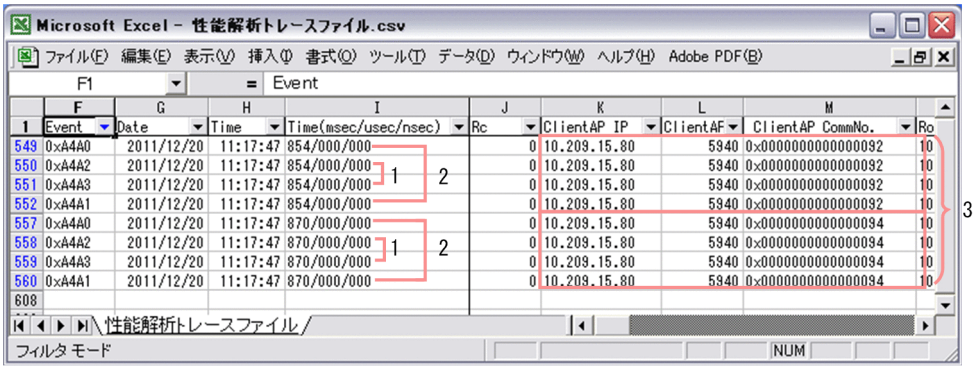
イベント ID「0xA4A0」，「0xA4A2」，「0xA4A3」および「0xA4A1」をキーにして，性能解析トレースファイルをフィルタリングします。各イベント ID に対するトレース取得ポイントを次の表に示します。

表 39-30 フィルタリングするイベント ID に対応するトレース取得ポイント

項番	イベント ID	トレース取得ポイント
1	0xA4A0	JAX-RS エンジンのサーブレット開始時
2	0xA4A2	Web リソース呼び出し前
3	0xA4A3	Web リソース呼び出し後
4	0xA4A1	JAX-RS エンジンの HTTP メッセージ送信前

リクエストを 2 回送信した場合に，イベント ID「0xA4A0」，「0xA4A2」，「0xA4A3」および「0xA4A1」をキーにして，性能解析トレースファイルをフィルタリングした例を次の図に示します。

図 39-17 性能解析トレースをフィルタリングした例



1. 0xA4A2と0xA4A3の時間差
   作成したプログラムのレスポンスタイム
2. 0xA4A0と0xA4A1の時間差
   Webリソース全体のレスポンスタイム
3. クライアントAP情報が同一（同じリクエストが処理中）

クライアントアプリケーション情報から，リクエストの処理状況を確認できます。同じリクエストを処理している場合，同じクライアントアプリケーション情報が出力されます。

同一クライアントアプリケーション情報の中のイベント ID「0xA4A0」および「0xA4A1」のトレース取得時刻から，JAX-RS エンジンを含めた Web リソース全体のレスポンスタイムを解析できます。また，

「0xA4A2」および「0xA4A3」のトレース取得時刻から，Web リソース（UP）のレスポンスタイムを解析できます。

付録

付録 A　旧バージョンからの移行

旧バージョンの機能である SOAP アプリケーション開発支援機能，および SOAP 通信基盤を利用して開発した SOAP アプリケーションの移行について，および JAX-WS エンジンを利用した Application Server を 08-00～08-70 から 09-00 以降にバージョンアップする場合の注意事項について説明します。

SOAP アプリケーション開発支援機能／SOAP 通信基盤で開発した SOAP アプリケーションおよび SOAP クライアントは，バージョンアップインストールして同じ環境で利用できます。この場合，特に設定は不要です。

SOAP アプリケーション開発支援機能／SOAP 通信基盤で開発した SOAP アプリケーション，および SOAP クライアントを JAX-WS エンジン上で利用する場合，バージョンアップインストールしたあとに，JAX-WS エンジンの環境に切り替えます。また，JAX-WS 2.2 仕様および Application Server の JAX-WS 機能のサポート範囲に従って，改めて Web サービスおよび Web サービスクライアントを作成し直す必要があります。

付録 A.1　バージョンアップインストール

SOAP アプリケーション開発支援機能／SOAP 通信基盤，JAX-WS 機能，および JAX-RS 機能は，Application Server のインストール時に同時にインストールされます。

ここでは，旧バージョンがインストールされているマシンに対して，バージョンアップインストールする場合の手順および注意事項について説明します。

(1)　J2EE サーバの移行

Application Server をバージョンアップインストールする場合，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」の「10.2.2　更新インストールの場合」に記載されている移行手順に従ってインストールしてください。

バージョンアップインストールすると，SOAP アプリケーション開発支援機能および SOAP 通信基盤を利用するように設定されます。

(2)　Web サービスのアンデプロイ

Web サービス（または SOAP アプリケーション）を移行する場合，あらかじめ J2EE サーバにデプロイされている Web サービス（または SOAP アプリケーション）をアンデプロイしてください。

(3)　動作環境の切り替え

SOAP アプリケーション開発支援機能／SOAP 通信基盤，JAX-WS エンジン，および JAX-RS エンジンの切り替えは，J2EE サーバ用オプション定義ファイルで定義します。ここでは，J2EE サーバ用オプション定義ファイルの指定内容，および指定方法について説明します。

J2EE サーバ用オプション定義ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「2.2.2　usrconf.cfg（J2EE サーバ用オプション定義ファイル)」を参照してください。

- JAX-WS エンジンおよび JAX-RS エンジンを利用する場合

  JAX-WS エンジンおよび JAX-RS エンジンを利用する場合，J2EE サーバ用オプション定義ファイルの add.class.path の「cjjaxws.jar」と「cjjaxrs.jar」の行を有効にし，「hitsaaj.jar」の行を無効にします。次の例に従って定義してください。

```
...
#add.class.path=<cosminexus.home>¥c4web¥lib¥hitsaaj.jar
add.class.path=<cosminexus.home>¥jaxws¥lib¥cjjaxws.jar
add.class.path=<cosminexus.home>¥jaxrs¥lib¥cjjaxrs.jar
...
```

- SOAP アプリケーション開発支援機能／SOAP 通信基盤を利用する場合

  SOAP アプリケーション開発支援機能／SOAP 通信基盤を利用する場合，J2EE サーバ用オプション定義ファイルの add.class.path の「hitsaaj.jar」の行を有効にし，「cjjaxws.jar」の行を無効にします。次の例に従って定義してください。

```
...
add.class.path=<cosminexus.home>¥c4web¥lib¥hitsaaj.jar
#add.class.path=<cosminexus.home>¥jaxws¥lib¥cjjaxws.jar
...
```

注意事項

有効，無効にする行に矛盾がないように設定してください。例えば，「hitsaaj.jar」，「cjjaxws.jar」，および「cjjaxrs.jar」すべての行を無効にした場合，またはすべての行を有効にした場合の動作は保証されません。

J2EE サーバ用オプション定義ファイルの編集方法を説明します。

ここで説明するどの方法でも，J2EE サーバ用オプション定義ファイルを編集（保存）したあとに，J2EE サーバを再起動してください。J2EE サーバを再起動しない場合は，編集内容が反映されません。

(a)　直接編集する場合

直接編集する場合，次の場所に格納された J2EE サーバ用オプション定義ファイルをテキストエディタで開き，内容を変更します。

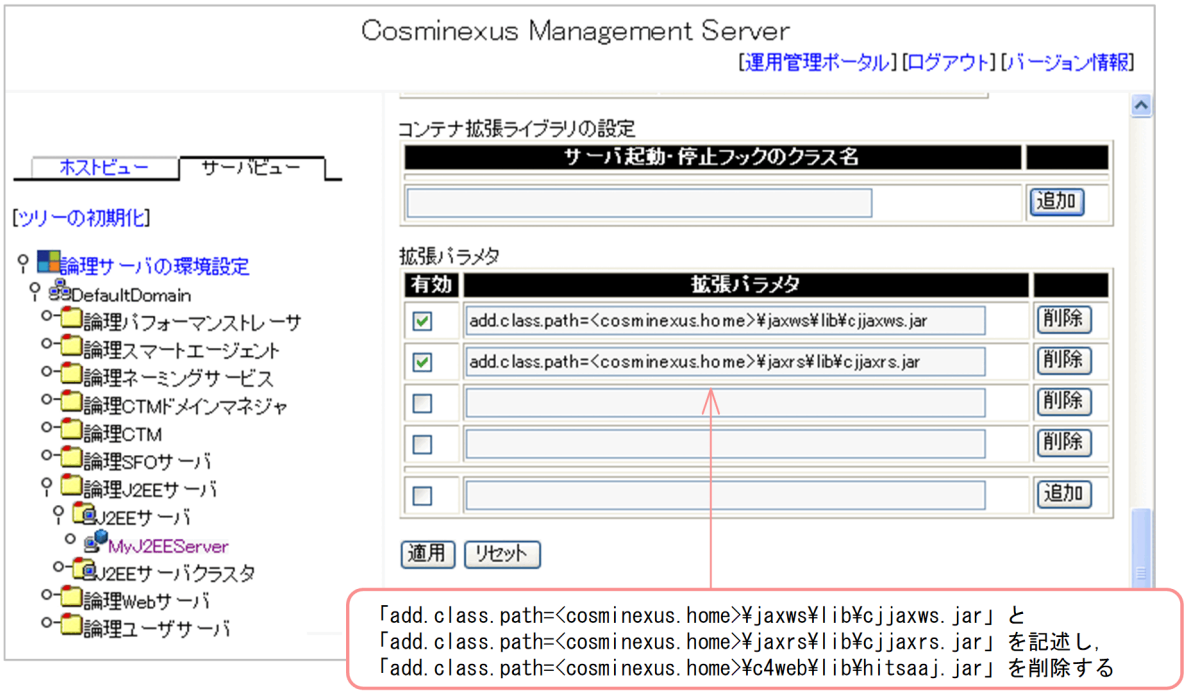
<Application Server のインストールディレクトリ>/CC/server/usrconf/ejb/<J2EE サーバ名>/usrconf.cfg

(b)　Management Server の運用管理ポータルを利用する場合

Management Server の運用管理ポータルを利用する場合，［J2EE コンテナの設定］画面の「拡張パラメタ」で設定します。

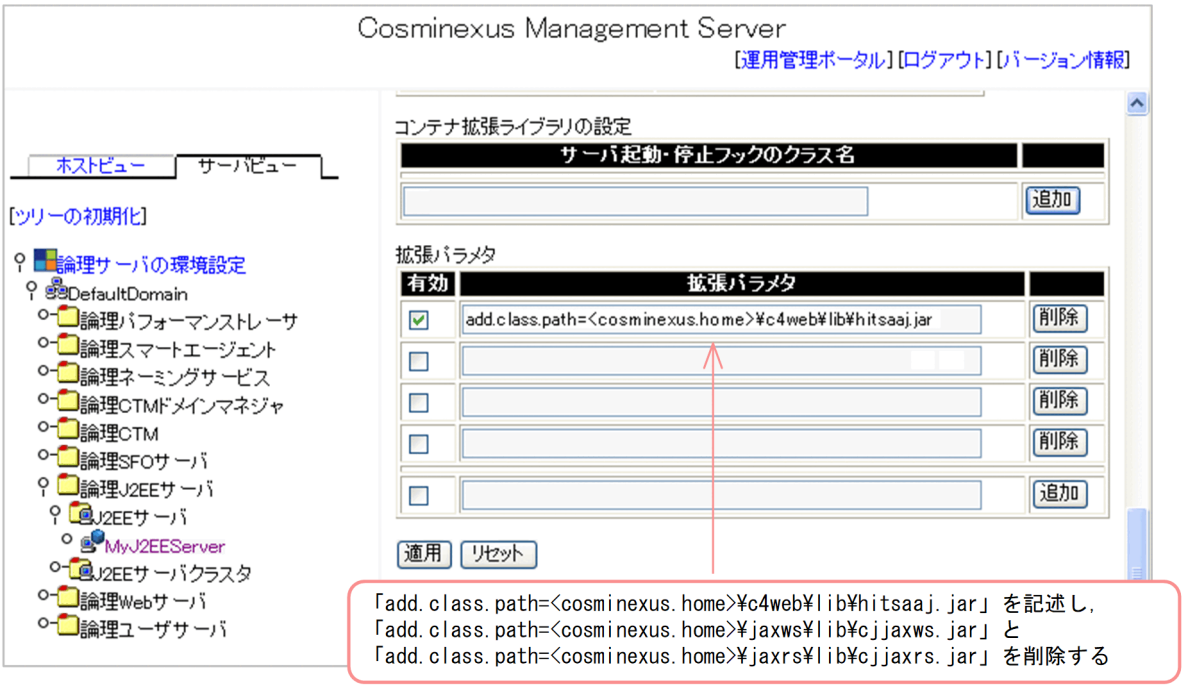
JAX-WS エンジンおよび JAX-RS エンジンを利用する場合の設定例を次に示します。

図 A-1　運用管理ポータルによる JAX-WS エンジンおよび JAX-RS エンジンの設定例



SOAP アプリケーション開発支援機能および SOAP 通信基盤を利用する場合の設定例を次に示します。

図 A-2　運用管理ポータルによる SOAP アプリケーション開発支援機能／SOAP 通信基盤の設定例



運用管理ポータルの［J2EE コンテナの設定］画面については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」の「10.8.2　J2EE コンテナの設定」を参照してください。

(c) Smart Composer 機能を利用する場合

Smart Composer 機能を利用する場合は，簡易構築定義ファイルに，J2EE の拡張パラメタとして追加します。Smart Composer 機能については，マニュアル「アプリケーションサーバ システム構築・運用ガイド」を参照してください。簡易構築定義ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の「4. Smart Composer 機能で使用するファイル」を参照してください。

JAX-WS エンジンおよび JAX-RS エンジンを利用する場合の設定例を次に示します。

```
<param>
 <param-name>add.class.path</param-name>
 <param-value>&lt;cosminexus.home&gt;¥jaxws¥lib¥cjjaxws.jar</param-value>
</param>
<param>
 <param-name>add.class.path</param-name>
 <param-value>&lt;cosminexus.home&gt;¥jaxrs¥lib¥cjjaxrs.jar</param-value>
</param>
```

SOAP アプリケーション開発支援機能および SOAP 通信基盤を利用する場合の設定例を次に示します。

```
<param>
 <param-name>add.class.path</param-name>
 <param-value>&lt;cosminexus.home&gt;¥c4web¥lib¥hitsaaj.jar</param-value>
</param>
```

付録 A.2 旧バージョンで作成した WSDL の互換性

旧バージョンで作成した WSDL を JAX-WS エンジン上で使用できます。ただし，WSDL に記述された XML Schema には制限があります。データ型，制約ファセット，および出現回数の制限について示します。

データ型の制限を次の表に示します。

表 A-1 XML Schema のデータ型の制限

項番	データ型			最大値	最大桁数
1	duration 型 （形式：PnYnMnDTnHnMnS）	nY（年）		2147483647	−
2		nM（月）			
3		nD（日）			
4		nH（時）			
5		nM（分）			
6		nS（秒）	1 秒以上		
7			1 秒未満		

項番	データ型		最大値	最大桁数
8	date 型 （形式：CCYY-MM-DD）	CCYY※1		
9	gYearMonth 型 （形式：CCYY-MM）	CCYY※1		
10	gYear 型 （形式：CCYY）	CCYY※1		
11	dateTime 型 （形式：CCYY-MM-DDThh:mm:ss）	CCYY※1		
		1 秒未満※2		
12	dateTime 型 （形式：CCYY-MM-DDThh:mm:ss）	1 秒未満※2		
13	time 型（形式：hh:mm:ss）		規定はありません。	メモリに依存します。
14	base64Binary 型			
15	hexBinary 型			
16	decimal 型			
17	integer 型			
18	nonPositiveInteger 型			
19	nonNegativeInteger 型			
20	negativeInteger 型			
21	positiveInteger 型			

（凡例）

－：該当しないことを示します。

注※1

"9999"を超える場合，桁数を追加できます。

注※2

"ss"に続く小数点以下に，1 秒未満の値を指定できます。

制約ファセットに関する制限を次の表に示します。

表 A–2　XML Schema の制約ファセットに関する制限

項番	制約ファセット	最大値	最大桁数
1	length	2147483647	－
2	minLength		
3	maxLength		
4	totalDigits		

項番	制約ファセット	最大値	最大桁数
5	fractionDigits		
6	maxInclusive	規定はありません。	メモリに依存します。
7	maxExclusive		
8	minInclusive		
9	minExclusive		

（凡例）

－：該当しないことを示します。

出現回数の指定の制限を次の表に示します。

表 A–3　XML Schema の出現回数の指定に関する制限

項番	出現回数の指定	最大値	最大桁数
1	minOccurs	2147483647	－
2	maxOccurs		

（凡例）

－：該当しないことを示します。

付録 A.3　JAX-WS エンジンを利用した Application Server を 08-00～08-70 から 09-00～09-70 にバージョンアップする場合

ここでは，JAX-WS エンジンを利用した Application Server を 08-00～08-70 から 09-00～09-70 にバージョンアップインストールする場合の注意事項を説明します。

(1)　SSL 接続におけるホスト名検証

SSL 接続におけるホスト名検証（Hostname Verification）は 08-00～08-70 と 09-00 以降とで動作が異なります。

08-00～08-70

Web サービスクライアントから SSL プロトコルに対応した Web サービスに接続する場合，エンドポイントアドレスに含まれるホスト名と，証明書のホスト名が一致しているかどうかは検証されません。

09-00 以降

Web サービスクライアントから SSL プロトコルに対応した Web サービスに接続する場合，エンドポイントアドレスに含まれるホスト名と，証明書のホスト名が一致しているかどうかが検証されます。使用される HostnameVerifier は JDK のデフォルトの実装です。JDK のデフォルトの HostnameVerifier の動作については JDK のドキュメントを参照してください。

09-00 以降で 08-00～08-70 と同様に SSL 接続におけるホスト名検証が行われないようにするには，com.cosminexus.xml.ws.client.http.HostnameVerificationProperty プロパティに"true"を設定してください。

(2) フォルトから例外クラスへのマッピング

cjwsimport コマンドを実行すると，WSDL のフォルトは，JAX-WS 2.2 仕様に従って Java 型にマッピングされます。このとき，ラッパ例外クラスが生成される条件は，08-00～08-70 と 09-00 以降とで異なります。

08-00～08-70

wsdl:portType 要素と wsdl:binding 要素の孫要素に，対応関係にある wsdl:fault 要素が定義されている場合，ラッパ例外クラスが生成されます。どちらか一方だけに定義されている場合は生成されません。

09-00 以降

wsdl:portType 要素の孫要素に wsdl:fault 要素が定義されている場合，ラッパ例外クラスが生成されます。wsdl:binding 要素の孫要素として，対応関係にある wsdl:fault 要素が定義されている必要はありません。

(3) javax.activation.DataHandler オブジェクトの動作

javax.activation.DataHandler オブジェクトは，08-00～08-70 と 09-00 以降とで動作が異なります。

08-00～08-70

wsi:swaRef 形式または MTOM/XOP 仕様形式の添付ファイルで受信したデータが javax.activation.DataHandler オブジェクトの場合，入力ストリームからすべてのデータを読み込みます。

09-00 以降

wsi:swaRef 形式または MTOM/XOP 仕様形式の添付ファイルで受信したデータが javax.activation.DataHandler オブジェクトの場合，入力ストリームから順次読み込みます。

wsi:swaRef 形式または MTOM/XOP 仕様形式の添付ファイルの受信側では，javax.activation.DataHandler オブジェクトが持つ入力ストリームからすべてのデータを読み込まないと受信処理が完了しないため，送信側の送信処理は受信側の受信処理が完了するまで待ち状態が続きます。

この状態を解消するためには，javax.activation.DataHandler オブジェクトが持つ java.io.InputStream オブジェクトからすべてのデータを読み込むか，javax.activation.DataHandler クラスの writeTo(java.io.OutputStream)メソッドを使用して入力ストリームのデータを出力ストリームに書き込む必要があります。

(4) javax.xml.transform.Source オブジェクトの動作

javax.xml.transform.Source オブジェクトは，08-00～08-70 と 09-00 以降とで動作が異なります。

08-00～08-70

MTOM/XOP 仕様形式の添付ファイルで受信したデータが javax.xml.transform.Source オブジェクトの場合，入力ストリームからすべてのデータを読み込みます。

09-00 以降

MTOM/XOP 仕様形式の添付ファイルで受信したデータが javax.xml.transform.Source オブジェクトの場合，入力ストリームから順次読み込みます。javax.xml.transform.Source オブジェクトは，JAXB が javax.xml.transform.stream.StreamSource オブジェクトに変換します。

MTOM/XOP 仕様形式の添付ファイルの受信側では，javax.xml.transform.stream.StreamSource オブジェクトが持つ入力ストリームからすべてのデータを読み込まないと受信処理が完了しないため，送信側の送信処理は受信側の受信処理が完了するまで待ち状態が続きます。

この状態を解消するためには，javax.xml.transform.stream.StreamSource オブジェクトが持つ java.io.Reader オブジェクトからすべてのデータを読み込む必要があります。

(5) アドレッシング機能を使用している場合のサービス側ランタイムの動作の違い

アドレッシング機能を使用していて，次の条件がすべて重なった場合，サービス側ランタイムの動作が 08-50～08-70 と 09-00 以降とで異なります。

- SOAP 1.1 仕様のメッセージである。
- リクエストメッセージのアドレッシングヘッダに wsa:ReplyTo 要素が存在しない。
- リクエストメッセージのアドレッシングヘッダに wsa:FaultTo 要素が存在し，Web サービスの URI を設定している。
- サービス側でエラーが発生し，SOAP フォルトを返信する。

08-50～08-70

HTTP ステータスコード：202 を返します。

09-00 以降

HTTP ステータスコード：500 を返します。

(6) MIME Multipart/Related 構造の SOAP メッセージ受信時の動作の違い

1048576 バイト以上のサイズの MIME ボディを含む MIME Multipart/Related 構造の SOAP メッセージを受信した場合，JAX-WS エンジンの動作が 08-00～08-70 と 09-00 以降とで異なります。

08-00～08-70

1048576 バイト以上のサイズの MIME ボディを含む MIME Multipart/Related 構造の SOAP メッセージを受信した場合，JAX-WS エンジンは SOAP メッセージに含まれる MIME ボディを一時ファイルに出力しないで，Java ヒープに展開します。

09-00 以降

1048576 バイト以上のサイズの MIME ボディを含む MIME Multipart/Related 構造の SOAP メッセージを受信した場合，JAX-WS エンジンは SOAP メッセージに含まれる MIME ボディを一時ファイルに出力します。

添付ファイルパートの MIME ボディに対応する一時ファイルは，com.sun.xml.ws.developer.StreamingDataHandler#close()メソッドで添付ファイルをクローズするときに削除されます。

詳細については，「10.24　MIME Multipart/Related 構造の SOAP メッセージの受信」を参照してください。

(7)　MIME Multipart/Related 構造の SOAP メッセージ送信時の動作の違い

MTOM/XOP 仕様形式の添付ファイル機能を使用して MIME Multipart/Related 構造の SOAP メッセージを送信する場合，JAX-WS エンジンの動作が 08-70 と 09-00 以降とで異なります。

08-70

レスポンスメッセージは，常に MIME Multipart/Related 構造の SOAP メッセージです。

09-00 以降

Web サービス側で次の条件をすべて満たすリクエストメッセージを受信した場合，レスポンスメッセージに含まれるバイナリデータが Base64 形式のデータになります。

- リクエストメッセージの HTTP ヘッダにある Accept フィールドに application/xop+xml がない。
- リクエストメッセージのルートパートにある Content-Type に application/xop+xml がない。

付録 A.4　JAX-WS エンジンを利用した Application Server を 09-70 以前から 09-87 以降にバージョンアップする場合

ここでは，JAX-WS エンジンを利用した Application Server を 09-70 以前から 09-87 以降にバージョンアップインストールする場合の注意事項を説明します。

(1)　cjapt コマンドの廃止

JAX-WS 機能の cjapt コマンドは使用できません。JAX-WS 2.2 仕様に対応した Web サービス実装クラスをコンパイルする場合，javac コマンドを使用してください。javac コマンドについては，JDK のドキュメントを参照してください。

(2)　cjwsgen コマンドの廃止および hwsgen コマンドの追加

JAX-WS 機能の cjwsgen コマンドは使用できません。コンパイルした Web サービス実装クラスに対して事前にエラーチェックをする場合，hwsgen コマンドを使用してください。hwsgen コマンドに指定できるオプションは 09-70 の cjwsgen コマンドと同じです。ただし，cjwsgen コマンドの-soap オプション

および-soap12binding オプションは使用できません。また，動作定義ファイルに設定する cjwsgen コマンドの定義は hwsgen コマンドでは無効となり，hwsgen コマンドではログを出力しません。hwsgen コマンドについては，「14.1.2　hwsgen コマンド」を参照してください。

(3)　java.activation モジュールを使用するアプリケーションのコンパイル

Java SE が提供していた java.activation モジュールに含まれる API を使用するアプリケーションをコンパイルする場合は，次のライブラリをクラスパスに追加してコンパイルしてください。

```
<Cosminexusインストールディレクトリ>¥jdk¥lib¥hcompatlib¥hjdk.act.jar
```

このライブラリは，コンパイル以外の用途では使用しないでください。

付録 B　POJO の Web サービスから EJB の Web サービスへの移行

付録 B.1　移行方法の詳細

POJO で開発された Web サービスを，EJB の Web サービスとして動作させることで，EJB の機能を利用できます。ここでは，POJO の Web サービスを EJB の Web サービスに移行する方法について説明します。なお，POJO の Web サービスで十分な場合，EJB への移行は不要です。

POJO の Web サービスを EJB の Web サービスに移行する方法を次の表に示します。

表 B-1　EJB の Web サービスに移行する方法

項番	POJO の Web サービスの構成物	EJB の Web サービスへの移行方法
1	Web サービス実装クラス	ソースコードに javax.ejb.Stateless アノテーションを追加し，コンパイルします。
2	項番 1 以外の Java クラス（SEI，JavaBeans クラス（スタブ）など）	EJB の Web サービスへの移行後もクラスファイルをそのまま使用できます。
3	WSDL	POJO の Web サービスの場合と EJB の Web サービスの場合とでは Web サービスとして公開する URL が異なるため，soap:address 要素の location 属性などの URL に関する値を変更して使用します。EJB の URL については，「10.2.2(1) ディスカバリ」を参照してください。
4	cosminexus-jaxws.xml，web.xml，application.xml などの DD	POJO の Web サービス用に作成された DD は，EJB の Web サービスではそのまま使用できません。EJB の Web サービス用に新たに DD を作成してください。

POJO の Web サービスを EJB の Web サービスに移行する場合，Web サービス実装クラスのソースコードの修正が必要です。ここでは，POJO の Web サービスの実行に必要なクラスファイルと，Web サービス実装クラスのソースコードがあることを前提に説明します。

移行手順の流れを次に示します。

(1)　EJB の Web サービス実装クラスを作成する

POJO の Web サービス実装クラスのソースコードに javax.ejb.Stateless アノテーションでアノテートします。アノテートしたソースコードをコンパイルしてクラスファイルを作成します。

(2)　Web サービス実行に必要なクラスファイルおよび WSDL を流用する

POJO の Web サービスを実行するために必要な SEI，JavaBeans クラス（スタブ）などのクラスファイルは，EJB の Web サービスとして実行する場合にも使用できます。

POJO の Web サービスを実行するために使用していた WSDL がある場合は，EJB の Web サービスとして実行する場合にも使用できます。ただし，POJO の Web サービスを EJB の Web サービスとして実行

した場合に，Web サービスとして公開する URL が異なるため，WSDL の soap:address 要素の location 属性などの URL に関する値を変更して使用します。EJB の Web サービスの URL については，「10.2.2(1)　ディスカバリ」を参照してください。

(3)　EJB JAR ファイルを作成する

(1)(2)のクラスファイルおよび WSDL を EJB JAR ファイルに格納します。EJB JAR ファイルの構成については，「3.5.2　EJB JAR ファイルの構成」を参照してください。

(4)　デプロイメントディスクリプタを作成する

(3)で作成した EJB JAR ファイルを指定する application.xml を作成します。application.xml の例を次に示します。「statelessjava.jar」には(3)で作成した EJB JAR ファイルの名前を指定します。

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="6" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/ap
plication_6.xsd">

  <description>Sample application &quot;statelessjava_dynamic_generate&quot;</description>
  <display-name>Sample_application_statelessjava_dynamic_generate</display-name>
  <module>
    <ejb>statelessjava_dynamic_generate.jar</ejb>
  </module>
</application>
```

バージョン 5 の application.xml を作成する場合は，application 要素の version 属性を"5"に，xsd:schemaLocation 属性の二つ目のロケーション情報を"http://java.sun.com/xml/ns/javaee/application_5.xsd"にしてください。

EJB の Web サービスの動作には，web.xml は必須ではありません。EJB の Web サービスで，サーブレットフィルタなどの設定を web.xml に加える場合に web.xml を作成します。詳細については，「3.5.4　EJB の Web サービスの設定用 WAR ファイルの作成」を参照してください。また，cosminexus-jaxws.xml は EJB の Web サービスには適用されません。

(5)　EAR ファイルを作成する

(3)(4)で作成した EJB JAR ファイルと application.xml を含む EAR ファイルを作成します。EAR ファイルの構成については，「3.5.3　EAR ファイルの作成」を参照してください。

付録 C　JAX-WS エンジンのメモリ使用量の算出

Java ヒープ領域での JAX-WS エンジンのメモリ使用量（Web サービス実行側）を次の場合を用いて説明します。

- アプリケーション起動時のメモリ使用量
- 1 リクエスト当たりのメモリ使用量
- 添付ファイル使用時の 1 リクエスト当たりのメモリ使用量
- 単位時間当たりのメモリ使用量

なお，JAX-WS エンジンに関する設定は，すべてデフォルト値とします。以降で説明するメモリ使用量には，JAX-WS エンジンの内部クラスで使用するメモリ使用量，内部クラスが呼び出す Java API のメモリ使用量，および JavaVM の内部実装クラスで使用するメモリ量も含みます。

また，以降で説明する JAX-WS エンジン固有のメモリ使用量は目安であり，実際のシステムでは若干異なる場合があります。

付録 C.1　アプリケーション起動時のメモリ使用量

Web サービスを含むアプリケーション起動時のメモリ使用量について説明します。アプリケーション起動時とは，cjstartapp コマンド実行時，または Management Server から J2EE アプリケーションを起動した直後のことを指します。

アプリケーション起動時には，JAX-WS エンジンの実行に必要なクラスの初期化などが行われます。初期化に必要なメモリ量は，Web サービス実装クラスの処理に関係なく，約 21.2MB となります。また，Java ヒープ内の Metaspace 領域でのメモリ使用量は，J2EE サーバにロードされるクラスファイルの合計サイズです。JAX-WS エンジンのコンテナ拡張ライブラリで指定する cjjaxws.jar のクラスファイルの総和は約 7MB になります。

付録 C.2　1 リクエスト当たりのメモリ使用量

1 リクエスト当たりの処理に必要なメモリ使用量について説明します。1 リクエストとは，Web サービスクライアントからリクエストを受け付け，処理をし，レスポンスを返すまでのことを指します。

初回リクエスト時と 2 回目以降のリクエスト時では，使用される JAX-WS エンジン固有のメモリ使用量が異なります。

初回リクエスト時のメモリ使用量を示します。

```
1リクエスト当たりのメモリ使用量（初回）＝
    Webサービス実装クラスが使用するメモリ使用量＋JAX-WSエンジン固有のメモリ使用量(2.43MB)
```

2 回目以降のリクエスト時のメモリ使用量を示します。

```
1リクエスト当たりのメモリ使用量（2回目以降）＝
    Webサービス実装クラスが使用するメモリ使用量＋JAX-WSエンジン固有のメモリ使用量(119.9KB)
```

また，単位時間当たりに複数のリクエストが到着する場合のメモリ使用量の算出式は，次のとおりです。

```
単位時間当たりのメモリ使用量＝
    1リクエスト当たりのメモリ使用量（初回）+{1リクエスト当たりのメモリ使用量（2回目以降）×
（単位時間当たりのリクエスト処理数-1）}
```

付録 C.3　添付ファイル使用時の 1 リクエスト当たりのメモリ使用量

添付ファイルを使用する Web サービスのアプリケーションで，1 リクエスト当たりの処理に必要なメモリ使用量について説明します。1 リクエストとは，Web サービスクライアントからリクエストを受け付け，処理をし，レスポンスを返すまでのことを指します。

ここで説明する算出方法では，Web サービスクライアントから添付ファイルを受け付け，添付ファイルをそのまま Web サービスクライアントに返す処理をする Web サービスを想定しています。

(1)　添付ファイルサイズが 10KB の場合

初回リクエスト時のメモリ使用量を示します。

```
1リクエスト当たりのメモリ使用量（初回）＝
    Webサービス実装クラスが使用するメモリ使用量＋JAX-WSエンジン固有のメモリ使用量（2.68MB）
```

2 回目以降のリクエスト時のメモリ使用量を示します。

```
1リクエスト当たりのメモリ使用量（2回目以降）＝
    Webサービス実装クラスが使用するメモリ使用量＋JAX-WSエンジン固有のメモリ使用量（193KB）
```

(2)　添付ファイルサイズが 100KB の場合

初回リクエスト時のメモリ使用量を示します。

```
1リクエスト当たりのメモリ使用量（初回）＝
    Webサービス実装クラスが使用するメモリ使用量＋JAX-WSエンジン固有のメモリ使用量（2.73MB）
```

2 回目以降のリクエスト時のメモリ使用量を示します。

```
1リクエスト当たりのメモリ使用量（2回目以降）＝
    Webサービス実装クラスが使用するメモリ使用量＋JAX-WSエンジン固有のメモリ使用量（411KB）
```

(3) 添付ファイルサイズが 1MB の場合

初回リクエスト時のメモリ使用量を示します。

```
1リクエスト当たりのメモリ使用量（初回）＝
    Webサービス実装クラスが使用するメモリ使用量＋JAX-WSエンジン固有のメモリ使用量（4.54MB）
```

2 回目以降のリクエスト時のメモリ使用量を示します。

```
1リクエスト当たりのメモリ使用量（2回目以降）＝
    Webサービス実装クラスが使用するメモリ使用量＋JAX-WSエンジン固有のメモリ使用量（2.24MB）
```

付録 C.4 単位時間当たりのメモリ使用量の算出

単位時間当たりに複数のリクエストが到着する場合のメモリ使用量の算出式は，次のとおりです。

```
単位時間当たりのメモリ使用量 ＝
    1リクエスト当たりのメモリ使用量（初回）+{1リクエスト当たりのメモリ使用量（2回目以降）×
（単位時間当たりのリクエスト処理数-1）}
```

ここでは例として，1 分間に 60 リクエストが到着し，平均約 10KB の添付ファイルを扱うシステムの 1 時間当たりのメモリ使用量を算出します。このシステムでは，Web サービス実装クラスが使用するメモリ使用量が，1 リクエスト当たり 100KB とします。

この場合，初回時の 1 リクエスト当たりのメモリ使用量は次のとおりとなります。

```
100KB＋2.68MB＝2.78MB
```

また，2 回目以降の 1 リクエスト当たりのメモリ使用量は次のとおりとなります。

```
100KB＋193KB＝293KB
```

この結果から，1 時間当たりのメモリ使用量は，次のように算出できます。

```
2.78MB＋{293KB×（60×60-1)}＝1,033MB
```

1 時間当たりのメモリ使用量に，アプリケーション起動時のメモリ量を加えると，このアプリケーションが起動してから 1 時間で使用する Java ヒープのメモリ量がわかります。

```
21.2MB＋1,033MB＝1,054.2MB
```

付録 D　用語解説

マニュアルで使用する用語について

マニュアル「アプリケーションサーバ & BPM/ESB 基盤 用語解説」を参照してください。

索引

D

E

F

G

H

I

J

T

U

V

W

X

あ

い

う

え

お

か

き

せ

そ

た

ち

つ

て

と

な

は

ひ

ふ

へ

ほ

ま

め

ゆ

よ

ら

り

る

れ

ろ