HITACHI
Inspire the Next

Cosminexus アプリケーションサーバ V8

Web サービス開発の手引

手引書

3020-3-U31-40

■対象製品

●適用 OS：Windows Server 2003，Windows Server 2003 R2，Windows Server 2003（x64）※1，Windows Server 2003 R2（x64）※1，Windows Server 2008 x86，Windows Server 2008 x64 ※1，Windows Server 2008 R2 ※1

P-2443-7B84 uCosminexus Application Server Standard-R 08-70

P-2443-7D84 uCosminexus Application Server Standard 08-70

P-2443-7K84 uCosminexus Application Server Enterprise 08-70

P-2443-7S84 uCosminexus Service Platform 08-70 ※2

●適用 OS：Windows Server 2003，Windows Server 2003 R2，Windows Vista，Windows XP，Windows 7（32bit），Windows 7（x64）※1

P-2443-7E84 uCosminexus Developer Standard 08-70

P-2443-7F84 uCosminexus Developer Professional 08-70

P-2443-7T84 uCosminexus Service Architect 08-70 ※2

●適用 OS：Windows Server 2003，Windows Server 2003 R2，Windows Server 2003（x64）※1，Windows Server 2003 R2（x64）※1，Windows Server 2008 x86，Windows Server 2008 x64 ※1，Windows Server 2008 R2 ※1，Windows Vista，Windows XP，Windows 7（32bit），Windows 7（x64）※1

P-2443-7H84 uCosminexus Client 08-70

●適用 OS：Windows Server 2003（x64），Windows Server 2003 R2（x64），Windows Server 2008 x64，Windows Server 2008 R2

P-2943-7B84 uCosminexus Application Server Standard-R 08-70

P-2943-7D84 uCosminexus Application Server Standard 08-70

P-2943-7K84 uCosminexus Application Server Enterprise 08-70

P-2943-7S84 uCosminexus Service Platform 08-70 ※2

●適用 OS：AIX 5L V5.3，AIX V6.1，AIX V7.1

P-1M43-7D81 uCosminexus Application Server Standard 08-70 ※2

P-1M43-7K81 uCosminexus Application Server Enterprise 08-70 ※2

P-1M43-7S81 uCosminexus Service Platform 08-70 ※2

●適用 OS：HP-UX 11i V2（IPF），HP-UX 11i V3（IPF）

P-1J43-7D81 uCosminexus Application Server Standard 08-70

P-1J43-7K81 uCosminexus Application Server Enterprise 08-70

P-1J43-7S81 uCosminexus Service Platform 08-70 ※2

●適用 OS：Red Hat Enterprise Linux AS 4（x86），Red Hat Enterprise Linux ES 4（x86），Red Hat Enterprise Linux AS 4（AMD64 & Intel EM64T），Red Hat Enterprise Linux ES 4（AMD64 & Intel EM64T），Red Hat Enterprise Linux 5 Advanced Platform（x86），Red Hat Enterprise Linux 5（x86），Red Hat Enterprise Linux 5 Advanced Platform（AMD/Intel 64），Red Hat Enterprise Linux 5（AMD/Intel 64），Red Hat Enterprise Linux Server 6（32-bit x86），Red Hat Enterprise Linux Server 6（64-bit x86_64）

P-9S43-7B81 uCosminexus Application Server Standard-R 08-70 ※2

P-9S43-7D81 uCosminexus Application Server Standard 08-70 ※2

P-9S43-7K81 uCosminexus Application Server Enterprise 08-70 ※2

P-9S43-7S81 uCosminexus Service Platform 08-70 ※2

注※1　WOW64（Windows On Windows 64）環境だけで使用できます。

注※2　この製品については，サポート時期をご確認ください。

これらのプログラムプロダクトのほかにもこのマニュアルをご利用になれる場合があります。詳細は「リリースノート」でご確認ください。
本製品では日立トレース共通ライブラリをインストールします。

■輸出時の注意

本製品を輸出される場合には，外国為替および外国貿易法ならびに米国の輸出管理関連法規などの規制をご確認の上，必要な手続きをお取りください。
なお，ご不明な場合は，弊社担当営業にお問い合わせください。

■商標類

AIX は，米国およびその他の国における International Business Machines Corporation の商標です。
AIX 5L は，米国およびその他の国における International Business Machines Corporation の商標です。
AMD は，Advanced Micro Devices,Inc. の商標です。
CORBA は，Object Management Group が提唱する分散処理環境アーキテクチャの名称です。
gzip は，米国 FSF(Free Software Foundation) が配布しているソフトウェアです。
HP-UX は，Hewlett-Packard Company のオペレーティングシステムの名称です。
Itanium は，アメリカ合衆国およびその他の国における Intel Corporation の商標です。
J2EE は，Oracle Corporation 及びその子会社，関連会社の米国 及びその他の国における登録商標または商標です。
Java は，Oracle Corporation 及びその子会社，関連会社の米国 及びその他の国における登録商標または商標です。
JDK は，Oracle Corporation 及びその子会社，関連会社の米国 及びその他の国における登録商標または商標です。
JSP は，Oracle Corporation 及びその子会社，関連会社の米国 及びその他の国における登録商標または商標です。
Linux は，Linus Torvalds 氏の日本およびその他の国における登録商標または商標です。
Microsoft は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
Red Hat は，米国およびその他の国で Red Hat,Inc. の登録商標もしくは商標です。
SOAP（Simple Object Access Protocol）は，分散ネットワーク環境において XML ベースの情報を交換するための通信プロトコルの名称です。
Solaris は，Oracle Corporation 及びその子会社，関連会社の米国 及びその他の国における登録商標または商標です。
Sun は，Oracle Corporation 及びその子会社，関連会社の米国 及びその他の国における登録商標または商標です。
UNIX は，The Open Group の米国ならびに他の国における登録商標です。
W3C は，World Wide Web Consortium の商標（多数の国において登録された）です。
Windows は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
Windows Server は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
Windows Vista は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
X/Open は，The Open Group の英国ならびに他の国における登録商標です。

■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

Microsoft Corporation のガイドラインに従って画面写真を使用しています。

■マイクロソフト製品の表記について

このマニュアルでは，マイクロソフト製品の名称を次のように表記しています。

製品名	表記		
Microsoft(R) Windows(R) 7 Enterprise	Windows 7		Windows
Microsoft(R) Windows(R) 7 Professional			
Microsoft(R) Windows(R) 7 Ultimate			
Microsoft(R) Windows Server(R) 2003, Enterprise Edition 日本語版	Windows Server 2003 Enterprise Edition	Windows Server 2003	
Microsoft(R) Windows Server(R) 2003, Standard Edition 日本語版	Windows Server 2003 Standard Edition		
Microsoft(R) Windows Server(R) 2003 R2, Enterprise Edition 日本語版	Windows Server 2003 R2 Enterprise Edition	Windows Server 2003 R2	
Microsoft(R) Windows Server(R) 2003 R2, Standard Edition 日本語版	Windows Server 2003 R2 Standard Edition		
Microsoft(R) Windows Server(R) 2003, Enterprise x64 Edition 日本語版	Windows Server 2003 Enterprise x64 Edition	Windows Server 2003 (x64)	
Microsoft(R) Windows Server(R) 2003, Standard x64 Edition 日本語版	Windows Server 2003 Standard x64 Edition		
Microsoft(R) Windows Server(R) 2003 R2, Enterprise x64 Edition 日本語版	Windows Server 2003 R2 Enterprise x64 Edition	Windows Server 2003 R2 (x64)	
Microsoft(R) Windows Server(R) 2003 R2, Standard x64 Edition 日本語版	Windows Server 2003 R2 Standard x64 Edition		
Microsoft(R) Windows Server(R) 2008 Enterprise 32-bit 日本語版	Windows Server 2008 x86	Windows Server 2008	
Microsoft(R) Windows Server(R) 2008 Standard 32-bit 日本語版			
Microsoft(R) Windows Server(R) 2008 Enterprise 日本語版	Windows Server 2008 x64		
Microsoft(R) Windows Server(R) 2008 Standard 日本語版			
Microsoft(R) Windows Server(R) 2008 R2 Enterprise 日本語版	Windows Server 2008 R2		

製品名	表記		
Microsoft(R) Windows Server(R) 2008 R2 Standard 日本語版			
Microsoft(R) Windows Vista(R) Business	Windows Vista Business	Windows Vista	
Microsoft(R) Windows Vista(R) Enterprise	Windows Vista Enterprise		
Microsoft(R) Windows Vista(R) Ultimate	Windows Vista Ultimate		
Microsoft(R) Windows(R) XP Professional Operating System	Windows XP		

なお，製品名に x64 を含む製品を「Windows(x64)」，含まない製品を「Windows(x86)」と区別して表記していることがあります。

■発行

2011 年 7 月 3020-3-U31-40

■著作権

変更内容

変更内容 (3020-3-U31-40)uCosminexus Application Server Enterprise 08-70，uCosminexus Application Server Standard 08-70，uCosminexus Application Server Standard-R 08-70，uCosminexus Client 08-70，uCosminexus Developer Professional 08-70，uCosminexus Developer Standard 08-70，uCosminexus Service Architect 08-70，uCosminexus Service Platform 08-70

追加・変更内容	変更個所
EJB の Web サービスをサポートしました。	1.2，1.3.2，2.1.2，2.1.3，3.1，3.3，3.5.2，3.5.3，3.5.4，8.，10.2.2，10.3，10.4，10.4.3，10.4.4，10.6，10.8，10.9，10.19，11.2，29.1，29.4.2，付録 C
WSEE（JSR-109）仕様をサポートしました。	1.2，1.3.2
複数スレッドでポートまたはディスパッチを共有する場合について記載を追加しました。	3.6.1，3.6.2，3.6.3
64 ビット版の記載を追加しました。	5.3.2，7.3.2，9.3.2，11.2，19.3.2
HTTP ヘッダについて記載を追加しました。	10.16
HTTP リクエストボディの gzip 圧縮をサポートしました。	10.17，15.5.1
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の記載を追加しました。	12.2.9
wrapper スタイルと non-wrapper スタイルの記載を追加しました。	13.1.10，18.4.3
ストリーミングをサポートしました。	13.2.2，15.1，15.4.1，15.4.2，15.4.3，22.，23.
WS-RM 1.2 仕様をサポートしました。	14.6，14.7，16.，24.，25.，29.3.4，29.4.2
WS-Addressing 機能の注意事項を追加しました。	15.2.4
添付ファイル機能のクラスおよびインタフェースの記載を追加しました。	15.4.15，15.4.16
メッセージコンテキストに，オペレーションの記述を追加しました。	15.5.1，15.5.2
次の製品の適用 OS に AIX，HP-UX を追加しました。 • uCosminexus Application Server Standard • uCosminexus Application Server Enterprise • uCosminexus Service Platform	－

追加・変更内容	変更個所
次の製品の適用 OS に Red Hat Enterprise Linux 6 を追加しました。 • uCosminexus Application Server Standard-R • uCosminexus Application Server Standard • uCosminexus Application Server Enterprise • uCosminexus Service Platform	－

uCosminexus Application Server Enterprise 08-53，uCosminexus Application Server Standard 08-53，uCosminexus Application Server Standard-R 08-53，uCosminexus Client 08-53，uCosminexus Developer Professional 08-53，uCosminexus Developer Standard 08-53，uCosminexus Service Architect 08-53，uCosminexus Service Platform 08-53

追加・変更内容	変更個所
MTOM/XOP 仕様形式の添付ファイルをサポートしました。	1.3.2，10.2.1，10.2.2，12.1.10，13.1.10，13.2.1，13.2.9，13.2.17，14.1.31，15.1，20.，21.
com.cosminexus.jaxws.publish_wsdl.soap12binding をサポートしました。	10.1.2
HTTP レスポンス圧縮機能との連携をサポートしました。	10.18
-soap12binding について記述を追加しました。	11.3
SOAP バインディングに，transport 属性値から BindingType アノテーションへのマッピングを追加しました。	12.1.8
SOAP11HTTP_MTOM_BINDING と SOAP12HTTP_MTOM_BINDING の記述を追加しました。また，javax.xml.ws.BindingType アノテーションから soap:binding 要素または soap12:binding 要素の transport 属性値へのマッピングについて記述を追加しました。	13.2.11
「http://schemas.xmlsoap.org/soap/http」の記述を追加しました。	14.1.26
対象製品として uCosminexus Application Server Standard-R を追加しました。	－
次の製品の適用 OS から AIX，HP-UX，Linux（IPF）を削除しました。 • uCosminexus Application Server Standard • uCosminexus Application Server Enterprise • uCosminexus Service Platform	－

単なる誤字・脱字などはお断りなく訂正しました。
今版 (3020-3-U31-40) では，前版 (3020-3-U31-20) との対応は次のようになっています。

旧 (3020-3-U31-20)	新 (3020-3-U31-40)
13.6　WS-RM 1.1 仕様のサポート範囲	付録 B.7　WS-RM 1.1 仕様のサポート範囲
15.　WS-RM 機能の API	付録 B.4　WS-RM 1.1 機能の API 一覧 付録 B.5　WS-RM 1.1 機能のプロパティ設定

旧 (3020-3-U31-20)	新 (3020-3-U31-40)
19.　WS-RM 機能	付録 B.　旧バージョンの WS-RM 機能 付録 B.1　WS-RM 1.1 機能を使用したメッセージの流れ 付録 B.2　WS-RM 1.1 機能を使用する場合のシステム構成 付録 B.3　WS-RM 1.1 機能の送達保証
20.　SEI を起点とした開発の例（WS-RM 機能使用時）	削除
27.4.2(3)　性能解析トレースのトレース取得ポイント（WS-RM 機能使用時）	付録 B.6　WS-RM 1.1 機能の性能解析トレース（PRF）
付録 C.4　常用漢字以外の漢字の使用について	削除

変更内容 (3020-3-U31-20)uCosminexus Application Server Enterprise 08-50，uCosminexus Application Server Standard 08-50，uCosminexus Client 08-50，uCosminexus Developer Professional 08-50，uCosminexus Developer Standard 08-50，uCosminexus Service Architect 08-50，uCosminexus Service Platform 08-50

追加・変更内容
プロバイダ実装クラスをサポートしました。
ディスパッチベースの Web サービスクライアントをサポートしました。
WS-Addressing 1.0 仕様をサポートしました。
cjwsgen コマンドをサポートしました。
SOAP 1.2 仕様をサポートしました。
SAAJ 1.3 仕様をサポートしました。
web.xml の要件についての説明を追加しました。
cjwsimport コマンドに -wsdllocation オプションを追加しました。
javax.xml.ws.Service オブジェクトおよびポートの再利用についての説明を追加しました。
Web サービスクライアント実装時の注意事項を追加しました。
配列または java.util.List を利用する場合の注意事項を追加しました。
HTTP ステータスコードの説明を追加しました。
WAR ファイルでエラーページをカスタマイズする場合の注意事項を追加しました。
non-wrapper スタイルの配列についての説明を追加しました。
環境変数で cjwsimport コマンドにオプションを追加できるようになりました。
Windows のユーザアカウント制御が有効な場合に，Cosminexus の JAX-WS 機能のコマンドラインインタフェースを実行するときの注意事項を追加しました。
Java から WSDL へのマッピングをカスタマイズするときに使用できるアノテーションを追加しました。
javax.jws.WebMethod アノテーションの exclude 要素において，public メソッドが一つもない場合の動作を変更しました。

追加・変更内容
soap:fault 要素の name 属性には親要素の name 属性と同じ値を指定する必要があることを追加しました。
soap:header 要素または soap12:header 要素の part 属性に指定する値に関する注意事項を追加しました。
ハンドラチェイン設定ファイルの javaee:soap-role 要素に記述する値に関する注意事項を追加しました。
WS-RM 1.1 仕様をサポートしました。
サービス API を追加しました。
クライアント API を追加しました。
javax.xml.ws.Service クラスで新しいメソッドをサポートしました。
コア API を追加しました。
メッセージコンテキスト使用時の注意事項を追加しました。
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を追加しました。
ハンドラチェインの設定項目を追加しました。
通信ログの文字エンコーディングの説明を追加しました。
次の製品の適用 OS に Windows Server 2008 R2 を追加しました。 • uCosminexus Application Server Standard • uCosminexus Application Server Enterprise • uCosminexus Service Platform • uCosminexus Client
次の製品の適用 OS から Solaris を削除しました。 • uCosminexus Application Server Standard • uCosminexus Application Server Enterprise
次の製品の適用 OS に Windows 7 を追加しました。 • uCosminexus Developer Standard • uCosminexus Developer Professional • uCosminexus Service Architect • uCosminexus Client

はじめに

このマニュアルは，Cosminexus で提供する機能を利用して，JAX-WS 2.1 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。
Cosminexus では，次のプログラムプロダクトを使用して JAX-WS 2.1 仕様に対応した Web サービスを開発できます。

- uCosminexus Developer Standard
- uCosminexus Developer Professional
- uCosminexus Service Architect

Cosminexus では，次のプログラムプロダクトを使用して JAX-WS 2.1 仕様に対応した Web サービスを実行できます。

- uCosminexus Application Server Standard
- uCosminexus Application Server Standard-R
- uCosminexus Application Server Enterprise
- uCosminexus Service Platform

なお，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（SOAP アプリケーション）を開発，実行する方法については，マニュアル「Cosminexus 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。

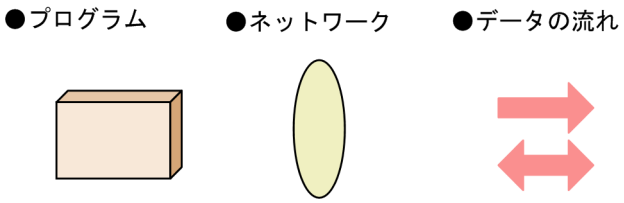
■対象読者

このマニュアルは，Cosminexus を使用して JAX-WS 2.1 仕様に対応した Web サービスを開発，実行する方を対象としています。このマニュアルをご利用になる方は，次の内容を理解されていることを前提とします。

- J2EE サーバに関する基本的な知識
- Java EE 仕様に関する基本的な知識
- XML 仕様に関する基本的な知識
- WS-RM 1.1 仕様または WS-RM 1.2 仕様に関する基本的な知識
- SOAP 1.1 仕様または SOAP 1.2 仕様，WSDL 1.1 仕様，および XML Schema に関する知識
- JAX-WS 2.1 仕様（JSR-224），JAXB 2.1 仕様（JSR-222），および Web サービスメタデータ仕様に関する知識（JSR-181）
- Java 言語によるオブジェクト指向プログラミングの知識（アノテーションも含む）

■図中で使用している記号

このマニュアルの図中で使用している記号について次に示します。



■コーディング例で使用している記号

このマニュアルでは，次に示す記号を使用して，コーディング例を記述しています。

記号	意味
… リーダ	記述が省略されていることを示します。この記号の直前に示された項目を繰り返し複数個指定できます。 （例） 　　値…では，「値を必要個指定する」ことを示します。
/*　*/ //	コメント文として扱うことを示します。 （例） 　　// メッセージを送信する
太字	例題で使用する指定値および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。

■コマンドの形式で使用している記号

このマニュアルでは，次に示す記号を使用して，コマンドの形式を記述しています。

記号	意味
[　]	この記号で囲まれている項目は，省略してもよいことを示します。
<　>	この記号で囲まれている項目は，可変値を指定することを示します。

■ JAX-WS 2.1 仕様の対応バージョンについて

このマニュアルで「JAX-WS 2.1 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-000224 - Java™API for XML-Based Web ServicesVersion:
2.1Status: Maintenance Release 2Release: 7 May 2007
```

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)
```

■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。

プレフィクス	名前空間 URI
cwsrm	http://jaxws.cosminexus.com/cwsrm
javaee	http://java.sun.com/xml/ns/javaee
jaxb	http://java.sun.com/xml/ns/jaxb
jaxws	http://java.sun.com/xml/ns/jaxws
jaxwsdd	http://java.sun.com/xml/ns/jax-ws/ri/runtime
net35rmp	http://schemas.microsoft.com/ws-rx/wsrmp/200702
soap	http://schemas.xmlsoap.org/wsdl/soap/
soap12	http://schemas.xmlsoap.org/wsdl/soap12/
soapenv	http://schemas.xmlsoap.org/soap/envelope/
soapenv12	http://www.w3.org/2003/05/soap-envelope
wsa	http://www.w3.org/2005/08/addressing
wsam	http://www.w3.org/2007/05/addressing/metadata
wsaw	http://www.w3.org/2006/05/addressing/wsdl
wsdl	http://schemas.xmlsoap.org/wsdl/
wsi	http://ws-i.org/profiles/basic/1.1/xsd
wsp	http://www.w3.org/ns/ws-policy
wsrm	http://docs.oasis-open.org/ws-rx/wsrm/200702
wsrmp	http://docs.oasis-open.org/ws-rx/wsrmp/200702
wsu	http://docs.oasis-open.org/wss/2004/01/ oasis-200401-wss-wssecurity-utility-1.0.xsd
xmime	http://www.w3.org/2005/05/xmlmime
xop	http://www.w3.org/2004/08/xop/include
xsd	http://www.w3.org/2001/XMLSchema
xsi	http://www.w3.org/2001/XMLSchema-instance

目次

第 1 編　概要

1 Web サービスの開発および実行の概要 1

1.1 Web サービスの開発の概要 2
1.2 Web サービスの開発時および実行時に使用する機能 3
1.3 Web サービスの開発および実行の前提条件 6
1.3.1 前提となる構成ソフトウェア 6
1.3.2 機能および仕様に関する前提条件 6
1.4 Web サービスと Web サービスクライアントの形態 11
1.4.1 Web サービスの形態 11
1.4.2 Web サービスクライアントの形態 13
1.5 JAX-WS エンジンの設定 16

2 開発の流れ 17

2.1 Web サービス開発の流れ 18
2.1.1 WSDL を起点とした開発の流れ 20
2.1.2 SEI を起点とした開発の流れ 23
2.1.3 SEI を起点とした開発の流れ（cjwsgen コマンドを使用する場合） 25
2.1.4 プロバイダを起点とした開発の流れ 27
2.2 Web サービスクライアント開発の流れ 29
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 29
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 30
2.2.3 API ベースの Web サービスクライアントの開発の流れ 30

第 2 編　開発と実行

3 開発のポイント 31

3.1 WSDL の作成 32
3.2 WSDL と Java ソースのマッピング 33
3.2.1 WSDL から Java ソースへのマッピング例 33

3.2.2 Java ソースから WSDL へのマッピング例 34
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 36
3.4 web.xml の作成 37
3.5 アーカイブの作成 42
3.5.1 WAR ファイルの構成 42
3.5.2 EJB JAR ファイルの構成 43
3.5.3 EAR ファイルの作成 43
3.5.4 EJB の Web サービスの設定用 WAR ファイルの作成 43
3.6 Web サービスクライアントの実装 51
3.6.1 スタブベースの実装例 51
3.6.2 ディスパッチベースの実装例 61
3.6.3 JAX-WS API を使用する場合の実装例 63
3.6.4 注意事項 65
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 66

4 WSDL を起点とした開発の例 67
4.1 開発例の構成（WSDL 起点） 68
4.2 開発例の流れ（WSDL 起点） 70
4.3 Web サービスの開発例（WSDL 起点） 71
4.3.1 WSDL ファイルを作成する 71
4.3.2 SEI を生成する 78
4.3.3 Web サービス実装クラスを作成する 79
4.3.4 Web サービス実装クラスをコンパイルする 80
4.3.5 web.xml を作成する 80
4.3.6 application.xml を作成する 81
4.3.7 EAR ファイルを作成する 82
4.4 デプロイと開始の例（WSDL 起点） 83
4.4.1 EAR ファイルをデプロイする 83
4.4.2 Web サービスを開始する 83
4.5 Web サービスクライアントの開発例（WSDL 起点） 84
4.5.1 サービスクラスを生成する 84
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 85
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 86
4.6 Web サービスの実行例（WSDL 起点） 87
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 87
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 87

4.6.3 Web サービスクライアントを実行する 88

5 SEI を起点とした開発の例 89

5.1 開発例の構成（SEI 起点） 90

5.2 開発例の流れ（SEI 起点） 92

5.3 Web サービスの開発例（SEI 起点） 93

5.3.1 Web サービス実装クラスを作成する 93

5.3.2 Java ソースを生成する 94

5.3.3 web.xml を作成する 95

5.3.4 application.xml を作成する 96

5.3.5 WSDL ファイルを作成する（任意） 97

5.3.6 EAR ファイルを作成する 97

5.4 デプロイと開始の例（SEI 起点） 99

5.4.1 EAR ファイルをデプロイする 99

5.4.2 Web サービスを開始する 99

5.5 Web サービスクライアントの開発例（SEI 起点） 100

5.5.1 サービスクラスを生成する 100

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 101

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 102

5.6 Web サービスの実行例（SEI 起点） 103

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 103

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 103

5.6.3 Web サービスクライアントを実行する 103

6 SEI を起点とした開発の例（cjwsgen コマンドを使用する場合） 105

6.1 開発例の構成（SEI 起点・cjwsgen コマンド） 106

6.2 開発例の流れ（SEI 起点・cjwsgen コマンド） 108

6.3 Web サービスの開発例（SEI 起点・cjwsgen コマンド） 109

6.3.1 コンパイル済みのクラスファイルを保存する 109

6.3.2 Java ソースを生成する 109

6.3.3 web.xml を作成する 110

6.3.4 application.xml を作成する 111

6.3.5 EAR ファイルを作成する 111

6.4 デプロイと開始の例（SEI 起点・cjwsgen コマンド） 112

6.4.1 EAR ファイルをデプロイする 112

6.4.2 Web サービスを開始する 112
6.5 Web サービスクライアントの開発例（SEI 起点・cjwsgen コマンド） 113
6.5.1 サービスクラスを生成する 113
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 114
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 115
6.6 Web サービスの実行例（SEI 起点・cjwsgen コマンド） 116
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 116
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 116
6.6.3 Web サービスクライアントを実行する 117

7 SEI を起点とした開発の例（カスタマイズする場合） 119
7.1 開発例の構成（SEI 起点・カスタマイズ） 120
7.2 開発例の流れ（SEI 起点・カスタマイズ） 122
7.3 Web サービスの開発例（SEI 起点・カスタマイズ） 123
7.3.1 Web サービス実装クラスを作成する 123
7.3.2 Java ソースを生成する 124
7.3.3 web.xml を作成する 126
7.3.4 application.xml を作成する 126
7.3.5 EAR ファイルを作成する 127
7.4 デプロイと開始の例（SEI 起点・カスタマイズ） 128
7.4.1 EAR ファイルをデプロイする 128
7.4.2 Web サービスを開始する 128
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） 129
7.5.1 サービスクラスを生成する 129
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 130
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 131
7.6 Web サービスの実行例（SEI 起点・カスタマイズ） 132
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 132
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 132
7.6.3 Web サービスクライアントを実行する 133

8 SEI を起点とした開発の例（EJB の Web サービスの場合） 135
8.1 開発例の構成（SEI 起点・EJB の Web サービス） 136
8.2 開発例の流れ（SEI 起点・EJB の Web サービス） 138
8.3 Web サービスの開発例（SEI 起点・EJB の Web サービス） 139

8.3.1 Web サービス実装クラスを作成する 139
8.3.2 Java ソースを生成する 140
8.3.3 application.xml を作成する 141
8.3.4 WSDL ファイルを作成する（任意） 142
8.3.5 EAR ファイルを作成する 142
8.4 デプロイと開始の例（SEI 起点・EJB の Web サービス） 144
8.4.1 EAR ファイルをデプロイする 144
8.4.2 Web サービスを開始する 144
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） 145
8.5.1 サービスクラスを生成する 145
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 146
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 147
8.6 Web サービスの実行例（SEI 起点・EJB の Web サービス） 148
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 148
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 148
8.6.3 Web サービスクライアントを実行する 149

9 プロバイダを起点とした開発の例（SAAJ を利用した場合） 151
9.1 開発例の構成（プロバイダ起点・SAAJ） 152
9.2 開発例の流れ（プロバイダ起点・SAAJ） 154
9.3 Web サービスの開発例（プロバイダ起点・SAAJ） 155
9.3.1 プロバイダ実装クラスを作成する 155
9.3.2 Java ソースを生成する 157
9.3.3 web.xml を作成する 158
9.3.4 application.xml を作成する 159
9.3.5 EAR ファイルを作成する 159
9.4 デプロイと開始の例（プロバイダ起点・SAAJ） 160
9.4.1 EAR ファイルをデプロイする 160
9.4.2 Web サービスを開始する 160
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） 161
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 161
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 163
9.6 Web サービスの実行例（プロバイダ起点・SAAJ） 164
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 164
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 164
9.6.3 Web サービスクライアントを実行する 165

10 JAX-WS 機能の設定と動作 167
10.1 動作定義ファイル 169
10.1.1 動作定義ファイルの記述規則 169
10.1.2 共通定義ファイルの設定項目 170
10.1.3 プロセス別の定義ファイルの設定 183
10.2 JAX-WS エンジンの動作 184
10.2.1 JAX-WS エンジンの動作とサポート範囲 184
10.2.2 ディスカバリとディスパッチ 195
10.3 cosminexus-jaxws.xml によるカスタマイズ 201
10.3.1 cosminexus-jaxws.xml のファイル名と格納先 201
10.3.2 cosminexus-jaxws.xml の書式 201
10.4 フォルトと例外の処理 206
10.4.1 Web サービス側のフォルトおよび例外の処理 206
10.4.2 Web サービスクライアント側のフォルトの処理 214
10.4.3 Java 例外の伝搬 215
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 217
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 217
10.5 インタフェースの透過性 219
10.6 メタデータの発行 223
10.7 Web サービスの情報表示 229
10.8 使用できる HTTP メソッド 230
10.9 Web サービスの初期化と破棄 231
10.10 プロキシサーバ経由の接続 233
10.11 SSL プロトコルによる接続 238
10.12 ベーシック認証による接続 240
10.13 SOAP のバージョン選択 242
10.13.1 SOAP のバージョン選択（Web サービス開発時） 242
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） 243
10.13.3 SOAP のバージョン選択（実行時） 244
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 245
10.14.1 コマンドライン利用時の設定 245
10.14.2 コマンドラインの実行 245
10.14.3 コマンドライン利用時の注意事項 246
10.15 HTTP ステータスコード 247
10.16 HTTP ヘッダ 248

10.17 HTTP リクエストボディの gzip 圧縮 249
10.18 HTTP レスポンス圧縮機能との連携 250
10.19 EJB の Web サービス呼び出し 251

第 3 編 リファレンス

11 コマンド 253

11.1 cjwsimport コマンド 254
11.2 apt コマンド 263
11.3 cjwsgen コマンド 266
11.4 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 279
11.4.1 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 279
11.4.2 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 279

12 WSDL から Java へのマッピング 281

12.1 WSDL から Java へのデフォルトマッピング 282
12.1.1 名前空間からパッケージ名へのマッピング 282
12.1.2 ポートタイプから SEI 名へのマッピング 284
12.1.3 オペレーションからメソッド名へのマッピング 285
12.1.4 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） 287
12.1.5 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） 293
12.1.6 スキーマ型から Java 型へのマッピング 295
12.1.7 フォルトから例外クラスへのマッピング 297
12.1.8 バインディング拡張要素からパラメタへのマッピング 300
12.1.9 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 302
12.1.10 WSDL から Java へのマッピングに関する注意事項 304
12.2 WSDL から Java へのマッピングのカスタマイズ 309
12.2.1 埋め込みによるバインディング宣言でのカスタマイズ 309
12.2.2 外部バインディングファイルによるカスタマイズ 314
12.2.3 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 320
12.2.4 jaxws:bindings 要素に指定できる値 321

12.2.5 カスタマイズ対象となった要素の値 324
12.2.6 名前衝突時の対応 324
12.2.7 jaxws:provider 要素を記述した場合の動作 324
12.2.8 SEI 名をカスタマイズする場合の注意事項 324
12.2.9 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 325

13 Java から WSDL へのマッピング 327
13.1 Java から WSDL へのデフォルトマッピング 328
13.1.1 パッケージ名から名前空間へのマッピング 328
13.1.2 Web サービス実装クラスから SEI へのマッピング 329
13.1.3 SEI 名からポートタイプへのマッピング 330
13.1.4 SEI のメソッド名からオペレーションへのマッピング 331
13.1.5 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） 334
13.1.6 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） 339
13.1.7 Java のラッパ例外クラスからフォルトへのマッピング 342
13.1.8 SEI からバインディングへのマッピング 345
13.1.9 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 346
13.1.10 Java から WSDL へのマッピングに関する注意事項 347
13.2 Java から WSDL へのマッピングのカスタマイズ 349
13.2.1 アノテーション一覧 350
13.2.2 com.sun.xml.ws.developer.StreamingAttachment アノテーション 354
13.2.3 javax.jws.HandlerChain アノテーション 356
13.2.4 javax.jws.soap.SOAPBinding アノテーション 356
13.2.5 javax.jws.WebMethod アノテーション 357
13.2.6 javax.jws.WebParam アノテーション 359
13.2.7 javax.jws.WebResult アノテーション 362
13.2.8 javax.jws.WebService アノテーション 365
13.2.9 javax.xml.bind.annotation.XmlMimeType アノテーション 368
13.2.10 javax.xml.ws.Action アノテーション 373
13.2.11 javax.xml.ws.BindingType アノテーション 375
13.2.12 javax.xml.ws.FaultAction アノテーション 376
13.2.13 javax.xml.ws.RequestWrapper アノテーション 377
13.2.14 javax.xml.ws.ResponseWrapper アノテーション 379
13.2.15 javax.xml.ws.ServiceMode アノテーション 381
13.2.16 javax.xml.ws.soap.Addressing アノテーション 382

13.2.17 javax.xml.ws.soap.MTOM アノテーション 383
13.2.18 javax.xml.ws.WebFault アノテーション 384
13.2.19 javax.xml.ws.WebServiceProvider アノテーション 386

14 標準仕様のサポート範囲 389
14.1 WSDL 1.1 仕様のサポート範囲 390
14.1.1 wsdl:definitions 要素 390
14.1.2 wsdl:import 要素 391
14.1.3 wsdl:types 要素 392
14.1.4 wsdl:message 要素 393
14.1.5 wsdl:part 要素 394
14.1.6 wsdl:portType 要素 395
14.1.7 wsdl:operation 要素（wsdl:portType 要素の子要素の場合） 396
14.1.8 wsdl:input 要素（wsdl:portType 要素の孫要素の場合） 397
14.1.9 wsdl:output 要素（wsdl:portType 要素の孫要素の場合） 398
14.1.10 wsdl:fault 要素（wsdl:portType 要素の孫要素の場合） 399
14.1.11 wsdl:binding 要素 400
14.1.12 wsdl:operation 要素（wsdl:binding 要素の子要素の場合） 402
14.1.13 wsdl:input 要素（wsdl:binding 要素の孫要素の場合） 403
14.1.14 wsdl:output 要素（wsdl:binding 要素の孫要素の場合） 404
14.1.15 wsdl:fault 要素（wsdl:binding 要素の孫要素の場合） 405
14.1.16 wsdl:service 要素 406
14.1.17 wsdl:port 要素 407
14.1.18 wsdl:documentation 要素 409
14.1.19 soap:binding 要素 409
14.1.20 soap:operation 要素 410
14.1.21 soap:body 要素 411
14.1.22 soap:header 要素 412
14.1.23 soap:fault 要素 413
14.1.24 soap:address 要素 414
14.1.25 soap12:operation 要素 415
14.1.26 soap12:binding 要素 416
14.1.27 soap12:body 要素 417
14.1.28 soap12:header 要素 418
14.1.29 soap12:fault 要素 419
14.1.30 soap12:address 要素 420

14.1.31 xsd:schema 要素 421
14.2 WSDL 作成時の注意事項 426
14.3 JAX-WS 2.1 仕様のサポート範囲 431
14.3.1 JAX-WS 2.1 仕様の機能のサポート範囲 431
14.3.2 Conformance への対応 436
14.4 ハンドラチェイン設定ファイルのサポート範囲 446
14.4.1 javaee:handler-chains 要素 446
14.4.2 javaee:handler-chain 要素 446
14.4.3 javaee:handler 要素 447
14.4.4 javaee:handler-name 要素 447
14.4.5 javaee:handler-class 要素 448
14.4.6 javaee:soap-header 要素 448
14.4.7 javaee:soap-role 要素 449
14.5 SAAJ 1.3 仕様のサポート範囲 450
14.5.1 Detail インタフェース 454
14.5.2 Node インタフェース 454
14.5.3 SOAPBody インタフェース 455
14.5.4 SOAPElement インタフェース 455
14.5.5 SOAPEnvelope インタフェース 457
14.5.6 SOAPFault インタフェース 457
14.5.7 SOAPHeader インタフェース 458
14.5.8 SOAPHeaderElement インタフェース 459
14.5.9 AttachmentPart クラス 459
14.5.10 MessageFactory クラス 460
14.5.11 MimeHeader クラス 460
14.5.12 MimeHeaders クラス 460
14.5.13 SAAJResult クラス 460
14.5.14 SOAPFactory クラス 461
14.5.15 SOAPMessage クラス 461
14.5.16 SOAPPart クラス 463
14.5.17 添付ファイルを使用する場合のサポート範囲 463
14.6 WS-RM 1.2 仕様のサポート範囲 467
14.7 WS-RM Policy 1.2 仕様のサポート範囲 470

15 JAX-WS API のサポート範囲 473
15.1 インタフェースおよびクラスの一覧（JAX-WS） 474

15.2 クライアント API 478
15.2.1 javax.xml.ws.BindingProvider インタフェース 478
15.2.2 javax.xml.ws.Dispatch インタフェース 478
15.2.3 javax.xml.ws.EndpointReference クラス 479
15.2.4 javax.xml.ws.Service クラス 480
15.2.5 javax.xml.ws.wsaddressing.W3CEndpointReference クラス 486
15.3 サービス API 487
15.3.1 javax.xml.ws.Provider インタフェース 487
15.3.2 javax.xml.ws.wsaddressing.W3CEndpointReferenceBuilder クラス 487
15.4 コア API 489
15.4.1 com.sun.xml.ws.developer.StreamingAttachmentFeature クラス 489
15.4.2 com.sun.xml.ws.developer.StreamingDataHandler クラス 491
15.4.3 org.jvnet.mimepull.MIMEConfig クラス 492
15.4.4 javax.xml.ws.handler.Handler<C extends MessageContext> インタフェース 494
15.4.5 javax.xml.ws.handler.HandlerResolver インタフェース 495
15.4.6 javax.xml.ws.handler.LogicalMessageContext インタフェース 495
15.4.7 javax.xml.ws.handler.MessageContext インタフェース 495
15.4.8 javax.xml.ws.handler.PortInfo インタフェース 496
15.4.9 javax.xml.ws.handler.soap.SOAPHandler<T extends SOAPMessageContext> インタフェース 497
15.4.10 javax.xml.ws.handler.soap.SOAPMessageContext インタフェース 497
15.4.11 javax.xml.ws.Holder<T> クラス 498
15.4.12 javax.xml.ws.LogicalMessage インタフェース 498
15.4.13 javax.xml.ws.ProtocolException クラス 499
15.4.14 javax.xml.ws.soap.AddressingFeature クラス 499
15.4.15 javax.xml.ws.soap.MTOMFeature クラス 500
15.4.16 javax.xml.ws.soap.SOAPBinding インタフェース 501
15.4.17 javax.xml.ws.soap.SOAPFaultException クラス 502
15.4.18 javax.xml.ws.WebServiceException クラス 502
15.5 メッセージコンテキストの使用 504
15.5.1 メッセージコンテキストのプロパティのサポート範囲 504
15.5.2 メッセージコンテキスト使用時の注意事項 508

16 WS-RM 1.2 機能の API と設定 511
16.1 com.sun.xml.ws.Closeable クラス 512
16.2 WS-Policy による設定 513

第 4 編　拡張機能

17 WSDL インポート機能 515

17.1 WSDL インポート機能とは 516
17.2 インポートできる WSDL 定義 517
17.3 wsdl:import 要素の書式 519

18 添付ファイル機能（wsi:swaRef 形式） 521

18.1 添付ファイル機能とは（wsi:swaRef 形式） 522
18.2 添付ファイルの Java インタフェース（wsi:swaRef 形式） 524
18.3 添付ファイルの WSDL（wsi:swaRef 形式） 526
18.3.1 添付ファイル使用時の WSDL の記述（wsi:swaRef 形式） 526
18.3.2 添付ファイルの Java 型と WSDL のマッピング（wsi:swaRef 形式） 527
18.3.3 WSDL から添付ファイルの Java 型へのマッピング（wsi:swaRef 形式） 528
18.4 添付ファイル付き SOAP メッセージ（wsi:swaRef 形式） 530
18.4.1 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） 531
18.4.2 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） 535
18.4.3 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） 539
18.5 添付ファイルの Java インスタンスの生成と取得（wsi:swaRef 形式） 540
18.5.1 添付ファイルのインスタンスを生成する方法（wsi:swaRef 形式） 540
18.5.2 添付ファイルデータを取得する方法（wsi:swaRef 形式） 542

19 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） 545

19.1 開発例の構成（SEI 起点・wsi:swaRef 形式の添付ファイル） 546
19.2 開発例の流れ（SEI 起点・wsi:swaRef 形式の添付ファイル） 548
19.3 Web サービスの開発例（SEI 起点・wsi:swaRef 形式の添付ファイル） 549
19.3.1 Web サービス実装クラスを作成する 549
19.3.2 Java ソースを生成する 551
19.3.3 web.xml を作成する 552
19.3.4 application.xml を作成する 553
19.3.5 EAR ファイルを作成する 553
19.4 デプロイと開始の例（SEI 起点・wsi:swaRef 形式の添付ファイル） 555
19.4.1 EAR ファイルをデプロイする 555

19.4.2 Web サービスを開始する 555
19.5 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） 556
19.5.1 サービスクラスを生成する 556
19.5.2 Web サービスクライアントの実装クラスを作成する 557
19.5.3 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 557
19.6 Web サービスの実行例（SEI 起点・wsi:swaRef 形式の添付ファイル） 559
19.6.1 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 559
19.6.2 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 559
19.6.3 Web サービスクライアントを実行する 560

20 添付ファイル機能（MTOM/XOP） 561
20.1 添付ファイル機能とは（MTOM/XOP） 562
20.2 添付ファイルの Java インタフェース（MTOM/XOP） 563
20.3 添付ファイルの WSDL（MTOM/XOP） 566
20.3.1 non-wrapper スタイルでの MTOM/XOP 仕様形式の添付ファイル（MTOM/XOP） 566
20.4 JAX-WS エンジンの動作 567
20.4.1 Web サービス側 JAX-WS エンジンの動作 567
20.4.2 Web サービスクライアント側 JAX-WS エンジンの動作 569
20.5 MTOM/XOP 仕様形式の添付ファイルの SOAP メッセージ 572
20.5.1 添付ファイルから SOAP メッセージへのマッピング（MTOM/XOP） 573
20.5.2 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） 576
20.5.3 SOAP メッセージから添付ファイルへのマッピング（MTOM/XOP） 580
20.6 注意事項 581

21 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） 583
21.1 開発例の構成（SEI 起点・MTOM/XOP 仕様形式の添付ファイル） 584
21.2 開発例の流れ（SEI 起点・MTOM/XOP 仕様形式の添付ファイル） 586
21.3 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） 587
21.3.1 Web サービス実装クラスを作成する 587
21.3.2 Java ソースを生成する 589
21.3.3 web.xml を作成する 590
21.3.4 application.xml を作成する 591
21.3.5 EAR ファイルを作成する 591
21.4 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） 593

21.4.1 EAR ファイルをデプロイする 593
21.4.2 Web サービスを開始する 593
21.5 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） 594
21.5.1 サービスクラスを生成する 594
21.5.2 Web サービスクライアントの実装クラスを作成する 595
21.5.3 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 595
21.6 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） 597
21.6.1 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 597
21.6.2 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 597
21.6.3 Web サービスクライアントを実行する 598

22 ストリーミング 599
22.1 ストリーミングとは 600
22.2 ストリーミングの使用方法 602
22.2.1 Web サービス側 602
22.2.2 Web サービスクライアント側 603
22.2.3 parseEagerly による変化 604
22.2.4 ストリーミングされた添付ファイルの操作 604
22.3 一時ファイル（ストリーミング） 607
22.3.1 命名規則 607
22.3.2 出力と削除 608
22.3.3 見積もり方法 608

23 SEI を起点とした開発の例（ストリーミング使用時） 609
23.1 開発例の構成（SEI 起点・ストリーミング） 610
23.2 開発例の流れ（SEI 起点・ストリーミング） 613
23.3 Web サービスの開発例（SEI 起点・ストリーミング） 614
23.3.1 Web サービス実装クラスを作成する 614
23.3.2 Java ソースを生成する 616
23.3.3 web.xml を作成する 618
23.3.4 application.xml を作成する 618
23.3.5 EAR ファイルを作成する 619
23.4 デプロイと開始の例（SEI 起点・ストリーミング） 620
23.4.1 EAR ファイルをデプロイする 620

23.4.2 Web サービスを開始する 620
23.5 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） 621
23.5.1 サービスクラスを生成する 621
23.5.2 Web サービスクライアントの実装クラスを作成する 622
23.5.3 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 622
23.6 Web サービスの実行例（SEI 起点・ストリーミング） 624
23.6.1 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 624
23.6.2 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 624
23.6.3 Web サービスクライアントを実行する 625

24 WS-RM 1.2 機能 627
24.1 WS-RM 1.2 機能とは 628
24.2 WS-RM 1.2 機能を使用したメッセージの流れ 629
24.3 WS-RM 1.2 機能の送達保証 631
24.4 WS-RM Policy の追加方法 633

25 WSDL を起点とした開発の例（WS-RM 1.2 機能使用時） 635
25.1 開発例の構成（WSDL 起点・WS-RM 1.2） 636
25.2 開発例の流れ（WSDL 起点・WS-RM 1.2） 638
25.3 Web サービスの開発例（WSDL 起点・WS-RM 1.2） 639
25.3.1 WSDL ファイルを作成する 639
25.3.2 WSDL ファイルに WS-RM Policy を追加する 645
25.3.3 SEI を生成する 645
25.3.4 Web サービス実装クラスを作成する 646
25.3.5 Web サービス実装クラスをコンパイルする 647
25.3.6 web.xml を作成する 647
25.3.7 application.xml を作成する 648
25.3.8 EAR ファイルを作成する 649
25.4 デプロイと開始の例（WSDL 起点・WS-RM 1.2） 650
25.4.1 EAR ファイルをデプロイする 650
25.4.2 Web サービスを開始する 650
25.5 Web サービスクライアントの開発例（WSDL 起点・WS-RM 1.2） 651
25.5.1 サービスクラスを生成する 651
25.5.2 Web サービスクライアントの実装クラスを作成する 652
25.5.3 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 653

25.5.4 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 654
25.6 Web サービスの実行例（WSDL 起点・WS-RM 1.2） 655
25.6.1 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 655
25.6.2 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 655
25.6.3 Web サービスクライアントを実行する 656

26 ハンドラフレームワーク 657
26.1 ハンドラフレームワークとは 658
26.2 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 660
26.3 EJB の Web サービスに適用する場合の注意事項 661
26.4 ハンドラの型 662
26.5 ハンドラチェインの編成と実行順序 664
26.5.1 handleMessage メソッドの処理 665
26.5.2 handleFault メソッドの処理 673
26.5.3 close メソッドの処理 677
26.6 ハンドラの初期化と破棄 678
26.7 SOAP ヘッダが含まれる場合のハンドラの動作と設定 679
26.7.1 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） 679
26.7.2 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） 683
26.7.3 処理できる SOAP ヘッダの設定方法 687
26.8 ハンドラの配置 690
26.9 ハンドラチェインの設定 691
26.9.1 Web サービス側のハンドラチェインの設定 691
26.9.2 Web サービスクライアント側のハンドラチェインの設定 693

27 アドレッシング機能 695
27.1 アドレッシング機能とは 696
27.1.1 同期通信 696
27.1.2 非同期通信 697
27.2 WSDL の拡張要素と拡張属性 700
27.2.1 WSDL の拡張要素 700
27.2.2 WSDL の拡張属性 701
27.3 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 703
27.4 フォルトメッセージ 704

27.4.1 サポートしていないサブサブコード 704
27.4.2 フォルトメッセージの注意事項 704
27.5 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） 705
27.5.1 リクエストメッセージ受信時の動作 705
27.5.2 レスポンスメッセージ 706
27.5.3 wsaw:Anonymous 要素指定時の動作 707
27.5.4 Addressing アノテーション指定時の動作 708
27.5.5 Action アノテーション指定時の動作 708
27.5.6 wsa:Action 要素指定時の動作 708
27.5.7 wsa:MessageID 要素を指定していない場合の動作 709
27.6 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） 710
27.6.1 メッセージ送受信時の動作 710
27.6.2 AddressingFeature クラスと匿名 URI 711
27.6.3 wsaw:Action 属性を指定していない場合の動作 711
27.6.4 wsa:Action 要素の注意事項 711
27.6.5 SEI の取得に関する注意事項 711

28 SEI を起点とした開発の例（アドレッシング機能使用時） 713
28.1 開発例の構成（SEI 起点・アドレッシング） 714
28.2 開発例の流れ（SEI 起点・アドレッシング） 716
28.3 Web サービスの開発例（SEI 起点・アドレッシング） 717
28.3.1 Web サービス実装クラスを作成する 717
28.3.2 Java ソースを生成する 719
28.3.3 web.xml を作成する 720
28.3.4 application.xml を作成する 721
28.3.5 EAR ファイルを作成する 721
28.4 デプロイと開始の例（SEI 起点・アドレッシング） 722
28.4.1 EAR ファイルをデプロイする 722
28.4.2 Web サービスを開始する 722
28.5 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） 723
28.5.1 サービスクラスを生成する 723
28.5.2 Web サービスクライアントの実装クラスを作成する 724
28.5.3 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 727
28.6 Web サービスの実行例（SEI 起点・アドレッシング） 728
28.6.1 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 728

28.6.2 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 728
28.6.3 Web サービスクライアントを実行する 729

第 5 編　トラブルシュート

29 障害対策 731

29.1 障害の種類と対策 732
29.1.1 プログラムの実行中に異常終了する場合 732
29.1.2 プログラムが意図したとおりに動作しない場合 733
29.1.3 期待した性能が出ない場合 734
29.2 障害発生時に取得する資料 736
29.3 ログ 737
29.3.1 ログの種類 737
29.3.2 ログの出力先 738
29.3.3 ログの重要度と出力条件 740
29.3.4 ログのフォーマット 742
29.3.5 ログの設定方法 744
29.3.6 ログの見積もり 745
29.4 性能解析トレース（PRF） 748
29.4.1 性能解析トレースの取得レベル 748
29.4.2 性能解析トレースのトレース出力情報 748
29.4.3 性能解析トレースによる性能解析方法 760

付録 763

付録 A　旧バージョンからの移行 764
付録 A.1　バージョンアップインストール 764
付録 A.2　旧バージョンで作成した WSDL の互換性 767
付録 B　旧バージョンの WS-RM 機能 770
付録 B.1　WS-RM 1.1 機能を使用したメッセージの流れ 771
付録 B.2　WS-RM 1.1 機能を使用する場合のシステム構成 772
付録 B.3　WS-RM 1.1 機能の送達保証 775
付録 B.4　WS-RM 1.1 機能の API 一覧 776
付録 B.5　WS-RM 1.1 機能のプロパティ設定 785

付録 B.6　WS-RM 1.1 機能の性能解析トレース（PRF）　786
付録 B.7　WS-RM 1.1 仕様のサポート範囲　791
付録 C　POJO の Web サービスから EJB の Web サービスへの移行　795
付録 D　JAX-WS エンジンのメモリ使用量の算出　797
付録 D.1　アプリケーション起動時のメモリ使用量　797
付録 D.2　1 リクエスト当たりのメモリ使用量　797
付録 D.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　798
付録 D.4　単位時間当たりのメモリ使用量の算出　799
付録 E　このマニュアルの参考情報　801
付録 E.1　関連マニュアル　801
付録 E.2　このマニュアルでの表記　802
付録 E.3　英略語　804
付録 E.4　KB（キロバイト）などの単位表記について　805
付録 F　用語解説　806

索引　811

1 Web サービスの開発および実行の概要

Cosminexus の JAX-WS 機能を使用して，JAX-WS 2.1 仕様に従った Web サービスを開発できます。
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　Web サービスの開発の概要
1.2　Web サービスの開発時および実行時に使用する機能
1.3　Web サービスの開発および実行の前提条件
1.4　Web サービスと Web サービスクライアントの形態
1.5　JAX-WS エンジンの設定

1.1　Web サービスの開発の概要

Cosminexus では，Web サービスの通信基盤として JAX-WS エンジンを提供しています。JAX-WS エンジンは，JAX-WS 2.1 仕様に従って，Web サービスクライアントと Web サービス間の SOAP 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。

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，Web サービスおよび Web サービスクライアントに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。さらに，コマンドで WSDL から Web サービスクライアントの実装に必要な Java ソースを生成します。
- **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。作成したプロバイダ実装クラスをコマンドでコンパイルします。

それぞれの開発の流れについては，「2.　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「Cosminexus 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。

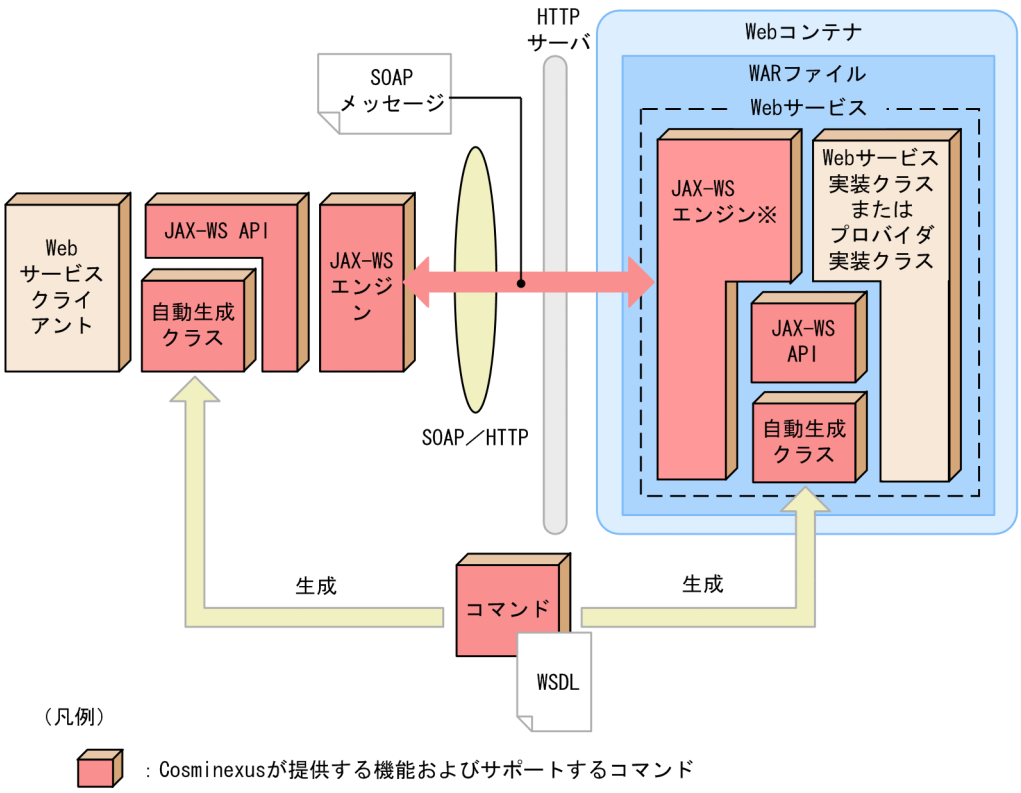
1.2　Web サービスの開発時および実行時に使用する機能

Cosminexus の Web サービスは，次のどちらの仕様範囲でも作成できます。

- POJO
  Java の仕様範囲で Java オブジェクトを作成する方法です。
- EJB
  ビジネスアプリケーション向けに拡張した Java オブジェクトの仕様です。

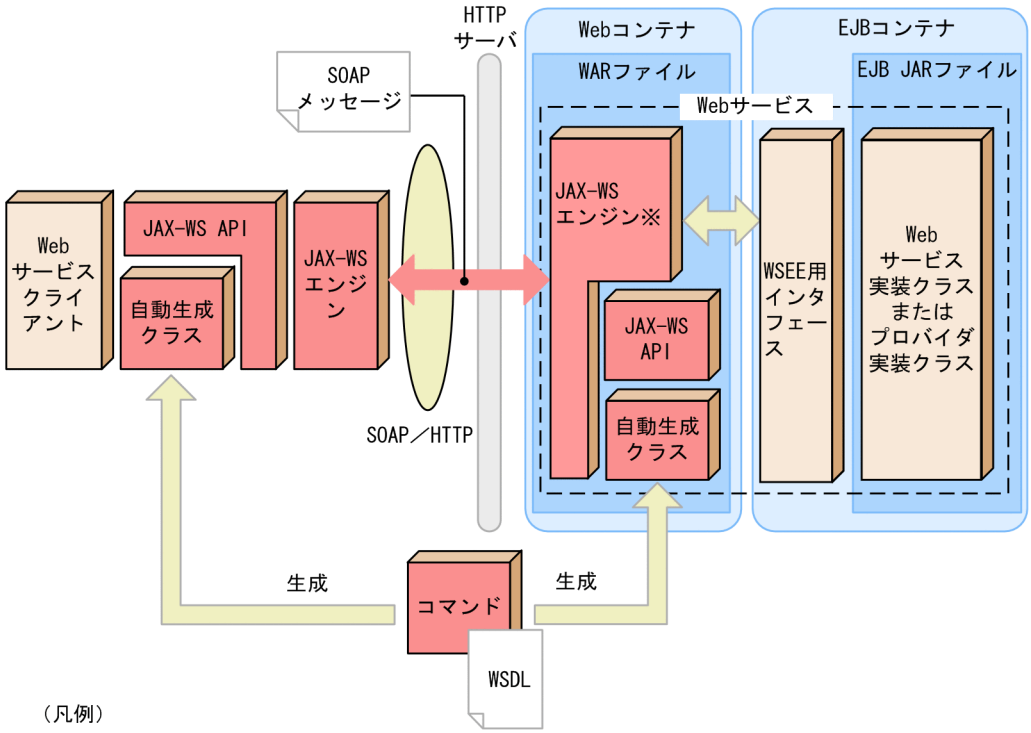
次の図に示す Web サービスの構成を基に，JAX-WS エンジンおよび開発時および実行時に使用する機能について説明します。

図 1-1　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 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-WS エンジン**

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

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
- apt コマンド
- cjwsgen コマンド

cjwsimport コマンドおよび apt コマンドを実行すると，JAX-WS 2.1 仕様に従って，WSDL と Java ソースがマッピングされます。
cjwsgen コマンドを実行すると，コンパイルした Web サービス実装クラスから，Web サービスの実装に必要なスタブや，開発に必要な WSDL が生成されます。
コマンドの使用方法については，「11. 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 ソースに，Web サービスおよび Web サービスクライアントの処理を実装します。

● JAX-WS API

JAX-WS 2.1 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 Web サービスの開発および実行の前提条件

Cosminexus 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。

1.3.1 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「Cosminexus 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　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「Cosminexus アプリケーションサーバ 概説」の「2.3　構成ソフトウェア」を参照してください。

1.3.2 機能および仕様に関する前提条件

Cosminexus 上で 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，「14.　標準仕様のサポート範囲」を参照してください。

(1) デフォルトマッピング

Cosminexus の JAX-WS 機能で提供しているコマンドは，JAX-WS 2.1 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 2.1 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 2.1 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，「12.1　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，「13.1　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。

(2) マッピングのカスタマイズ

Cosminexus の JAX-WS 機能で提供しているコマンドは，JAX-WS 2.1 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 2.1 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 2.1 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，「12.2　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，「13.2　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。

(3) Java と WSDL 間のバインディング

Cosminexus 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 2.1 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。

Cosminexus 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。

(4) WSDL 仕様

Cosminexus 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，「14.1　WSDL 1.1 仕様のサポート範囲」を参照してください。

(5) SOAP 仕様

Cosminexus 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。

(6) Message Exchange Pattern（MEP）

Cosminexus の JAX-WS 機能では，MEP として request-response パターンだけサポートしています。

(7) 非同期関連の機能

Cosminexus の JAX-WS 機能では，JAX-WS 2.1 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。

（8） Dispatch ／ Provider インタフェース関連の機能

Cosminexus の JAX-WS 機能では，JAX-WS 2.1 仕様の 4 章に記載されている Dispatch インタフェース，JAX-WS 2.1 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 2.1 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

（9） Endpoint クラスおよび発行関連の機能

Cosminexus の JAX-WS 機能では，JAX-WS 2.1 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。

（10） ハンドラ関連の機能

Cosminexus 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。

（11） 添付ファイルの使用

Cosminexus 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，「18.　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，「20.　添付ファイル機能（MTOM/XOP）」を参照してください。

（12） メッセージコンテキスト

Cosminexus の JAX-WS 機能では，JAX-WS 2.1 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，「15.5　メッセージコンテキストの使用」を参照してください。

（13）API

Cosminexus の JAX-WS 機能では，JAX-WS 2.1 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，「15.　JAX-WS API のサポート範囲」を参照してください。

（14）XML ／ HTTP バインディング

Cosminexus の JAX-WS 機能では，JAX-WS 2.1 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。

（15）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
Cosminexus の JAX-WS 機能では，JAX-WS 2.1 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
Cosminexus の JAX-WS 機能では，JAX-WS 2.1 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，「27.　アドレッシング機能」を参照してください。

● XML Catalogs 1.1 仕様
Cosminexus の JAX-WS 機能では，JAX-WS 2.1 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はサポートしていません。

● WSEE（JSR-109）仕様
Cosminexus 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 1-1　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。		－
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	－

項番	項目	サポート
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

Cosminexus 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，「14.5　SAAJ 1.3 仕様のサポート範囲」を参照してください。

1.4　Web サービスと Web サービスクライアントの形態

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

1.4.1　Web サービスの形態

Cosminexus では，次の二つの形態の Web サービスをサポートしています。

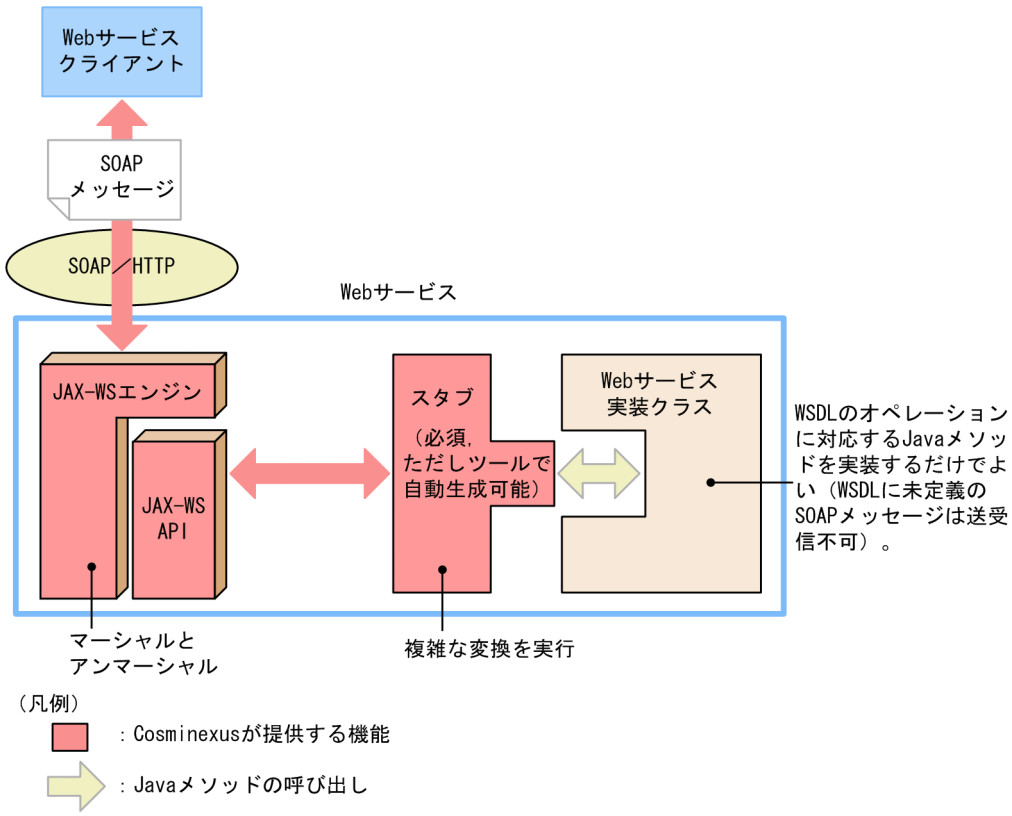
- Web サービス実装クラスを使用した Web サービス
- プロバイダ実装クラスを使用した Web サービス

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 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 1-3　Web サービスの形態（Web サービス実装クラス）



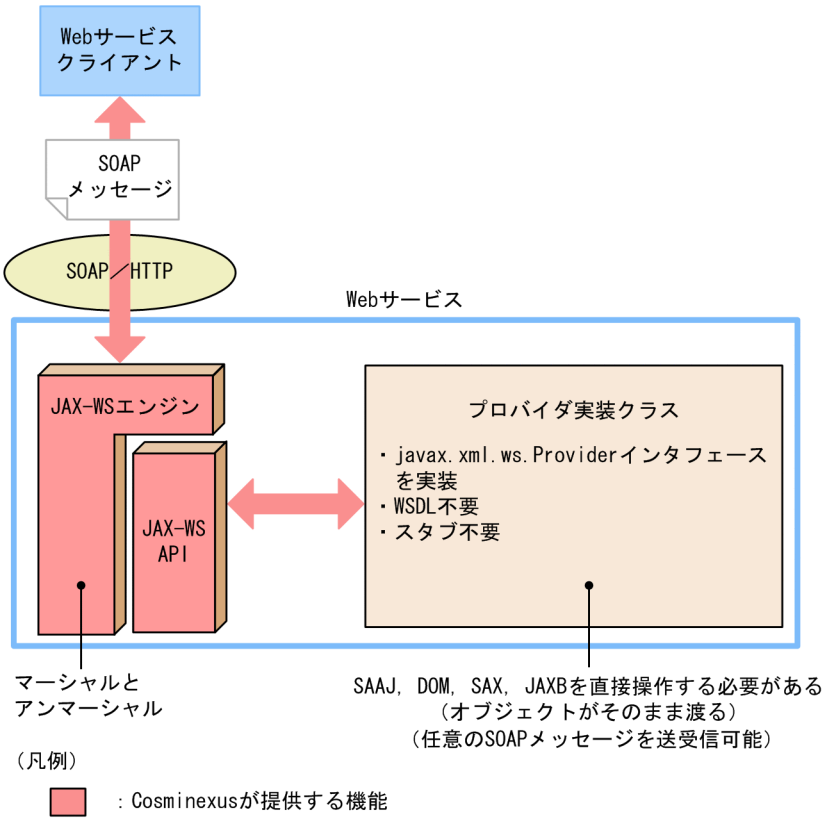
Web サービス実装クラスを使用すると，WSDL のオペレーションに対応する Java メソッドを実装するだけで Web サービスを実現できます。複雑な変換は自動生成されるスタブが実施するため，Web サービス開発時に，API を利用して SOAP メッセージを構成する XML を組み立てるような，複雑なプログラミングをする必要はありません。

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。

(2) プロバイダ実装クラスを使用した 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 1-4 Web サービスの形態（プロバイダ実装クラス）



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。

1.4.2　Web サービスクライアントの形態

Cosminexus では，次の三つの形態の Web サービスクライアントをサポートしています。

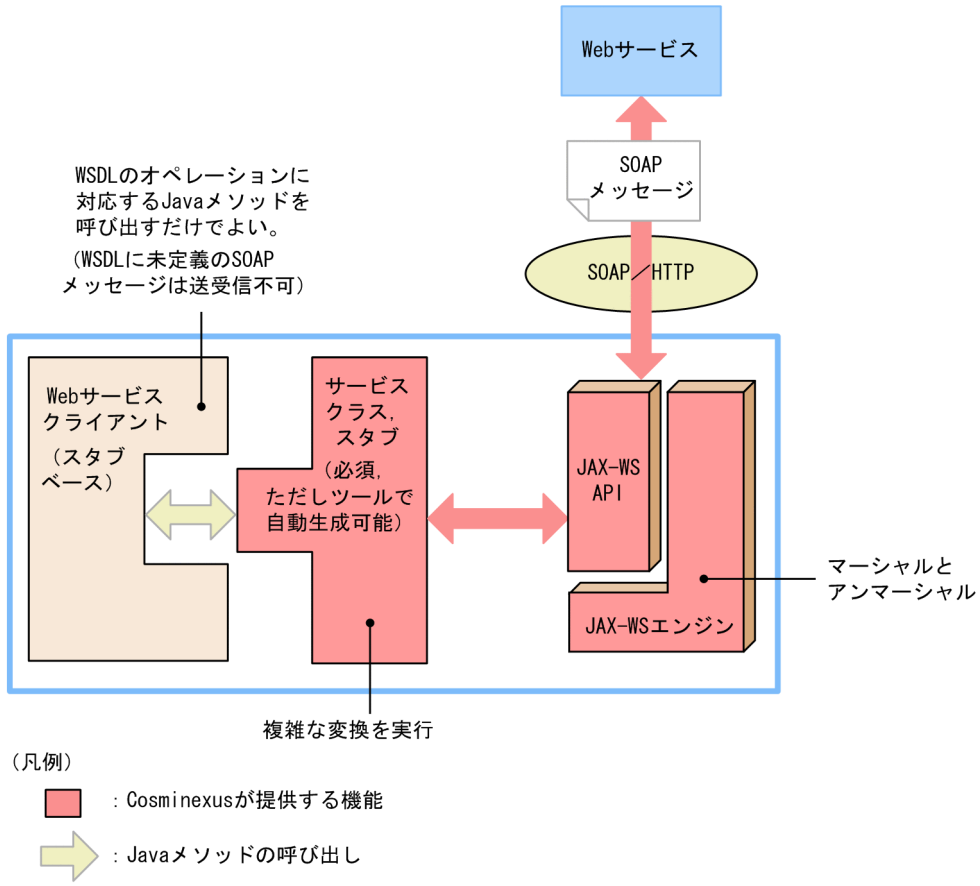
- スタブベースの Web サービスクライアント
- ディスパッチベースの Web サービスクライアント
- API ベースの Web サービスクライアント

それぞれの形態について説明します。

(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 1-5　Web サービスクライアント（スタブベース）



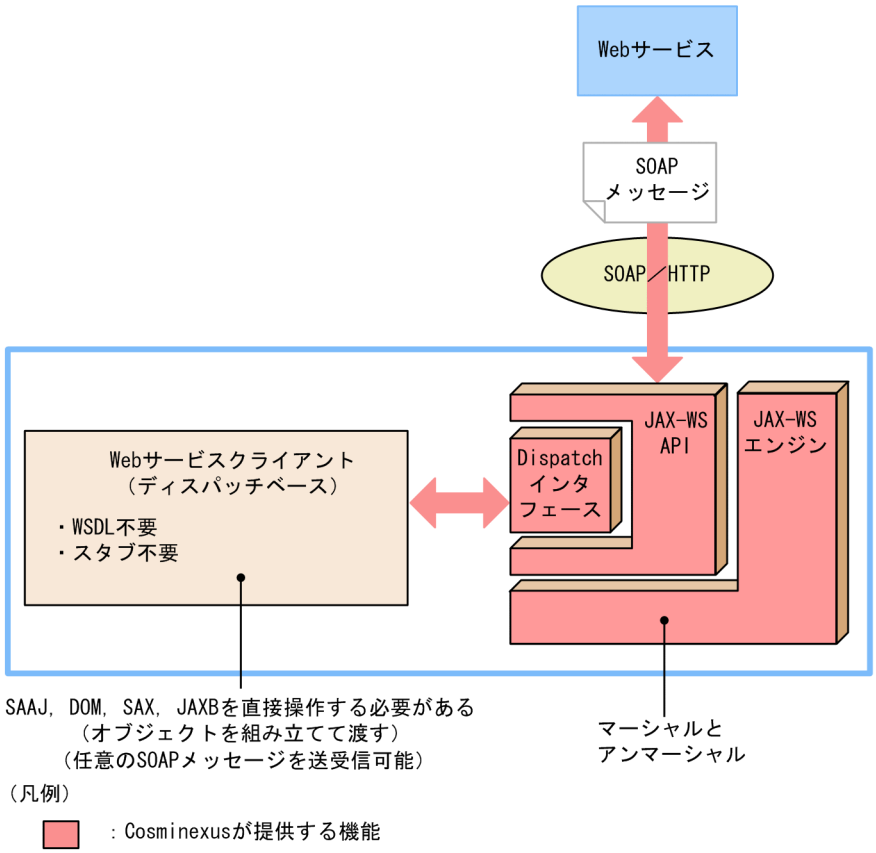
スタブベースの Web サービスクライアントを使用すると，WSDL のオペレーションに対応する Java メソッドを呼び出すだけで，Web サービスを呼び出せます。複雑な変換は自動生成されるスタブが実施するため，Web サービスクライアント開発時に，API を利用して SOAP メッセージを構成する XML を組み立てるような，複雑なプログラミング

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 1-6 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。

（3） API ベースの Web サービスクライアント

Cosminexus の JAX-WS 機能の API のうち，javax.xml.ws.Dispatch インタフェース以外の API を使用する Web サービスクライアントを API ベースの Web サービスクライアントと呼びます。API ベースの Web サービスクライアントは，JAX-WS 機能がサポートしている任意の API を呼び出せます。

1.5 JAX-WS エンジンの設定

JAX-WS 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

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。Web サービスクライアントは，スタブベース，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のどれかを開発します。
この章では，Web サービスおよび Web サービスクライアントの開発の流れについて説明します。

2.1 Web サービス開発の流れ

2.2 Web サービスクライアント開発の流れ

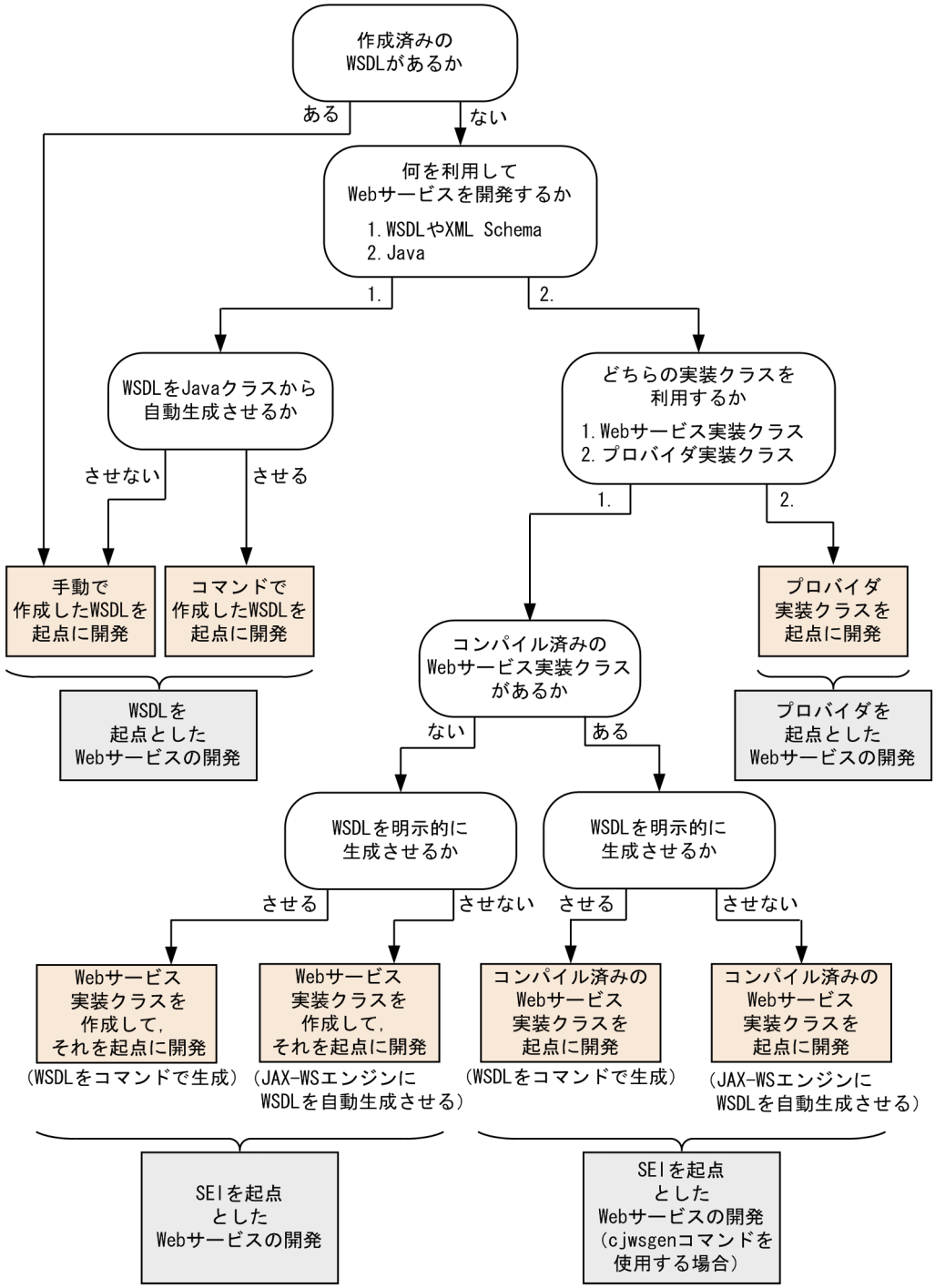
2.1 Web サービス開発の流れ

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（cjwsgen 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。

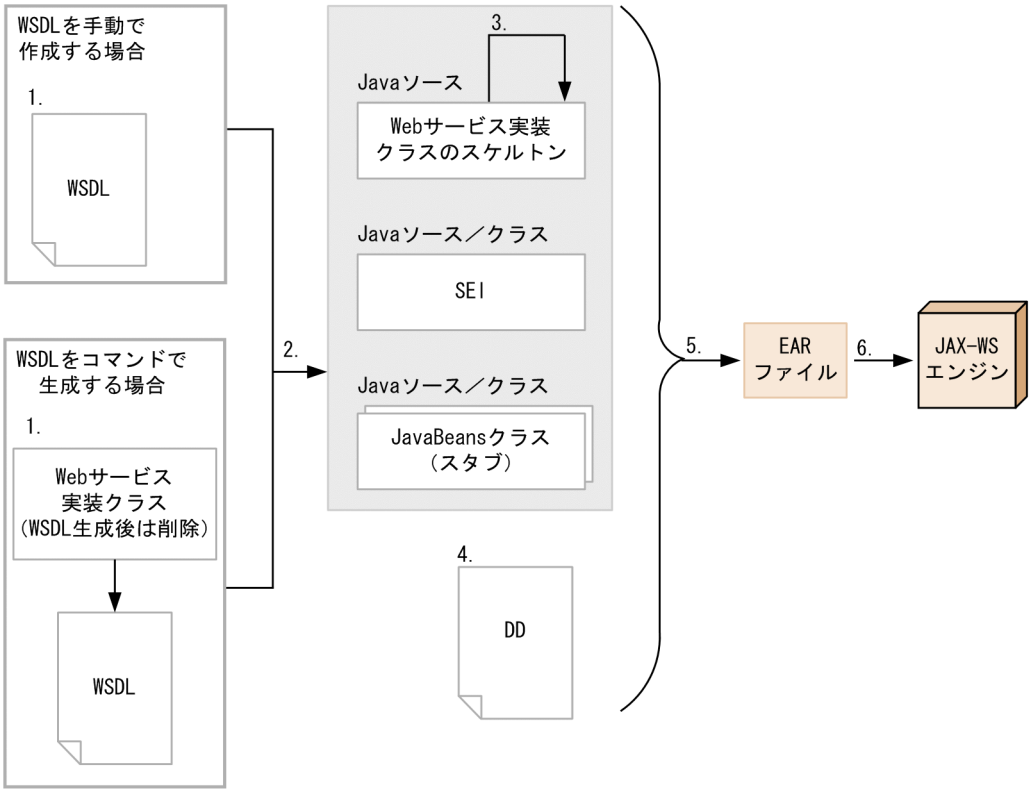
図 2-1 Web サービス開発方法の選択の流れ



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，「14.1　WSDL1.1 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 cjwsgen 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，cjwsgen 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，「11.1　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。cjimportapp コマンドおよび cjstartapp コマンドについては，マニュアル「Cosminexus 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「Cosminexus 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「Cosminexus 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。

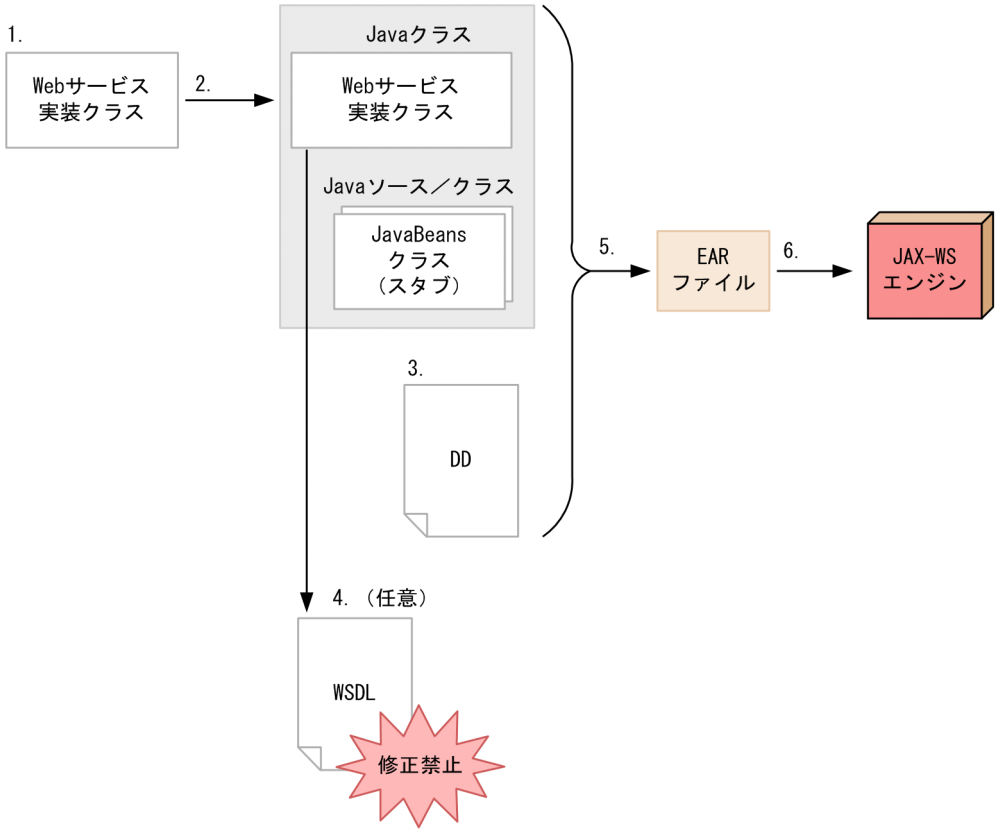
WSDL を起点とした Web サービスの開発例については，次に示す個所を参照してください。

- 「4.3　Web サービスの開発例（WSDL 起点）」
- 「25.3　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 2.1 仕様および JAXB 2.1 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 2.1 仕様のサポート範囲については，「14.3　JAX-WS 2.1 仕様のサポート範囲」を参照してください。JAXB 2.1 仕様のサポート範囲については，マニュアル「Cosminexus 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. apt コマンドを実行する（追加の Java ソースの生成）
   apt 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，JavaBeans クラス（スタブ）など，Web サービスの実行に必要な追加の Java ソースを生成します。apt コマンドについては，「11.2　apt コマンド」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. cjwsgen 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 cjwsgen コマンドを実行し，Web サービス実装クラスから WSDL を生成します。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。
   cjwsgen 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。
   cjwsgen コマンドについては，「11.3　cjwsgen 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。cjimportapp コマンドおよび cjstartapp コマンドについては，マニュアル「Cosminexus 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「Cosminexus 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「Cosminexus 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。

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。

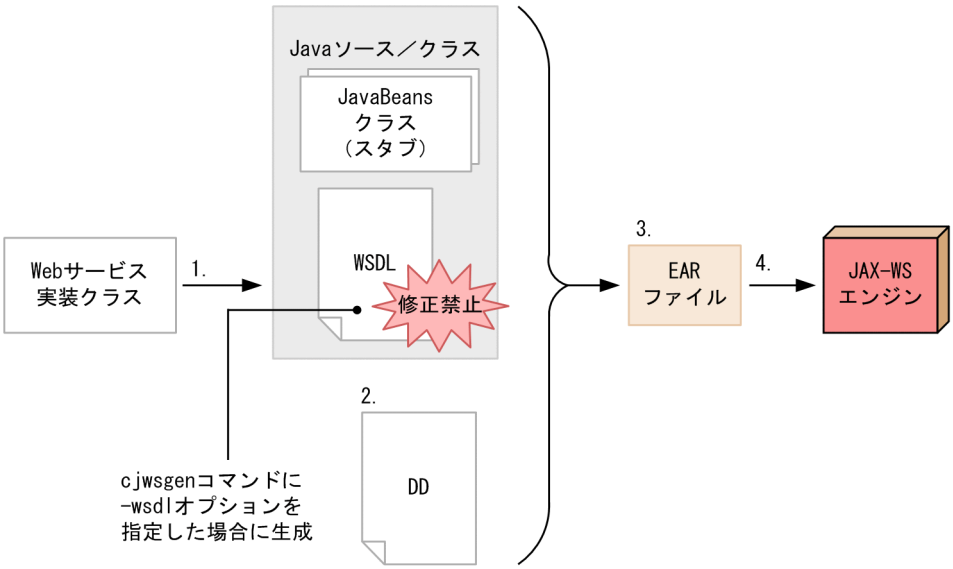
- 「5.3　Web サービスの開発例（SEI 起点）」
- 「6.3　Web サービスの開発例（SEI 起点・cjwsgen コマンド）」
- 「7.3　Web サービスの開発例（SEI 起点・カスタマイズ）」
- 「8.3　Web サービスの開発例（SEI 起点・EJB の Web サービス）」

- 「19.3　Web サービスの開発例（SEI 起点・wsi:swaRef 形式の添付ファイル）」
- 「23.3　Web サービスの開発例（SEI 起点・ストリーミング）」
- 「28.3　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（cjwsgen 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，cjwsgen 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（cjwsgen コマンドを使用する場合）



1. cjwsgen コマンドを実行する（追加の Java ソースの生成）

   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，cjwsgen コマンドを実行して，コンパイルが完了している Web サービス実装クラスから，JavaBeans クラス（スタブ）など，Web サービスの実装に必要な追加の Java ソースを生成します。このとき，-wsdl オプションを指定すると，WSDL も生成できます。この場合，生成した WSDL は変更しないでください。

   cjwsgen コマンドについては，「11.3　cjwsgen 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。cjimportapp コマンドおよび cjstartapp コマンドについては，マニュアル「Cosminexus 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「Cosminexus 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「Cosminexus 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。

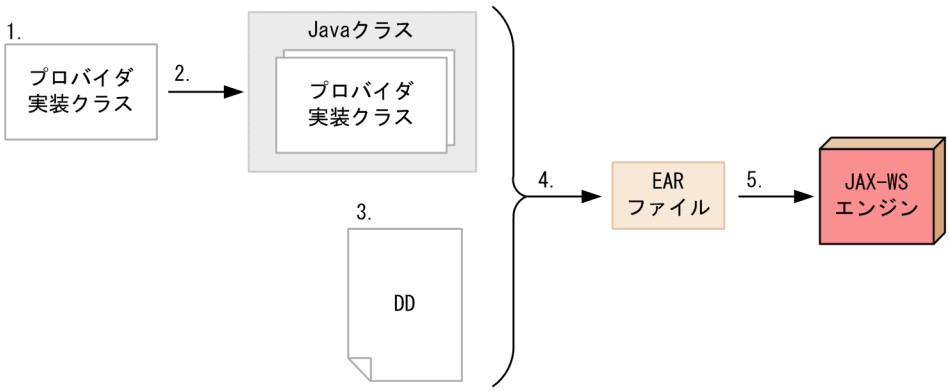
SEI を起点とした Web サービスの開発例については，次に示す個所を参照してください。

- 「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・cjwsgen 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）」
- 「19.3　Web サービスの開発例（SEI 起点・wsi:swaRef 形式の添付ファイル）」
- 「23.3　Web サービスの開発例（SEI 起点・ストリーミング）」
- 「28.3　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 2.1 仕様および JAXB 2.1 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 2.1 仕様のサポート範囲については，「14.3　JAX-WS 2.1 仕様のサポート範囲」を参照してください。JAXB 2.1 仕様のサポート範囲については，マニュアル「Cosminexus 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. **apt コマンドを実行する（コンパイルとエラーチェック）**
   apt 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。cjimportapp コマンドおよび cjstartapp コマンドについては，マニュアル「Cosminexus 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「Cosminexus 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「Cosminexus 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

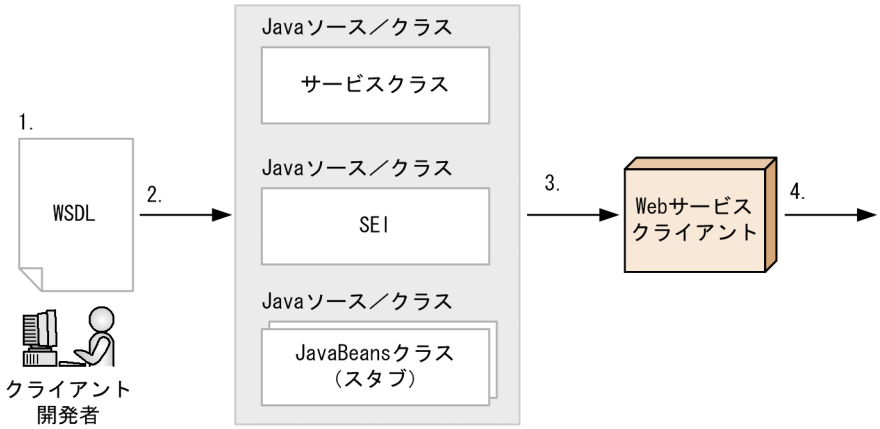
Web サービスクライアントには，次に示す開発方法があります。

- 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）
- API ベースの Web サービスクライアントの開発（2.2.3）

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，「11.1　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・cjwsgen 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）」
- 「19.5　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）」
- 「23.3　Web サービスの開発例（SEI 起点・ストリーミング）」
- 「28.5　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 2.1 仕様および JAXB 2.1 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。

2.2.3 API ベースの Web サービスクライアントの開発の流れ

API 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 2.1 仕様および JAXB 2.1 仕様に従って，API を使用して Web サービスクライアントを実装してください。

3 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

3.2　WSDL と Java ソースのマッピング

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

3.4　web.xml の作成

3.5　アーカイブの作成

3.6　Web サービスクライアントの実装

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，「14.1　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 2.1 仕様に従った WSDL が自動生成されます。

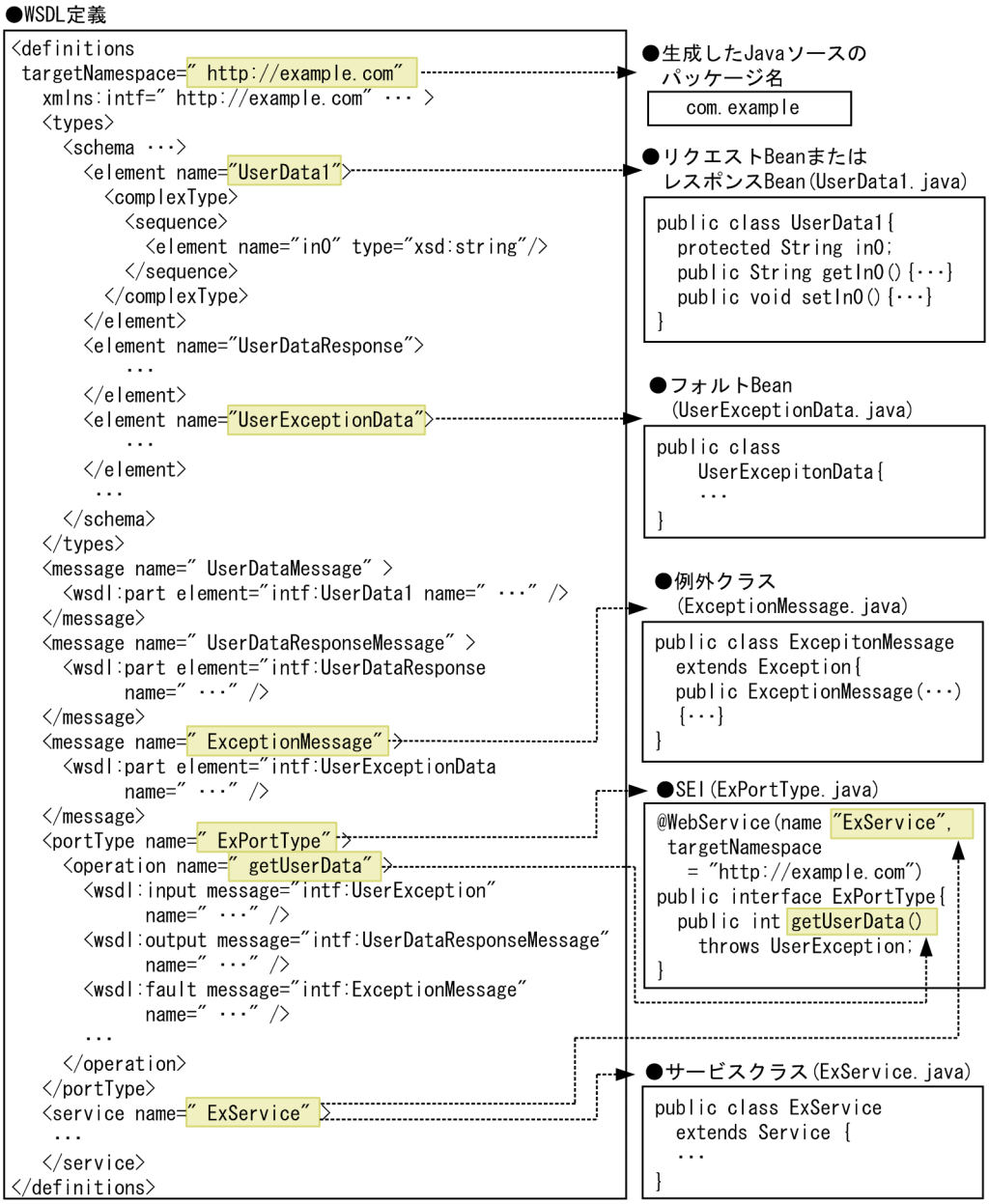
3.2 WSDL と Java ソースのマッピング

cjwsimport コマンドおよび apt 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，「12.　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

apt コマンドを実行すると，Java ソースから WSDL へマッピングされます。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,「13. 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。

（1）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。

（2）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。

- classes 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。

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。

（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 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。

（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="2.5" 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_2_5.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>
```

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） web.xml を WAR ファイルに含めない場合の動作

Cosminexus 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 version="2.5" 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_2_5.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.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。
- Cosminexus 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。

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

Cosminexus 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。

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「Cosminexus 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「1.3.2　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。※
│ └ 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 version="2.5" 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_2_5.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>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nameA+ "/" + WebサービスnのクラスnameA</
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 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="2.5" 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_2_5.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>
```

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 サービスを開発できます。

- Java アプリケーション
- 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）

Cosminexus 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

スタブベースの Web サービスクライアントの実装例について説明します。

(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。

- サービスクラスのどちらのコンストラクタを利用するのか
- 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

各条件の組み合わせと，参照される WSDL の対応を次の表に示します。それぞれの条件の場合の例については，「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
絶対的な 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。
相対的な 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的な 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) サービスクラスおよびポートの再利用

サービスクラス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，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変更は，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することがあります。

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。

(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

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 2.1 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/AddNumbersImplService?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。

次に，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示します。

実行例 1

```
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" -wsdllocation file:/home/
wsdl4runtime/master.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変更は，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することがあります。

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 +
" = " + returnValue );
        }
        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) サービスクラスのインスタンス化およびポートの初期化

サーブレットから Web サービスを呼び出す実装では，サービスクラスおよびポートをサーブレットのメンバ変数として保持し，初期化します。

サービスクラスのインスタンス化およびポートの初期化の例を次に示します。

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

    AddNumbersImplService service;
    AddNumbersImpl port;

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

(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

サーブレットのメンバ変数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 {

    AddNumbersImplService service;
    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 {

    AddNumbersImplService service;
    AddNumbersImpl port;

    @Override
    public void init() {
        // サービスクラスを初期化する
        service = new AddNumbersImplService();
        // ポートを取得する
        port = service.getAddNumbersImplPort();
    }

    @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
```

注意事項

Web サービスクライアント（サーブレット）と 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，Web サービスクライアントが含まれる Web アプリケーションの DD（web.xml）に <load-on-startup> タグを指定し，J2EE サーバ起動時にサーブレットを初期化する設定にすると，Web サービスクライアント（サーブレット）の init() 実行時に例外が発生します。そのため，DD（web.xml）には <load-on-startup> タグを指定しないでください。性能などの理由で <load-on-startup> タグを使用したい場合は，Web サービスクライアントと Web サービスを別々の J2EE サーバ上にデプロイして実行してください。

3.6.2 ディスパッチベースの実装例

Cosminexus でサポートしている javax.xml.ws.Dispatch インタフェース，JAX-WS 2.1 仕様，JAXB 2.1 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変更は，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することがあります。

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

Cosminexus 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，「15. JAX-WS 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変更は，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することがあります。

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，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) サービスクラスおよびポートの再利用」
- 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，これら設定は OS 全体に影響が及ぶため，注意が必要です。

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）

4.2 開発例の流れ（WSDL 起点）

4.3 Web サービスの開発例（WSDL 起点）

4.4 デプロイと開始の例（WSDL 起点）

4.5 Web サービスクライアントの開発例（WSDL 起点）

4.6 Web サービスの実行例（WSDL 起点）

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。

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	「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。
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**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，「14.1　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「Cosminexus 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，cjwsgen 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¥client¥lib
¥j2ee-javax.jar" -d .¥temporary¥classes
.¥temporary¥src¥com¥example¥sample¥TestJaxWsImpl.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，cjwsgen コマンドの WSDL 生成機能で WSDL ファイルを作成します。

cjwsgen 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¥cjwsgen.bat" -r .¥WEB-INF¥wsdl -d
.¥temporary¥classes -cp .¥temporary¥classes com.example.sample.TestJaxWsImpl
```

cjwsgen 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:tns="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://example.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，「11.1 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 WEB-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，「12.1.1 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 2.1 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，「12. 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://example.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¥client¥lib
¥j2ee-javax.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="2.5" 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_2_5.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>
```

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="5" 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_5.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>
```

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「Cosminexus 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「5.2.4　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「Cosminexus 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「Cosminexus 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「Cosminexus 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「Cosminexus 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，「11.1 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，「12.1.1　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。

ファイル名	説明
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 2.1 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。
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。

- 「12.1.2　ポートタイプから SEI 名へのマッピング」
- 「12.1.3　オペレーションからメソッド名へのマッピング」
- 「12.1.4　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）」
- 「12.1.5　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¥client¥lib
¥j2ee-javax.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=<Cosminexus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=<Cosminexusのインストールディレクトリ>
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

<Cosminexus のインストールディレクトリ > の部分は，Cosminexus 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「Cosminexus 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「15.2　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「Cosminexus 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「15.3　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「Cosminexus 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）

5.2 開発例の流れ（SEI 起点）

5.3 Web サービスの開発例（SEI 起点）

5.4 デプロイと開始の例（SEI 起点）

5.5 Web サービスクライアントの開発例（SEI 起点）

5.6 Web サービスの実行例（SEI 起点）

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
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。

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	「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　Java ソースを生成する」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	「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　Java ソースを生成する」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.ear	「5.3.6　EAR ファイルを作成する」で作成します。
fromjava.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。

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**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. apt コマンドを実行し，追加の Java ソースを生成する（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;
    }
}
```

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

```
package com.sample;

@javax.jws.WebService
@javax.xml.ws.BindingType(javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_BINDI
NG)
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 Java ソースを生成する

apt コマンドを実行して，Web サービス実装クラスから Web サービスの開発に必要な追加の Java ソースを生成します。また，Web サービス実装クラスを含めてコンパイルします。apt コマンドについては，「11.2　apt コマンド」を参照してください。

apt コマンドの実行例を次に示します。

● Windows(x86) の場合

```
> set HNTRLIB2_HOME=<HNTRLib2インストールディレクトリ>
> cd c:¥temp¥jaxws¥works¥fromjava¥server¥
> mkdir WEB-INF¥classes¥
> apt -J-Dcosminexus.home="%COSMINEXUS_HOME%" -cp
"%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥client¥lib
¥j2ee-javax.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%
¥jaxp¥lib¥csmjaxp.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar;%HNTRLIB2_H
OME%¥classes¥hntrlib2j.jar;%HNTRLIB2_HOME%¥classes¥hntrlibMj.jar" -d
WEB-INF¥classes¥ -s src src¥com¥sample¥AddNumbersImpl.java
src¥com¥sample¥AddNumbersFault.java
```

● Windows(x64) の場合

```
> set HNTRLIB2_HOME=<HNTRLib2インストールディレクトリ>
> cd c:¥temp¥jaxws¥works¥fromjava¥server¥
> mkdir WEB-INF¥classes¥
> apt -factory com.cosminexus.istack.ws.AnnotationProcessorFactoryImpl
-J-Dcosminexus.home="%COSMINEXUS_HOME%" -cp
"%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥client¥lib
¥j2ee-javax.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%
¥jaxp¥lib¥csmjaxp.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar;%HNTRLIB2_H
OME%¥classes¥hntrlib2j64.jar;%HNTRLIB2_HOME%¥classes¥hntrlibMj64.jar" -d
WEB-INF¥classes¥ -s src src¥com¥sample¥AddNumbersImpl.java
src¥com¥sample¥AddNumbersFault.java
```

<HNTRLib2 インストールディレクトリ > の部分には，次のコマンドの実行結果を指定します。

● Windows(x86) の場合

```
> "%COSMINEXUS_HOME%¥common¥bin¥gethntr2conf.exe" HNTR2INSTDIR
```

● Windows(x64) の場合

```
> "%COSMINEXUS_HOME%¥common¥bin¥gethntr2conf64.exe" HNTR2INSTDIR
```

apt コマンドが正常に終了すると，
c:¥temp¥jaxws¥works¥fromjava¥server¥src¥com¥sample¥jaxws¥ 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 5-3　Java ソース生成時の生成物（SEI 起点）

ファイル名	説明
Add.java	add メソッドに対応するリクエスト bean です。
AddResponse.java	add メソッドに対応するレスポンス bean です。
AddNumbersFaultBean.java	AddNumbersFault に対応するフォルト bean 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。

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="2.5" 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_2_5.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>/AddNumbersImplService</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

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="5" 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_5.xsd">

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

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「Cosminexus 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「5.2.4　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，cjwsgen 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。cjwsgen コマンドについては，「11.3　cjwsgen コマンド」を参照してください。

cjwsgen 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¥cjwsgen.bat" -r .¥WEB-INF¥wsdl -d
.¥temporary -cp .¥WEB-INF¥classes com.sample.AddNumbersImpl
> rmdir /S /Q .¥temporary
```

cjwsgen 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 5-4　cjwsgen 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.war .¥WEB-INF
> jar cvf fromjava.ear .¥fromjava.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.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.ear
```

cjimportapp コマンドについては，マニュアル「Cosminexus 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「Cosminexus 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
```

cjstartapp コマンドについては，マニュアル「Cosminexus 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「Cosminexus 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，「11.1 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/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，「12.1.1　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 5-5　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 2.1 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。

- 「12.1.2　ポートタイプから SEI 名へのマッピング」
- 「12.1.3　オペレーションからメソッド名へのマッピング」
- 「12.1.4　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）」
- 「12.1.5　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¥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¥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=<Cosminexus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=<Cosminexusのインストールディレクトリ>
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

<Cosminexus のインストールディレクトリ > の部分は，Cosminexus 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「Cosminexus 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「15.2　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「Cosminexus 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「15.3　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「Cosminexus 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（cjwsgen コマンドを使用する場合）

この章では，SEI を起点として cjwsgen 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・cjwsgen コマンド）

6.2 開発例の流れ（SEI 起点・cjwsgen コマンド）

6.3 Web サービスの開発例（SEI 起点・cjwsgen コマンド）

6.4 デプロイと開始の例（SEI 起点・cjwsgen コマンド）

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

6.6 Web サービスの実行例（SEI 起点・cjwsgen コマンド）

6.1 開発例の構成（SEI 起点・cjwsgen コマンド）

この章で説明する開発例では，Cosminexus が提供する cjwsgen 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。cjwsgen コマンドの使用方法については，「11.3　cjwsgen 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・cjwsgen 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・cjwsgen 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**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・cjwsgen 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. cjwsgen 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・cjwsgen コマンド）

ここでは，SEI を起点とし，cjwsgen 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

cjwsgen 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 cjwsgen 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¥cjwsgen.bat" -r WEB-INF¥wsdl -d
WEB-INF¥classes -cp WEB-INF¥classes com.sample.AddNumbersImpl
```

cjwsgen 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，「12.1.1　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・cjwsgen 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。

ファイル名	説明
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 は，オペレーション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のマッピングについては，「12.1.3　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="2.5" 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_2_5.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>
```

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="5" 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_5.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>
```

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「Cosminexus 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「5.2.4　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・cjwsgen コマンド）

ここでは，SEI を起点として cjwsgen 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「Cosminexus 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「Cosminexus 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「Cosminexus 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「Cosminexus 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・cjwsgen コマンド）

ここでは，SEI を起点とし，cjwsgen 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，「11.1 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，「12.1.1　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・cjwsgen 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 2.1 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。

ファイル名	説明
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。

- 「12.1.2　ポートタイプから SEI 名へのマッピング」
- 「12.1.3　オペレーションからメソッド名へのマッピング」
- 「12.1.4　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）」
- 「12.1.5　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・cjwsgen コマンド）

ここでは，SEI を起点とし，cjwsgen 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=<Cosminexus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=<Cosminexusのインストールディレクトリ>
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

<Cosminexus のインストールディレクトリ > の部分は，Cosminexus 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「Cosminexus 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「15.2　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「Cosminexus 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「15.3　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「Cosminexus 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）

7.2 開発例の流れ（SEI 起点・カスタマイズ）

7.3 Web サービスの開発例（SEI 起点・カスタマイズ）

7.4 デプロイと開始の例（SEI 起点・カスタマイズ）

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）

7.6 Web サービスの実行例（SEI 起点・カスタマイズ）

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		fromjava
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。

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	「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　Java ソースを生成する」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	「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　Java ソースを生成する」で使用します。
annotations.ear	「7.3.5　EAR ファイルを作成する」で作成します。
annotations.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**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. apt コマンドを実行し，追加の Java ソースを生成する（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_BINDI
NG)
@javax.jws.soap.SOAPBinding(style =
javax.jws.soap.SOAPBinding.Style.DOCUMENT, use =
javax.jws.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="num2")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;
    }
}
```

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

```
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_BINDI
NG)
@javax.jws.soap.SOAPBinding(style =
javax.jws.soap.SOAPBinding.Style.DOCUMENT, use =
javax.jws.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="num2")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 Java ソースを生成する

apt コマンドを実行して，Web サービス実装クラスから Web サービスの開発に必要な追加の Java ソースを生成します。また，Web サービス実装クラスを含めてコンパイルします。apt コマンドについては，「11.2　apt コマンド」を参照してください。

apt コマンドの実行例を次に示します。

● Windows(x86) の場合

```
> set HNTRLIB2_HOME=<HNTRLib2インストールディレクトリ>
> cd c:¥temp¥jaxws¥works¥annotations¥server¥
> mkdir WEB-INF¥classes¥
> apt -J-Dcosminexus.home="%COSMINEXUS_HOME%" -cp
"%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥client¥lib
¥j2ee-javax.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%
¥jaxp¥lib¥csmjaxp.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar;%HNTRLIB2_H
OME%¥classes¥hntrlib2j.jar;%HNTRLIB2_HOME%¥classes¥hntrlibMj.jar" -d
WEB-INF¥classes¥ -s src src¥com¥sample¥AddNumbersImpl.java
src¥com¥sample¥AddNumbersFault.java
```

● Windows(x64) の場合

```
> set HNTRLIB2_HOME=<HNTRLib2インストールディレクトリ>
> cd c:¥temp¥jaxws¥works¥annotations¥server¥
> mkdir WEB-INF¥classes¥
> apt -factory com.cosminexus.istack.ws.AnnotationProcessorFactoryImpl
-J-Dcosminexus.home="%COSMINEXUS_HOME%" -cp
"%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥client¥lib
¥j2ee-javax.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%
¥jaxp¥lib¥csmjaxp.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar;%HNTRLIB2_H
OME%¥classes¥hntrlib2j64.jar;%HNTRLIB2_HOME%¥classes¥hntrlibMj64.jar" -d
WEB-INF¥classes¥ -s src src¥com¥sample¥AddNumbersImpl.java
src¥com¥sample¥AddNumbersFault.java
```

<HNTRLib2 インストールディレクトリ > の部分には，次のコマンドの実行結果を指定します。

● Windows(x86) の場合

```
> "%COSMINEXUS_HOME%¥common¥bin¥gethntr2conf.exe" HNTR2INSTDIR
```

● Windows(x64) の場合

```
> "%COSMINEXUS_HOME%¥common¥bin¥gethntr2conf64.exe" HNTR2INSTDIR
```

apt コマンドが正常に終了すると，
c:¥temp¥jaxws¥works¥annotations¥server¥src¥com¥sample¥jaxws¥ 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 7-3 Java ソース生成時の生成物（SEI 起点・カスタマイズ）

生成物	説明
Add.java	add メソッドに対応するリクエスト bean です。
AddResponse.java	add メソッドに対応するレスポンス bean です。
AddNumbersFaultBean.java	AddNumbersFault に対応するフォルト bean 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。

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="2.5" 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_2_5.xsd">
  <description>Sample web service &amp;quot;annotations&amp;quot;</description>
  <display-name>Sample_web_service_annotations</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>
```

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="5" 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_5.xsd">

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

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「Cosminexus 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「5.2.4　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.war .¥WEB-INF
> jar cvf annotations.ear .¥annotations.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.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.ear
```

cjimportapp コマンドについては，マニュアル「Cosminexus 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「Cosminexus 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
```

cjstartapp コマンドについては，マニュアル「Cosminexus 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「Cosminexus 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，「11.1 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/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，「12.1.1　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 7-4　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 2.1 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。

ファイル名	説明
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。

- 「12.1.2　ポートタイプから SEI 名へのマッピング」
- 「12.1.3　オペレーションからメソッド名へのマッピング」
- 「12.1.4　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）」
- 「12.1.5　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¥client¥lib
¥j2ee-javax.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=<Cosminexus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=<Cosminexusのインストールディレクトリ>
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

<Cosminexus のインストールディレクトリ > の部分は，Cosminexus 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「Cosminexus 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「15.2　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「Cosminexus 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「15.3　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「Cosminexus 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）

8.2 開発例の流れ（SEI 起点・EJB の Web サービス）

8.3 Web サービスの開発例（SEI 起点・EJB の Web サービス）

8.4 デプロイと開始の例（SEI 起点・EJB の Web サービス）

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）

8.6 Web サービスの実行例（SEI 起点・EJB の Web サービス）

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		fromjava
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。

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	「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　Java ソースを生成する」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　Java ソースを生成する」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.ear	「8.3.5　EAR ファイルを作成する」で作成します。
statelessjava.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**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. apt コマンドを実行し，追加の Java ソースを生成する（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 Java ソースを生成する

apt コマンドを実行して，Web サービス実装クラスから Web サービスの開発に必要な追加の Java ソースを生成します。また，Web サービス実装クラスを含めてコンパイルします。apt コマンドについては，「11.2 apt コマンド」を参照してください。

apt コマンドの実行例を次に示します。

● Windows(x86) の場合

```
> set HNTRLIB2_HOME=<HNTRLib2インストールディレクトリ>
> cd c:¥temp¥jaxws¥works¥statelessjava¥server¥
> mkdir .¥jar¥
> apt -J-Dcosminexus.home="%COSMINEXUS_HOME% -cp"
"%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥client¥lib
¥j2ee-javax.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%
¥jaxp¥lib¥csmjaxp.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar;%HNTRLIB2_H
OME%¥classes¥hntrlib2j.jar;%HNTRLIB2_HOME%¥classes¥hntrlibMj.jar" -d
jar¥ -s src src¥com¥sample¥AddNumbersImpl.java
src¥com¥sample¥AddNumbersFault.java
```

● Windows(x64) の場合

```
> set HNTRLIB2_HOME=<HNTRLib2インストールディレクトリ>
> cd c:¥temp¥jaxws¥works¥statelessjava¥server¥
> mkdir .¥jar¥
> apt -factory com.cosminexus.istack.ws.AnnotationProcessorFactoryImpl
-J-Dcosminexus.home="%COSMINEXUS_HOME%" -cp
"%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥client¥lib
¥j2ee-javax.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%
¥jaxp¥lib¥csmjaxp.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar;%HNTRLIB2_H
OME%¥classes¥hntrlib2j64.jar;%HNTRLIB2_HOME%¥classes¥hntrlibMj64.jar" -d
.¥jar¥ -s src src¥com¥sample¥AddNumbersImpl.java
src¥com¥sample¥AddNumbersFault.java
```

apt コマンドを実行すると，Web サービス実装クラスに javax.ejb.Stateless アノテーションが付いているため次のような警告が出力されます。

```
警告：プロセッサなしの注釈タイプです：[javax.ejb.Stateless]
```

<HNTRLib2 インストールディレクトリ > の部分には，次のコマンドの実行結果を指定

します。

● Windows(x86) の場合

```
> "%COSMINEXUS_HOME%¥common¥bin¥gethntr2conf.exe" HNTR2INSTDIR
```

● Windows(x64) の場合

```
> "%COSMINEXUS_HOME%¥common¥bin¥gethntr2conf64.exe" HNTR2INSTDIR
```

apt コマンドが正常に終了すると，
c:¥temp¥jaxws¥works¥statelessjava¥server¥src¥com¥sample¥jaxws¥ 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 8-3　Java ソース生成時の生成物（SEI 起点）

ファイル名	説明
Add.java	add メソッドに対応するリクエスト bean です。
AddResponse.java	add メソッドに対応するレスポンス bean です。
AddNumbersFaultBean.java	AddNumbersFault に対応するフォルト bean 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。

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="5" 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_5.xsd">

  <description>Sample application &quot;statelessjava&quot;</description>
  <display-name>Sample_application_statelessjava</display-name>
  <module>
    <ejb>statelessjava.jar</ejb>
  </module>
</application>
```

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「Cosminexus

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「5.2.4　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，cjwsgen 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。cjwsgen コマンドについては，「11.3　cjwsgen コマンド」を参照してください。

cjwsgen 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¥cjwsgen.bat" -r .¥jar¥META-INF¥wsdl -d
.¥temporary -cp .¥jar com.sample.AddNumbersImpl
> rmdir /S /Q .¥temporary
```

cjwsgen 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 8-4　cjwsgen 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.jar -C jar com
> jar cvf statelessjava.ear .¥statelessjava.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.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.ear
```

cjimportapp コマンドについては，マニュアル「Cosminexus 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「Cosminexus 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
```

cjstartapp コマンドについては，マニュアル「Cosminexus 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「Cosminexus 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，「11.1 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/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，「12.1.1　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 8-5　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 2.1 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。

ファイル名	説明
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。

- 「12.1.2　ポートタイプから SEI 名へのマッピング」
- 「12.1.3　オペレーションからメソッド名へのマッピング」
- 「12.1.4　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）」
- 「12.1.5　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¥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¥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=<Cosminexus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=<Cosminexusのインストールディレクトリ>
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

<Cosminexus のインストールディレクトリ > の部分は，Cosminexus 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「Cosminexus 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「15.2　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「Cosminexus 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「15.3　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「Cosminexus 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**）**

9.2 **開発例の流れ（プロバイダ起点・**SAAJ**）**

9.3 Web **サービスの開発例（プロバイダ起点・**SAAJ**）**

9.4 **デプロイと開始の例（プロバイダ起点・**SAAJ**）**

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**）**

9.6 Web **サービスの実行例（プロバイダ起点・**SAAJ**）**

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。

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	「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。
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**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。

● Windows(x86) の場合

```
> set HNTRLIB2_HOME=<HNTRLib2インストールディレクトリ>
> cd c:¥temp¥jaxws¥works¥dispatch_provider¥server¥
> mkdir WEB-INF¥classes¥
> apt -J-Dcosminexus.home="%COSMINEXUS_HOME%" -cp
"%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥client¥lib
¥j2ee-javax.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%
¥jaxp¥lib¥csmjaxp.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar;%HNTRLIB2_H
OME%¥classes¥hntrlib2j.jar;%HNTRLIB2_HOME%¥classes¥hntrlibMj.jar" -d
WEB-INF¥classes¥ -s src src¥com¥sample¥AddNumbersImpl.java
src¥com¥sample¥AddNumbersFault.java
```

● Windows(x64) の場合

```
> set HNTRLIB2_HOME=<HNTRLib2インストールディレクトリ>
> cd c:¥temp¥jaxws¥works¥dispatch_provider¥server¥
> mkdir WEB-INF¥classes¥
> apt -factory com.cosminexus.istack.ws.AnnotationProcessorFactoryImpl
-J-Dcosminexus.home="%COSMINEXUS_HOME%" -cp
"%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥client¥lib
¥j2ee-javax.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%
¥jaxp¥lib¥csmjaxp.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar;%HNTRLIB2_H
OME%¥classes¥hntrlib2j64.jar;%HNTRLIB2_HOME%¥classes¥hntrlibMj64.jar" -d
WEB-INF¥classes¥ -s src src¥com¥sample¥AddNumbersImpl.java
src¥com¥sample¥AddNumbersFault.java
```

<HNTRLib2 インストールディレクトリ > の部分には，次のコマンドの実行結果を指定します。

● Windows(x86) の場合

```
> "%COSMINEXUS_HOME%¥common¥bin¥gethntr2conf.exe" HNTR2INSTDIR
```

● Windows(x64) の場合

```
> "%COSMINEXUS_HOME%¥common¥bin¥gethntr2conf64.exe" HNTR2INSTDIR
```

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="2.5" 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_2_5.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>
```

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="5" 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_5.xsd">

  <description>Sample application &amp;quot;dispatch_provider&amp;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>
```

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「Cosminexus 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「5.2.4 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「Cosminexus 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「Cosminexus 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「Cosminexus 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「Cosminexus 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¥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¥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=<Cosminexus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=<Cosminexusインストールディレクトリ>
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

<Cosminexus のインストールディレクトリ > の部分は，Cosminexus 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「Cosminexus 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「15.2　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「Cosminexus 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「15.3　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「Cosminexus 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，Web サービスを開発，運用するときの各種設定と，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

10.2 JAX-WS エンジンの動作

10.3 cosminexus-jaxws.xml によるカスタマイズ

10.4 フォルトと例外の処理

10.5 インタフェースの透過性

10.6 メタデータの発行

10.7 Web サービスの情報表示

10.8 使用できる HTTP メソッド

10.9 Web サービスの初期化と破棄

10.10 プロキシサーバ経由の接続

10.11 SSL プロトコルによる接続

10.12 ベーシック認証による接続

10.13 SOAP のバージョン選択

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

10.15 HTTP ステータスコード

10.16 HTTP ヘッダ

10.17 HTTP リクエストボディの gzip 圧縮

10.18 HTTP レスポンス圧縮機能との連携

10.19 EJB の Web サービス呼び出し

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，「15.5.1　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。

<Cosminexus 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	稼働ログのログ出力レベル	com.cosminexus.jaxws.logger.runtime.message.level	稼働ログのログ出力レベルを指定します。 ERROR，WARN，INFO，DEBUG，NONE を指定します。各指定値に対応した出力内容については，「29.3.3　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）で指定します。	16777216	―
7	例外ログのログ出力レベル	com.cosminexus.jaxws.logger.runtime.exception.level	例外ログのログ出力レベルを指定します。ERROR，WARN，INFO，DEBUG，NONE を指定します。各指定値に対応した出力内容については，「29.3.3　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 の数字で指定します。	2	―

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
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）で指定します。	16777216	―
10	通信ログの出力（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通信ログを出力するかどうかを指定します。 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。	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
11	通信ログの出力（Web サービス側）	com.cosminexus.jaxws.logger.runtime.transport.server_dump	Web サービス側で通信ログを出力するかどうかを指定します。 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。 **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。	ERROR_HEADER	―
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 の数字で指定します。	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）で指定します。	16777216	―

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
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 5.0 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 5.0 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	稼働ログのログ出力レベル（cjwsimport）	com.cosminexus.jaxws.logger.cjwsimport.message.level	cjwsimport コマンドの稼働ログのログ出力レベルを指定します。ERROR，WARN，INFO，DEBUG，NONE を指定します。各指定値に対応した出力内容については，「29.3.3　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 の数字で指定します。	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）で指定します。	2097152	―
18	例外ログのログ出力レベル（cjwsimport）	com.cosminexus.jaxws.logger.cjwsimport.exception.level	cjwsimport コマンドの例外ログのログ出力レベルを指定します。ERROR，WARN，INFO，DEBUG，NONE を指定します。各指定値に対応した出力内容については，「29.3.3　ログの重要度と出力条件」を参照してください。	INFO	―
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 の数字で指定します。	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）で指定します。	16777216	―

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
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 の数字で指定します。	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）で指定します。	16777216	―
24	稼働ログのログ出力レベル（apt）	com.cosminexus.jaxws.logger.apt.message.level	apt コマンドの稼働ログのログ出力レベルを指定します。ERROR，WARN，INFO，DEBUG，NONE を指定します。各指定値に対応した出力内容については，「29.3.3　ログの重要度と出力条件」を参照してください。	INFO	―
25	稼働ログの面数（apt）	com.cosminexus.jaxws.logger.apt.message.file_num	apt コマンドの稼働ログの面数を指定します。1 ～ 16 の数字で指定します。	2	―
26	稼働ログの容量（apt）	com.cosminexus.jaxws.logger.apt.message.file_size	apt 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）で指定します。	2097152	―
27	例外ログのログ出力レベル（apt）	com.cosminexus.jaxws.logger.apt.exception.level	apt コマンドの例外ログのログ出力レベルを指定します。ERROR，WARN，INFO，DEBUG，NONE を指定します。各指定値に対応した出力内容については，「29.3.3　ログの重要度と出力条件」を参照してください。	INFO	―
28	例外ログの面数（apt）	com.cosminexus.jaxws.logger.apt.exception.file_num	apt コマンドの例外ログの面数を指定します。1 ～ 16 の数字で指定します。	2	―

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
29	例外ログの容量（apt）	com.cosminexus.jaxws.logger.apt.exception.file_size	apt 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）で指定します。	16777216	―
30	保守ログの出力（apt）	com.cosminexus.jaxws.logger.apt.maintenance.level	apt 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	―
31	保守ログの面数（apt）	com.cosminexus.jaxws.logger.apt.maintenance.file_num	apt コマンドの保守ログの面数を指定します。1 ～ 16 の数字で指定します。	2	―
32	保守ログの容量（apt）	com.cosminexus.jaxws.logger.apt.maintenance.file_size	apt 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）で指定します。	16777216	―
33	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	―

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
34	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）で指定します。 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	○※
35	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）で指定します。 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	○※
36	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	○※

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
37	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	○※
38	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。 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	○※
39	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 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。	false	―
40	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	―

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
41	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	—
42	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	—
43	ログの出力先ディレクトリ	com.cosminexus.jaxws.tool.log.directory	cjwsimport コマンド，apt コマンド，および cjwsgen 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。	<Cosminexus のインストール先 >/jaxws/logs	—
44	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。	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
45	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	―
46	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	―

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
47	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	―
48	稼働ログのログ出力レベル（cjwsgen）	com.cosminexus.jaxws.logger.cjwsgen.message.level	cjwsgen コマンドの稼働ログのログ出力レベルを指定します。ERROR，WARN，INFO，DEBUG，NONE を指定します。各指定値に対応した出力内容については，「29.3.3　ログの重要度と出力条件」を参照してください。	INFO	―
49	稼働ログの面数（cjwsgen）	com.cosminexus.jaxws.logger.cjwsgen.message.file_num	cjwsgen コマンドの稼働ログの面数を指定します。1 ～ 16 の数字で指定します。	2	―
50	稼働ログの容量（cjwsgen）	com.cosminexus.jaxws.logger.cjwsgen.message.file_size	cjwsgen 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）で指定します。	2097152	―
51	例外ログのログ出力レベル（cjwsgen）	com.cosminexus.jaxws.logger.cjwsgen.exception.level	cjwsgen コマンドの例外ログのログ出力レベルを指定します。ERROR，WARN，INFO，DEBUG，NONE を指定します。各指定値に対応した出力内容については，「29.3.3　ログの重要度と出力条件」を参照してください。	INFO	―
52	例外ログの面数（cjwsgen）	com.cosminexus.jaxws.logger.cjwsgen.exception.file_num	cjwsgen コマンドの例外ログの面数を指定します。1 ～ 16 の数字で指定します。	2	―

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
53	例外ログの容量（cjwsgen）	com.cosminexus.jaxws.logger.cjwsgen.exception.file_size	cjwsgen 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）で指定します。	16777216	―
54	保守ログの出力（cjwsgen）	com.cosminexus.jaxws.logger.cjwsgen.maintenance.level	cjwsgen コマンドの保守ログを出力するかどうか指定します。 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	―
55	保守ログの面数（cjwsgen）	com.cosminexus.jaxws.logger.cjwsgen.maintenance.file_num	cjwsgen コマンドの保守ログの面数を指定します。1 ～ 16 の数字で指定します。	2	―
56	保守ログの容量（cjwsgen）	com.cosminexus.jaxws.logger.cjwsgen.maintenance.file_size	cjwsgen 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）で指定します。	16777216	―
57	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	―

（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。

(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

Cosminexus の JAX-WS エンジンの動作，および Cosminexus 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，GET HTTP 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

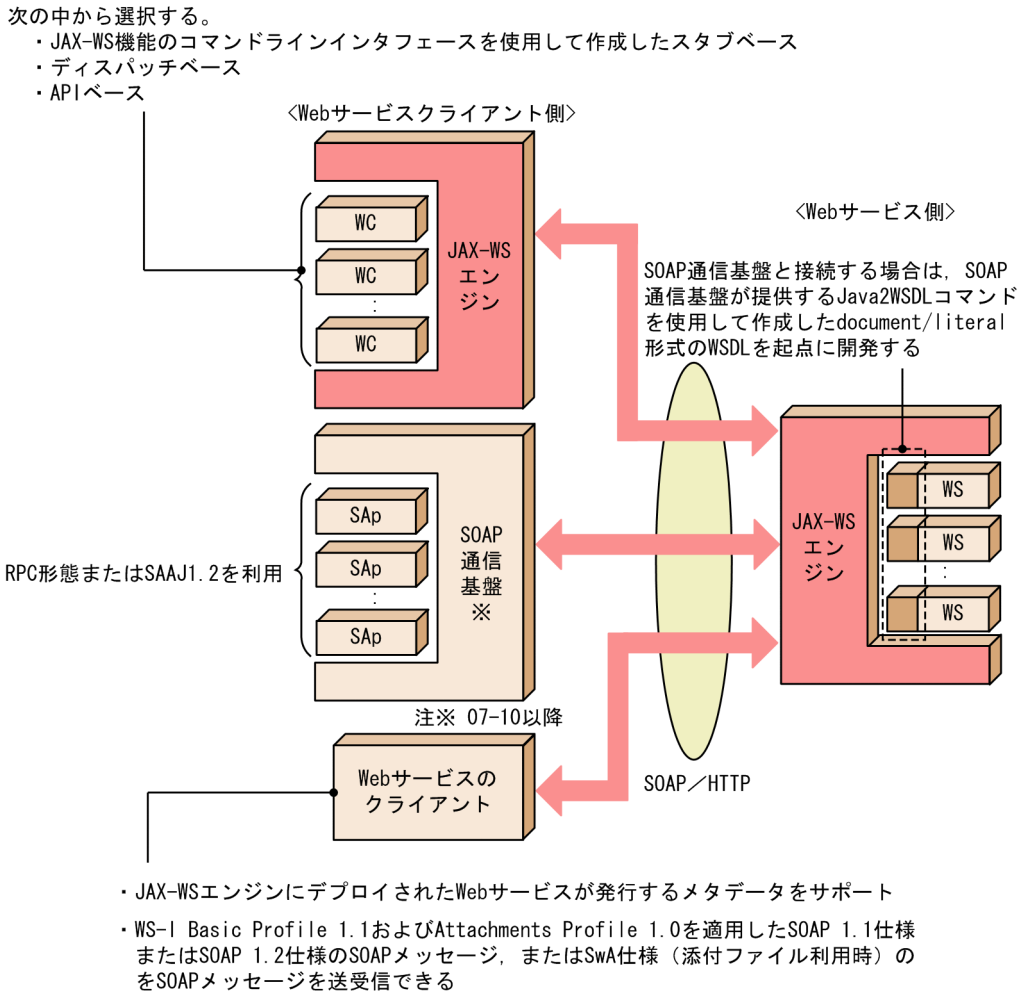
Web サービス側の JAX-WS エンジンと Web サービスクライアントの関係について説明

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。

● Cosminexus の JAX-WS 機能で動作する Web サービスクライアント
Cosminexus の JAX-WS 機能が提供しているコマンドで開発し，Cosminexus 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 Cosminexus 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
Cosminexus 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）

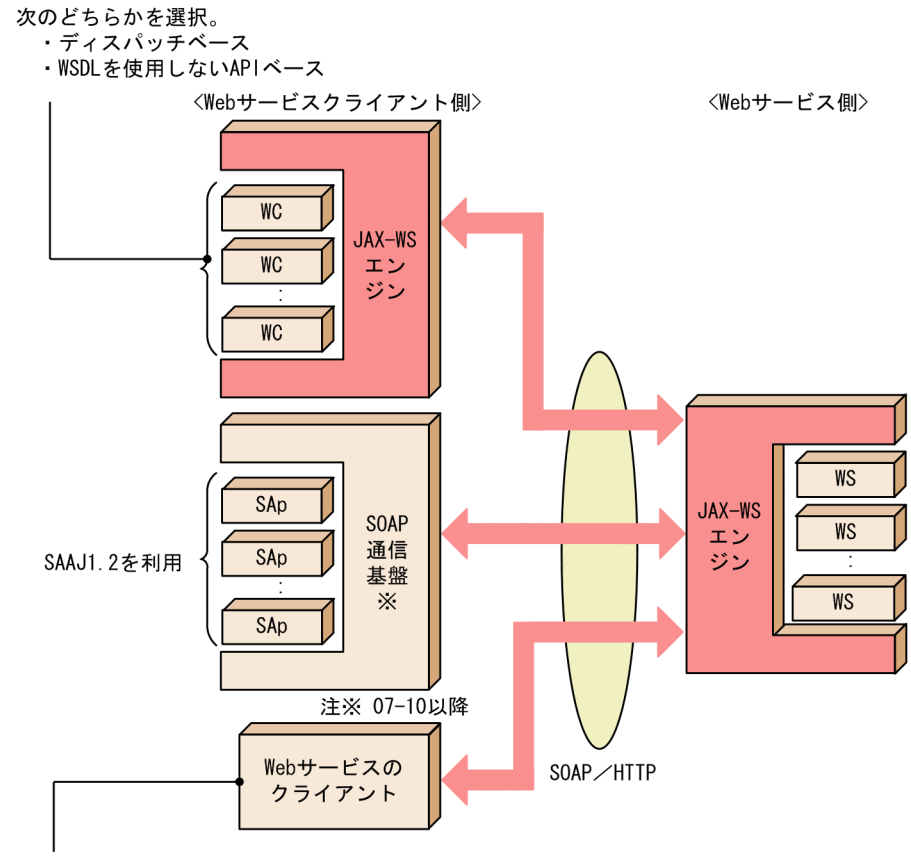
WSDL のサポート範囲については，「14.1　WSDL 1.1 仕様のサポート範囲」を参照してください。

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）



WS-I Basic Profile 1.1およびAttachments Profile 1.0を適用したSOAP 1.1 仕様またはSOAP 1.2仕様のSOAPメッセージ，またはSwA仕様（添付ファイル利用時）のSOAPメッセージを送受信できる

（凡例）
WC：Webサービスクライアント
SAp：SOAPアプリケーション
WS：プロバイダ実装クラス

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。

- Cosminexus の JAX-WS 機能で動作する Web サービスクライアント
  Cosminexus の JAX-WS 機能が提供している API で開発し，Cosminexus 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 Cosminexus 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。
- Cosminexus 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 Cosminexus 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 2.1 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。

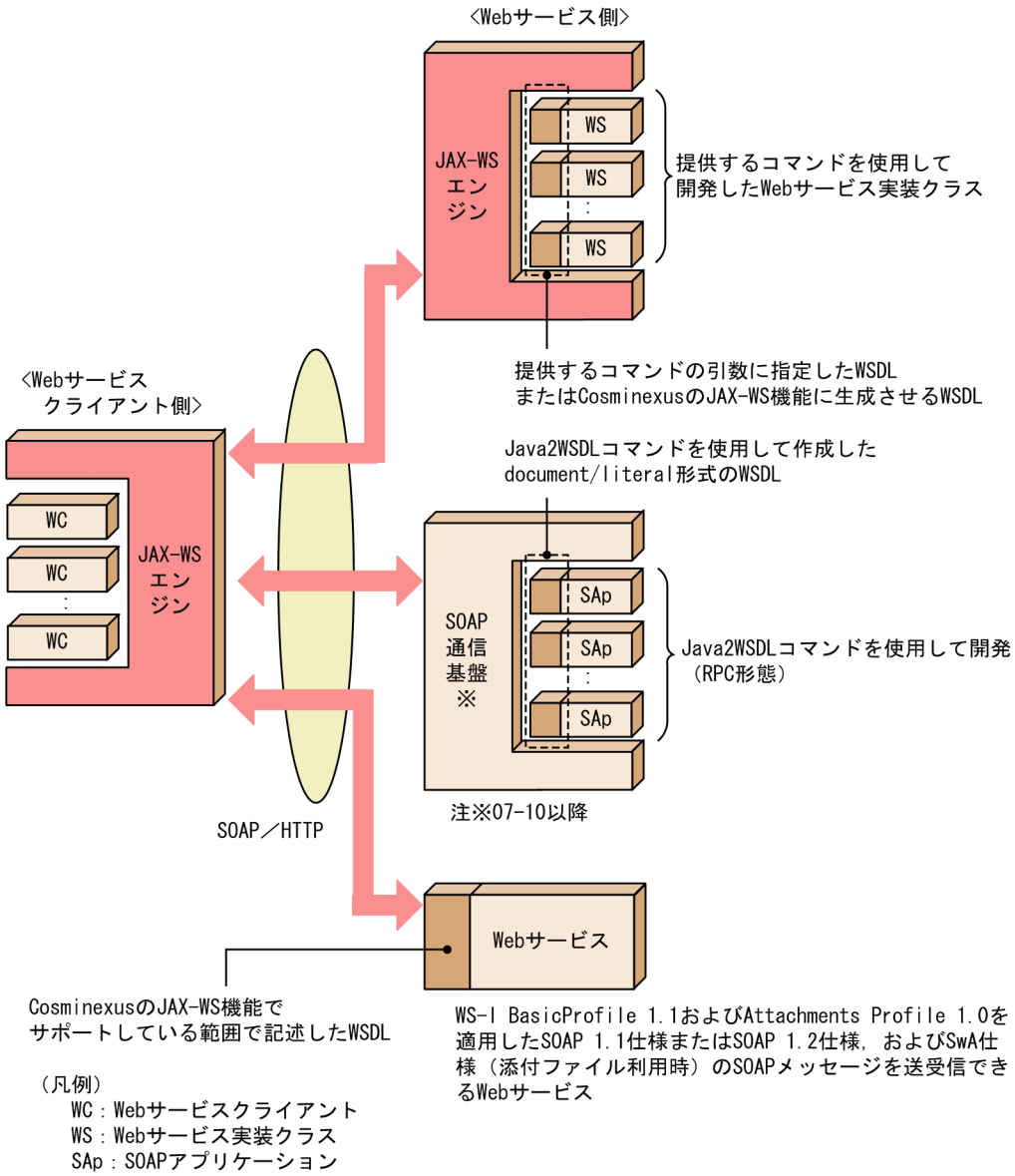
(b) Web サービスクライアント側の JAX-WS エンジンのサポート範囲

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。

● Cosminexus 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

Cosminexus 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 Cosminexus 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
Cosminexus 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）

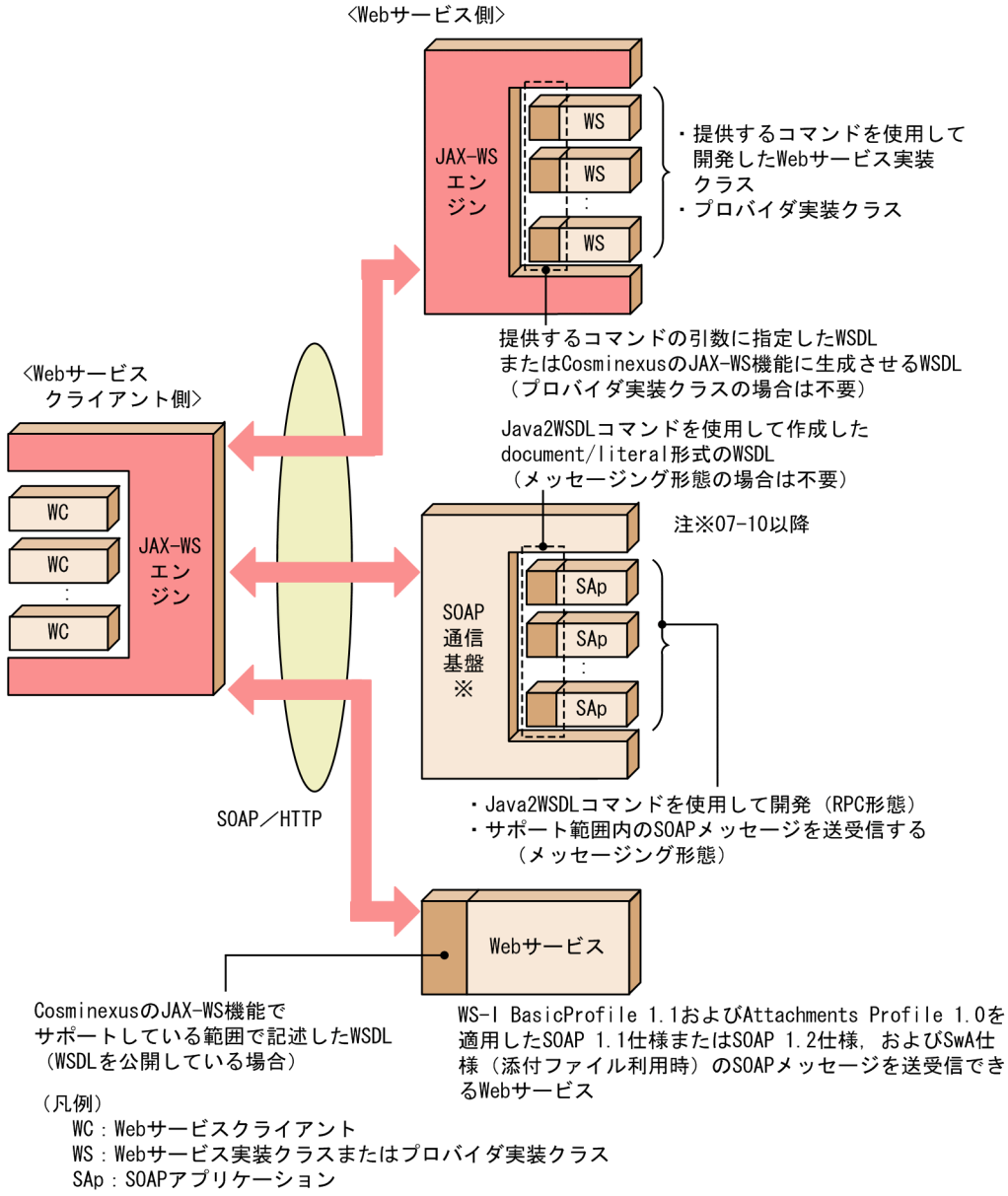
WSDL のサポート範囲については，「14.1　WSDL 1.1 仕様のサポート範囲」を参照してください。

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。

● Cosminexus 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

Cosminexus 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。Cosminexus 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。

● Cosminexus 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
Cosminexus 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 Cosminexus 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
Cosminexus 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，「14.1　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 Cosminexus 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 Cosminexus 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

項番	条件
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 Cosminexus の JAX-WS 機能を使用する場合

Cosminexus 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 Cosminexus 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）

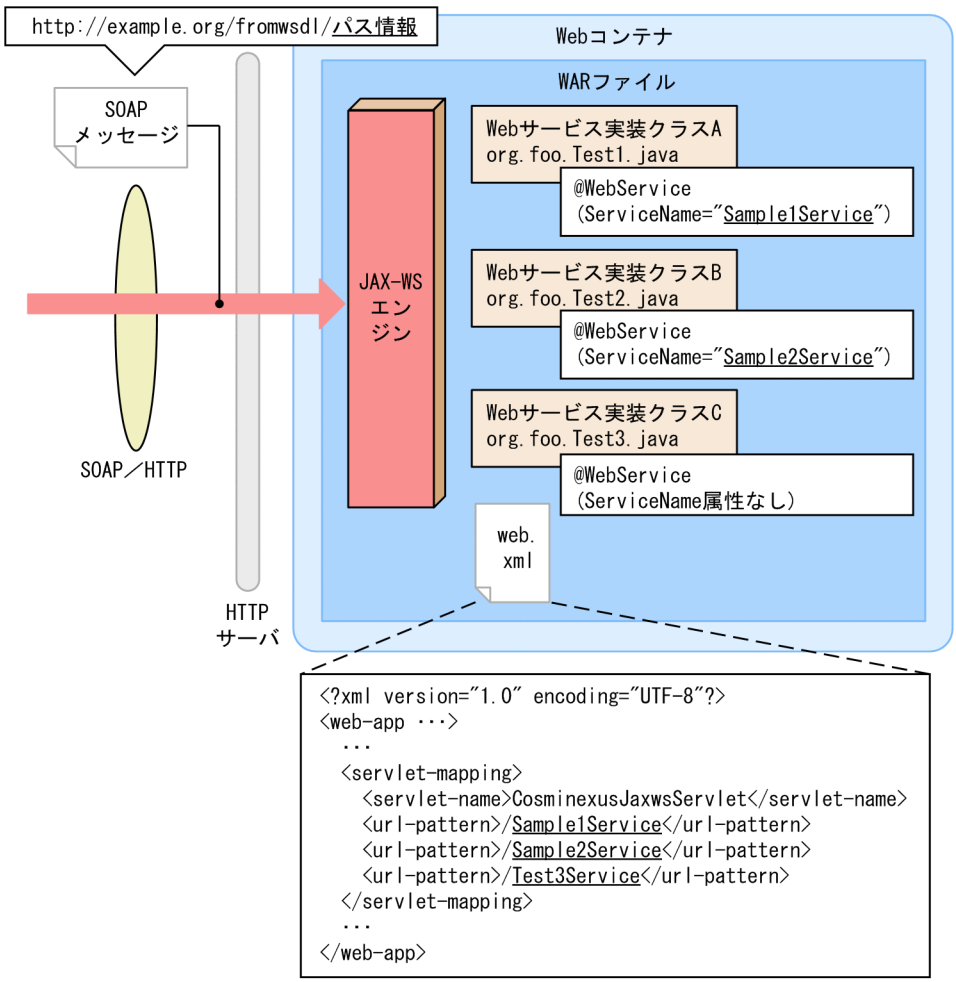
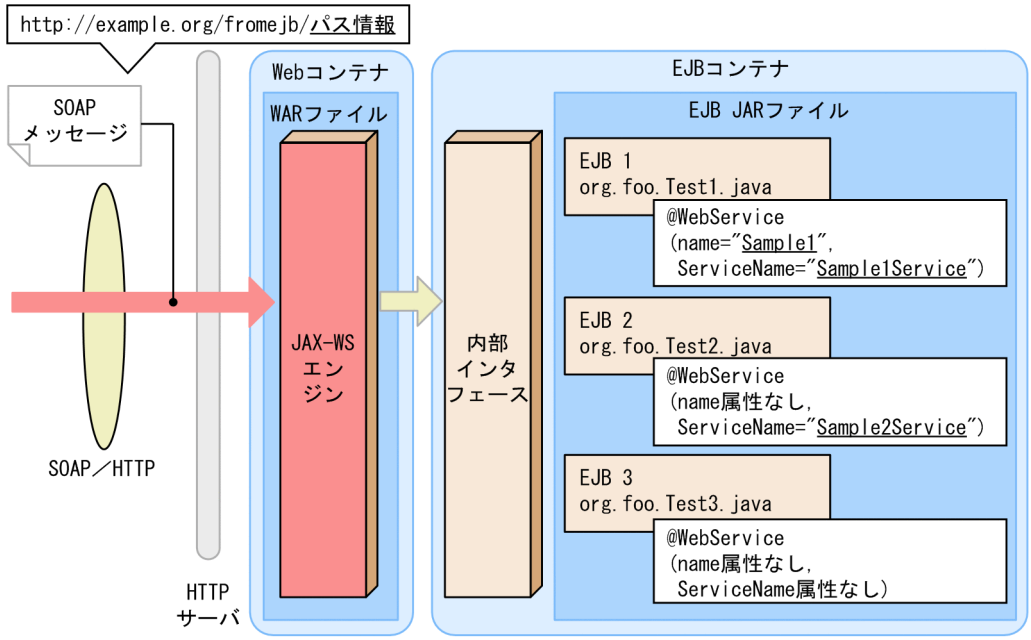


図 10-6　Web サービス実装クラスのディスカバリ（EJB の Web サービス）



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 アノテーションの serviceName 属性は省略できるので，省略されている場合は，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" を付加した文字列が，serviceName 属性の値と見なされます。また，javax.jws.WebService アノテーションおよび javax.xml.ws.WebServiceProvider アノテーションの name 属性は省略できるので，省略されている場合は，JSR-181 仕様に基づき，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，「18.4　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。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，「18.4　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。
└	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="2.5" 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_2_5.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>/test1</url-pattern>
    <url-pattern>/test2</url-pattern>
    <url-pattern>/test3</url-pattern>
  </servlet-mapping>
</web-app>
```

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>
```

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 2.1 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。

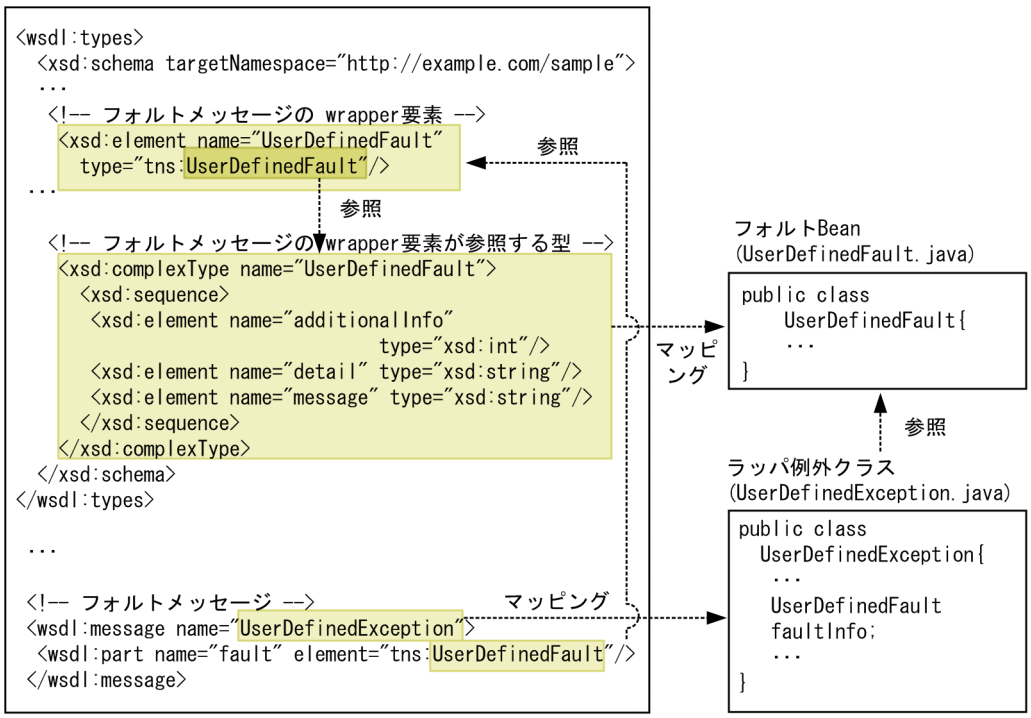
10.4.1　Web サービス側のフォルトおよび例外の処理

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 2.1 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，「12.1.7　フォルトから例外クラスへのマッピング」，および「13.1.7　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 soapenv12:Receiver 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" ?>
<S:Envelope xmlns:S= "http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
      <faultcode>ns2: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>
    </ns2: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" ?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Body>
    <ns3:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
      <ns3:Code>
        <ns3:Value>ns3:Receiver</ns3:Value>
      </ns3:Code>
      <ns3:Reason>
        <ns3:Text xml:lang="ja">Something happens.</ns3:Text>
      </ns3:Reason>
      <ns3:Detail>
        <env:UserDefinedFault xmlns:env="http://example.com/sample">
          <additionalInfo>257</additionalInfo>
          <detail>Failed by some reason.</detail>
          <message>Contact your administrator.</message>
        </env:UserDefinedFault>
      </ns3:Detail>
    </ns3: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 2.1 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 soapenv12:Receiver 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。
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" ?>
<S:Envelope xmlns:S= "http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
      <faultcode>ns2:Server</faultcode>
      <faultstring>Something illegal.</faultstring>
    </ns2: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" ?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Body>
    <ns3:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
      <ns3:Code>
        <ns3:Value>ns3:Receiver</ns3:Value>
      </ns3:Code>
      <ns3:Reason>
        <ns3:Text xml:lang="ja">Something illegal.</ns3:Text>
      </ns3:Reason>
    </ns3: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 2.1 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 soapenv12:Receiver 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。
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" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
      <faultcode>ns2:Server</faultcode>
      <faultstring>Web Service Exception.</faultstring>
    </ns2: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" ?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Body>
    <ns3:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/
               xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
      <ns3:Code>
        <ns3:Value>ns3:Receiver</ns3:Value>
      </ns3:Code>
      <ns3:Reason>
        <ns3:Text xml:lang="ja">Something illegal.</ns3:Text>
      </ns3:Reason>
    </ns3: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 2.1 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 soapenv12:Sender 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。
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" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
               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>
    </ns2: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" ?>
<S:Envelope xmlns:S= "http://www.w3.org/2003/05/soap-envelope">
  <S:Body>
    <ns3:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
      <ns3:Code>
        <ns3:Value>ns3:Sender</ns3:Value>
        <ns3:Subcode>
          <ns3:Value xmlns:ns0="http://sample.org">ns0:UserDefined</
ns3:Value>
        </ns3:Subcode>
      </ns3:Code>
      <ns3:Reason>
        <ns3:Text xml:lang="ja">SOAPFaultException happens.</ns3:Text>
      </ns3:Reason>
      <ns3:Role>http://example.com/sample</ns3:Role>
      <ns3:Detail>
        <env:Detail xmlns:env="http://www.w3.org/2003/05/soap-envelope">
          <detailTest>TEST.</detailTest>
        </env:Detail>
      </ns3:Detail>
    </ns3: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 2.1 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 2.1 仕様に従っています）。

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，Cosminexus 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 2.1 仕様で定められた機能ではないため，Cosminexus 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）も含まれる可能性があります。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，detail 要素，または soapenv12:Detail 要素の子要素として，Cosminexus 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 例外の情報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" ?>
<S:Envelope xmlns:S= "http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
      <faultcode>ns2: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.captureStackTrace property
to false">
          <message>Something illegal</message>
          <ns2:stackTrace>
            <ns2:frame class="com.example.sample.TestJaxWsImpl"
file="TestJaxWsImpl.java" line="32" method="jaxWsTest1"/>
            <ns2:frame class="sun.reflect.NativeMethodAccessorImpl"
file="NativeMethodAccessorImpl.java" line="native" method="invoke0"/>
            <ns2:frame class="sun.reflect.NativeMethodAccessorImpl"
file="NativeMethodAccessorImpl.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" line="32" method="jaxWsTest1"/>
              <ns2:frame class="sun.reflect.NativeMethodAccessorImpl"
file="NativeMethodAccessorImpl.java" line="native" method="invoke0"/>
  ...
             <ns2:frame class="sun.reflect.DelegatingMethodAccessorImpl"
file="DelegatingMethodAccessorImpl.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"
file="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>
    </ns2: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" ?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope">
  <S:Body>
    <ns3:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
      <ns3:Code>
        <ns3:Value>ns3:Receiver</ns3:Value>
      </ns3:Code>
      <ns3:Reason>
        <ns3:Text xml:lang="ja">Something illegal.</ns3:Text>
      </ns3:Reason>
      <ns3:Detail>
        <env:Detail xmlns:env="http://www.w3.org/2003/05/soap-envelope">
          <detailTest>TEST.</detailTest>
        </env:Detail>
        <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.captureStackTrace property
to false">
           ...
           （SOAP 1.1仕様のSOAPフォルトと同じ）
           ...
        </ns2:exception>
      </ns3:Detail>
    </ns3: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例外（Web サービス側で発生した例外）が設定されます。

Web サービスクライアントは，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 Cosminexus 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。

(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。

(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。

(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。

(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。

(a) 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

(b) 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

(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	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。	○	○
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。

（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 0850 (2010.01.01 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。

（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。

（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。

（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。

10.7　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。

(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

(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。

(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
- 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，「29.3.1　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「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「5.2　アプリケーションサーバのログの内容」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「Cosminexus アプリケーションサーバ メッセージ 2」の「3.　KDJE30000 から KDJE39999 までのメッセージ」を参照してください。

(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。

(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 Cosminexus 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。	△	×

項番	プロパティ	指定内容	非SSLの場合	SSLの場合
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	△	×
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。	×	△

項番	プロパティ	指定内容	非 SSL の場合	SSL の場合
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

Cosminexus 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 5.0 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。

(2) プロパティの設定方法

Cosminexus 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.hitachi.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。

(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 5.0 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 5 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;
  }
} );
```

（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

Cosminexus 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，Cosminexus 以外の HTTP 接続にも影響します。Cosminexus 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，Cosminexus 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。

(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	必須
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。	△※

（凡例）

△：プロパティの指定が任意であることを示します。

注※

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。

(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。

10.12　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。

(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 10-16　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。

(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,「15.5　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

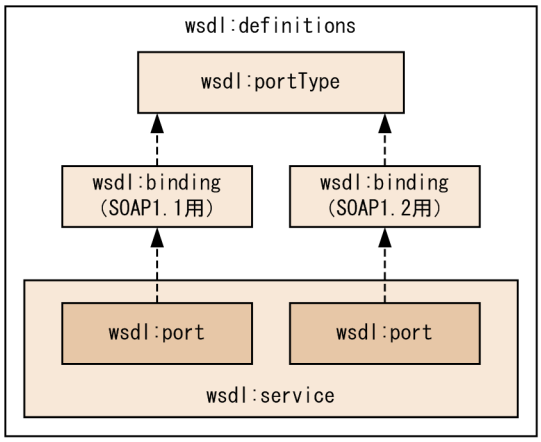
SOAP 1.1 仕様または SOAP 1.2 仕様のどちらのメッセージを送受信するのかを，WSDL に記述するバインディングで決定します。

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



（凡例）
----------► ：バインド

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=<Cosminexus のインストールディレクトリ >/jaxws/lib/cjjaxws.jar
- add.jvm.arg=-Dcosminexus.home=<Cosminexus 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。

<Cosminexus 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。

<Cosminexus 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。

<Cosminexus 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，「29.3.2(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 10-17　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	Web サービスの呼び出しが正常終了した場合。
2	202 Accepted	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。

（1） 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 Cosminexus 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。

(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
- Timer Service

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

(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。

11 コマンド

Web サービスに必要な SEI や JavaBean クラスは，cjwsimport コマンドおよび apt コマンドで生成できます。また，cjwsgen 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，apt コマンド，および cjwsgen コマンドの使用方法について説明します。

11.1　cjwsimport コマンド
11.2　apt コマンド
11.3　cjwsgen コマンド
11.4　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

11.1 cjwsimport コマンド

cjwsimport コマンドは，JAX-WS 2.1 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，RFC2396 仕様で規定されている文字，および

RFC2396 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 RFC2396 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。RFC2396 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 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 11-1 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 11-3 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。

オプション	設定項目	指定内容
-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。 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，「11.1(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。

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/

次に，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）。
- 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。
- 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 11-2　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 11-3 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（.wsdl）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。

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 11-4 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。	○	○

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
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 2.1 仕様のファクトリクラスです。	○	○
6	JAXB 2.1 仕様のそのほかのクラス	JAXB 2.1 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 クラス	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，Cosminexus 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，「11.1(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，設定したログ出力レベル（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。

11.2 apt コマンド

apt コマンドは，アノテーションを解釈して，追加の Java コードを生成し，基の Java コードを含めてコンパイルする JDK のコマンドです。Web サービスの開発では，SEI を起点とした開発をする場合に使用します。Web サービス実装クラスに記述されたアノテーション（JAX-WS 2.1 仕様に従って記述）を解釈し，必要な JavaBean クラスを追加して生成します。

apt コマンドの形式，引数，およびオプションについては，JDK のドキュメントを参照してください。ここでは，JDK のドキュメントで規定されていない内容や，コマンド実行時の注意事項について説明します。

(1) 引数に指定する Web サービス実装クラスおよび SEI

apt コマンドの引数には，Web サービス実装クラスおよび SEI（SEI を参照している場合）をそれぞれ 1 個指定します。Web サービス実装クラスを 2 個以上指定した場合は，ログにエラーメッセージが出力され，エラーメッセージが apt コマンドに返されます（KDJW61002-E）。ただし，SEI だけ指定した場合など Web サービス実装クラスがない場合は，標準エラー出力とログに警告メッセージが出力され，処理が続行されます（KDJW61001-W）。この場合，JavaBean クラスは生成されません。

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，「13.1　Java から WSDL へのデフォルトマッピング」および「13.2　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。

apt コマンドでは，javax.ejb.Stateless アノテーションの付いた EJB の Web サービス実装クラスを引数に指定した場合，次の警告が出力されます。

```
警告：プロセッサなしの注釈タイプです：[javax.ejb.Stateless]
```

(2) 必須オプション

apt コマンドを実行するときは，-classpath オプションおよび -J-Dcosminexus.home オプションの指定が必須です。それぞれの指定値を次に示します。

-classpath オプション

- <Cosminexus のインストールディレクトリ >/jaxws/lib/cjjaxws.jar
- <Cosminexus のインストールディレクトリ >/jaxp/lib/csmjaxb.jar
- <Cosminexus のインストールディレクトリ >/jaxp/lib/csmjaxp.jar
- <Cosminexus のインストールディレクトリ >/jaxp/lib/csmstax.jar
- <Cosminexus のインストールディレクトリ >/CC/client/lib/j2ee-javax.jar
- <Cosminexus のインストールディレクトリ >/CC/client/lib/HiEJBClientStatic.jar

Windows(x86) の場合

- < HNTRLib2 インストールディレクトリ >※ /classes/hntrlib2j.jar
- < HNTRLib2 インストールディレクトリ >※ /classes/hntrlibMj.jar

Windows(x64) の場合

- < HNTRLib2 インストールディレクトリ >※ /classes/hntrlib2j64.jar
- < HNTRLib2 インストールディレクトリ >※ /classes/hntrlibMj64.jar

-J-Dcosminexus.home

<Cosminexus のインストールディレクトリ >

注※

<HNTRLib2 インストールディレクトリ > の部分は次に示すコマンドの実行結果を指定します。

● Windows(x86) の場合

```
> "%COSMINEXUS_HOME%¥common¥bin¥gethntr2conf.exe" HNTR2INSTDIR
```

● Windows(x64) の場合

```
> "%COSMINEXUS_HOME%¥common¥bin¥gethntr2conf64.exe" HNTR2INSTDIR
```

(3) 生成されるファイル

apt 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 11-5 apt コマンドの生成ファイル一覧

項番	Java コード	内容
1	リクエスト bean クラス	要求メッセージの JavaBean クラスです。wrapper スタイル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 JavaBean クラスです。wrapper スタイル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 JavaBean クラスです。指定した Java コードでラッパ例外クラスが定義されている場合で，かつフォルト bean がない場合に出力されます。

ファイル生成時のディレクトリの作成

apt 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
  apt -d ./output -s ./output/ -sourcepath . com/example/test.java

- 出力先
  JavaBean クラス以外のソースファイル，およびコンパイル済みクラスファイルは次のディレクトリに出力されます。
  ./output/com/example/
  ただし，コマンドに指定した Java コードに 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そのコンパイル済みクラスファイルは SEI のパッケージの 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/

Cosminexus の JAX-WS 機能が提供するアノテーションプロセッサでは，Java コードの出力先ディレクトリがない場合やディレクトリではない場合などで不正な場合，ログにエラーメッセージが出力され，apt コマンドにエラーが通知されます（KDJW61003-E）。
Java コードの出力先は，apt コマンドの引数に指定した -s オプションの値となります。-s オプションが指定されていない場合は，-d オプションに指定された値となります。-s オプション，-d オプションのどちらも指定されていない場合，カレントディレクトリが出力先となります。

Javadoc のヘッダ情報の出力

生成されるファイルでは，Cosminexus に関する情報が，Javadoc としてヘッダ情報に出力されます。

(4) 続行できる軽微なエラーが検出された場合の動作

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，設定したログ出力レベル（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。

(5) 異常終了時の対処

apt コマンドの実行時にエラーが発生した場合，エラーメッセージが出力され，処理が終了されます。

エラーメッセージが出力された場合，出力されたエラーの要因を取り除き，apt コマンドを再実行してください。表示されたエラーの要因を一つずつ取り除き，正常終了するまで，apt コマンドを繰り返し実行してください。すでにファイルが生成されている場合は，apt コマンドを実行する前に生成されたファイルを削除してください。

なお，設定したログ出力レベル（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。

11.3 cjwsgen コマンド

cjwsgen 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，cjwsgen 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

cjwsgen コマンドの指定形式を次に示します。

```
cjwsgen [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
  cjwsgen -wsdl -cp . com.example.UserInfoImpl
- 既存の Web サービス（SOAP1.1）から Web サービス（SOAP1.2）の Java コードおよびリソースファイルを生成する場合
  cjwsgen -soap 1.2 -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
  cjwsgen -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
  cjwsgen -portname {http://example.com/}MyServicePort -cp . com.example.UserInfoImpl

cjwsgen コマンド実行時の注意

cjwsgen 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。

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。
- 引数が指定されていない場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW71023-E）。
- 引数にクラスファイル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，処理が終了されます（KDJW71026-E）。
- 引数にサービス実装クラス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，処理が終了されます（KDJW71025-E）。
- 指定されたサービス実装クラス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（KDJW71026-E）。
- 指定されたサービス実装クラスに対するアクセス権がない場合は，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（KDJW71026-E）。
- javax.jws.WebService アノテーションが指定されていないサービス実装クラスを指定した場合は，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（KDJW71029-E）。
- 引数にサービス実装クラスを含む複数のクラスを指定した場合は，警告メッセージが出力され，処理が続行されます（KDJW71027-W）。どのサービス実装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，「13.1 Java から WSDL へのデフォルトマッピング」および「13.2 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

cjwsgen 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 11-6 cjwsgen 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」および「-d ／ -r ／ -soap ／ -servicename ／ -portname ／ -soap12binding ／ -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。
-soap < バージョン >	SOAP バインディングのバージョン	Web サービスが通信時に使用する SOAP バインディングのバージョンを指定します。
-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。
-soap12binding < soap spec >	soap12:binding 要素の transport 属性に設定する URL を示す値	SOAP1.2 の場合に，wsdl:binding 要素の子要素である soap12:binding 要素の transport 属性に，どの URL を設定するかを指定します。
-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。

次に，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（KDJW71001-E）。
- 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。

- 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（KDJW71001-E）。

-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。
- 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（KDJW71002-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，標準出力にヘルプが出力され，処理が終了されます（KDJW71003-E）。
- -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。

-soap オプション指定時の注意事項

-soap オプションの指定値についての注意事項を示します。

- SOAP バインディングのバージョンは，1.1 または 1.2 だけが指定できます。それ以外の値を指定した場合，標準エラー出力とログにエラーメッセージが，標準出力にヘルプが出力され，処理が終了されます（KDJW71004-E）。
- soap オプションと javax.xml.ws.BindingType アノテーションの両方に値が指定されている場合は，-soap オプションの値が優先されます。
- -soap オプションを省略した場合は，javax.xml.ws.BindingType アノテーションの指定値が有効になります。
- javax.xml.ws.BindingType の指定値が SOAP 1.2 over HTTP バインディングの場合，-soap オプションに 1.1 を指定すると，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（KDJW71007-E）。

-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。名前空間を省略，または括弧（{}）で囲んでいない場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW71009-E）。
- 括弧が閉じられていない場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW71008-E）。
- 括弧（{}）の中に値がない場合は，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（KDJW71008-E）。
- 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。http:// または urn: 以外で始まる名前空間 URI 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（KDJW71011-E）。
- -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。相対パスで名前空間 URI 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（KDJW71012-E）。

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。これらの情報を名前空間 URI に記述した場合は，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（KDJW71013-E）。

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 11-7　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:// 日立 .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表示されません）。
2	Java の予約語以外の文字列	http://hitachi.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://1hitachi.com	

サービス名

- サービス名を省略した場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW71014-E）。
- 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。名前空間を省略，または括弧（{}）で囲んでいない場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW71016-E）。
- 括弧が閉じられていない場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW71015-E）。
- 括弧（{}）の中に値がない場合は，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（KDJW71015-E）。
- 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。WSDL ファイルの service 要素と異なる名前空間 URI 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（KDJW71022-E）。
- 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。http:// または urn: 以外で始まる名前空間 URI 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（KDJW71018-E）。
- -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。相対パスで名前空間 URI 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（KDJW71019-E）。

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。これらの情報を名前空間 URI に記述した場合は，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（KDJW71020-E）。

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 11-8 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:// 日立 .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表示されません）。
2	Java の予約語以外の文字列	http://hitachi.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://1hitachi.com	

ポート名

- ポート名を省略した場合は，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW71021-E）。
- 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。

-soap12binding の指定値の注意事項

-soap12binding オプションは "DEFAULT" または "WSI_BP20_TRANSPORT" だけを指定できます。ほかの値を指定すると，標準エラー出力とログにエラーメッセージが出力され，さらに標準出力にヘルプが出力されて，処理が終了されます（KDJW71030-E）。

-soap12binding オプションと，cjwsgen コマンドが生成する WSDL の soap12:binding 要素の transport 属性値の関係を次の表に示します。

表 11-9 -soap12binding オプションと transport 属性値の関係

項番	オプション指定有無	オプション指定値	設定される 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

-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。
  # % & ^
- 指定したクラスパスが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，処理が終了されます（KDJW71026-E）。

-d ／ -s ／ -keep オプションの組み合わせと出力先ディレクトリ

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 11-10 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	-s オプション	-keep オプション		
指定あり	指定あり	指定あり	○	-s 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	○	-s 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	○	カレントディレクトリ

オプションと指定有無			出力有無	出力先
-d オプション	-s オプション	-keep オプション		
		指定なし	×	―

（凡例）

―：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。

○：ソースファイルが出力されることを示します。

×：ソースファイルが出力されないことを示します。

-d ／ -r ／ -soap ／ -servicename ／ -portname ／ -soap12binding ／ -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 11-11　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	-soap，-servicename，-portname，-soap12binding 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		
		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

cjwsgen 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 11-12　cjwsgen 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。
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

cjwsgen 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
  cjwsgen -d ./output -s ./output -keep -cp . com.example.UserInfoImpl
- 出力先
  cjwsgen 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，cjwsgen 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
  cjwsgen -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，Cosminexus 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 11-13 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

cjwsgen 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

（KDJW70001-I）が出力され，JavaBeans の生成と削除，WSDL および XSD の生成が実行されます。それぞれの処理について，次に示します。

JavaBeans の生成処理

JavaBeans 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（KDJW70004-I）。JavaBeans 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（KDJW70005-E）。

WSDL および XSD の生成処理

生成された JavaBeans の内容に対応したリソースファイル（WSDL および XSD）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（KDJW70006-I）。リソースファイル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（KDJW70007-E）。

JavaBeans の削除処理

生成した JavaBeans のソースファイルを削除します。ただし，オプションの指定内容によっては，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，「11.3(2)　オプション一覧」を参照してください。

(6) 終了コード

cjwsgen 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（KDJW70002-I）。

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。
- 処理の途中で続行できないエラーが検出された場合は，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（KDJW70003-E）。異常終了時の対処については，「11.3(7)　異常終了時の対処」を参照してください。
- 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。
- cjwsgen コマンドを実行すると，apt コマンドが呼び出されるため，apt コマンドのエラーメッセージが出力される場合があります。
- 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。

なお，設定したログ出力レベル（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

cjwsgen 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，cjwsgen 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，cjwsgen 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。

11.4 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，Cosminexus の JAX-WS 機能のコマンドラインインタフェースを実行するときの注意事項を説明します。

11.4.1 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。

11.4.2 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，Cosminexus 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。
- Cosminexus 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。

12 WSDL から Java へのマッピング

cjwsimport コマンドを実行すると，JAX-WS 2.1 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。

12.1 WSDL から Java へのデフォルトマッピング

12.2 WSDL から Java へのマッピングのカスタマイズ

12.1　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 12-1　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	12.1.1
2	ポートタイプ	SEI 名	12.1.2
3	オペレーション	メソッド名	12.1.3
4	パート	パラメタおよび戻り値	12.1.4，12.1.5
5	型	パラメタおよび戻り値	12.1.6
6	フォルト	例外クラス	12.1.7
7	バインディング	javax.jws.soap.SOAPBinding アノテーション	12.1.8
8	サービス	javax.jws.WebService アノテーションの serviceName 属性	12.1.9

12.1.1　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。

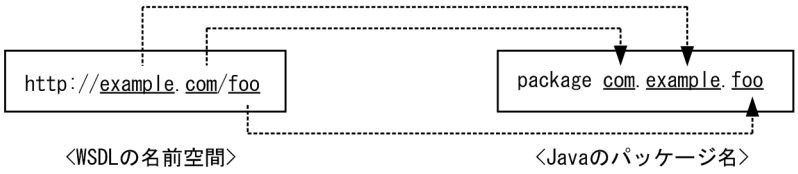
(1) マッピング

WSDL の名前空間とパッケージ名は，JAX-WS 2.1 仕様に従ってマッピングされます。詳細は，JAX-WS 2.1 仕様を参照してください。

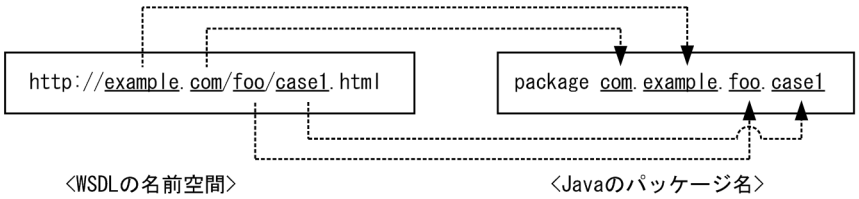
マッピング例を次の図に示します。

図 12-1　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，RFC2396 仕様の規則に従った文字列を記述できます。

表 12-2　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:// 日立 .com	動作は保証されません（エラーメッセージは表示されません）。
2	Java の予約語以外の文字列	http://hitachi.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.hitachi._abstract
3	先頭が数字でない文字列	http://1hitachi.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._1hitachi

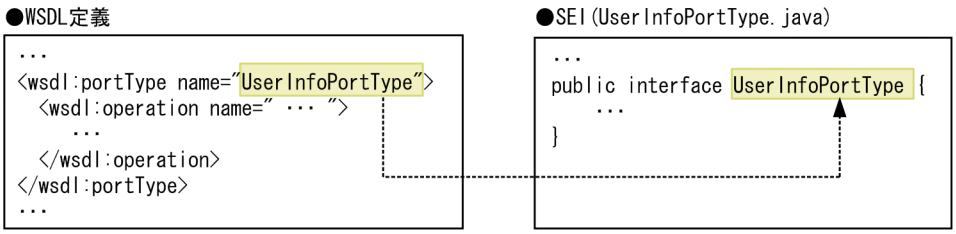
12.1.2　ポートタイプから SEI 名へのマッピング

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 2.1 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 12-2　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 12-3　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	日立 _portType	動作は保証されません（エラーメッセージは表示されません）。
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 12-4　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）。

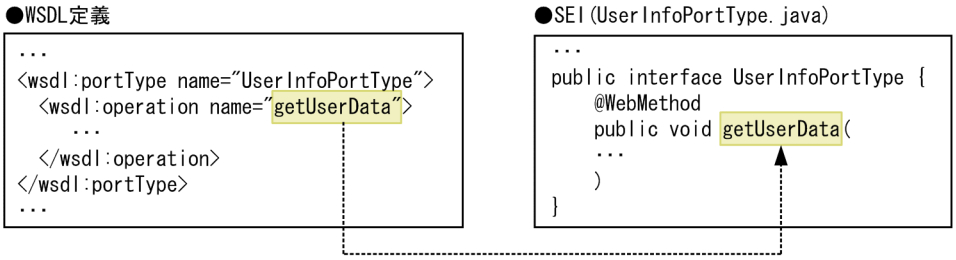
12.1.3　オペレーションからメソッド名へのマッピング

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 2.1 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 12-3　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 12-5　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	日立 _operation	動作は保証されません（エラーメッセージは表示されません）。
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 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 12-6　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 12-7　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	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	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）。

12.1.4　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

wrapper スタイルの場合，WSDL 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 12-4　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（wrapper スタイル）

マッピングするときに，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 12-8 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 12-9 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	日立 _wrapper	動作は保証されません（エラーメッセージは表示されません）。
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 12-10　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。

12.1.5　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）

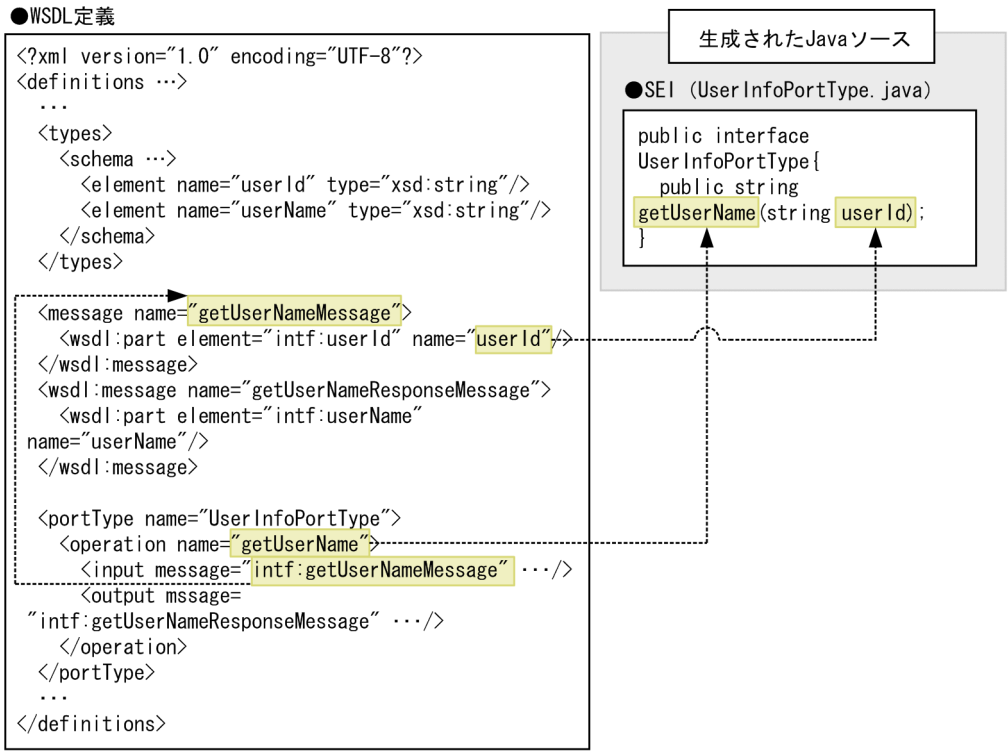
WSDL のメッセージのパート（wsdl:message 要素の wsdl:part 子要素）から Java のメソッドのパラメタおよび戻り値へのマッピングについて説明します。

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

non-wrapper スタイルの場合，WSDL 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 12-5　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（non-wrapper スタイル）



マッピングするときに，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 12-11 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 12-12 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	日立 _part	動作は保証されません（エラーメッセージは表示されません）。
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。

12.1.6　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

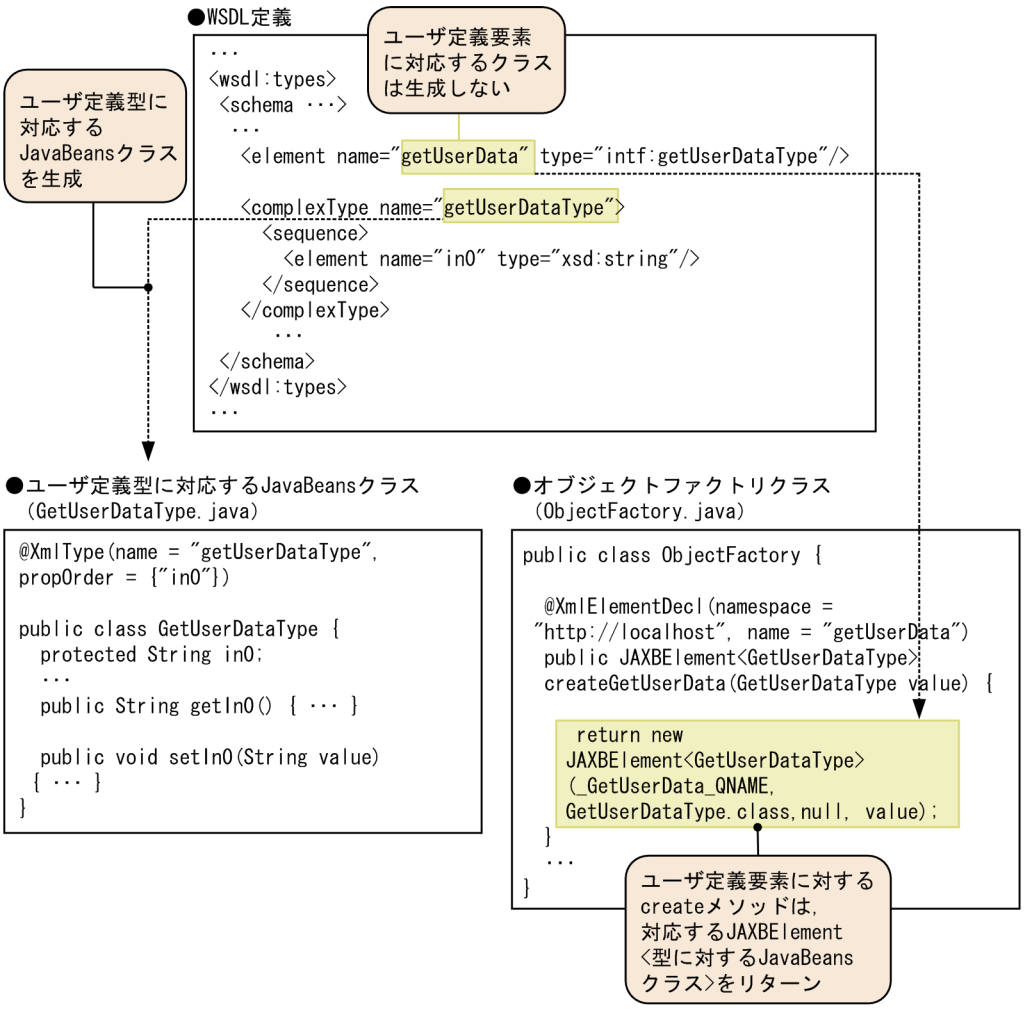
WSDL のスキーマの型と Java 型は，JAXB 2.1 仕様に従ってマッピングされます。

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 12-6　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。

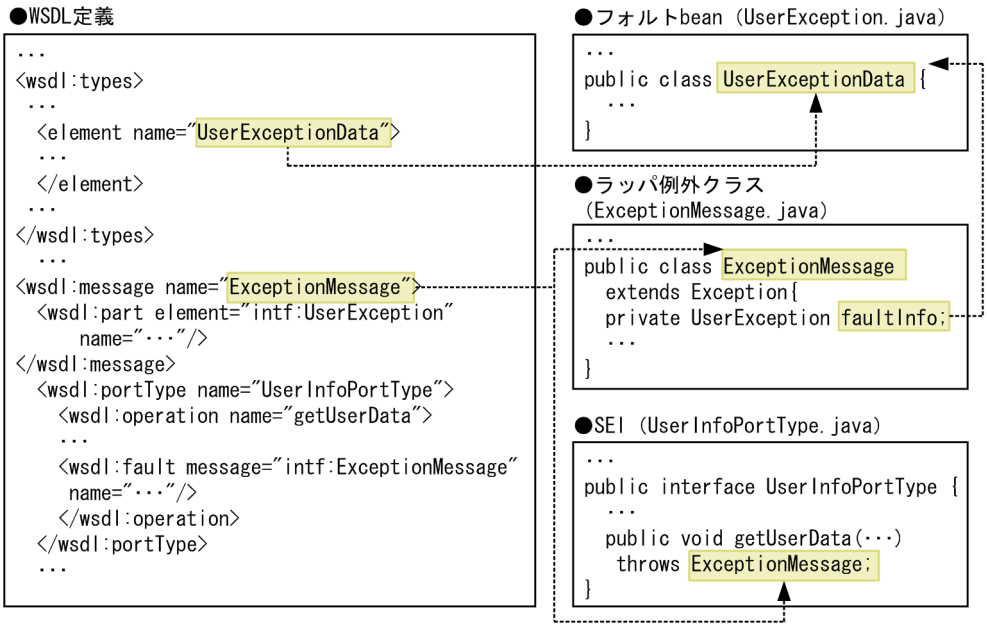
12.1.7　フォルトから例外クラスへのマッピング

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 2.1 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 12-7　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 2.1 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 2.1 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 12-13 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	日立 _fault	動作は保証されません（エラーメッセージは表示されません）。
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。

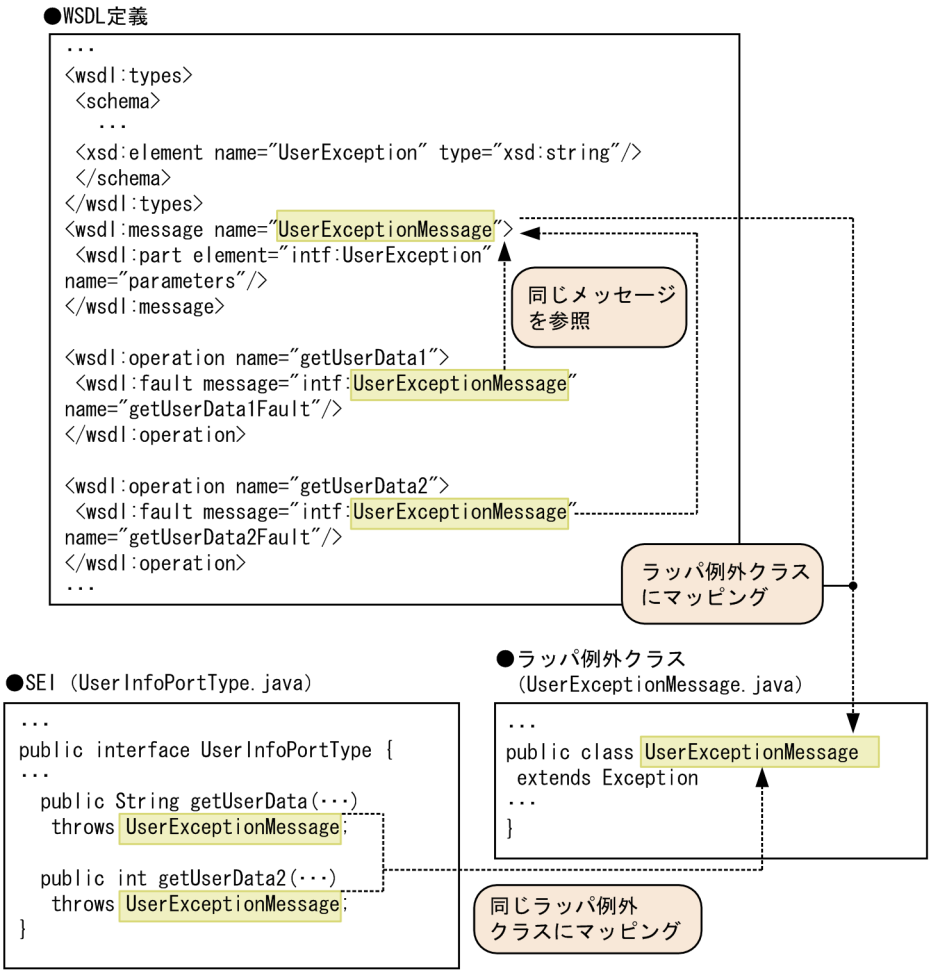
表 12-14 フォルトから参照されるメッセージのパートの個数と動作

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 12-8　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 12-15 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。

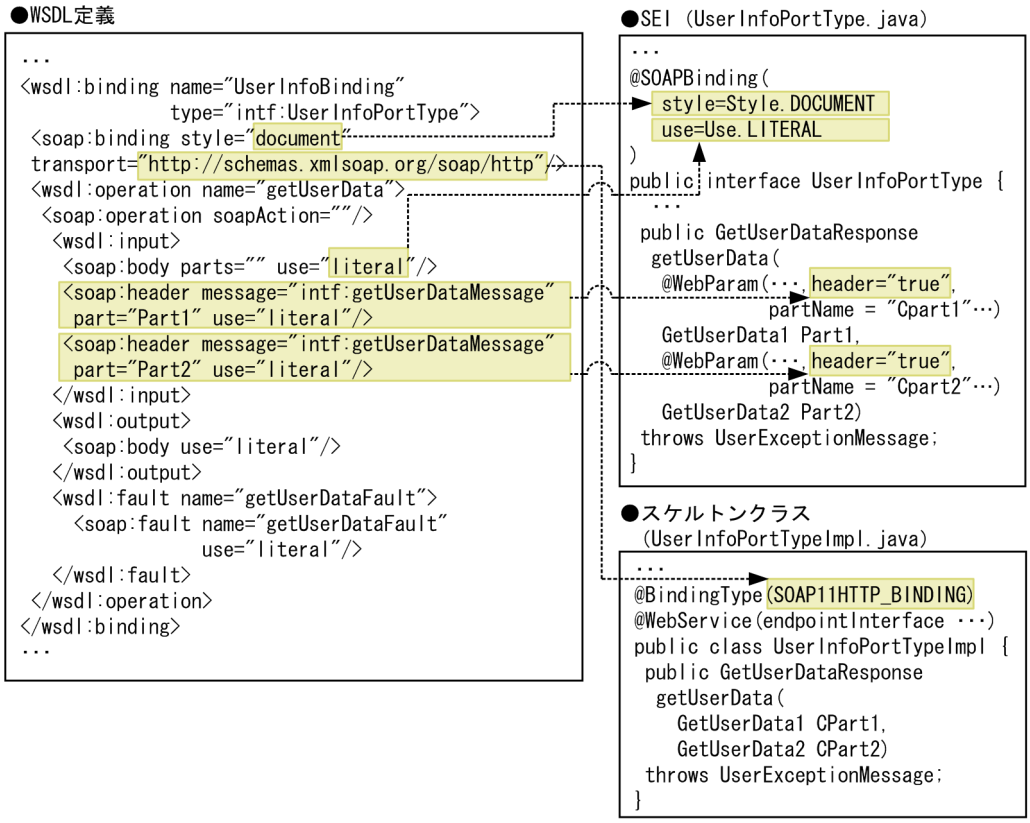
12.1.8 バインディング拡張要素からパラメタへのマッピング

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 2.1 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 12-9　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 12-16　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。

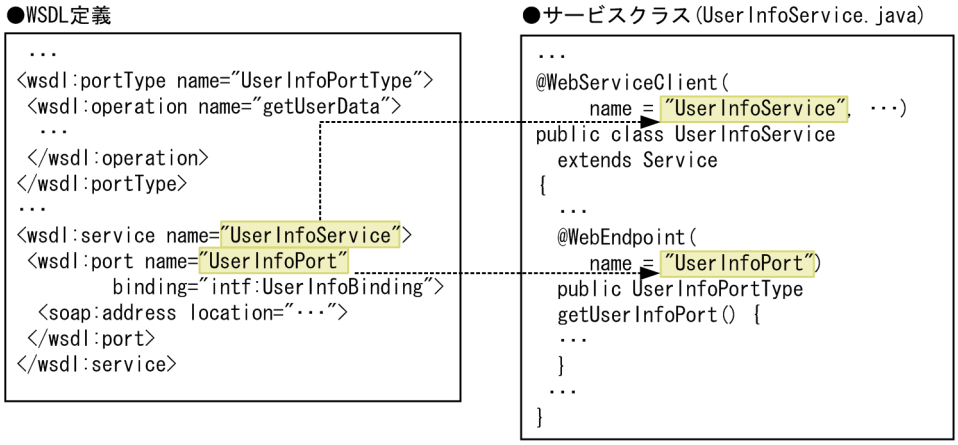
12.1.9　サービスおよびポートからサービスクラスへのマッピング

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

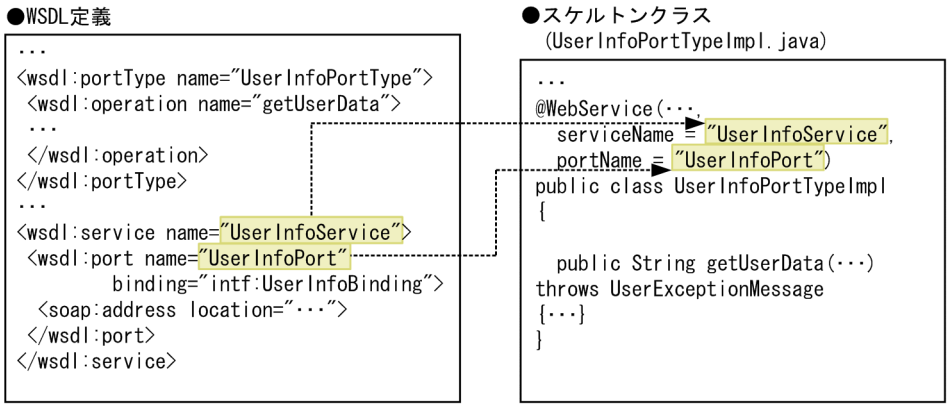
WSDL のサービスおよびポートと，サービスクラスは，JAX-WS 2.1 仕様に従ってマッピングされます。マッピング例を次の図に示します。

図 12-10　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 12-11　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は，次のメソッドを持ちます。

- ServiceName() のコンストラクタ※1
  引数はありません。このコンストラクタ中で親クラス javax.xml.ws.Service のコンストラクタが呼び出されます。
- ServiceName(URL wsdlLocation, QName serviceName) のコンストラクタ※1
  引数として，WSDL ロケーションを示す URL とサービスの QName 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.Service のコンストラクタが呼び出されます。

- getPortName() メソッド※2
  引数はありません。戻り値は SEI を実装するプロキシです。このメソッド中で，親クラス javax.xml.ws.Service の getPort メソッドが呼び出されます。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。このメソッドでエラーが発生した場合，javax.xml.ws.WebServiceException がスローされます。
- getPortName(WebServiceFeature ・・・ features) メソッド※2
  引数として可変長の javax.xml.ws.WebServiceFeature 型を持ちます。戻り値は SEI を実装するプロキシです。また，このメソッド中で，親クラス javax.xml.ws.Service の getPort メソッドが呼び出されます。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。このメソッドでエラーが発生した場合，javax.xml.ws.WebServiceException がスローされます。
  また，JAX-WS 2.1 仕様に従ってこのメソッドは Java ソースに出力されますが，javax.xml.ws.WebServiceFeature 型は使用できません。

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

"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 12-17　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	日立 _service 日立 _port	動作は保証されません（エラーメッセージは表示されません）。
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）。

12.1.10　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。

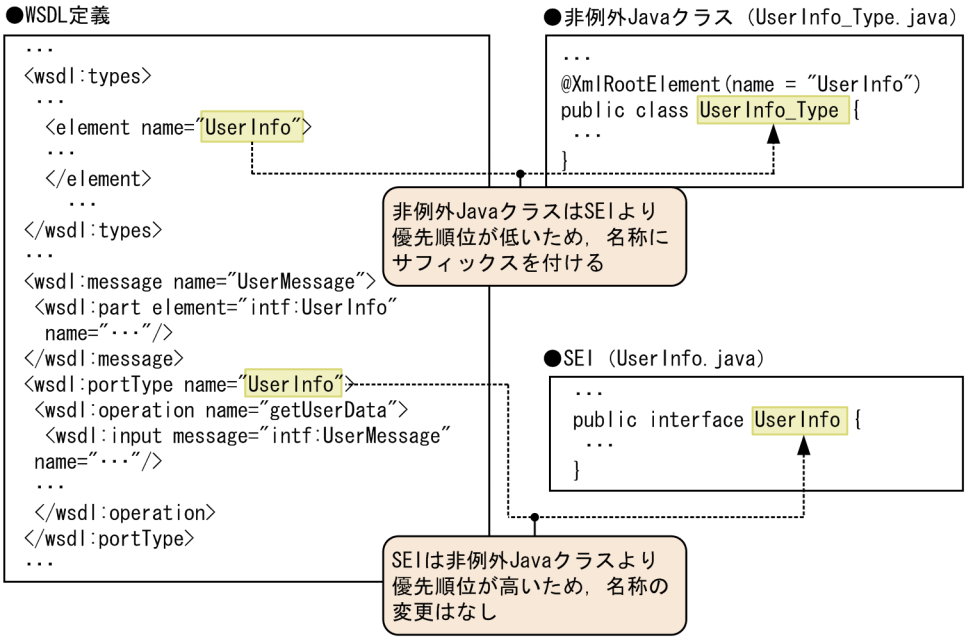
名前衝突時の優先順位と解決方法を次の表に示します。項番は優先順位を表します（項番 1 がいちばん高い）。

表 12-18 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 12-12　SEI 名と非例外 Java クラス名の衝突時の名前解決例

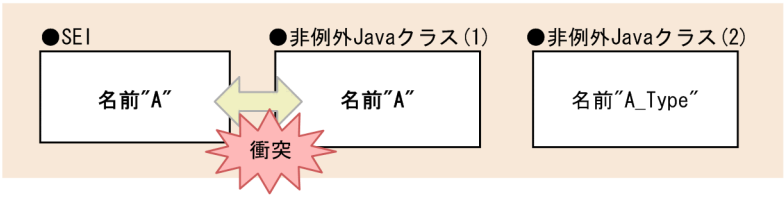


名前解決によってサフィックスが付加されたクラス名と，同名のクラス名が定義されていた場合は，再度，名前衝突が発生します。この場合，定義済みのクラス名からアンダースコアを削除することで解決します。

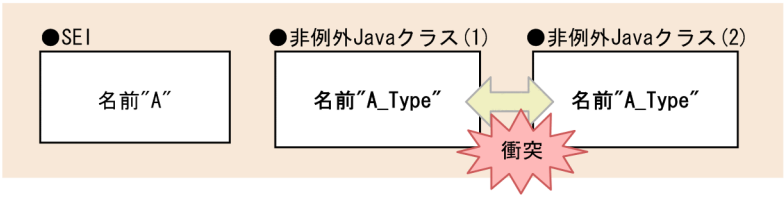
サフィックスが付加されたあとに，名前衝突する場合の名前解決例を示します。

図 12-13　サフィックス付加後に名前衝突する場合の名前解決例

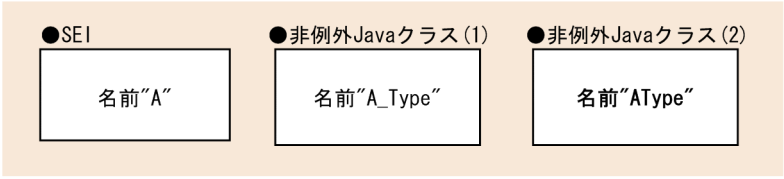
1. SEI，非例外Javaクラス(1)，非例外Javaクラス(2)を定義したが，SEIと非例外Javaクラス(1)の名前が衝突。



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 2.1 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，Cosminexus 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）。

12.2　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

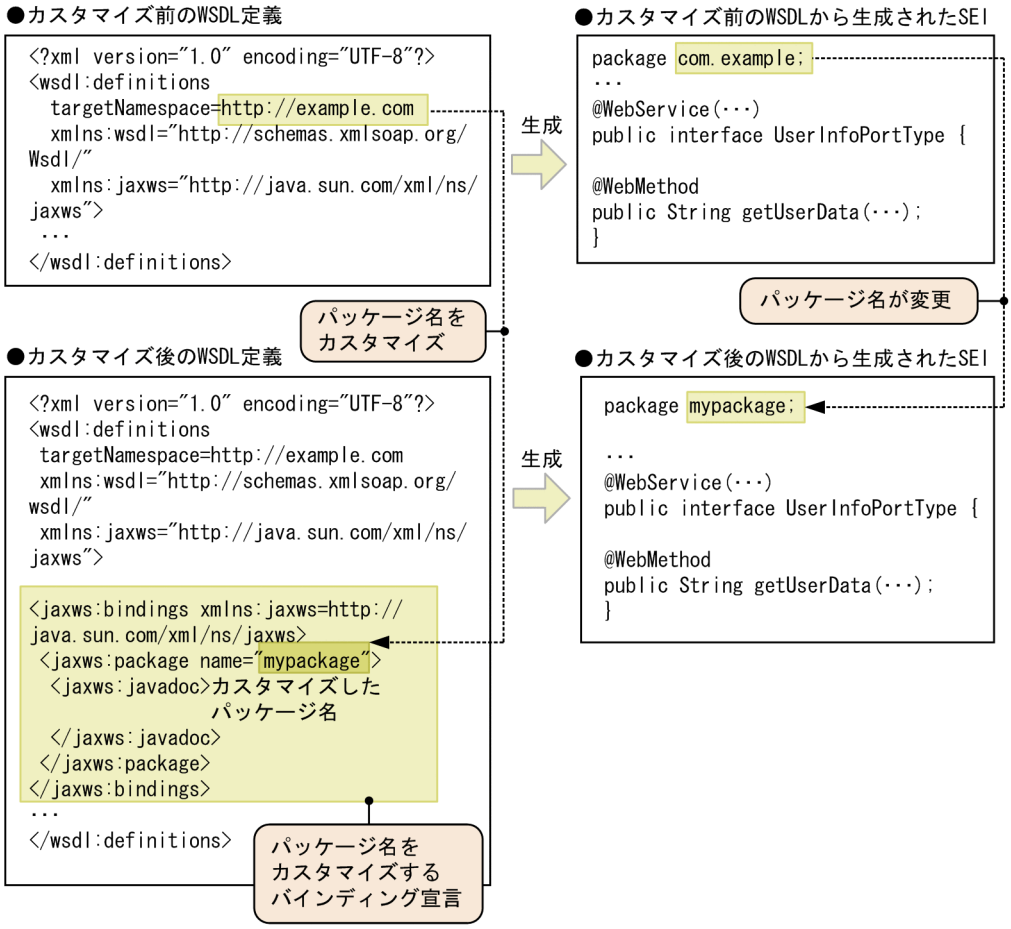
それぞれの方法について，カスタマイズ方法と注意事項について説明します。

12.2.1　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 12-14　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 12-19　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，Cosminexus 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 2.1 仕様を参照してください。

表 12-20　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。
├ 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。

要素名	属性名	説明
├ jaxws:class └ jaxws:javadoc	name	name 属性には，wsdl:portType 要素に対応する SEI 名を記述します。
	—	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 └ jaxws:javadoc	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。
	—	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。
	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。

要素名	属性名	説明
└ jaxws:class 　|	name	name 属性には，wsdl:fault 要素に対応する例外クラス名を記述します。
└ 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インタフェース名を記述します。
└ jaxws:javadoc	—	サービスインタフェースに付加する 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。 jaxws:provider 要素については，「12.2.7　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 2.1 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-W）。

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を重複して記述した場合，標準エラー出力とログに警告メッセージが出力され，処理が続行されます（KDJW51036-W）。

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 Cosminexus 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。

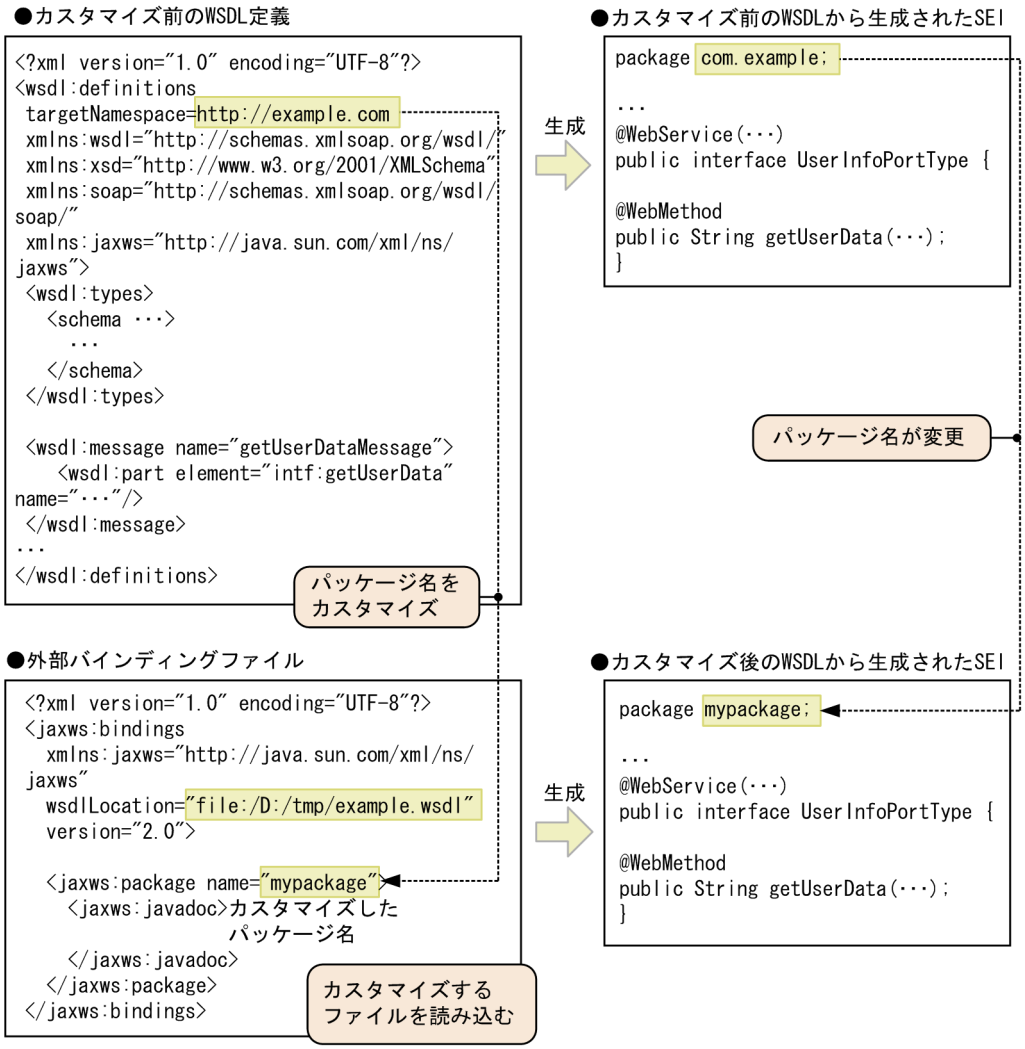
12.2.2 外部バインディングファイルによるカスタマイズ

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 12-15　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 12-21　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，「12.2.2(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，「12.2.2(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，「12.2.2(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。

項番	記述位置	要素	属性の指定	動作
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，「12.2.2(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，RFC2396 仕様の規則に従った文字列を使用してください。RFC2396 仕様の規則に従っていない文字列を使用する場合は，RFC2396 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。RFC2396 仕様の規則に従わず，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，Cosminexus 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 2.1 仕様を参照してください。

表 12-22　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，外部バインディング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	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。
├ 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	name	name 属性には，各要素に対応するクラス名を記述します。
│ └ jaxws:javadoc	—	クラスに付加する Javadoc 文字列です。
├ 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，「12.2.7　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 2.1 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-W）。

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警告メッセージが出力され，処理が続行されます（KDJW51036-W）。

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 Cosminexus 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）。

12.2.3　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。

12.2.4　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 12-23　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	—	○
├	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	—	×

要素名			属性名	指定可否
└	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（$）

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。

（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

Cosminexus 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。

12.2.5　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。

12.2.6　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，「12.1.10(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。

12.2.7　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）。

12.2.8　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。

12.2.9　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。

13 Java から WSDL へのマッピング

apt コマンドまたは cjwsgen コマンドを実行すると，JAX-WS 2.1 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。

13.1 Java から WSDL へのデフォルトマッピング

13.2 Java から WSDL へのマッピングのカスタマイズ

13.1　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 13-1　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	13.1.1
2	SEI 名	ポートタイプ	13.1.3
3	SEI のメソッド名	オペレーション	13.1.4
4	SEI のメソッドのパラメタおよび戻り値	パート	13.1.5，13.1.6
5	SEI のラッパ例外クラス	フォルト	13.1.7
6	SEI および Web サービス実装クラス	バインディング	13.1.8
7	Web サービス実装クラス	サービスとポート	13.1.9

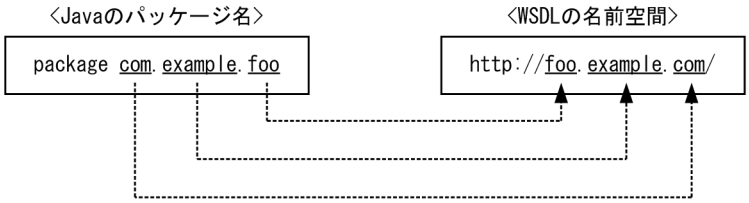
13.1.1　パッケージ名から名前空間へのマッピング

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 2.1 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 13-1　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 13-2　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. 日立	動作は保証されません（エラーメッセージは表示されません）。
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;	apt コマンド実行時にコンパイルエラーとなり，終了します。詳細は，JDK のドキュメントを参照してください。

（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 要素で記述していない場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW61004-E）。

javax.jws.WebService アノテーションの targetNamespace 要素については，「13.2.8 javax.jws.WebService アノテーション」を参照してください。

13.1.2　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。すべてのメソッドを実装していない場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW61011-E）。
- 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。定義した場合は，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（KDJW61012-E）。
- 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。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ます（KDJW61013-E）。
- 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。Web サービス実装クラスが一つもないと判断された場

合，警告メッセージが出力されます（KDJW61001-W）。

- 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。static でないインナークラスの場合は，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（KDJW61015-E）。
- 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。public 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（KDJW61016-E）。
- Web サービス実装クラスは，SEI を implements してもかまいません。implements しなかった場合，apt コマンド実行時に -sourcepath オプションを指定しても，引数に SEI と Web サービス実装クラスの両方を指定する必要があります。implements しないで SEI を apt コマンドの引数に指定しなかった場合，javax.jws.WebService アノテーションの endpointInterface 要素によって SEI の情報だけは参照しているが，コンパイル対象には含んでいないため，SEI のクラスファイルは生成されません。

（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，「13.2.8 javax.jws.WebService アノテーション」を参照してください。

13.1.3　SEI 名からポートタイプへのマッピング

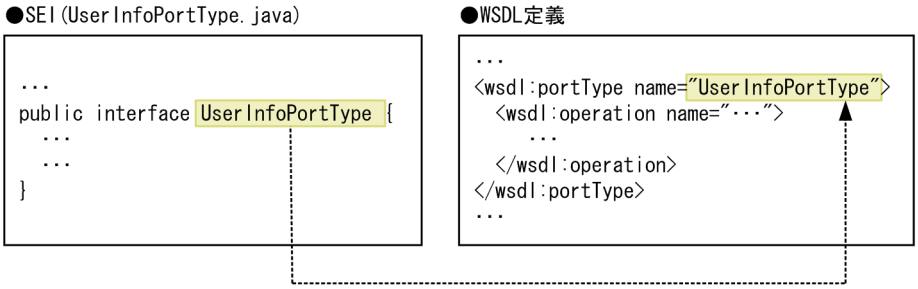
Java の SEI 名から WSDL のポートタイプ名（wsdl:portType 要素の name 属性）へのマッピングについて説明します。

（1）マッピング

Java の SEI 名と WSDL のポートタイプは，JAX-WS 2.1 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 13-2　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。記述していない場合は，標準エラー出力とログにエラーメッセージが出力されます（KDJW61020-E)。
- 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 13-3　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	日立 _sei	動作は保証されません（エラーメッセージは表示されません)。
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	apt コマンド実行時にコンパイルエラーとなり，終了します。詳細は，JDK のドキュメントを参照してください。

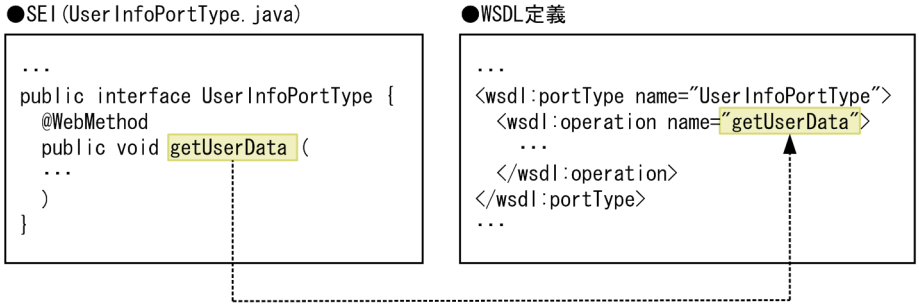
13.1.4　SEI のメソッド名からオペレーションへのマッピング

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 2.1 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 13-3　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 public なメソッドが 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 public 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，「13.2.5　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 public なメソッド。
- SEI の public なメソッドだけ WSDL のオペレーションへマッピングされます。それ以外のアクセス修飾子の場合は，WSDL へマッピングされません。public メソッドが一つもない場合，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（KDJW61093-E）。
- 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 public なメソッドが 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 に定義できるメソッドおよび Web サービス実装クラスで，exclude 要素が true である javax.jws.WebMethod アノテーションでアノテートされていない public メ

ソッドは，255 個まで定義できます。256 個以上定義した場合は，標準エラー出力とログに警告メッセージが出力され，処理が続行されます（KDJW61026-W）。

● 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。名前衝突が発生した場合，標準エラー出力とログにエラーメッセージが出力されます（KDJW61060-E）。

(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 13-4　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	日立 _sei	動作は保証されません（エラーメッセージは表示されません）。
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	apt コマンド実行時にコンパイルエラーとなり，終了します。詳細は，JDK のドキュメントを参照してください。

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 13-5　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	13.2.5(2)	
2	input のグローバル要素※	13.2.13(1)，13.2.13(2)	13.2.6(2)，13.2.6(5)
3	output のグローバル要素※	13.2.14(1)，13.2.14(2)	13.2.7(2)，13.2.7(4)
4	リクエスト bean クラス名	13.2.13(3)	
5	レスポンス bean クラス名	13.2.14(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。

13.1.5　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。

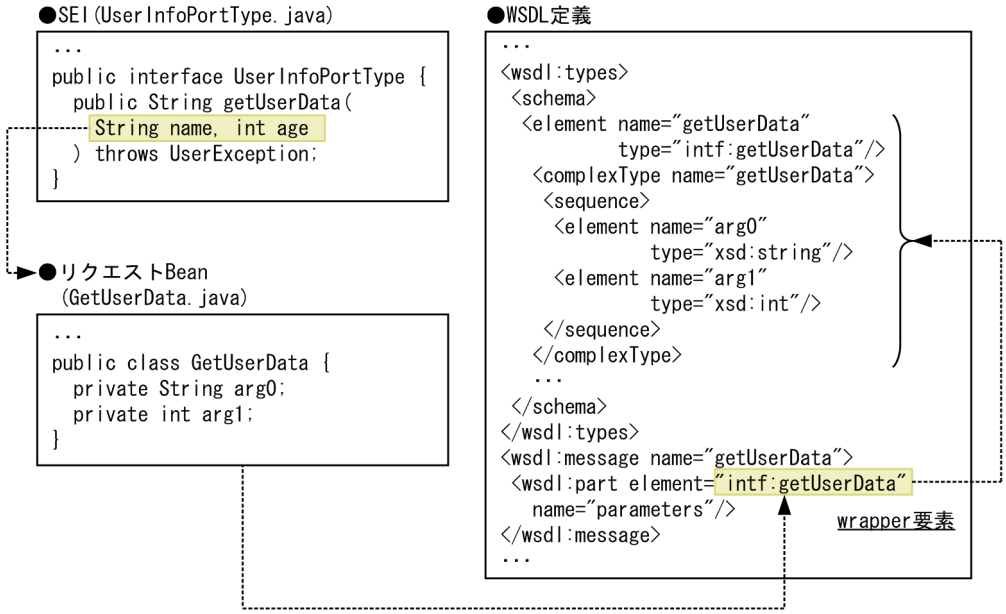
ここでは，wrapper スタイルの場合について説明します。

(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。

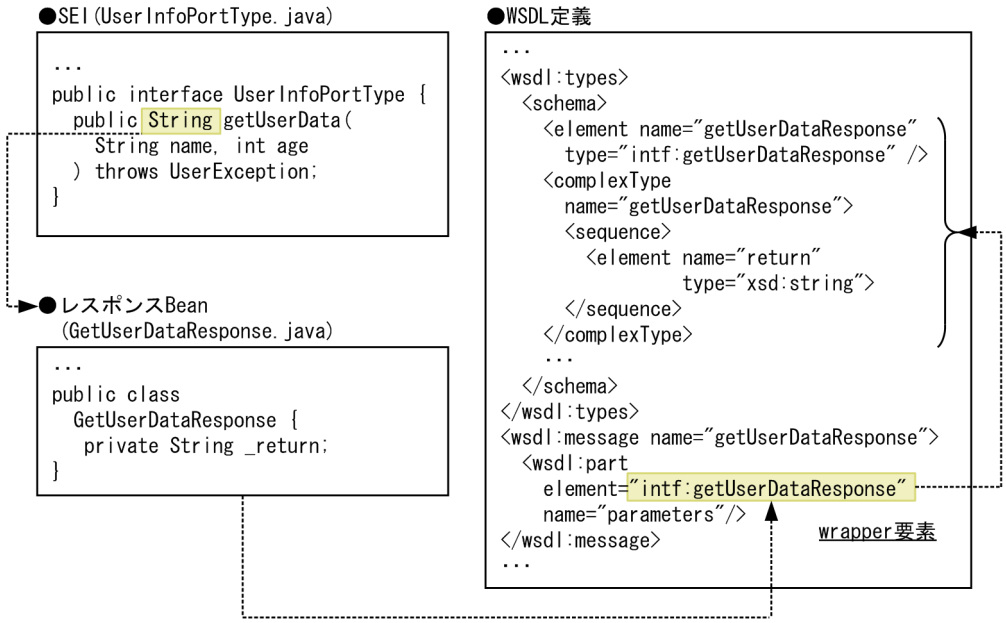
メソッドのパラメタとメッセージのパートのマッピング例を次の図に示します。

図 13-4　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 13-5　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。
- wrapper 要素は，"parameters" という固定値で input メッセージおよび output メッセージのパート名に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。

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。

(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 2.1 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。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ます（KDJW61035-E）。
- 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，apt 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。それ以外の方法で指定した場合は，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（KDJW61035-E）。

(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。mode 要素を指定していない場合，または mode 要素に Mode.IN 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ます（KDJW61031-E）。
  また，javax.jws.WebParam アノテーションを指定していない場合，
  javax.xml.ws.Holder の型パラメタは inout パラメタとして解釈されます。

● 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

(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，「13.2.6(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，apt 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。リクエスト bean またはレスポンス bean が，パッケージ内で同時に生成されるほかの JavaBean クラスと名前衝突した場合，標準エラー出力とログにエラーメッセージが出力されます（KDJW61083-E）。

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 apt コマンドの引数に含まれていた場合は，apt コマンド実行時にエラー

チェックされます。

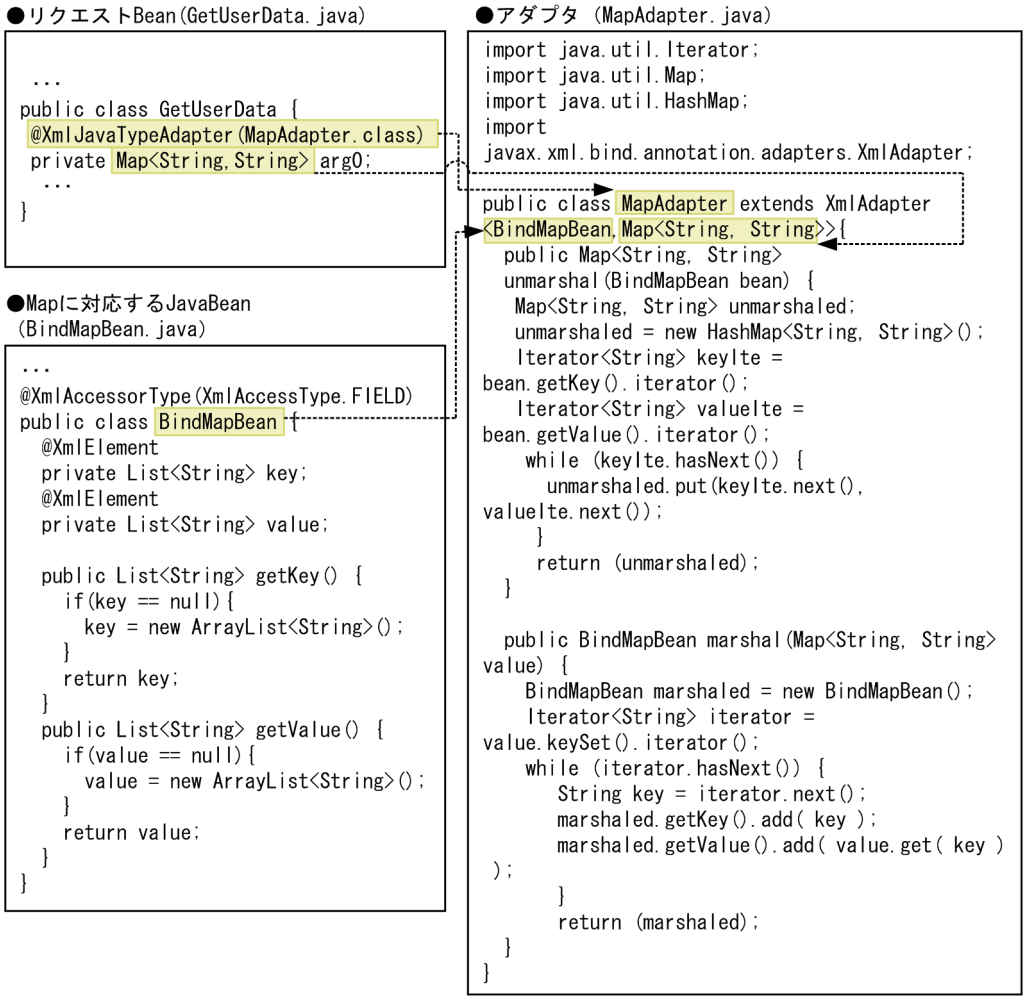
● グローバル要素（ローカル名および名前空間）
グローバル要素（ローカル名および名前空間）は，WSDL 内でユニークである必要があります。ユニークでない場合の動作は保証されません。

（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 2.1 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. **apt コマンドを実行します。**
   アノテート済みの SEI を apt コマンドで解釈します。
   value type，アダプタ，および apt コマンドによって 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 13-6　java.util.Map の使用例



13.1.6　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。

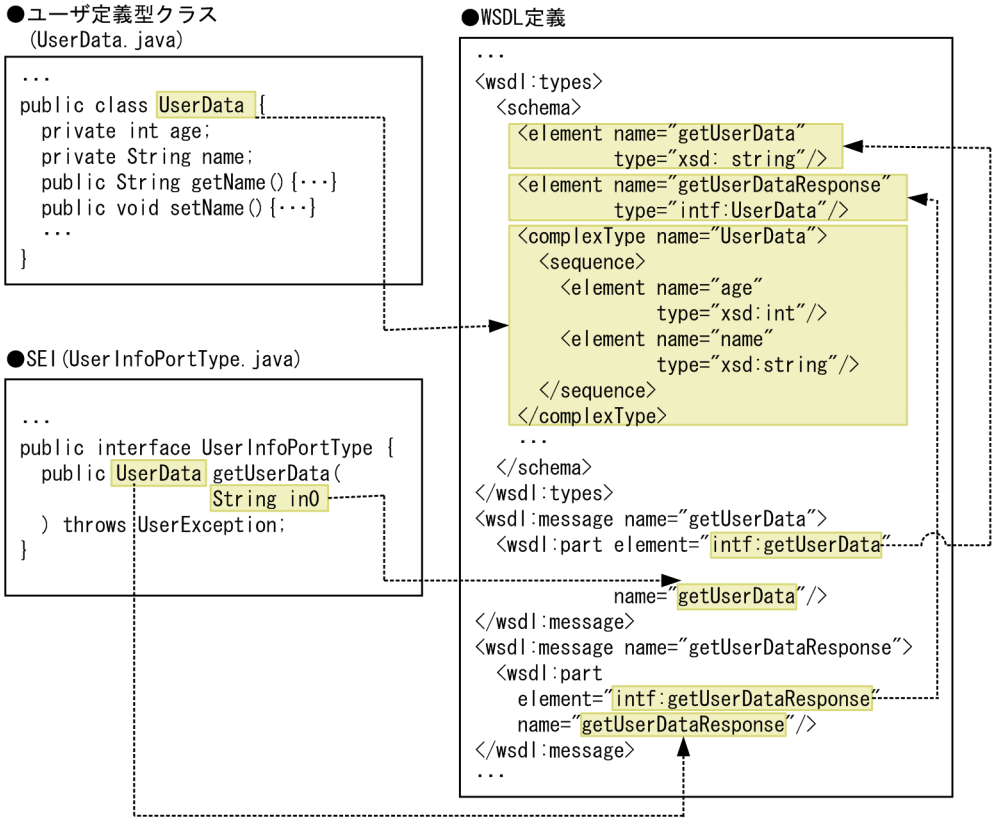
ここでは，non-wrapper スタイルの場合について説明します。

(1) マッピング

non-wrapper スタイルの場合，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。マッピング例を次の図に示します。

図 13-7　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。「13.1.5(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 13-6　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。※ 1 • 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
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。※ 1 • 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
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。※ 3

注※ 1

合計で 0 個または 2 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ます（KDJW61056-E）。

注※ 2

合計で 0 個または 2 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ます（KDJW61057-E）。

注※ 3

out パラメタまたは inout パラメタを指定した場合，標準エラー出力とログにエラーメッセージが出力されます（KDJW61059-E）。

non-wrapper スタイルで out パラメタまたは inout パラメタを使用する場合，javax.jws.WebParam アノテーションの name 要素を指定する必要があります。javax.jws.WebParam アノテーションの name 要素を指定していない場合，標準エラー出力とログにエラーメッセージが出力されます（KDJW61085-E）。

(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 13-7　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，「13.1.5(6)　java.util.Map クラスの使用」を参照してください。

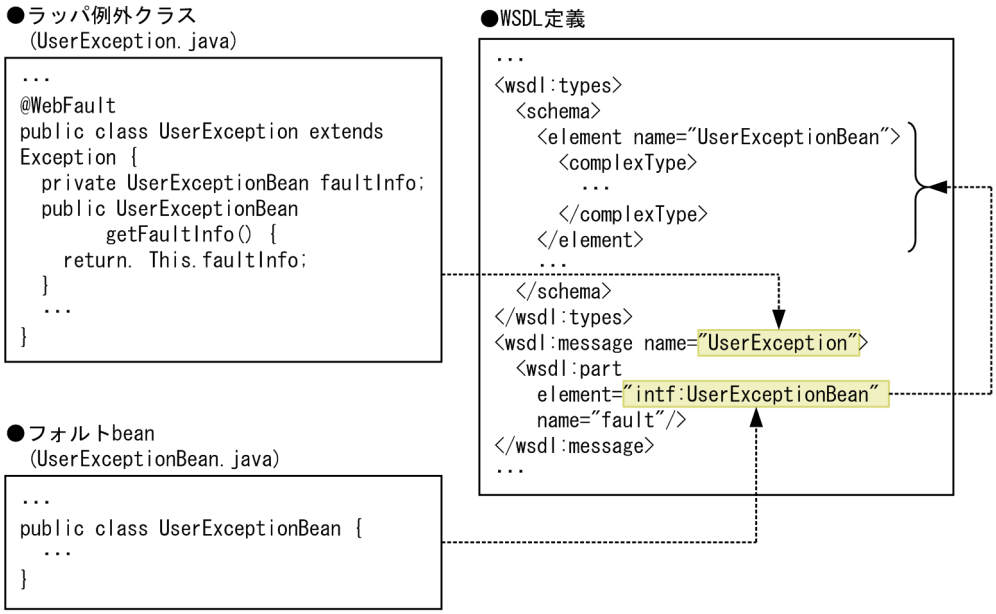
13.1.7　Java のラッパ例外クラスからフォルトへのマッピング

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 2.1 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 13-8　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 getCause，getLocalizedMessage，および getStackTrace という getter と，java.lang.Object から継承される getClass という getter 以外のすべての getter と同じ型／名称のプロパティを持ちます。
- 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。256 個以上定義した場合は，標準エラー出力とログに警告メッセージが出力され，処理が続行されます（KDJW61027-W）。
- 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 13-8 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	日立 _exception	動作は保証されません（エラーメッセージは表示されません）。
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	apt コマンド実行時にコンパイルエラーとなり，終了します。詳細は，JDK のドキュメントを参照してください。

(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 が，パッケージ内で同時に生成される JavaBean と名前衝突した場合，標準エラー出力とログにエラーメッセージが出力されます（KDJW61065-E）。

すでに存在するクラスと名前が重複していた場合，上書きされます。ただし，そのクラスが apt コマンドの引数に含まれている場合は，apt コマンドでエラーチェックされます。

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。

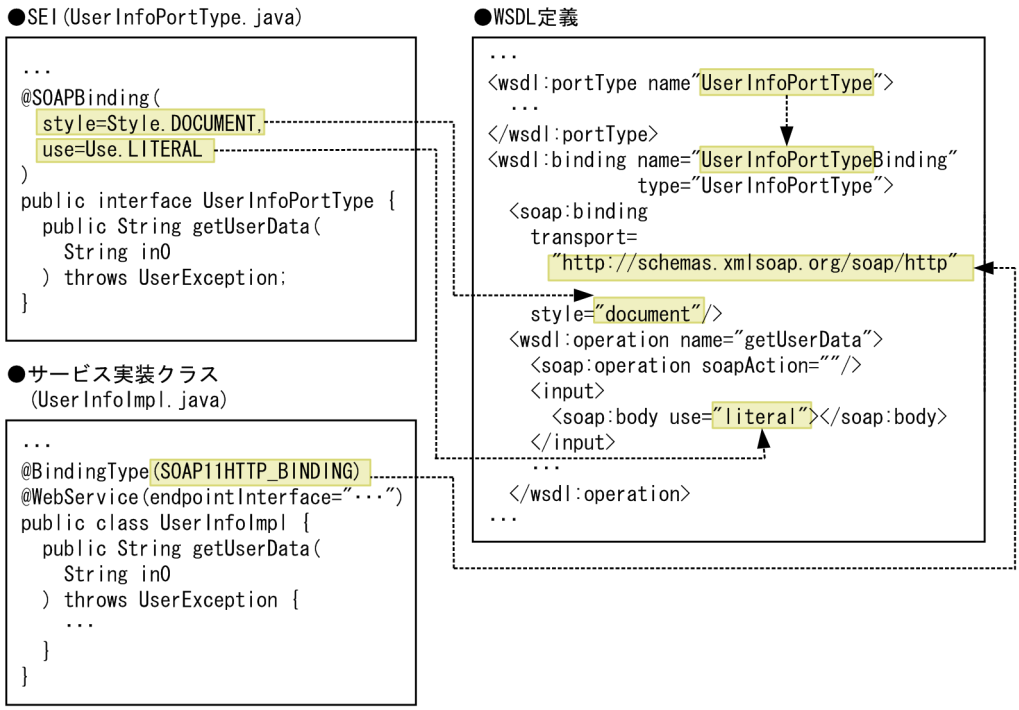
13.1.8　SEI からバインディングへのマッピング

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 2.1 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 13-9　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，「13.1.3　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。

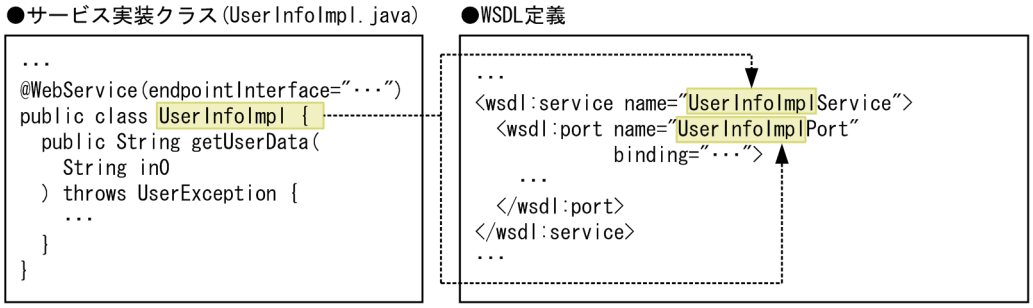
13.1.9　Web サービス実装クラスからサービスおよびポートへのマッピング

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 2.1 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 13-10　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 13-9　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	日立 _service	動作は保証されません（エラーメッセージは表示されません）。
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	apt コマンド実行時にコンパイルエラーとなり，終了します。詳細は，JDK のドキュメントを参照してください。

13.1.10　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 13-10　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

Cosminexus の JAX-WS 機能は，JAX-WS 2.1 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

Cosminexus 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。

Cosminexus 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。

Cosminexus 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

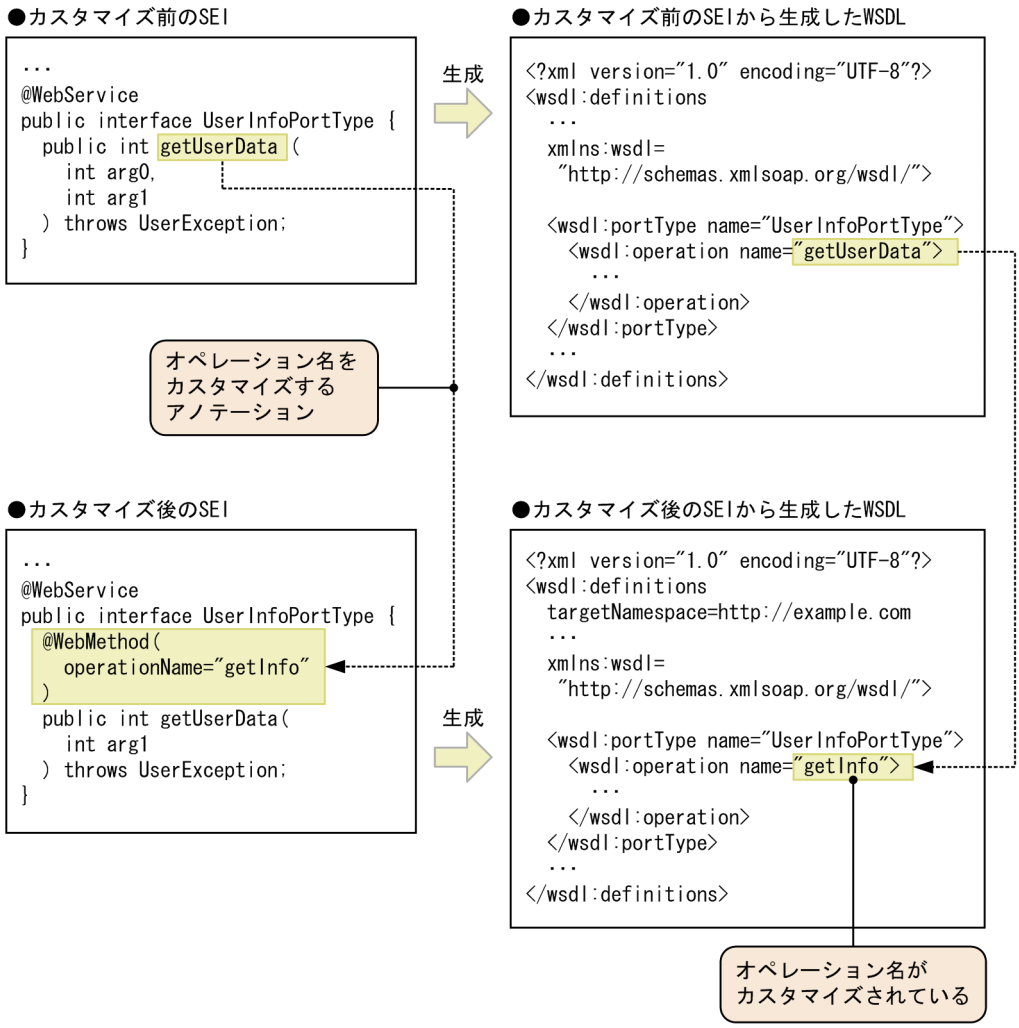
wrapper スタイルの場合にはメソッド引数や戻り値にジェネリクス型を使用できますが，non-wrapper スタイルの場合にはジェネリクス型を使用できません。

13.2　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 13-11　アノテーションを使用したカスタマイズ例



apt コマンドまたは cjwsgen コマンドを実行すると，cjwsimport コマンドで自動的に付与される javax.xml.ws.WebEndpoint および javax.xml.ws.WebServiceClient アノテーションは無視されます（警告メッセージは出力されません）。また，非サポートのアノ

テーションを指定した場合も無視されます。このとき，Cosminexus の JAX-WS 機能以外が提供するアノテーションプロセッサでも非サポートであった場合，apt コマンドの警告メッセージが出力されます。

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。このときに限り，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。

13.2.1 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 2.1 仕様を参照してください。

表 13-11 カスタマイズ時に使用する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。	○

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		
3	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。Cosminexus 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。Cosminexus の JAX-WS 機能では literal だけ指定できます。	○
4	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。	○
5	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。	○
		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。	○
6	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。	○

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		
7	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
8	javax.xml.bind.annotation.XmlMimeType	value	Java 型に関連づける MIME タイプのテキストを表します。	○
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。	○
		targetNamespace	対象要素の名前空間名です。	○

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		
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。	○
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。	○
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。	○
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
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

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。

JSR-181 仕様および JAX-WS 2.1 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 13-12　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.WebServiceRef	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.WebServiceRefs	JSR109 仕様に関連する機能はサポートしていません。
3	javax.xml.ws.spi.WebServiceFeatureAnnotation	このアノテーションを使用するための API（フィーチャーをパラメタに取るメソッド）はサポートしていません。
4	javax.jws.Oneway	one-way パターンはサポートしていません。
5	javax.jws.soap.InitParam	JSR181 仕様で推奨していません（廃止対象）。
6	javax.jws.soap.SOAPMessageHandler	JSR181 仕様で推奨していません（廃止対象）。
7	javax.jws.soap.SOAPMessageHandlers	JSR181 仕様で推奨していません（廃止対象）。

13.2.2　com.sun.xml.ws.developer.StreamingAttachment アノテーション

com.sun.xml.ws.developer.StreamingAttachment アノテーションは，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。

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 cjwsgen 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，apt コマンドおよび cjwsgen 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 13-12　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，「22.3　一時ファイル（ストリーミング）」を参照してください。

13.2.3　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。このとき，標準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（KDJW61076-W）。

（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。Cosminexus 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，apt コマンドまたは cjwsgen 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，「26.9.1　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。

13.2.4　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）

Cosminexus 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。したがって，style 要素に SOAPBinding.Style.RPC 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ます（KDJW61063-E）。また，SOAPBinding.Style.DOCUMENT および SOAPBinding.Style.RPC 以外の値を指定した場合，apt コマンドを実行するときにコンパイルエラーになります。

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）

Cosminexus 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。したがって，use 要素に SOAPBinding.Use.ENCODED 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（KDJW61063-E）。また，SOAPBinding.Use.LITERAL，SOAPBinding.Use.ENCODED 以外の値を指定した場合，apt コマンドを実行するときにコンパイルエラーになります。

（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。

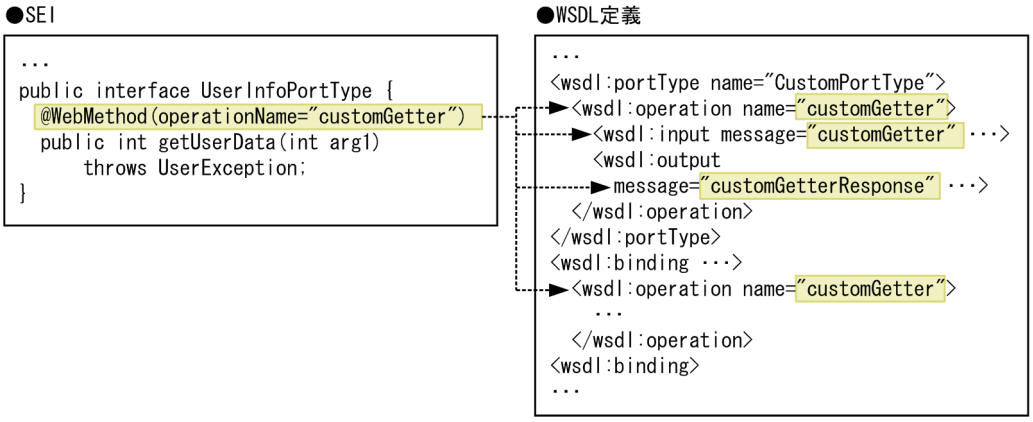
parameterStyle 要素に，SOAPBinding.ParameterStyle.WRAPPED，および SOAPBinding.ParameterStyle.BARE 以外の値を指定した場合，apt コマンドを実行するときにコンパイルエラーになります。

13.2.5　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 13-13　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。

- マッピングされるオペレーションから除外した結果，public メソッドが一つもなくなってしまった場合，標準エラー出力とログにエラーメッセージが出力され，処理が終了されます（KDJW61093-E）。
- 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。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ます（KDJW61010-E）。
- 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。"true" 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ます（KDJW61021-E）。
- 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場合，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 13-13　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	マッピングされません。
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。同じでない場合，標準エラー出力とログにエラーメッセージが出力されます（KDJW61047-E）。

（3） action 要素（javax.jws.WebMethod）

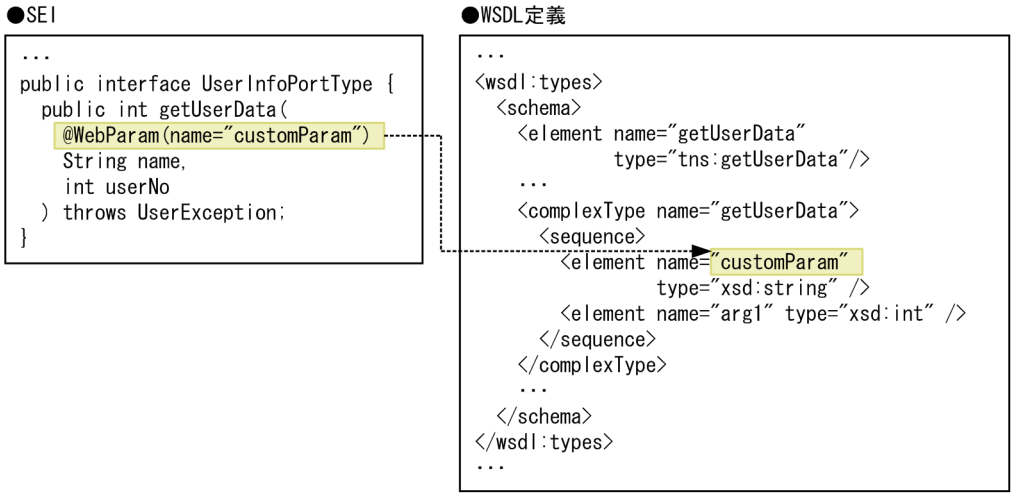
action 要素は，SOAP オペレーションのアクションにマッピングされます。action 要素には，RFC2396 で規定された xsd:anyURI を満たす文字を指定できますが，指定した値はランタイムでは無視されます。

13.2.6　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 13-14　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。wrapper スタイルで指定した場合，標準エラー出力とログにエラーメッセージが出力されます（KDJW61037-E）。

（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。
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ます（KDJW61087-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ます（KDJW61088-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ます（KDJW61089-E）。

- 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 13-14　名前空間に記述できる文字列の条件（javax.jws.WebParam）

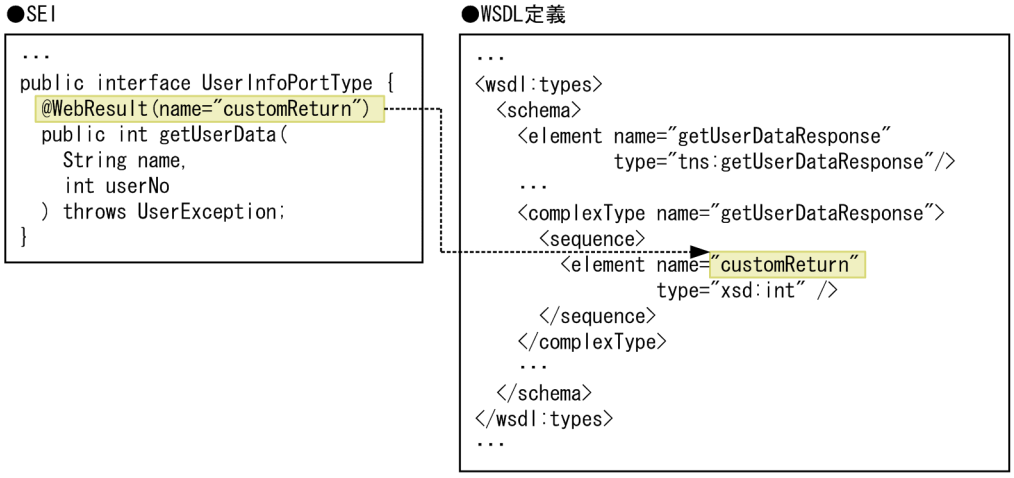
項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0 ～ 9，A ～ Z，a ～ z）だけを使用した文字列	http:// 日立 .com http://133.145.224.19/ http:// [1080:2C14:D30:BA04:275:806:270C:418A]/	動作は保証されません（エラーメッセージは表示されません）。
2	Java の予約語以外の文字列	http://hitachi.com/abstract	動作は保証されません。
3	先頭が数字でない文字列	http://1hitachi.com	

13.2.7　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 13-15　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。wrapper スタイルで header 要素の要素値に "true" 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ます（KDJW61038-E）。

（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。
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ます（KDJW61090-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ます（KDJW61091-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ます（KDJW61092-E）。

- 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 13-15　名前空間に記述できる文字列の条件（javax.jws.WebResult）

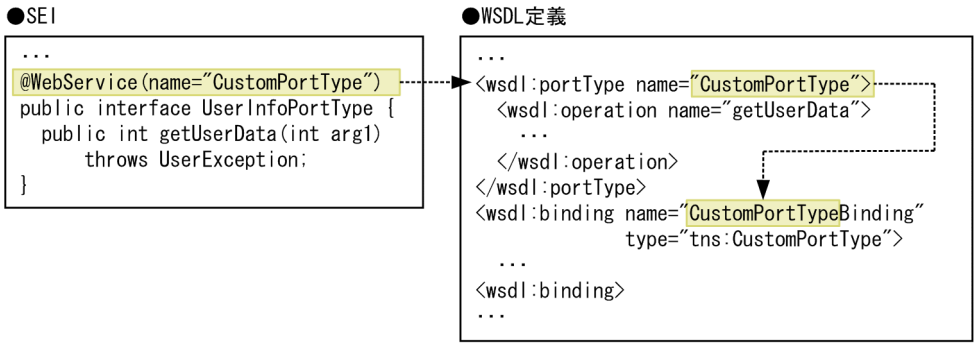
項番	条件	不正な文字列の例	不正な文字列を指定した場合の動作
1	半角英数字（0 ～ 9，A ～ Z，a ～ z）だけを使用した文字列	http:// 日立 .com http://133.145.224.19/ http:// [1080:2C14:D30:BA04:275:806:270C:418A]/	動作は保証されません（エラーメッセージは表示されません）。
2	Java の予約語以外の文字列	http://hitachi.com/abstract	動作は保証されません。
3	先頭が数字でない文字列	http://1hitachi.com	

13.2.8　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 13-16　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。javax.xml.ws.WebServiceProvider アノテーションと javax.jws.WebService アノテーションを同時に指定した場合は，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（KDJW61098-E）。

（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。
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ます（KDJW61005-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ます（KDJW61006-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ます（KDJW61007-E）。

- 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 13-16　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:// 日立 .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表示されません）。
2	Java の予約語以外の文字列	http://hitachi.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://1hitachi.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。クラスを指定した場合は，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（KDJW61009-E）。指定された SEI 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ます（KDJW61028-E）。

● 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。同時に指定した場合，標準エラー出力とログにエラーメッセージが出力されます（KDJW61019-E）。

● 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。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ます（KDJW61024-E）。

● 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。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ます（KDJW61075-E）。

- javax.jws.WebMethod
- javax.jws.WebParam
- javax.jws.WebResult
- 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。同時に指定した場合，標準エラー出力とログにエラーメッセージが出力されます（KDJW61019-E）。

（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。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ます（KDJW61023-E）。
- 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。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ます（KDJW61022-E）。

- 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，apt コマンドまたは cjwsgen コマンドの実行時には解釈されません。

13.2.9　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 cjwsgen ツールが生成する 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 13-17　javax.xml.bind.annotation.XmlMimeType アノテーションを使用した例

●SEI

```
・・・・・
interface ExamplePortType {

    public String getUserData(@XmlMimeType("Image/jpeg")Image in0)
throws ExampleException;
}
・・・・・
```

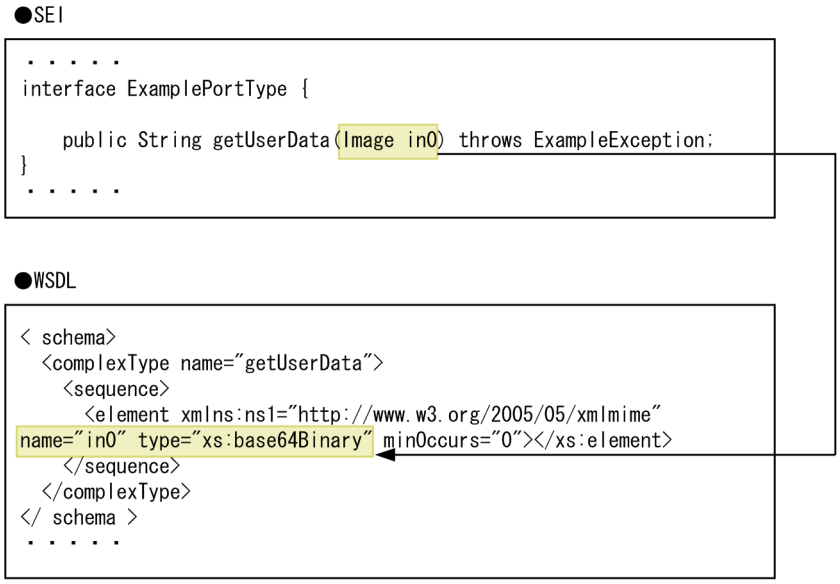
●WSDL

```
< schema>
  <complexType name="getUserData">
    <sequence>
      <element xmlns:ns1="http://www.w3.org/2005/05/xmlmime"
name="in0" ns1:expectedContentTypes="image/jpeg"
type="xs:base64Binary" minOccurs="0"></xs:element>
    </sequence>
  </complexType>
</ schema >
・・・・・
```

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 13-18　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 13-17　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

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
7	java.util.List<DataHandler>	○	○	○	△※2	×※3
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 13-18　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

項番	Java 型	value 要素に指定できる MIME タイプ
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。

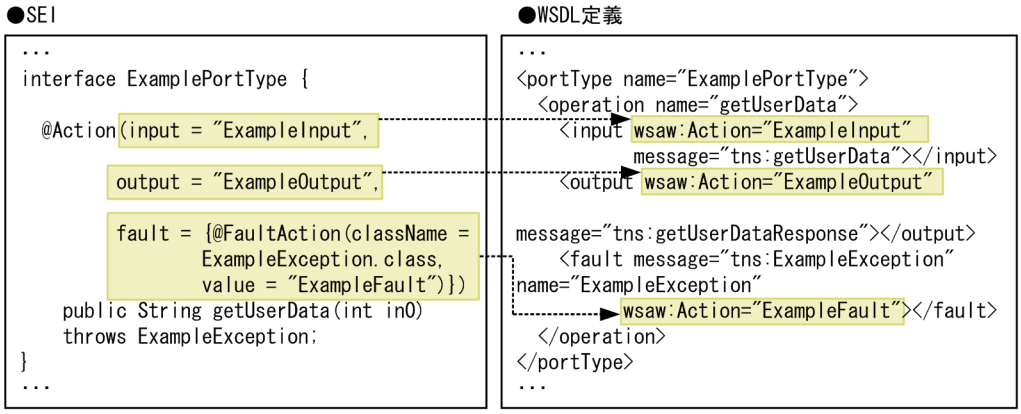
13.2.10　javax.xml.ws.Action アノテーション

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。サービス実装クラスに指定した場合は，標準エラー出力とログに警告メッセージが出力され，処理が続行されます（KDJW61095-W）。

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 13-19　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。apt コマンドまたは cjwsgen 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，RFC2396 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。apt コマンドまたは cjwsgen 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，RFC2396 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。apt コマンドまたは cjwsgen コマンドの実行時には解釈されません。

13.2.11　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。SEI に指定した場合，標準エラー出力とログに警告メッセージが出力され，処理が続行されます（KDJW61079-W）。

（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）

これら以外のフィールド値を指定した場合，標準エラー出力とログにエラーメッセージが出力されます。value 要素に不正な値を指定した場合に出力されるエラーメッセージを次の表に示します。

表 13-19　value 要素（javax.xml.ws.BindingType）に不正な値を指定した場合に出力されるエラーメッセージ

項番	value 要素の値	エラーメッセージ ID	
		apt コマンド実行時	cjwsgen コマンド実行時
1	javax.xml.ws.http.HTTPBinding.HTTP_BINDING	KDJW61072-E	KDJW71005-E
2	バインディング識別子以外の不正な値	KDJW61072-E	KDJW71005-E

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 13-20　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	-soap12binding オプション または 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	

注※

-soap12binding オプションの指定値と WSDL の transport 属性値の関係については，「11.3 cjwsgen コマンド」を参照してください。

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 13-21　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。

13.2.12　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）。

className 要素は Web サービス開始時に参照されるだけです。apt コマンドまたは cjwsgen 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，RFC2396 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。apt コマンドまたは cjwsgen コマンドの実行時には解釈されません。

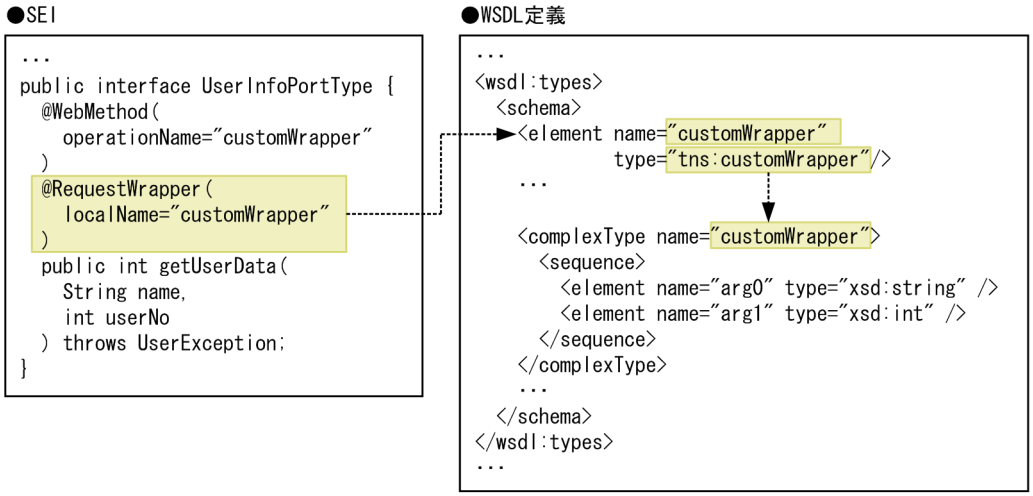
13.2.13　javax.xml.ws.RequestWrapper アノテーション

javax.xml.ws.RequestWrapper アノテーションは，wrapper スタイルで指定できます。non-wrapper スタイルで指定した場合，標準エラー出力とログに警告メッセージが出力され，処理が続行されます（KDJW61061-W）。

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。Web サービス実装クラスに指定した場合は無視されます。Web サービス実装クラスに指定した場合，標準エラー出力とログに警告メッセージが出力され，処理が続行されます（KDJW61077-W）。

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 13-20　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。名前が異なる場合，標準エラー出力とログにエラーメッセージが出力されます（KDJW61047-E）。

（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。
  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ます（KDJW61042-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ます（KDJW61043-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ます（KDJW61044-E）。

- 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 13-22　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:// 日立 .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表示されません）。
2	Java の予約語以外の文字列	http://hitachi.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://1hitachi.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。Java 識別子の命名規則に従っていない場合，標準エラー出力とログにエラーメッセージが出力されます（KDJW61040-E）。

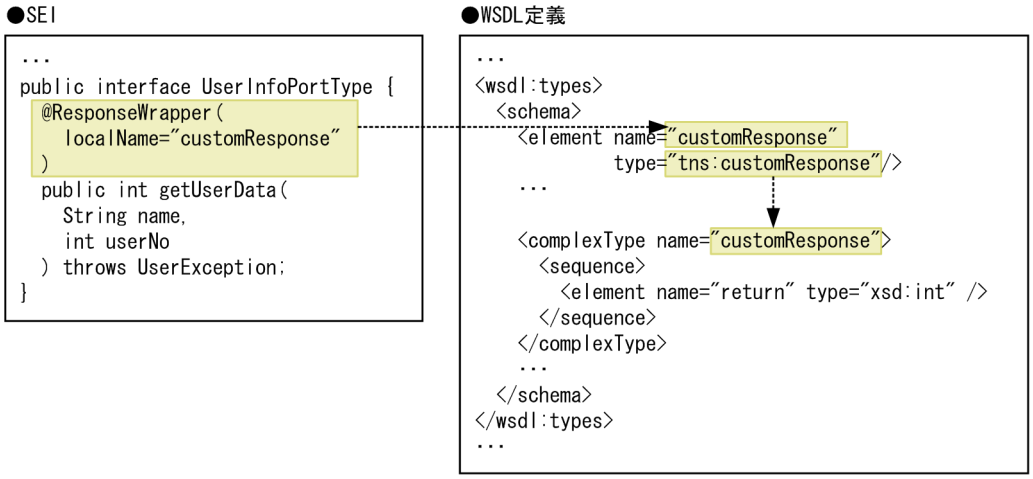
13.2.14　javax.xml.ws.ResponseWrapper アノテーション

javax.xml.ws.ResponseWrapper アノテーションは，wrapper スタイルで指定できます。non-wrapper スタイルで指定した場合，標準エラー出力とログに警告メッセージが出力され，処理が続行されます（KDJW61062-W）。

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。Web サービス実装クラスに指定した場合，標準エラー出力とログに警告メッセージが出力され，処理が続行されます（KDJW61078-W）。

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 13-21　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。
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ます（KDJW61049-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

れます（KDJW61050-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ます（KDJW61051-E）。

- 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 13-23　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:// 日立 .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表示されません）。
2	Java の予約語以外の文字列	http://hitachi.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://1hitachi.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。Java 識別子の命名規則に従っていない場合，標準エラー出力とログにエラーメッセージが出力されます（KDJW61041-E）。

13.2.15　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。apt コマンドまたは cjwsgen 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。

13.2.16　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 アノテーションは，サービス実装クラスでだけ指定できます。SEI に指定した場合は，標準エラー出力とログに警告メッセージが出力され，処理が続行されます（KDJW61094-W）。

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 13-22　javax.xml.ws.soap.Addressing アノテーションを使用したカスタマイズ例

●サービス実装クラス

```
...
@Addressing
@WebService(endpointInterface =
"jaxwstp.example.service.ExamplePortType",
targetNamespace =
"http://service.example.jaxwstp/",
 serviceName = "ExampleService",
 portName = "ExamplePort")
public class ExampleBinding implements
ExamplePortType {
```

●WSDL定義

```
...
<binding name="ExampleBinding"
      type="tns:ExamplePortType">
  <wsaw:UsingAddressing>
  </wsaw:UsingAddressing>
  <soap:binding transport=
      "http://schemas.xmlsoap.org/soap/http"
       style="document"></soap:binding>
  <operation name="getUserData">
    <soap:operation soapAction="">
    </soap:operation>
    <input>
      <soap:body use="literal"></soap:body>
    </input>
    <output>
       <soap:body use="literal"></soap:body>
    </output>
  </operation>
</binding>
...
```

（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。apt コマンドまたは cjwsgen 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。apt コマンドまたは cjwsgen コマンドの実行時には解釈されません。

13.2.17　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 アノテーションは 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。

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 cjwsgen 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。apt コマンドまたは cjwsgen コマンドの実行時には解釈されません。

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）

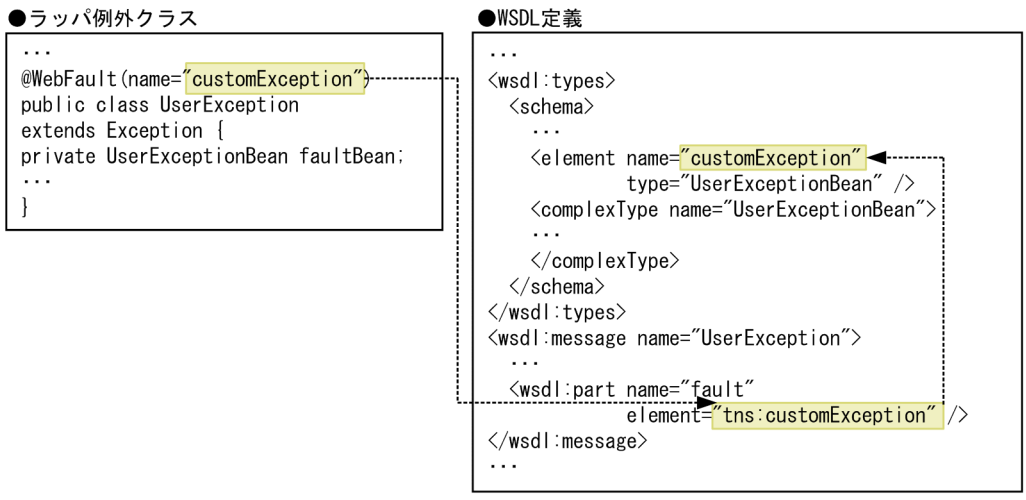
threshold 要素は，Web サービスで MTOM/XOP 仕様形式の添付ファイルが使用できるときに，バイナリデータを MTOM/XOP 仕様形式の添付ファイルとして送信するためのしきい値です。javax.activation.DataHandler クラス以外で，さらに指定された値を超えるバイナリデータ（threshold 要素値＜送信するデータのサイズ）の場合，MTOM/XOP 仕様形式の添付ファイルとして送信されます。デフォルト値は "0" です。

13.2.18　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 13-23　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。
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ます（KDJW61067-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ます（KDJW61068-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ます（KDJW61069-E）。

- 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 13-24　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:// 日立 .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表示されません）。

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
2	Java の予約語以外の文字列	http://hitachi.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://1hitachi.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。Java 識別子の命名規則に従っていない場合，標準エラー出力とログにエラーメッセージが出力されます（KDJW61039-E）。

13.2.19　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 アノテーションと javax.jws.WebService アノテーションを同時に指定した場合は，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（KDJW61098-E）。

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。apt コマンドまたは cjwsgen 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。
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ます（KDJW61099-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

れます（KDJW61100-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ます（KDJW61101-E）。

- 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 13-25　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:// 日立 .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表示されません）。
2	Java の予約語でない文字列	http://hitachi.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://1hitachi.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表示されません）。

（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表示されません）。

（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。

14 標準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，標準仕様のサポート範囲について説明します。

14.1 WSDL 1.1 仕様のサポート範囲

14.2 WSDL 作成時の注意事項

14.3 JAX-WS 2.1 仕様のサポート範囲

14.4 ハンドラチェイン設定ファイルのサポート範囲

14.5 SAAJ 1.3 仕様のサポート範囲

14.6 WS-RM 1.2 仕様のサポート範囲

14.7 WS-RM Policy 1.2 仕様のサポート範囲

14.1 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，Cosminexus 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 2.1 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。

14.1.1 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，Cosminexus 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 2.1 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，Cosminexus XML Processor のエラーになります。

● 指定できる値については，「14.2(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，Cosminexus XML Processor のエラーになります。

● 指定できる値については，「12.1.1(2)　名前空間の条件」を参照してください。

14.1.2　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，Cosminexus XML Processor のエラーになります。
- 指定できる値については，「17.3(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，Cosminexus 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，「17.3(2)　location 属性（wsdl:import 要素）」を参照してください。

14.1.3　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）。

14.1.4 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，Cosminexus XML Processor のエラーになります。
- 指定できる値については，「14.2(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）。

14.1.5　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，「14.2　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，Cosminexus XML Processor のエラーになります。
- 指定できる値については，「12.1.5(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，Cosminexus 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。

14.1.6　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，「12.1.2(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 2.1 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，Cosminexus XML Processor のエラーになります。
- 指定できる値については，「12.1.2(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）。

14.1.7　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，「12.1.3(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 2.1 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，Cosminexus XML Processor のエラーになります。
- 指定できる値については，「12.1.3(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）。

14.1.8　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，「12.1.3(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，「27.5.6　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，Cosminexus XML Processor のエラーになります。
- 指定できる値については，「14.2(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，Cosminexus 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。

14.1.9　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。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，「12.1.3(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，「27.5.6　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，Cosminexus XML Processor のエラーになります。
- 指定できる値については，「14.2(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，Cosminexus 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。

14.1.10　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，「12.1.3(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 2.1 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，「27.5.6　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，Cosminexus XML Processor のエラーになります。
- 指定できる値については，「14.2(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，Cosminexus 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。

14.1.11　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 2.1 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，Cosminexus XML Processor のエラーになります。

● 指定できる値については，「14.2(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，Cosminexus 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。

14.1.12 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 2.1 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）。

（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，Cosminexus XML Processor のエラーになります。

● 指定できる値については，「14.2(1)　NCName 型に指定できる値」を参照してください。

14.1.13　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，Cosminexus XML Processor のエラーになります。
- 指定できる値については，「14.2(1)　NCName 型に指定できる値」を参照してください。

14.1.14　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，Cosminexus XML Processor のエラーになります。

● 指定できる値については，「14.2(1)　NCName 型に指定できる値」を参照してください。

14.1.15　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，Cosminexus XML Processor のエラーになります。
- 指定できる値については，「14.2(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）。

14.1.16　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 2.1 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，Cosminexus XML Processor のエラーになります。

● 指定できる値については，「12.1.9(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）。

14.1.17　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 2.1 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，Cosminexus XML Processor のエラーになります。

● 指定できる値については，「12.1.9(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，Cosminexus 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。

14.1.18　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）。

14.1.19　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，Cosminexus 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，Cosminexus 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）。

14.1.20 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，Cosminexus 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，Cosminexus 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）。

14.1.21 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，Cosminexus 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，Cosminexus 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，「14.2(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。

14.1.22　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，「14.2(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，Cosminexus 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，「14.2(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，Cosminexus 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，「14.2(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，Cosminexus 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）。

14.1.23　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，Cosminexus 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，「14.2(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，Cosminexus 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）。

14.1.24　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，Cosminexus XML Processor のエラーになります。
- 指定できる値については，「14.2(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。

14.1.25　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，Cosminexus 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，Cosminexus 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）

14.1.26 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，Cosminexus 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，Cosminexus 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）。

14.1.27 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，Cosminexus 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，Cosminexus 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，「14.2(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。

14.1.28　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，「14.2(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，Cosminexus 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，「14.2(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，Cosminexus 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，「14.2(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，Cosminexus 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）。

14.1.29　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，Cosminexus 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，「14.2(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，Cosminexus 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）。

14.1.30　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，Cosminexus XML Processor のエラーになります。
- 指定できる値については，「14.2(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。

14.1.31　xsd:schema 要素

xsd:schema 要素のサポート範囲を説明します。

- JAXB 2.1 仕様に従い Java 型にマッピングしますが，マッピングは Cosminexus 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 14-1　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）

○：記述できる。

×：記述できない。

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 14-1　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 14-2　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※ 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 14-2　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。

cjwsgen 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>
   ・・・・・・
```

14.2 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。

(1) NCName 型に指定できる値

Cosminexus 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。

(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。

(a) 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 14-3　soap:header 要素を記述しない場合の記述例

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="TestJaxWsService" ･･･>
  <wsdl:types>
    ･･･
  </wsdl:types>

  <wsdl:message name="jaxWsTest1Request">
    <wsdl:part name="RequestBodyPart" element="･･･"/>
  </wsdl:message>

  <wsdl:message name="jaxWsTest1Response">
    <wsdl:part name="ResponseBodyPart" element="･･･"/>
  </wsdl:message>

  <wsdl:portType name="TestJaxWs">
    <wsdl:operation name="jaxWsTest1">
      <wsdl:input message="tns:jaxWsTest1Request" name="jaxWsTest1Request"/>
      <wsdl:output message="tns:jaxWsTest1Response" name="jaxWsTest1Response"/>
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="TestJaxWsBinding" type="tns:TestJaxWs">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="jaxWsTest1">
      <soap:operation/>
      <wsdl:input name="jaxWsTest1Request">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="jaxWsTest1Response">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>

  <wsdl:service name="TestJaxWsService">
    ･･･
  </wsdl:service>
</wsdl:definitions>
```

soap:body要素にparts属性を記述する場合も，記述しない場合も，wsdl:message要素の子要素には，wsdl:part要素を1個だけ記述する

(b) SOAP ヘッダを定義する場合

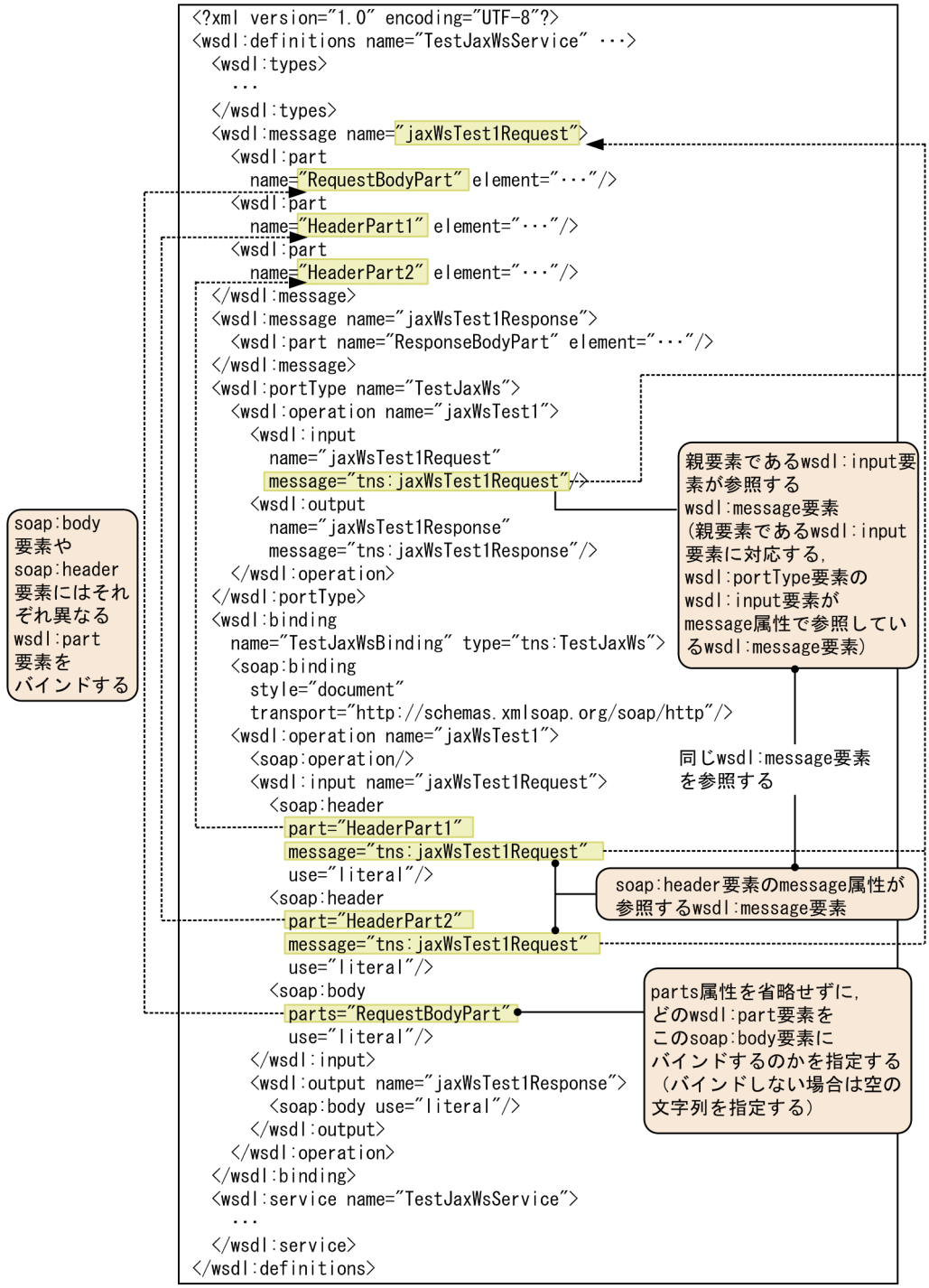
soap:header 要素を記述する場合，次に示す内容に従って定義してください。

- 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 14-4　soap:header 要素を記述する場合の記述例



（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※3＞
（例）http://hitachi.com/jaxws/service/UserInfoPort

●＜プロトコル※1＞://＜ホスト名※2＞:＜ポート番号※4＞/＜パス部分※3＞
（例）http://hitachi.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
RFC2396 仕様の規則に従った文字列を指定できます。また，IPv4 および IPv6 形式で指定できます。
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
RFC2396 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。

14.3　JAX-WS 2.1 仕様のサポート範囲

JAX-WS 2.1 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。

14.3.1　JAX-WS 2.1 仕様の機能のサポート範囲

JAX-WS 2.1 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，「1.3.2　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 14-3　JAX-WS 2.1 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，「12.2 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，「13.2.1　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，「12.1.1　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，「12.1.2　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，「12.1.3　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，「12.1.10(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 パターンのサポート	○	
2.3	MEP:one-way パターンのサポート	×	

分類			サポート	備考
大分類※1	小分類※2			
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	×	
		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，「15.4.11 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		×	
2.5	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，「12.1.7　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，「14.1.22 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，「12.1.9　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，「12.1.10(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，「13.2.1　アノテーション一覧」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		
3.1	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,「13.1.3 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。
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,「13.1.4 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,「13.1.5 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「13.1.6 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,「13.1.7 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,「13.1.8 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,「13.1.10(1) ジェネリクスの型削除」および「13.1.10(3) ジェネリクス型の制限」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			
3.10	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,「13.1.9 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,「15. JAX-WS API のサポート範囲」を参照してください。
5	サービス API		○	API のサポート範囲については,「15. JAX-WS 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,「15. JAX-WS 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,「13.2.1 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,「12.2 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,「26.4 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,「26.4 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,「26.9 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,「26.5 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。

分類			サポート	備考
大分類※1	小分類※2			
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，「26.9　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，「26.5　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，「15.5.1　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		○	
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，「15.5　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）

○：Cosminexus の JAX-WS 機能でサポートしています。

×：Cosminexus 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 2.1 仕様の該当個所（章節項）を示します。

注※ 2

JAX-WS 2.1 仕様の該当個所に記載されている内容を示します。

14.3.2 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 2.1 仕様の "Appendix A Conformance Requirements" に準じます。

表 14-4 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	○	
2.2	Customization required	○	バンディング宣言のサポート範囲については，「12.2 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，「13.2.1 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，「12.1.1(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，「17.3 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 2.1 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，「12.1.2(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	○	

分類			対応	備考
番号※1	タイトル※2			
2.10	Method naming		○	wsdl:operation 要素の name 属性に指定できる値については，「12.1.3(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，「12.1.5(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	use of JAXB annotations		○	アノテーションのサポート範囲については，「13.2.1　アノテーション一覧」を参照してください。
2.18	Non-wrapped parameter naming		○	wsdl:part 要素の name 属性に指定できる値については，「12.1.5(2)　パート名の条件」を参照してください。
2.19	Default mapping mode		○	
2.20	Disabling wrapper style		○	
2.21	Wrapped parameter naming		○	wrapper 子要素の name 属性に指定できる値については，「12.1.4(2)　wrapper 子要素名の条件」を参照してください。
2.22	Parameter name clash		○	
2.23	Using javax.xml.ws.RequestWrapper		○	
2.24	Using javax.xml.ws.ResponseWrapper		○	
2.25	Use of Holder		○	
2.26	Asynchronous mapping required		×	
2.27	Asynchronous mapping option		×	
2.28	Asynchronous method naming		－	非同期関連の機能はサポートしていません。
2.29	Asynchronous parameter naming		－	非同期関連の機能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			
2.30	Failed method invocation		―	非同期関連の機能はサポートしていません。
2.31	Response bean naming		―	非同期関連の機能はサポートしていません。
2.32	Asynchronous fault reporting		―	非同期関連の機能はサポートしていません。
2.33	Asynchronous fault cause		―	非同期関連の機能はサポートしていません。
2.34	JAXB class mapping		○	
2.35	JAXB customization use		○	
2.36	JAXB customization clash		○	
2.37	javax.xml.ws.wsaddressing.W3CEndpointReference		○	
2.38	javax.xml.ws.WebFault required		○	
2.39	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,「12.1.7(2)　フォルト名の条件」を参照してください。
2.40	Fault equivalence		○	
2.41	Fault equivalence		○	
2.42	Required WSDL extensions	SOAP	○	
		MIME	×	
2.43	Unbound message parts		○	
2.44	Duplicate headers in binding		○	
2.45	Duplicate headers in message		○	
2.46	Use of MIME type information		―	MIME バインディングはサポートしていません。
2.47	MIME type mismatch		―	MIME バインディングはサポートしていません。
2.48	MIME part identification		―	MIME バインディングはサポートしていません。
2.49	Service superclass required		○	
2.50	Service class naming		○	wsdl:service 要素の name 属性に指定できる値については,「12.1.9(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。

分類		対応	備考
番号※1	タイトル※2		
2.51	javax.xml.ws.WebServiceClient required	○	
2.52	—	○	
2.53	—	○	
2.54	Failed getPort Method	○	
2.55	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，「14.1 WSDL 1.1 仕様のサポート範囲」を参照してください。
3.2	Standard annotations	○	アノテーションのサポート範囲については，「13.2.1　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，「13.　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，「13.　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，Cosminexus 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，「13.　Java から WSDL へのマッピング」を参照してください。
3.12	One-way mapping	×	

分類		対応	備考
番号※1	タイトル※2		
3.13	One-way mapping errors	―	one-way パターンはサポートしていません。
3.14	use of JAXB annotations	○	
3.15	Parameter classification	○	
3.16	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,「13. Java から WSDL へのマッピング」を参照してください。
3.17	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,「13. Java から WSDL へのマッピング」を参照してください。
3.18	Header mapping of parameters and results	○	
3.19	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,「13. Java から WSDL へのマッピング」を参照してください。
3.20	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,「13. Java から WSDL へのマッピング」を参照してください。
3.21	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,「13. Java から WSDL へのマッピング」を参照してください。
3.22	Wrapper bean name clash	○	
3.23	Null Values in rpc/literal	―	rpc/literal スタイルはサポートしていません。
3.24	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,「13. Java から WSDL へのマッピング」を参照してください。
3.25	java.lang.RuntimeExceptions and java.rmi.RemoteExceptions	○	
3.26	Fault bean name clash	○	
3.27	Binding selection	○	

分類		対応	備考
番号※1	タイトル※2		
3.28	SOAP binding support	○	
3.29	SOAP binding style required	○	
3.30	Service creation	○	
3.31	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,「13.　Java から WSDL へのマッピング」を参照してください。
3.32	Port binding	○	
4.1	Service completeness	×	
4.2	Service Creation Failure	○	
4.3	Use of Executor	―	非同期関連の機能はサポートしていません。
4.4	Default Executor	―	非同期関連の機能はサポートしていません。
4.5	javax.xml.ws.BindingProvider.getEndpoint Reference	○	
4.6	Message context decoupling	○	
4.7	Required BindingProvider properties	○	
4.8	Optional BindingProvider properties	○	
4.9	Additional context properties	○	
4.10	Asynchronous response context	―	非同期関連の機能はサポートしていません。
4.11	Proxy support	○	
4.12	Implementing BindingProvider	○	
4.13	Service.getPort failure	○	
4.14	Remote Exceptions	○	
4.15	Exceptions During Handler Processing	○	
4.16	Other Exceptions	○	
4.17	Dispatch support	○	
4.18	Failed Dispatch.invoke	○	
4.19	Failed Dispatch.invokeAsync	―	非同期関連の機能はサポートしていません。
4.20	Failed Dispatch.invokeOneWay	―	非同期関連の機能はサポートしていません。
4.21	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。

分類		対応	備考
番号※1	タイトル※2		
4.22	Marshalling failure	○	
4.23	Use of the Catalog	×	
5.1	Provider support required	○	
5.2	Provider default constructor	○	
5.3	Provider implementation	○	
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。
6.1	Read-only handler chains	○	javax.xml.ws.Binding インタフェースの setHandlerChain メソッドはサポートしていません。
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	○	

分類		対応	備考
番号※1	タイトル※2		
6.7	One-way operations	―	one-way パターンはサポートしていません。
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。
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。
7.1	Correctness of annotations	○	アノテーションのサポート範囲については,「13.2.1　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,「13.2.1　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,「13.2.1　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,「15.　JAX-WS 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	○	

分類		対応	備考
番号※1	タイトル※2		
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	○	
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	×	one-way パターンはサポートしていません。
10.18	HTTP basic authentication support	○	
10.19	Authentication properties	○	
10.20	URL rewriting support	×	
10.21	Cookie support	○	

分類		対応	備考
番号※1	タイトル※2		
10.22	SSL session support	○	Conformance 自体には対応していますが，Cosminexus 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	×	
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 2.1 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。

14.4　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。

14.4.1　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）。

14.4.2　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）。

14.4.3 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）。

14.4.4 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）。

14.4.5 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）。

14.4.6 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）。

14.4.7 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）。

14.5　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。

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 14-5　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)	○
23		addChildElement(Name name)	○
24		addChildElement(SOAPElement element)	○

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
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)	○
54		setFaultString(String faultString, Locale locale)	○
55		addFaultReasonText(String text, Locale locale)	○
56		getFaultReasonLocales()	○
57		getFaultReasonText(Locale locale)	○

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
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）

○：Cosminexus 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 14-6　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)	×
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)	○

項番	クラス名	メソッド名／フィールド名	サポート
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)	○
35		createAttachmentPart(Object content, String contentType)	○
36		getAttachment(SOAPElement element)	○
37		getAttachments(MimeHeaders headers)	×

項番	クラス名	メソッド名／フィールド名	サポート
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）

○：Cosminexus の JAX-WS 機能でサポートしています。

×：Cosminexus 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。

14.5.1 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。

14.5.2 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。

14.5.3　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。

14.5.4　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。

14.5.5　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。

14.5.6　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，Cosminexus 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，Cosminexus 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，Cosminexus 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。

14.5.7 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。

14.5.8 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。

14.5.9 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。Cosminexus 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。

14.5.10 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。

14.5.11 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。

- RFC822 や RFC2045 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。

14.5.12 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。

14.5.13 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。

14.5.14　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。

14.5.15　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（RFC2392 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。

● Cosminexus 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。

14.5.16 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)

14.5.17 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 D.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 14-7　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

項番	添付ファイルの拡張子	対応する MIME タイプ
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);
```

14.6　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 14-8　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		×
		ExactryOnce		○
		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		○
		wsrm:Identifier		○
		wsrm:LastMsgNumber		○

分類			サポート
大分類	小分類		
		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	×
		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		×

分類		サポート
大分類	小分類	
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）

○：Cosminexus の WS-RM 1.2 機能でサポートしています。

×：Cosminexus の WS-RM 1.2 機能でサポートしていません。

△：Cosminexus 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

Cosminexus 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

Cosminexus 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。

14.7 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 14-9 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）

○：Cosminexus の WS-RM Policy 1.2 機能でサポートしています。

×：Cosminexus の WS-RM Policy 1.2 機能でサポートしていません。

注※

Cosminexus 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。

15 JAX-WS API 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 API のサポート範囲について説明します。

15.1 **インタフェースおよびクラスの一覧（JAX-WS）**

15.2 **クライアント** API

15.3 **サービス** API

15.4 **コア** API

15.5 **メッセージコンテキストの使用**

15.1　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。Cosminexus 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 15-1　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*	―	×
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	―	×
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ークラスです。	○
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ークラスです。	○
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	−	×
33	ServiceDelegate	−	×
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ークラスです。	○
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ークラスです。	○
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）。

○：Cosminexus の JAX-WS 機能でサポートしています。

×：Cosminexus の JAX-WS 機能でサポートしていません。

15.2　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。

15.2.1　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 2.1 仕様を参照してください。

表 15-2　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()		×
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。	

（凡例）

○：Cosminexus の JAX-WS 機能でサポートしています。

×：Cosminexus の JAX-WS 機能でサポートしていません。

15.2.2　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 2.1 仕様を参照してください。

表 15-3　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)	×

項番	戻り値の型	メソッド名	サポート
3	java.util.concurrent.Future<?>	invokeAsync(T msg, AsyncHandler<T> handler)	×
4	void	invokeOneWay(T msg)	×

（凡例）

○：Cosminexus の JAX-WS 機能でサポートしています。

×：Cosminexus の JAX-WS 機能でサポートしていません。

15.2.3　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 2.1 仕様を参照してください。

表 15-4　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。

○：Cosminexus の JAX-WS 機能でサポートしています。

×：Cosminexus の JAX-WS 機能でサポートしていません。

15.2.4　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 2.1 仕様を参照してください。

表 15-5　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。 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	

項番	戻り値の型	メソッド名／説明		サポート
2	void	addPort(javax.xml.namespace.QName portName, java.lang.String bindingId, java.lang.String endpointAddress)		○
		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。	
3	static Service	create(javax.xml.namespace.QName serviceName)		○
4	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）	
5	<T> Dispatch<T>	createDispatch (EndpointReference endpointReference, java.lang.Class<T> type, Service.Mode mode, WebServiceFeature... features)		×
6	Dispatch <java.lang.Object>	createDispatch (EndpointReference endpointReference, javax.xml.bind.JAXBContext context, Service.Mode mode, WebServiceFeature... features)		×
7	<T> Dispatch<T>	createDispatch (javax.xml.namespace.QName portName, java.lang.Class<T> type, 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。	
8	<T> Dispatch<T>	createDispatch (javax.xml.namespace.QName portName, java.lang.Class<T> type, Service.Mode mode, WebServiceFeature... features)		×
9	Dispatch <java.lang.Object>	createDispatch (javax.xml.namespace.QName portName, javax.xml.bind.JAXBContext context, 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	Dispatch <java.lang.Object>	createDispatch (javax.xml.namespace.QName portName, javax.xml.bind.JAXBContext context, Service.Mode mode, WebServiceFeature... features)		×
11	java.util.concurrent.Executor	getExecutor()		×
12	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。	
13	<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。	

項番	戻り値の型	メソッド名／説明		サポート
		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	
14	<T> T	getPort (java.lang.Class<T> serviceEndpointInterface, WebServiceFeature... features)		×
15	<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。 features： ポート上で構成する WebServiceFeature のリストです。	

項番	戻り値の型	メソッド名／説明		サポート
		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	
16	<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	
17	<T> T	getPort(javax.xml.namespace.QName portName, java.lang.Class<T> serviceEndpointInterface, WebServiceFeature... features)		×
18	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。	

項番	戻り値の型	メソッド名／説明		サポート
		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。	
19	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。	
20	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。	
21	void	setExecutor (java.util.concurrent.Executor executor)		×
22	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。

○：Cosminexus の JAX-WS 機能でサポートしています。

×：Cosminexus 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 15-6　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。

項番	指定方法	指定例	動作
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。

15.2.5　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 2.1 仕様を参照してください。

表 15-7　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。

○：Cosminexus の JAX-WS 機能でサポートしています。

15.3　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。

15.3.1　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 2.1 仕様を参照してください。

表 15-8　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）

○：Cosminexus の JAX-WS 機能でサポートしています。

15.3.2　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 2.1 仕様を参照してください。

表 15-9　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	W3CEndpointReference	build()	○
4	W3CEndpointReferenceBuilder	endpointName (javax.xml.namespace.QName endpointName)	○
5	W3CEndpointReferenceBuilder	metadata (org.w3c.dom.Element metadataElement)	○
6	W3CEndpointReferenceBuilder	referenceParameter (org.w3c.dom.Element referenceParameter)	○
7	W3CEndpointReferenceBuilder	serviceName (javax.xml.namespace.QName serviceName)	○

項番	戻り値の型	メソッド名	サポート
8	W3CEndpointReferenceBuilder	wsdlDocumentLocation (java.lang.String wsdlDocumentLocation)	○

（凡例）

－：戻り値の型がないことを示します。

○：Cosminexus の JAX-WS 機能でサポートしています。

15.4　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。

15.4.1　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 15-10　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()		○
		説明	StreamingAttachmentFeature の一意の識別子を取得します。	

項番	戻り値の型	メソッド名		サポート
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。	

項番	戻り値の型	メソッド名		サポート
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。

○：Cosminexus の JAX-WS 機能でサポートしています。

×：Cosminexus の JAX-WS 機能でサポートしていません。

15.4.2　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 15-11　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)	×

項番	戻り値の型	メソッド名		サポート
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()		○
		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。

○：Cosminexus の JAX-WS 機能でサポートしています。

×：Cosminexus の JAX-WS 機能でサポートしていません。

15.4.3 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 15-12　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。	
		引数	dir： 一時ファイルの出力先ディレクトリパスです。	

項番	戻り値の型	メソッド名		サポート
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。

○：Cosminexus の JAX-WS 機能でサポートしています。

×：Cosminexus の JAX-WS 機能でサポートしていません。

15.4.4　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 2.1 仕様を参照してください。

表 15-13　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）

○：Cosminexus の JAX-WS 機能でサポートしています。

15.4.5　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 2.1 仕様を参照してください。

表 15-14　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）

○：Cosminexus の JAX-WS 機能でサポートしています。

15.4.6　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 2.1 仕様を参照してください。

表 15-15　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）

○：Cosminexus の JAX-WS 機能でサポートしています。

15.4.7　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 2.1 仕様を参照してください。

表 15-16　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)		○
		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，およびユーザプログラムが追加したプロパティは，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，「15.5.1　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)		×

（凡例）

○：Cosminexus の JAX-WS 機能でサポートしています。

×：Cosminexus の JAX-WS 機能でサポートしていません。

15.4.8　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 2.1 仕様を参照してください。

表 15-17　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）

○：Cosminexus の JAX-WS 機能でサポートしています。

15.4.9　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 2.1 仕様を参照してください。

表 15-18　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）

○：Cosminexus の JAX-WS 機能でサポートしています。

15.4.10　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 2.1 仕様を参照してください。

表 15-19　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	

項番	戻り値の型	メソッド名／説明		サポート
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）

○：Cosminexus の JAX-WS 機能でサポートしています。

×：Cosminexus の JAX-WS 機能でサポートしていません。

15.4.11　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 2.1 仕様を参照してください。

表 15-20　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。

○：Cosminexus の JAX-WS 機能でサポートしています。

15.4.12　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 15-21　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()	○

項番	戻り値の型	メソッド名／説明	サポート
2	java.lang.Object	getPayload(javax.xml.bind.JAXBContext context)	○
3	void	setPayload(java.lang.Object payload, javax.xml.bind.JAXBContext context)	×
4	void	setPayload(javax.xml.transform.Source payload)	×

（凡例）

○：Cosminexus の JAX-WS 機能でサポートしています。

×：Cosminexus の JAX-WS 機能でサポートしていません。

15.4.13　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 2.1 仕様を参照してください。

表 15-22　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。

○：Cosminexus の JAX-WS 機能でサポートしています。

15.4.14　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 2.1 仕様を参照してください。

表 15-23　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()	○
2	—	AddressingFeature(boolean enabled)	○
3	—	AddressingFeature(boolean enabled, boolean required)	○
4	java.lang.String	getID()	○
5	boolean	isRequired()	○

（凡例）
－：戻り値の型がないことを示します。
○：Cosminexus の JAX-WS 機能でサポートしています。

15.4.15　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 2.1 仕様を参照してください。

表 15-24　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。	

項番	戻り値の型	メソッド名		サポート
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。

○：Cosminexus の JAX-WS 機能でサポートしています。

15.4.16　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 2.1 仕様を参照してください。

表 15-25　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。	

項番	戻り値の型	メソッド名		サポート
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）

○：Cosminexus の JAX-WS 機能でサポートしています。

×：Cosminexus の JAX-WS 機能でサポートしていません。

15.4.17　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 2.1 仕様を参照してください。

表 15-26　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。

○：Cosminexus の JAX-WS 機能でサポートしています。

15.4.18　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 2.1 仕様を参照してください。

表 15-27　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。

○：Cosminexus の JAX-WS 機能でサポートしています。

15.5　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，ハンドラおよび Web サービスクライアントからメッセージコンテキストにアクセスできます。

ハンドラの場合は，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は，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。

JAX-WS API については，JAX-WS 2.1 仕様を参照してください。また，Cosminexus の JAX-WS 機能での JAX-WS API のサポート範囲については，「15.1　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。Cosminexus の JAX-WS 機能では，JAX-WS 2.1 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 2.1 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。

15.5.1　メッセージコンテキストのプロパティのサポート範囲

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 15-28　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		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 ※5	in ※6	in ※7	out ※8
javax.xml.ws.handler.message									
1	.outbound ※9	9.4.1.1	○	×※9	×※9	△	△	△	△
javax.xml.ws.binding.attachments									
2	.inbound ※9	9.4.1.1	○	×※9	×※9	△	△	△	△
3	.outbound ※9	9.4.1.1	○	×※9	×※9	△	△	△	△
javax.xml.ws.reference									

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		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 ※5	in ※6	in ※7	out ※8
4	.parameters ※9	9.4.1.1	○	×※9	×※9	△	△	△	△
javax.xml.ws.wsdl									
5	.description ※9, ※17	9.4.1.1	—	×※9	×※9	×※10	×※10	×※10	×※10
6	.service ※9, ※17	9.4.1.1	—	×※9	×※9	△	△	△	△
7	.port ※9, ※17	9.4.1.1	—	×※9	×※9	△	△	△	△
8	.interface ※9, ※17	9.4.1.1	—	×※9	×※9	△	△	△	△
9	.operation ※9, ※17, ※18	9.4.1.1	—	×※9	△※18	△	△	△	△
javax.xml.ws.http.request									
10	.headers	9.4.1.1	○	◎※19	×※9	◎※19	△※11	△	△
11	.method ※9※10	9.4.1.1	○	×※9	×※9	△※11	△※11	△	△
12	.querystring ※9※10	9.4.1.1	○	×※9	×※9	△※11	△※11	△	△
13	.pathinfo ※9※10	9.4.1.1	○	×※9	×※9	△※11	△※11	△※12	△※12
javax.xml.ws.http.response									
14	.headers ※9	9.4.1.1	○	×※9	×※9	△※13	△	△※13	△※13
15	.code ※9	9.4.1.1	○	×※9	×※9	△※14	△	△	△
javax.xml.ws.servlet									
16	.context ※9※10	9.4.1.1	○	×※9	×※9	△※11	△※11	△	△
17	.request ※9	9.4.1.1	○	×※9	×※9	△※11	△※11	△	△

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		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 ※5	in ※6	in ※7	out ※8
18	.response ※9	9.4.1.1	○	×※9	×※9	△※11	△※11	△	△
javax.xml.ws.service.endpoint									
19	.address ※15	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	—	×※16	×※16	×※16	×※16	×※16	×※16
24	.uri	4.2.1.1	—	×※16	×※16	×※16	×※16	×※16	×※16
com.cosminexus.jaxws									
25	.connect.timeout			◎	△	◎	△	△	△
26	.request.timeout			◎	△	◎	△	△	△

（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：Cosminexus 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 2.1 仕様で定義されている個所を示します。

注※ 2

JAX-WS 2.1 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※ 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，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※ 7

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。

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，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。

注※ 9

「15.5.2(5)　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※ 10

常に null が返されます。

注※ 11

「15.5.2(1)　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※ 12

「15.5.2(2)　パス情報」を参照してください。

注※ 13

「15.5.2(3)　HTTP 応答ヘッダ」を参照してください。

注※ 14

「15.5.2(4)　HTTP ステータスコード」を参照してください。

注※ 15

「15.5.2(7)　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※ 16

「15.5.2(6)　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※ 17

「15.5.2(8)　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※ 18

「15.5.2(9)　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※ 19

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。

15.5.2　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。

(1) 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。

(2) 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。

(3) HTTP 応答ヘッダ

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。

(4) 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。

(5) 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，Cosminexus の JAX-WS 機能では，「15.5.1　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。

(6) 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

Cosminexus 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。

(7) 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，「14.2(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。

(8) 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。

(9) 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。

16 WS-RM 1.2 機能の API と設定

この章では，WS-RM 1.2 機能の API と設定について説明します。WS-RM 1.2 機能については，「24.　WS-RM 1.2 機能」を参照してください。

16.1　com.sun.xml.ws.Closeable クラス

16.2　WS-Policy による設定

16.1　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 16-1　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後に Web サービスメソッドを使用した場合に発生します。

16.2　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，「24.4　WS-RM Policy の追加方法」を参照してください。

表 16-2　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。 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

項番	プロパティ	説明	単位	範囲	デフォルト値
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。 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）。	－	1 ～ 100,000	10,000

（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>
```

17 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。

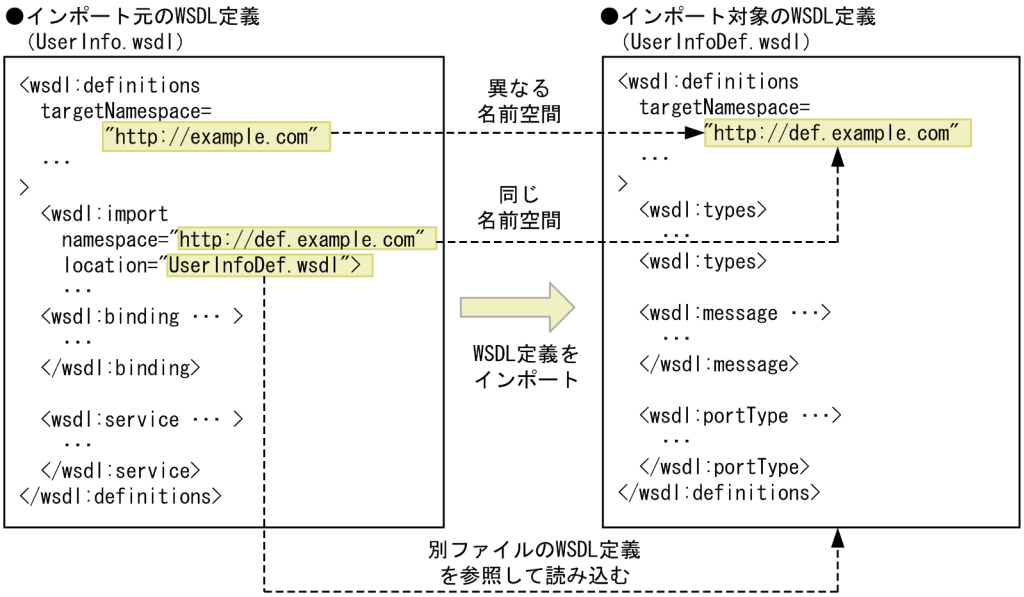
17.1　WSDL インポート機能とは
17.2　インポートできる WSDL 定義
17.3　wsdl:import 要素の書式

17.1　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 17-1　WSDL インポート機能のイメージ



17.2　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。

(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。

(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，「14.1.16　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 17-2　WSDL 定義の階層的なインポート（正しい場合の例）

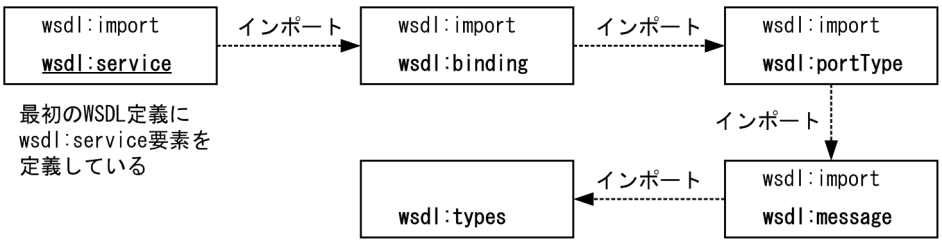
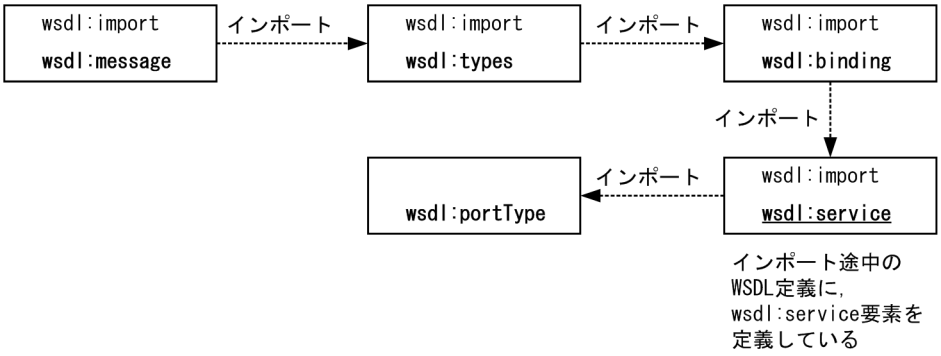


図 17-3　WSDL 定義の階層的なインポート（誤っている場合の例）

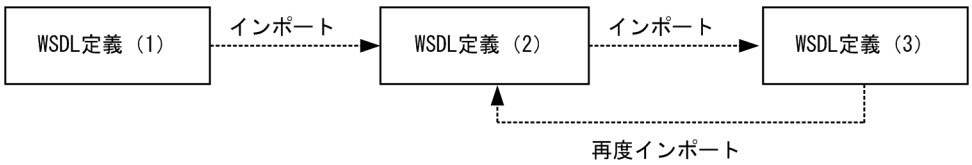


（3） WSDL 定義の再帰的なインポート

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 17-4　WSDL 定義の再帰的なインポートの例



17.3　wsdl:import 要素の書式

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 17-1　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。
- RFC2396 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。ただし，RFC2732（IPv6）は使用できません。
- 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 17-2　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）。

18 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。

18.1 添付ファイル機能とは（wsi:swaRef 形式）

18.2 添付ファイルの Java インタフェース（wsi:swaRef 形式）

18.3 添付ファイルの WSDL（wsi:swaRef 形式）

18.4 添付ファイル付き SOAP メッセージ（wsi:swaRef 形式）

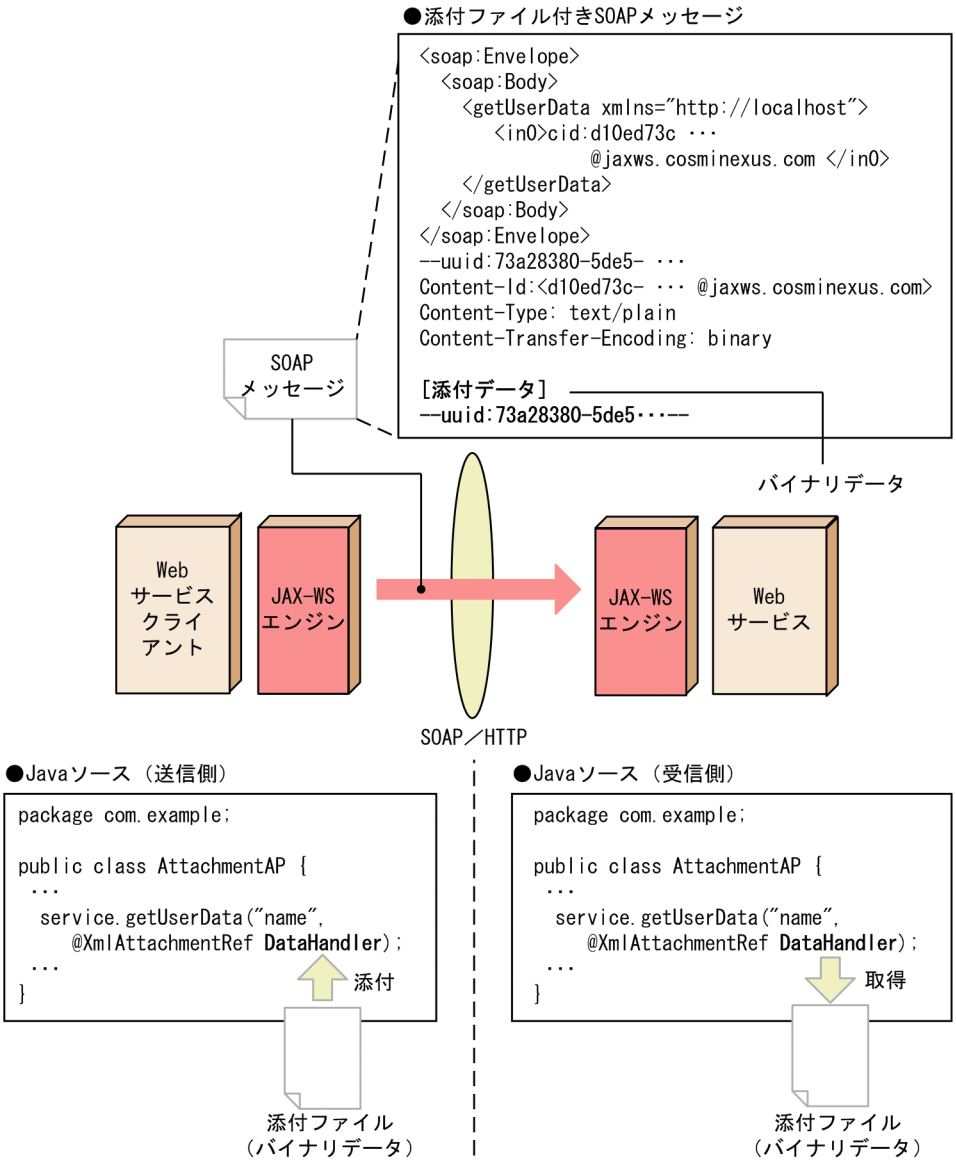
18.5 添付ファイルの Java インスタンスの生成と取得（wsi:swaRef 形式）

18.1　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 18-1　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，「14.5　SAAJ 1.3 仕様のサポート範囲」を参照してください。

18.2　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。

（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 18-1　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。

（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 18-2　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	○
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。

（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，「18.4.2(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。

18.3 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。

18.3.1 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，Cosminexus 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 18-3　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，Cosminexus 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，Cosminexus 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。

18.3.2　添付ファイルの Java 型と WSDL のマッピング（wsi:swaRef 形式）

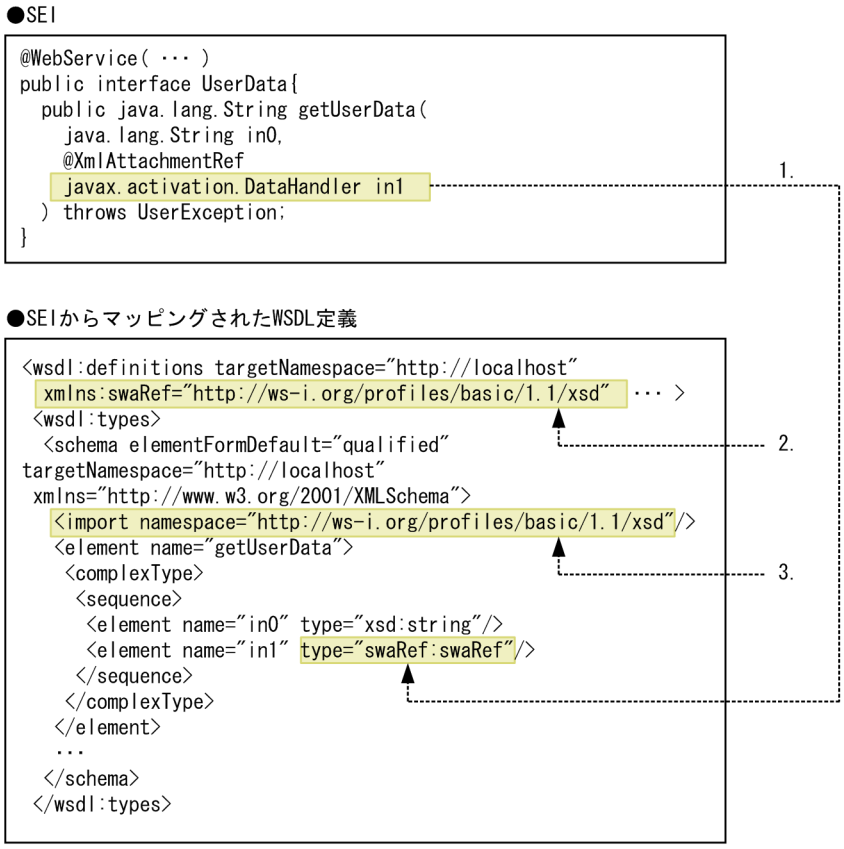
添付ファイルの Java 型から WSDL へのマッピング規則を次に示します。

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 18-2　添付ファイルの Java 型から WSDL へのマッピング例



18.3.3　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 18-3　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>
```

1.

●WSDLからマッピングされたSEI

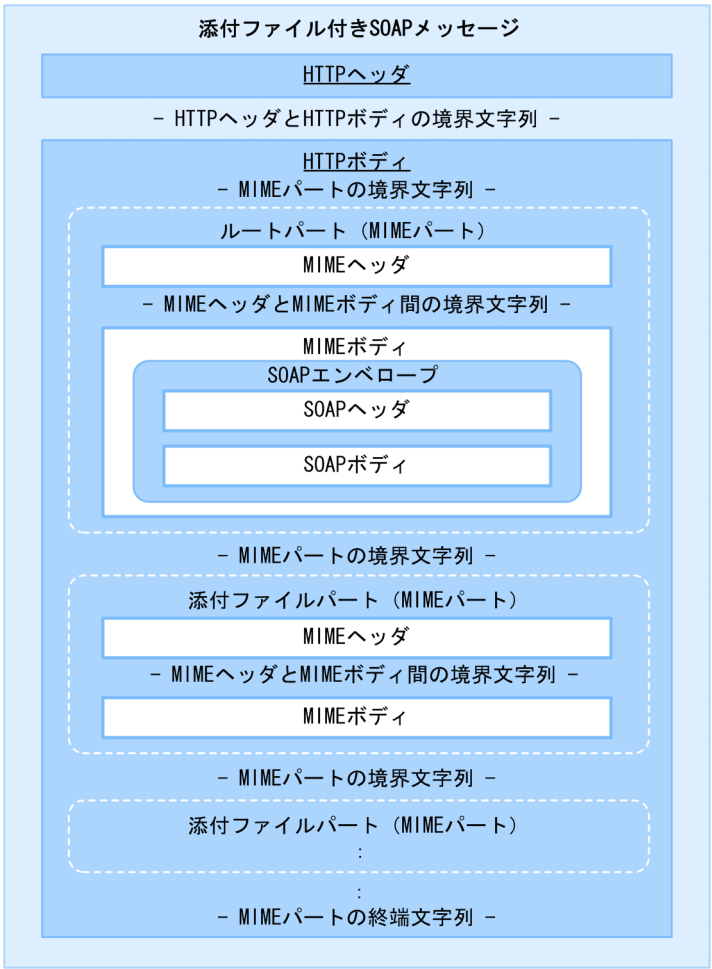
```
@WebService( ··· )
public interface UserData{
  @WebMethod
  public String getUserData(
    @WebParam(name = "in0", ··· )
    String in0,
    @WebParam(name = "in1", ··· )
    DataHandler in1
  ) throws UserException;
}
```

18.4　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 18-4　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 18-4　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。
├ 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。
├ 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。

18.4.1　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，「18.4.2　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 18-5 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 18-6 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 18-7　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，「18.4.2(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，「18.4.2(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。

18.4.2　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。

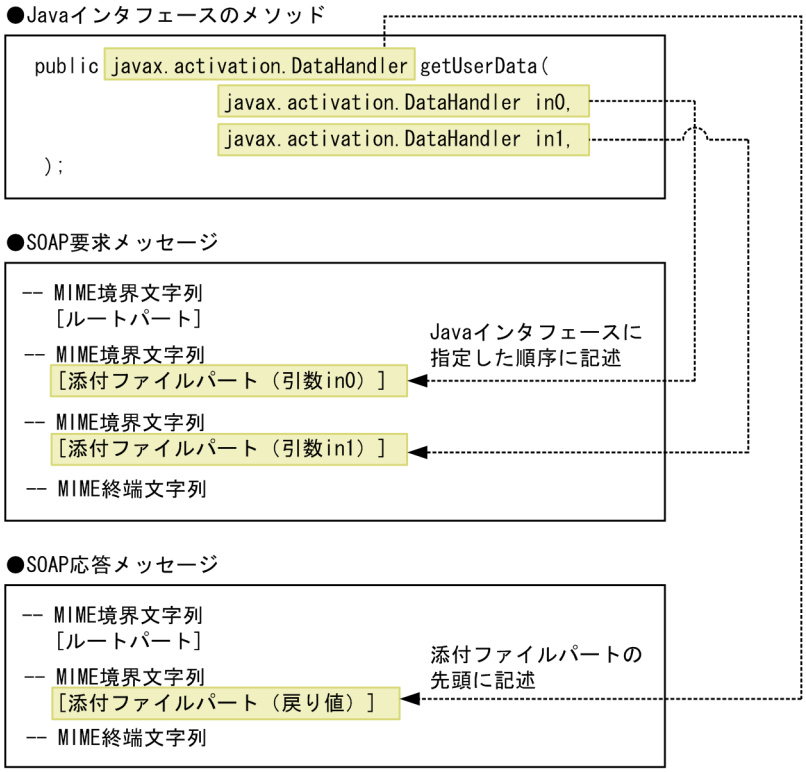
添付ファイルパートでは，Java インタフェースで指定された引数や戻り値がマッピングされます。Java インタフェースの指定内容と添付ファイルパートでの記述順序の対応を次の表に示します。

表 18-8　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。
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 18-5　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。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 18-9　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

項番	添付ファイルの拡張子	設定される MIME タイプ
19	qt，mov	video/quicktime
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。

18.4.3　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。

18.5　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。

18.5.1　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 18-10　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，「18.4.2(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, "text/plain; charset=UTF-8");
```

javax.activation.DataHandler(Object, String) コンストラクタの第 2 引数の記述形式を示します。

```
MIMEタイプ＋";"＋"charset"＋"="＋文字コード
```

注意事項

- javax.activation.DataHandler(Object, String) コンストラクタの第 1 引数に，日本語を含む java.lang.String オブジェクトを指定する場合は，第 2 引数にその MIME タイプと charset パラメタで適切な文字コードを指定する必要があります。
- charset パラメタを指定しない場合，添付ファイルの文字列がデフォルトの文字コード（US-ASCII）でエンコードされます。そのため，添付ファイルの文字列が US-ASCII 以外の文字を含んでいる場合，不正な添付ファイルが送信されます。添付ファイルとして String オブジェクトを送信する場合に指定できます。JPEG などのバイナリデータを送信する場合は指定できません。
- charset パラメタでは，大文字／小文字は区別されません。また，JDK がサポートしていない文字コードおよび空文字は指定できません。

18.5.2 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;
```

19 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。

19.1 開発例の構成（SEI 起点・wsi:swaRef 形式の添付ファイル）

19.2 開発例の流れ（SEI 起点・wsi:swaRef 形式の添付ファイル）

19.3 Web サービスの開発例（SEI 起点・wsi:swaRef 形式の添付ファイル）

19.4 デプロイと開始の例（SEI 起点・wsi:swaRef 形式の添付ファイル）

19.5 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）

19.6 Web サービスの実行例（SEI 起点・wsi:swaRef 形式の添付ファイル）

19.1　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 19-1　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 19-2　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 19-3　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。

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	「19.3.4　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。「19.3.1　Web サービス実装クラスを作成する」および「19.3.2　Java ソースを生成する」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	「19.3.3　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。「19.3.2　Java ソースを生成する」で使用します。
attachments.ear	「19.3.5　EAR ファイルを作成する」で作成します。
attachments.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。「19.5.1　サービスクラスを生成する」および「19.5.2　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。「19.5.3　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	「19.6.1　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	「19.6.2　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**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。

19.2 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（19.3.1）
2. apt コマンドを実行し，追加の Java ソースを生成する（19.3.2）
3. web.xml を作成する（19.3.3）
4. application.xml を作成する（19.3.4）
5. EAR ファイルを作成する（19.3.5）

デプロイと開始

1. EAR ファイルをデプロイする（19.4.1）
2. Web サービスを開始する（19.4.2）

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（19.5.1）
2. Web サービスクライアントの実装クラスを作成する（19.5.2）
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（19.5.3）

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（19.6.1）
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（19.6.2）
3. Web サービスクライアントを実行する（19.6.3）

19.3　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。

19.3.1　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="htt
p://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("Hitachi 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。

19.3.2　Java ソースを生成する

apt コマンドを実行して，Web サービス実装クラスから Web サービスの開発に必要な追加の Java ソースを生成します。また，Web サービス実装クラスを含めてコンパイルします。apt コマンドについては，「11.2　apt コマンド」を参照してください。

apt コマンドの実行例を次に示します。

● Windows(x86) の場合

```
> set HNTRLIB2_HOME=<HNTRLib2インストールディレクトリ>
> cd c:¥temp¥jaxws¥works¥attachments¥server¥
> mkdir WEB-INF¥classes¥
> apt -J-Dcosminexus.home="%COSMINEXUS_HOME%" -cp
"%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥client¥lib
¥j2ee-javax.jar;%COSMINEXUS_HOME%¥CC¥client¥lib¥HiEJBClientStatic.jar;%C
OSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxp.
jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar;%HNTRLIB2_HOME%¥classes¥hntrl
ib2j.jar;%HNTRLIB2_HOME%¥classes¥hntrlibMj.jar" -d WEB-INF¥classes¥ -s
src src¥com¥sample¥UserInfoImpl.java src¥com¥sample¥UserData.java
src¥com¥sample¥UserInfoException.java
```

● Windows(x64) の場合

```
> set HNTRLIB2_HOME=<HNTRLib2インストールディレクトリ>
> cd c:¥temp¥jaxws¥works¥attachments¥server¥
> mkdir WEB-INF¥classes¥
> apt -factory com.cosminexus.istack.ws.AnnotationProcessorFactoryImpl
-J-Dcosminexus.home="%COSMINEXUS_HOME%" -cp
"%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥client¥lib
¥j2ee-javax.jar;%COSMINEXUS_HOME%¥CC¥client¥lib¥HiEJBClientStatic.jar;%C
OSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxp.
jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar;%HNTRLIB2_HOME%¥classes¥hntrl
ib2j64.jar;%HNTRLIB2_HOME%¥classes¥hntrlibMj64.jar" -d WEB-INF¥classes¥
-s src src¥com¥sample¥UserInfoImpl.java src¥com¥sample¥UserData.java
src¥com¥sample¥UserInfoException.java
```

<HNTRLib2 インストールディレクトリ > の部分は次のコマンドの実行結果を指定しま

す。

● Windows(x86) の場合

```
> "%COSMINEXUS_HOME%¥common¥bin¥gethntr2conf.exe" HNTR2INSTDIR
```

● Windows(x64) の場合

```
> "%COSMINEXUS_HOME%¥common¥bin¥gethntr2conf64.exe" HNTR2INSTDIR
```

正常に終了すると，
c:¥temp¥jaxws¥works¥attachments¥server¥src¥com¥sample¥jaxws¥ 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 19-4　Java ソース生成時の生成物（SEI 起点・添付ファイル）

ファイル名	説明
GetUserData.java	getUserData メソッドに対応するリクエスト bean です。
GetUserDataResponse.java	getUserData メソッドに対応するレスポンス bean です。
UserInfoExceptionBean.java	UserInfoException に対応するフォルト bean です。

19.3.3　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="2.5" 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_2_5.xsd">
  <description>Sample web service &amp;quot;fromjava&amp;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>
```

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。

19.3.4　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="5" 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_5.xsd">

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

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「Cosminexus 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「5.2.4　application.xml 編集時の注意事項」を参照してください。

19.3.5　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.war .¥WEB-INF
> jar cvf attachments.ear .¥attachments.war .¥META-INF¥application.xml
```

正常に終了すると，c:¥temp¥jaxws¥works¥attachments¥server¥ ディレクトリに attachments.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。

19.4　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。

19.4.1　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.ear
```

cjimportapp コマンドについては，マニュアル「Cosminexus 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「Cosminexus 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。

19.4.2　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「Cosminexus 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「Cosminexus 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。

19.5　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。

19.5.1　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，「11.1 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/UserInfoImplService?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 19-5　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 2.1 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。

19.5.2 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。

19.5.3 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¥client¥lib
¥j2ee-javax.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。

19.6　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。

19.6.1　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=<Cosminexus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=<Cosminexusのインストールディレクトリ>
```

<Cosminexus のインストールディレクトリ > の部分は，Cosminexus 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「Cosminexus 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「15.2　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。

19.6.2　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「Cosminexus 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「15.3　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。

19.6.3　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:Hitachi 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「Cosminexus 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。

20 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。

20.1 **添付ファイル機能とは**（MTOM/XOP）

20.2 **添付ファイルの** Java **インタフェース**（MTOM/XOP）

20.3 **添付ファイルの** WSDL（MTOM/XOP）

20.4 JAX-WS **エンジンの動作**

20.5 MTOM/XOP **仕様形式の添付ファイルの** SOAP **メッセージ**

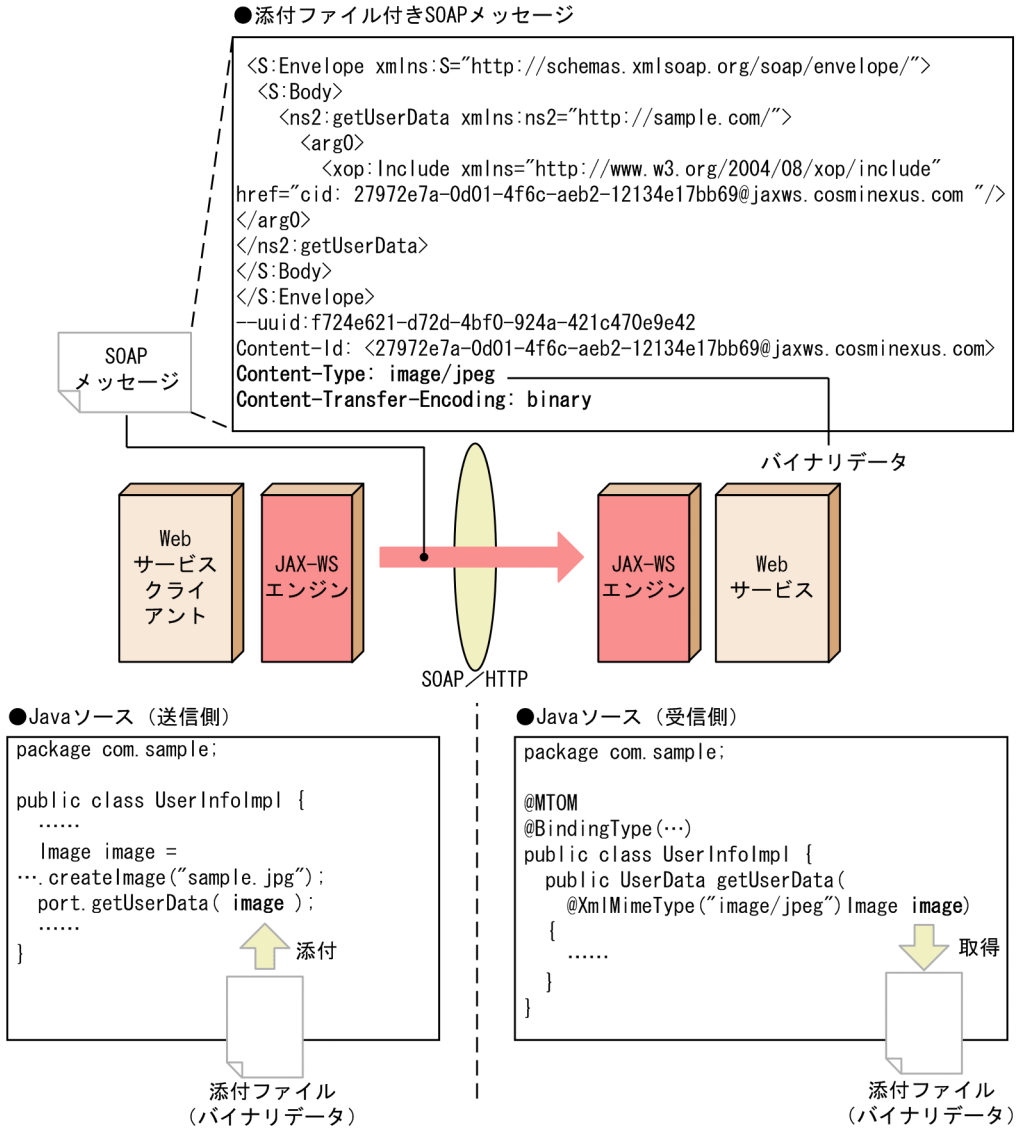
20.6 **注意事項**

20.1　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，バイナリデータを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 20-1　添付ファイル機能を使用したバイナリデータの送受信



20.2 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。

（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

複数の Base64 形式のデータがある場合，特定のデータだけを 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。

（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

（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 20-1 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
×：無効
－：なし

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 {
        ・・・・・・
    }
}
```

（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。

20.3　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。

20.3.1　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。

20.4　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。

20.4.1　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 20-2　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 20-3　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。

20.4.2　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 20-4　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）
－：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。

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。

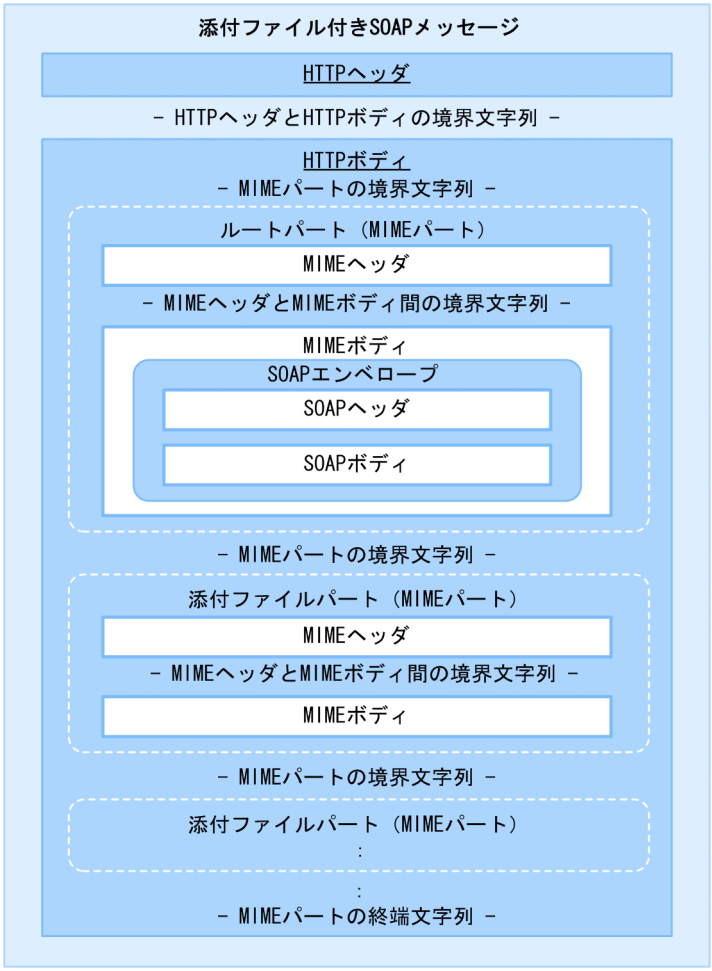
- xmime:expectedContentTypes 属性がある
  添付ファイルパートにある Content-Type フィールドの値は xmime:expectedContentTypes 属性に指定されている値となります。
- xmime:expectedContentTypes 属性がない
  添付ファイルパートにある Content-Type フィールドの値は "application/octet-stream" となります。

20.5　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 20-2　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 20-5　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。
├ 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。
├ 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。

20.5.1　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，「20.5.2　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 20-6　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 20-7　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。

項番	フィールド名	パラメタ名	設定値
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 20-8 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。

20.5.2 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。

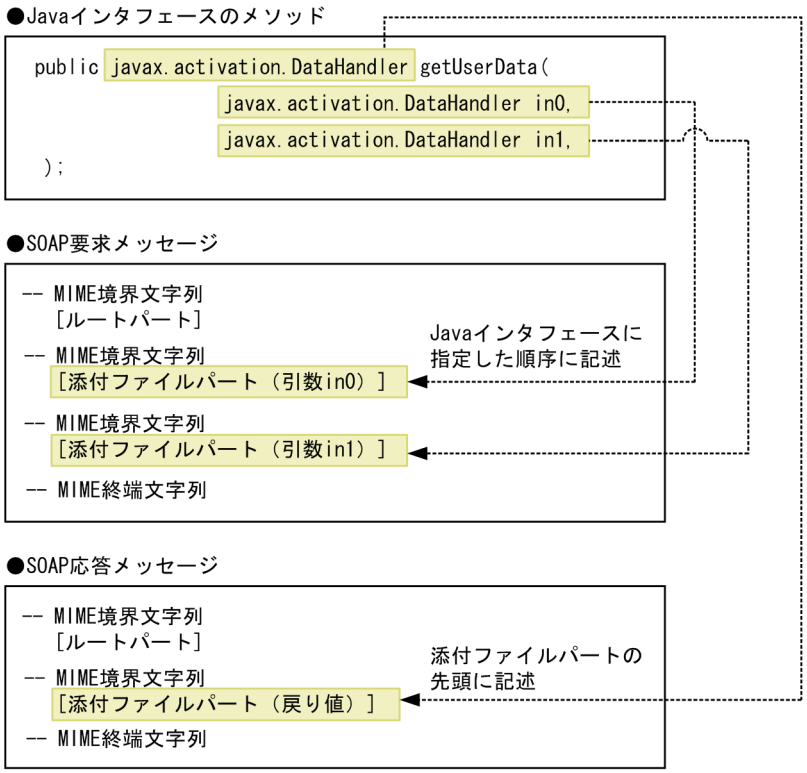
添付ファイルパートでは，Java インタフェースで指定された引数や戻り値がマッピングされます。Java インタフェースの指定内容と添付ファイルパートでの記述順序の対応を次の表に示します。

表 20-9　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。 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 20-3　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。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 20-10 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
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。

20.5.3 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。

20.6 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。
- WSS（Cosminexus 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。

21 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。

21.1 開発例の構成（SEI 起点・MTOM/XOP 仕様形式の添付ファイル）

21.2 開発例の流れ（SEI 起点・MTOM/XOP 仕様形式の添付ファイル）

21.3 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）

21.4 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）

21.5 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）

21.6 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）

21.1　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 21-1　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 21-2　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 21-3　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。

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	「21.3.4　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。 「21.3.1　Web サービス実装クラスを作成する」および 「21.3.2　Java ソースを生成する」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	「21.3.3　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。 「21.3.2　Java ソースを生成する」で使用します。
├ mtom.ear	「21.3.5　EAR ファイルを作成する」で作成します。
└ mtom.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。「21.5.1　サービスクラスを生成する」および 「21.5.2　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。 「21.5.3　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	「21.6.1　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	「21.6.2　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**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。

21.2 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（21.3.1）
2. apt コマンドを実行し，追加の Java ソースを生成する（21.3.2）
3. web.xml を作成する（21.3.3）
4. application.xml を作成する（21.3.4）
5. EAR ファイルを作成する（21.3.5）

デプロイと開始

1. EAR ファイルをデプロイする（21.4.1）
2. Web サービスを開始する（21.4.2）

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（21.5.1）
2. Web サービスクライアントの実装クラスを作成する（21.5.2）
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（21.5.3）

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（21.6.1）
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（21.6.2）
3. Web サービスクライアントを実行する（21.6.3）

21.3　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。

21.3.1　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("Hitachi 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。

21.3.2　Java ソースを生成する

apt コマンドを実行して，Web サービス実装クラスから Web サービスの開発に必要な追加の Java ソースを生成します。また，Web サービス実装クラスを含めてコンパイルします。apt コマンドについては，「11.2　apt コマンド」を参照してください。

apt コマンドの実行例を次に示します。

● Windows(x86) の場合

```
> set HNTRLIB2_HOME=<HNTRLib2インストールディレクトリ>
> cd c:¥temp¥jaxws¥works¥mtom¥server¥
> mkdir WEB-INF¥classes¥
> apt -J-Dcosminexus.home="%COSMINEXUS_HOME%" -cp
"%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥client¥lib
¥j2ee-javax.jar;%COSMINEXUS_HOME%¥CC¥client¥lib¥HiEJBClientStatic.jar;%C
OSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxp.
jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar;%HNTRLIB2_HOME%¥classes¥hntrl
ib2j.jar;%HNTRLIB2_HOME%¥classes¥hntrlibMj.jar" -d WEB-INF¥classes¥ -s
src src¥com¥sample¥UserInfoImpl.java src¥com¥sample¥UserData.java
src¥com¥sample¥UserInfoException.java
```

● Windows(x64) の場合

```
> set HNTRLIB2_HOME=<HNTRLib2インストールディレクトリ>
> cd c:¥temp¥jaxws¥works¥mtom¥server¥
> mkdir WEB-INF¥classes¥
> apt -factory com.cosminexus.istack.ws.AnnotationProcessorFactoryImpl
-J-Dcosminexus.home="%COSMINEXUS_HOME%" -cp
"%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥client¥lib
¥j2ee-javax.jar;%COSMINEXUS_HOME%¥CC¥client¥lib¥HiEJBClientStatic.jar;%C
OSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxp.
jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar;%HNTRLIB2_HOME%¥classes¥hntrl
ib2j64.jar;%HNTRLIB2_HOME%¥classes¥hntrlibMj64.jar" -d WEB-INF¥classes¥
-s src src¥com¥sample¥UserInfoImpl.java src¥com¥sample¥UserData.java
src¥com¥sample¥UserInfoException.java
```

<HNTRLib2 インストールディレクトリ > の部分は次のコマンドの実行結果を指定します。

● Windows(x86) の場合

```
> "%COSMINEXUS_HOME%¥common¥bin¥gethntr2conf.exe" HNTR2INSTDIR
```

● Windows(x64) の場合

```
> "%COSMINEXUS_HOME%¥common¥bin¥gethntr2conf64.exe" HNTR2INSTDIR
```

正常に終了すると，c:¥temp¥jaxws¥works¥mtom¥server¥src¥com¥sample¥jaxws¥ 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 21-4　Java ソース生成時の生成物（SEI 起点・添付ファイル）

ファイル名	説明
GetUserData.java	getUserData メソッドに対応するリクエスト bean です。
GetUserDataResponse.java	getUserData メソッドに対応するレスポンス bean です。
UserInfoExceptionBean.java	UserInfoException に対応するフォルト bean です。

21.3.3　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="2.5" 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_2_5.xsd">
  <description>Sample web service &amp;quot;mtom&amp;quot;</description>
  <display-name>Sample_web_service_mtom</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>
```

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。

21.3.4　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="5" 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_5.xsd">

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

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「Cosminexus 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「5.2.4　application.xml 編集時の注意事項」を参照してください。

21.3.5　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.war .¥WEB-INF
> jar cvf mtom.ear .¥mtom.war .¥META-INF¥application.xml
```

正常に終了すると，c:¥temp¥jaxws¥works¥mtom¥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。

21.4　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。

21.4.1　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.ear
```

cjimportapp コマンドについては，マニュアル「Cosminexus 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「Cosminexus 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。

21.4.2　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
```

cjstartapp コマンドについては，マニュアル「Cosminexus 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「Cosminexus 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。

21.5　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。

21.5.1　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，「11.1 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/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 21-5　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 2.1 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。

21.5.2 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。

21.5.3 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¥client¥lib
¥j2ee-javax.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。

21.6　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。

21.6.1　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=<Cosminexus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=<Cosminexusのインストールディレクトリ>
```

<Cosminexus のインストールディレクトリ > の部分は，Cosminexus 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「Cosminexus 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「15.2　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。

21.6.2　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「Cosminexus 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「15.3　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。

21.6.3　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:Hitachi 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「Cosminexus 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。

22 ストリーミング

この章では，Cosminexus 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。

22.1 **ストリーミングとは**

22.2 **ストリーミングの使用方法**

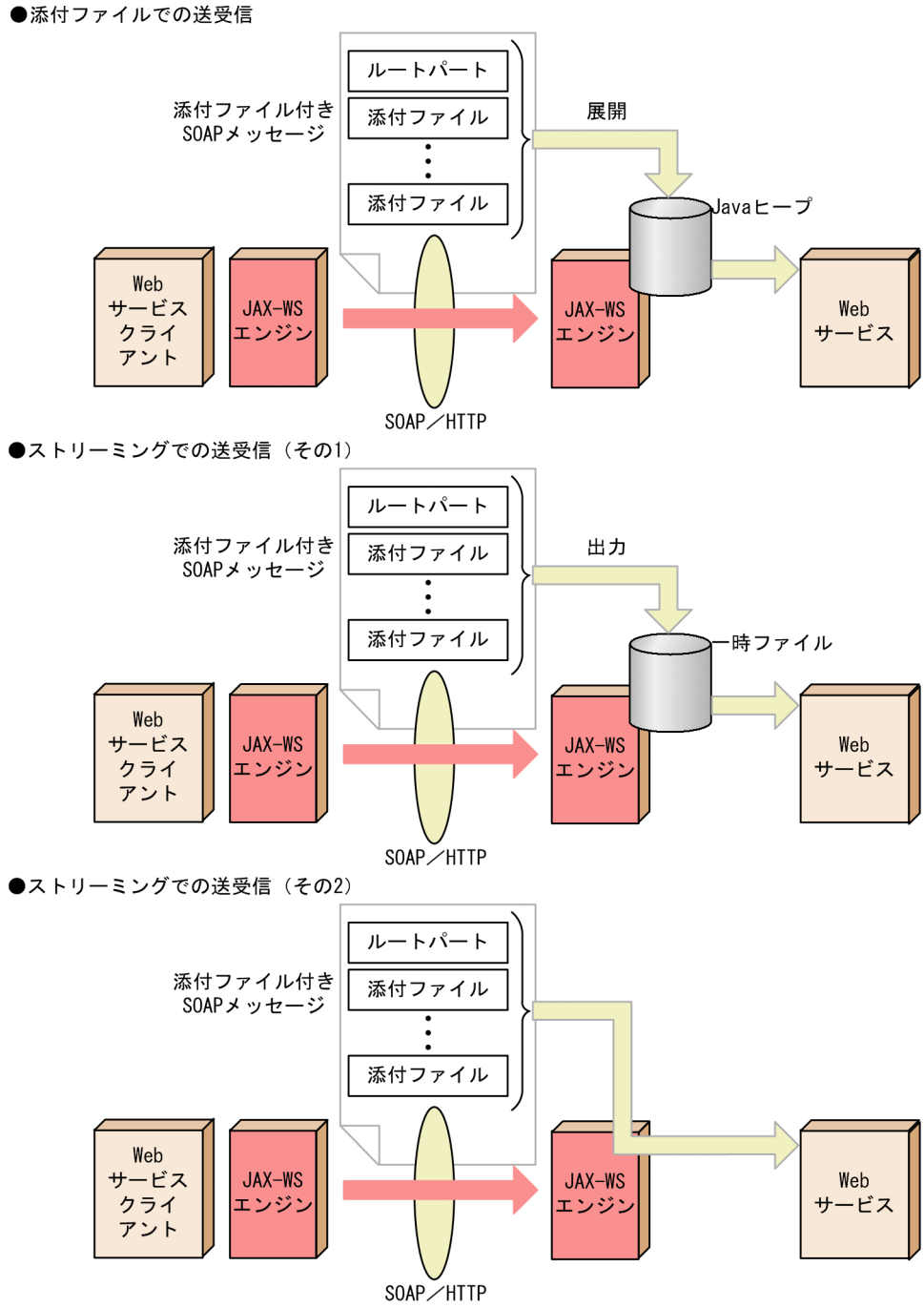
22.3 **一時ファイル（ストリーミング）**

22.1 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 22-1　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。

22.2 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。

22.2.1 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，「13.2.2 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。

22.2.2 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，「15.4.1 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
StreamingAttachmentFeature("C:/TMP", true, 50000L);

            UserInfoService service = new UserInfoService();
           UserInfoImpl port = service.getUserInfoImplPort(mtomFeature,
streamingAttachmentFeature);
```

```
            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();
        }
    }
}
```

22.2.3 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 22-1 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。

22.2.4 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，「15.4.2 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) {
                    ・・・・・・
                }
            }
        }
    }
}
```

注意事項

- 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 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。

22.3 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 に指定した値と 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 22-2 SOAP メッセージに含まれる MIME ボディの出力先

項番	parseEagerly に指定した値	MIME ボディのサイズ	MIME ボディの出力先
1	true	しきい値※より大きい	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※以下	一時ファイルに出力しないで，メモリ上に展開します。
3	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。

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。

22.3.1 命名規則

ストリーミング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 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。

ストリーミング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
```

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可能性があります。

- 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
- プロセスが異なる Web サービスクライアントでストリーミングを使用する場合
- 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，プロセスが異なる Web サービスクライアントでストリーミングを使用する場合にはクライアントごとに一時ファイルの出力先を変える必要があります。

22.3.2 出力と削除

一時ファイルは，添付ファイルを含む MIME Multipart/Related 構造の SOAP メッセージを受信したときに出力されます。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，ストリーミングされた添付ファイルをクローズするとき（StreamingDataHandler#close() メソッドの呼び出し時）に削除されます。

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可能性があります。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。

22.3.3 見積もり方法

ストリーミング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＞×＜ストリーミングを使用する 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＞

23 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。

23.1 **開発例の構成（SEI 起点・ストリーミング）**

23.2 **開発例の流れ（SEI 起点・ストリーミング）**

23.3 Web **サービスの開発例（SEI 起点・ストリーミング）**

23.4 **デプロイと開始の例（SEI 起点・ストリーミング）**

23.5 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）**

23.6 Web **サービスの実行例（SEI 起点・ストリーミング）**

23.1　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 2.1 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 23-1　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 23-2　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 23-3　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
5	スタイル	document/literal/wrapped

項番	項目		値
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
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 23-4　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	「23.3.4　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。 「23.3.1　Web サービス実装クラスを作成する」および 「23.3.2　Java ソースを生成する」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	「23.3.3　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。 「23.3.2　Java ソースを生成する」で使用します。
├ streaming.ear	「23.3.5　EAR ファイルを作成する」で作成します。
└ streaming.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。「23.5.1　サービスクラスを生成する」および「23.5.2　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。「23.5.3　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	「23.6.1　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	「23.6.2　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**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。

23.2　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（23.3.1）
2. apt コマンドを実行し，追加の Java ソースを生成する（23.3.2）
3. web.xml を作成する（23.3.3）
4. application.xml を作成する（23.3.4）
5. EAR ファイルを作成する（23.3.5）

デプロイと開始

1. EAR ファイルをデプロイする（23.4.1）
2. Web サービスを開始する（23.4.2）

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（23.5.1）
2. Web サービスクライアントの実装クラスを作成する（23.5.2）
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（23.5.3）

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（23.6.1）
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（23.6.2）
3. Web サービスクライアントを実行する（23.6.3）

23.3　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。

23.3.1　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")DataHandler 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("Hitachi 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。

23.3.2 Java ソースを生成する

apt コマンドを実行して，Web サービス実装クラスから Web サービスの開発に必要な追加の Java ソースを生成します。また，Web サービス実装クラスを含めてコンパイルします。apt コマンドについては，「11.2　apt コマンド」を参照してください。

apt コマンドの実行例を次に示します。

● Windows(x86) の場合

```
> set HNTRLIB2_HOME=<HNTRLib2インストールディレクトリ>
> cd c:¥temp¥jaxws¥works¥streaming¥server¥
> mkdir WEB-INF¥classes¥
> apt -encoding UTF-8 -J-Dcosminexus.home="%COSMINEXUS_HOME%" -cp
"%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥client¥lib
¥j2ee-javax.jar;%COSMINEXUS_HOME%¥CC¥client¥lib¥HiEJBClientStatic.jar;%C
OSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxp.
jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar;%HNTRLIB2_HOME%¥classes¥hntrl
ib2j.jar;%HNTRLIB2_HOME%¥classes¥hntrlibMj.jar" -d WEB-INF¥classes¥ -s
src src¥com¥sample¥UserInfoImpl.java src¥com¥sample¥UserData.java
src¥com¥sample¥UserInfoException.java
```

● Windows(x64) の場合

```
> set HNTRLIB2_HOME=<HNTRLib2インストールディレクトリ>
> cd c:¥temp¥jaxws¥works¥streaming¥server¥
> mkdir WEB-INF¥classes¥
> apt -factory com.cosminexus.istack.ws.AnnotationProcessorFactoryImpl
-J-Dcosminexus.home="%COSMINEXUS_HOME%" -cp
"%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥client¥lib
¥j2ee-javax.jar;%COSMINEXUS_HOME%¥CC¥client¥lib¥HiEJBClientStatic.jar;%C
OSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxp.
jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar;%HNTRLIB2_HOME%¥classes¥hntrl
ib2j64.jar;%HNTRLIB2_HOME%¥classes¥hntrlibMj64.jar" -d WEB-INF¥classes¥
-s src src¥com¥sample¥UserInfoImpl.java src¥com¥sample¥UserData.java
src¥com¥sample¥UserInfoException.java
```

<HNTRLib2 インストールディレクトリ > の部分は次のコマンドの実行結果を指定します。

● Windows(x86) の場合

```
> "%COSMINEXUS_HOME%¥common¥bin¥gethntr2conf.exe" HNTR2INSTDIR
```

● Windows(x64) の場合

```
> "%COSMINEXUS_HOME%¥common¥bin¥gethntr2conf64.exe" HNTR2INSTDIR
```

正常に終了すると，
c:¥temp¥jaxws¥works¥streaming¥server¥src¥com¥sample¥jaxws¥ 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 23-5　Java ソース生成時の生成物（SEI 起点・添付ファイル）

ファイル名	説明
GetUserData.java	getUserData メソッドに対応するリクエスト bean です。
GetUserDataResponse.java	getUserData メソッドに対応するレスポンス bean です。
UserInfoExceptionBean.java	UserInfoException に対応するフォルト bean です。

23.3.3　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="2.5" 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_2_5.xsd">
  <description>Sample web service &amp;quot;streaming&amp;quot;</description>
  <display-name>Sample_web_service_streaming</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>
```

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。

23.3.4　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="5" 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_5.xsd">

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

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「Cosminexus 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「5.2.4　application.xml 編集時の注意事項」を参照してください。

23.3.5　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.war .¥WEB-INF
> jar cvf streaming.ear .¥streaming.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.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。

23.4　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。

23.4.1　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.ear
```

cjimportapp コマンドについては，マニュアル「Cosminexus 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「Cosminexus 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。

23.4.2　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
```

cjstartapp コマンドについては，マニュアル「Cosminexus 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「Cosminexus 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。

23.5　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。

23.5.1　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，「11.1 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/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 23-6　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 2.1 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。

23.5.2 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。

23.5.3 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¥client¥lib
¥j2ee-javax.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。

23.6　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。

23.6.1　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=<Cosminexus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=<Cosminexusのインストールディレクトリ>
```

<Cosminexus のインストールディレクトリ > の部分は，Cosminexus 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「Cosminexus 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「15.2　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。

23.6.2　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「Cosminexus 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「15.3　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。

23.6.3　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:Hitachi 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「Cosminexus 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。

24 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。

24.1 WS-RM 1.2 機能とは

24.2 WS-RM 1.2 機能を使用したメッセージの流れ

24.3 WS-RM 1.2 機能の送達保証

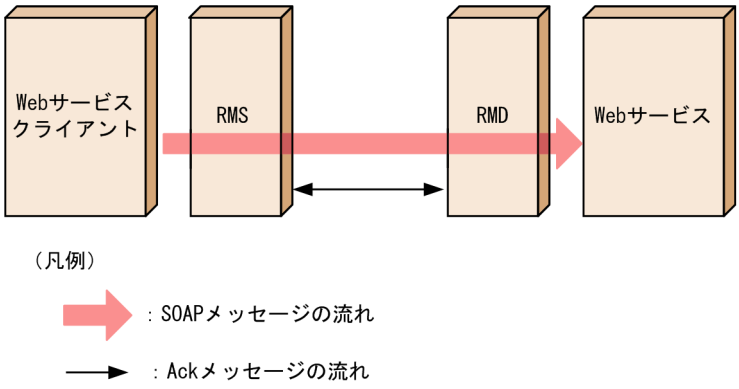
24.4 WS-RM Policy の追加方法

24.1　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 24-1　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



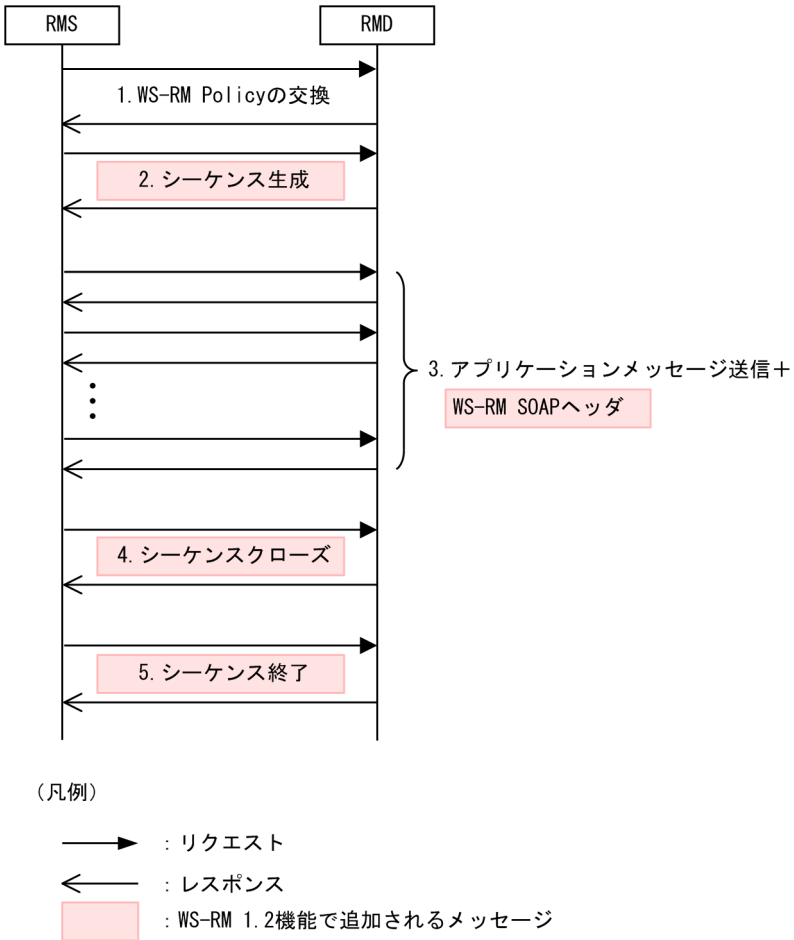
WS-RM 1.2 機能は，SOAP 1.2 メッセージにだけ対応しています。SOAP 1.1 メッセージには対応していません。また，Dispatch インタフェースおよび Provider インタフェースは利用できません。

08-70 より前のバージョンで提供していた WS-RM 機能（WS-RM 1.1 機能）とはソースコードとバイナリの互換性がありません。WS-RM 1.2 機能を使用するには，アプリケーションを作成し直してください。

24.2　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 24-2　WS-RM 1.2 機能を使用したメッセージの流れ



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。

24.3　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 Cosminexus 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 24-1　送達保証の種類と Cosminexus 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。

(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，最大 3 回まで，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。3 回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 202 を受信した場合

(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。

24.4 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。WS-RM Policy の追加方法を次に示します。

```
<?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"
  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"
  targetNamespace="http://example.com/sample">

 <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 -->

  <!-- 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">
    <wsp:PolicyReference URI="#WSRM_policy"/>

<!-- 後略 -->
```

太字で示す個所が，WS-RM Policy の定義または参照個所です。

WS-RM Policy の定義は wsdl:definitions 要素の子要素として記載します。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。

WS-RM Policy の参照個所は wsdl:binding の子要素として記載します。

「<!-- 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「16.2 WS-Policy による設定」を参照してください。

25 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。

25.1 開発例の構成（WSDL 起点・WS-RM 1.2）

25.2 開発例の流れ（WSDL 起点・WS-RM 1.2）

25.3 Web サービスの開発例（WSDL 起点・WS-RM 1.2）

25.4 デプロイと開始の例（WSDL 起点・WS-RM 1.2）

25.5 Web サービスクライアントの開発例（WSDL 起点・WS-RM 1.2）

25.6 Web サービスの実行例（WSDL 起点・WS-RM 1.2）

25.1　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 25-1　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_soap12
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 25-2　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_soap12	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。
server¥META-INF¥	EAR ファイルの META-INF ディレクトリに対応します。
server¥META-INF¥application.xml	「25.3.7　application.xml を作成する」で作成します。
server¥src¥	Web サービスのソースファイル（*.java）を格納します。「25.3.3　SEI を生成する」および「25.3.5　Web サービス実装クラスをコンパイルする」で使用します。
server¥WEB-INF¥	WAR ファイルの WEB-INF ディレクトリに対応します。
server¥WEB-INF¥web.xml	「25.3.6　web.xml を作成する」で作成します。
server¥WEB-INF¥classes¥	コンパイルしたクラスファイル（*.class）を格納します。
server¥WEB-INF¥wsdl¥	作成した wsdl を格納します。
server¥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。
server¥temporary¥src¥	
server¥temporary¥classes¥	
server¥wsrm_soap12.ear	「25.3.8　EAR ファイルを作成する」で作成します。
server¥wsrm_soap12.war	
client¥	Web サービスクライアントの開発で使用します。
client¥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。「25.5.1　サービスクラスを生成する」および「25.5.2　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。
client¥classes¥	コンパイルしたクラスファイル（*.class）を格納します。「25.5.4　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。
client¥usrconf.cfg	「25.6.1　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。
client¥usrconf.properties	「25.6.2　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**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。

25.2　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（25.3.1）
2. WSDL ファイルに WS-RM Policy を追加する（25.3.2）
3. cjwsimport コマンドを実行し，SEI を生成する（25.3.3）
4. Web サービス実装クラスを作成する（25.3.4）
5. Web サービス実装クラスをコンパイルする（25.3.5）
6. web.xml を作成する（25.3.6）
7. application.xml を作成する（25.3.7）
8. EAR ファイルを作成する（25.3.8）

デプロイと開始

1. EAR ファイルをデプロイする（25.4.1）
2. Web サービスを開始する（25.4.2）

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（25.5.1）
2. Web サービスクライアントの実装クラスを作成する（25.5.2）
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（25.5.3）
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（25.5.4）

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（25.6.1）
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（25.6.2）
3. Web サービスクライアントを実行する（25.6.3）

25.3　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。

25.3.1　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，「14.1　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「Cosminexus 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_soap12¥server¥WEB-INF¥wsdl¥ ディレクトリに保存してください。

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_soap12/
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，cjwsgen 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 アノテーションと 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。

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_BINDI
NG)
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_soap12¥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_soap12¥server¥
> mkdir .¥temporary
> mkdir .¥temporary¥classes
> javac -encoding UTF-8 -cp
"%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥client¥lib
¥j2ee-javax.jar" -d .¥temporary¥classes
.¥temporary¥src¥com¥example¥sample¥TestJaxWsImpl.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_soap12¥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，cjwsgen コマンドの WSDL 生成機能で WSDL ファイルを作成します。

cjwsgen コマンドの実行例を次に示します。

```
> cd c:¥temp¥jaxws¥works¥wsrm_soap12¥server¥
> mkdir .¥WEB-INF¥wsdl¥
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsgen.bat" -r .¥WEB-INF¥wsdl -d
.¥temporary¥classes -cp .¥temporary¥classes com.example.sample.TestJaxWsImpl
```

cjwsgen コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥wsrm_soap12¥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_soap12¥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:tns="http://example.com/sample"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://
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_soap12/
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://example.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_soap12¥server¥WEB-INF¥wsdl¥ ディレクトリに保存してください。

25.3.2　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"
  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"
  targetNamespace="http://example.com/sample">

 <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 -->

  <!-- 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">
  <wsp:PolicyReference URI="#WSRM_policy"/>
    <!-- document/literal/wrapped -->

<!-- 後略 -->
```

25.3.3　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，「11.1　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_soap12¥server¥
> mkdir src¥
> mkdir WEB-INF¥classes¥
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" -generateService -s src -d
WEB-INF¥classes WEB-INF¥wsdl¥input.wsdl
```

cjwsimport コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥wsrm_soap12¥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，「12.1.1　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 25-3　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 2.1 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，「12.　WSDL から Java へのマッピング」を参照してください。

25.3.4　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://example.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。

25.3.5　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_soap12¥server¥
> javac -encoding UTF-8 -cp
"%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥client¥lib
¥j2ee-javax.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_soap12¥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。

25.3.6　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="2.5" 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_2_5.xsd">
  <description>Sample web service &quot;wsrm_soap12&quot;</description>
  <display-name>Sample_web_service_wsrm_soap12</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>
```

作成した web.xml は，UTF-8 形式で
c:¥temp¥jaxws¥works¥wsrm_soap12¥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。

25.3.7　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="5" 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_5.xsd">

  <description>Sample application &quot;wsrm_soap12&quot;</description>
  <display-name>Sample_application_wsrm_soap12</display-name>
  <module>
    <web>
      <web-uri>wsrm_soap12.war</web-uri>
      <context-root>wsrm_soap12</context-root>
    </web>
  </module>
</application>
```

作成した application.xml は，UTF-8 形式で
c:¥temp¥jaxws¥works¥wsrm_soap12¥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「Cosminexus

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「5.2.4　application.xml 編集時の注意事項」を参照してください。

25.3.8　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_soap12¥server¥
> jar cvf wsrm_soap12.war .¥WEB-INF
> jar cvf wsrm_soap12.ear .¥wsrm_soap12.war .¥META-INF¥application.xml
```

jar コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥wsrm_soap12¥server¥ ディレクトリに wsrm_soap12.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。

25.4　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。

25.4.1　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_soap12¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjimportapp" jaxwsserver -nameserver
corbaname::testserver:900 -f wsrm_soap12.ear
```

cjimportapp コマンドについては，マニュアル「Cosminexus 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「Cosminexus 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。

25.4.2　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_soap12¥server¥
> "%COSMINEXUS_HOME%¥CC¥admin¥bin¥cjstartapp" jaxwsserver -nameserver
corbaname::testserver:900 -name Sample_application_wsrm_soap12
```

cjstartapp コマンドについては，マニュアル「Cosminexus 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「Cosminexus 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。

25.5　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。

25.5.1　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，「11.1 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_soap12¥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_soap12¥client¥
> mkdir src¥
> mkdir classes¥
> "%COSMINEXUS_HOME%¥jaxws¥bin¥cjwsimport.bat" -s src -d classes http://
webhost:8085/wsrm_soap12/TestJaxWsService?wsdl
```

正常に終了すると，c:¥temp¥jaxws¥works¥wsrm_soap12¥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，「12.1.1　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 25-4　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。

ファイル名	説明
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 2.1 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。
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。

- 「12.1.2　ポートタイプから SEI 名へのマッピング」
- 「12.1.3　オペレーションからメソッド名へのマッピング」
- 「12.1.4　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）」
- 「12.1.5　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）」

25.5.2　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_soap12¥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。

25.5.3　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();
       }
    }
     }

}
```

25.5.4　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_soap12¥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¥example¥sample¥client¥TestClient.java
```

javac コマンドが正常に終了すると，c:¥temp¥jaxws¥works¥wsrm_soap12¥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。

25.6　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。

25.6.1　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=<Cosminexus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=<Cosminexusのインストールディレクトリ>
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

<Cosminexus のインストールディレクトリ > の部分は，Cosminexus 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_soap12¥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「Cosminexus 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「15.2　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。

25.6.2　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_soap12¥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「Cosminexus 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「15.3　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。

25.6.3　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_soap12¥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_soap12¥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「Cosminexus 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。

26 ハンドラフレームワーク

Cosminexus の JAX-WS 機能では，JAX-WS 2.1 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。

26.1 ハンドラフレームワークとは

26.2 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

26.3 EJB の Web サービスに適用する場合の注意事項

26.4 ハンドラの型

26.5 ハンドラチェインの編成と実行順序

26.6 ハンドラの初期化と破棄

26.7 SOAP ヘッダが含まれる場合のハンドラの動作と設定

26.8 ハンドラの配置

26.9 ハンドラチェインの設定

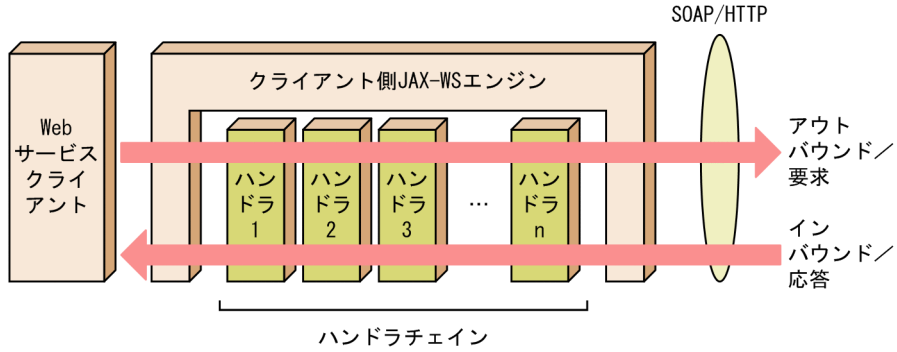
26.1　ハンドラフレームワークとは

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。

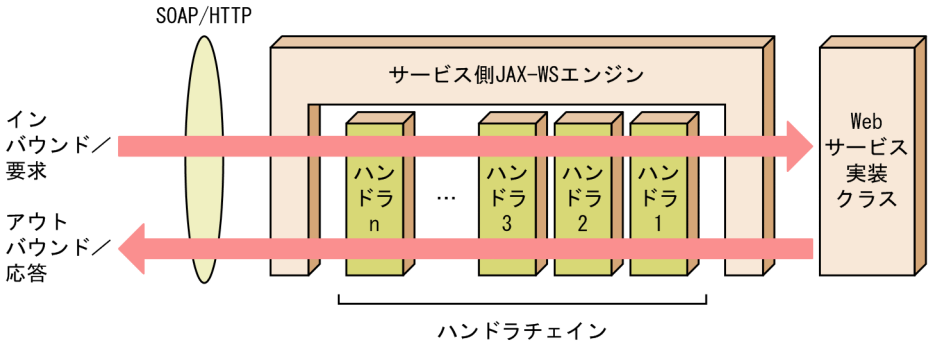
ハンドラフレームワークの処理の流れを次の図に示します。

図 26-1　ハンドラフレームワークの処理の流れ

●Webサービスクライアントの送受信



●Webサービス実装クラスの送受信



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 26-1 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。 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。
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。

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，「15.5　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 2.1 仕様を参照してください。

26.2 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 Cosminexus 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。

Cosminexus 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「Cosminexus アプリケーションサーバ Web サービスセキュリティ 使用の手引」を参照してください。

26.3　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。

26.4 ハンドラの型

JAX-WS 2.1 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

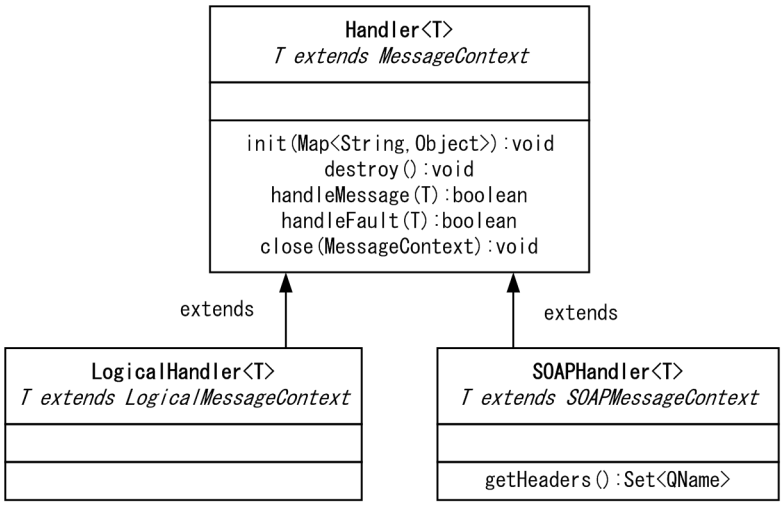
javax.xml.ws.handler.LogicalHandler インタフェースを除く，javax.xml.ws.handler.Handler のすべての継承インタフェースを実装するハンドラです。

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 26-2 ハンドラのクラス階層



Cosminexus 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。

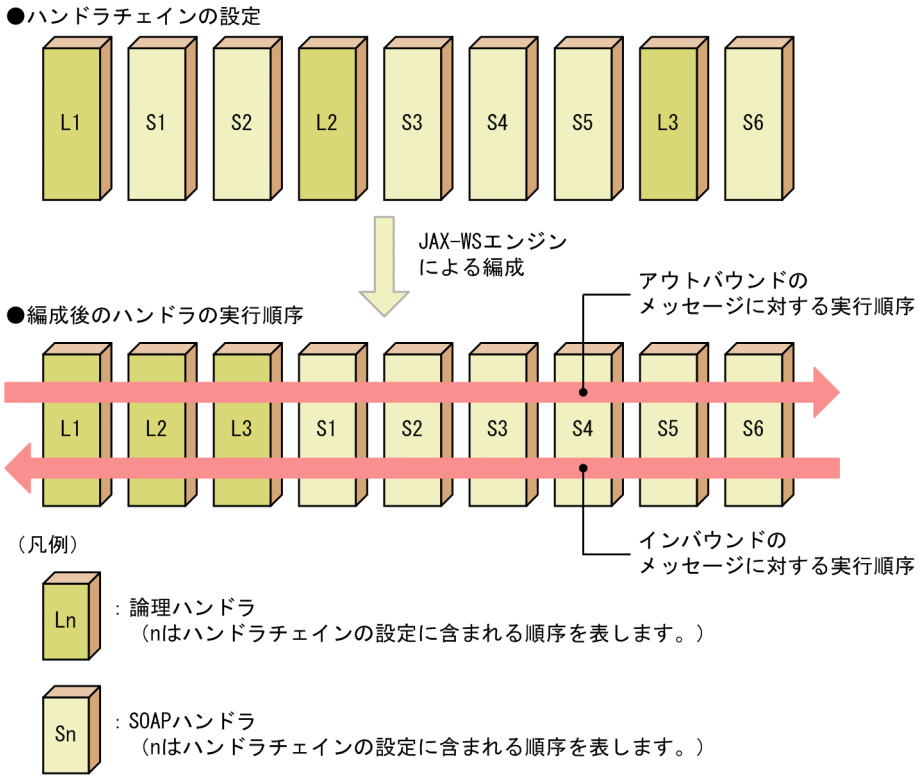
javax.xml.ws.handler.LogicalHandler インタフェースおよび
javax.xml.ws.handler.soap.SOAPHandler インタフェースの両方を実装するハンドラ
（論理ハンドラでも SOAP ハンドラでもあるハンドラ）は論理ハンドラと見なされます。

26.5　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 26-3　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。

26.5.1　handleMessage メソッドの処理

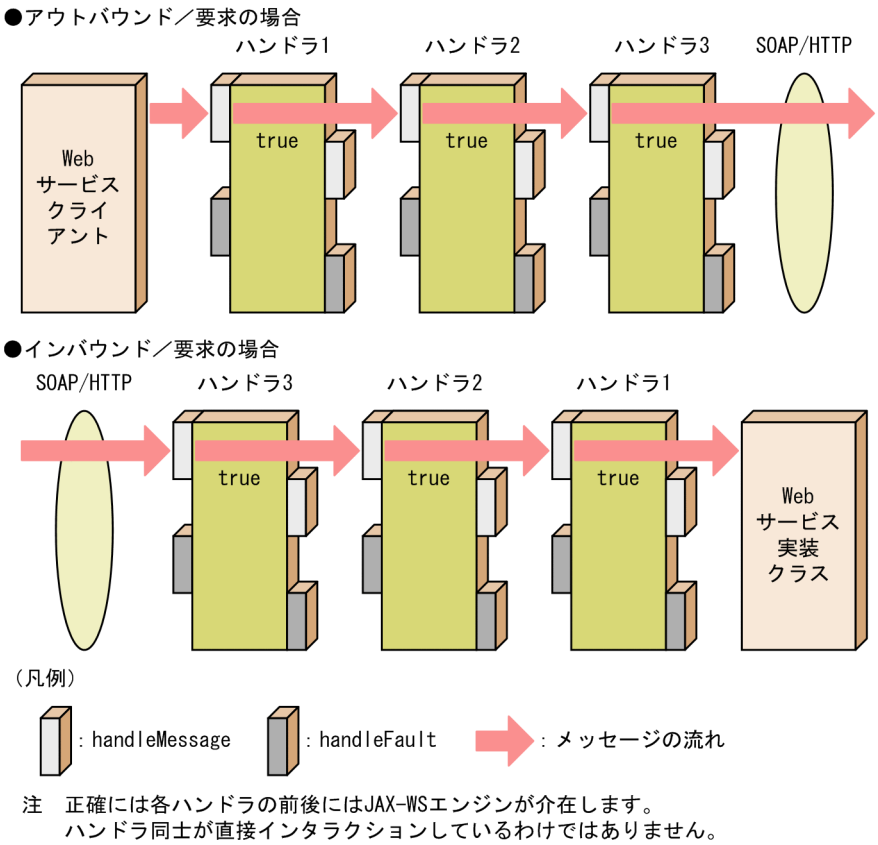
handleMessage メソッドについて，true を返す場合，false を返す場合，および例外をスローする場合の処理について説明します。

（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。

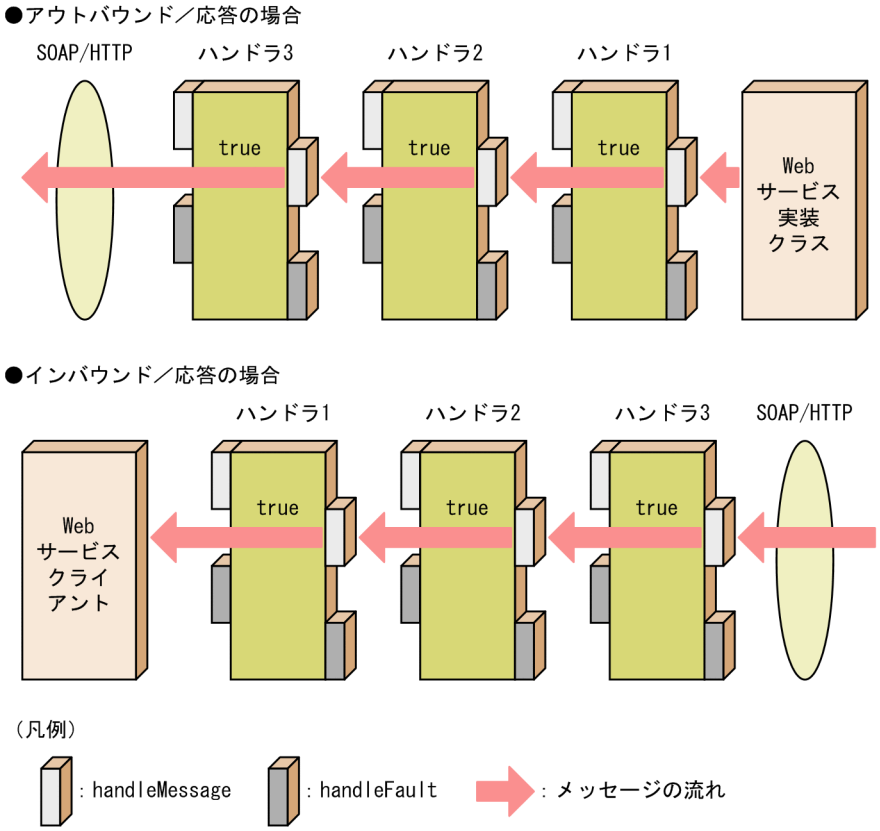
図 26-4　handleMessage で true を返す場合の処理（要求）



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。

応答処理の流れを次の図に示します。

図 26-5　handleMessage で true を返す場合の処理（応答）



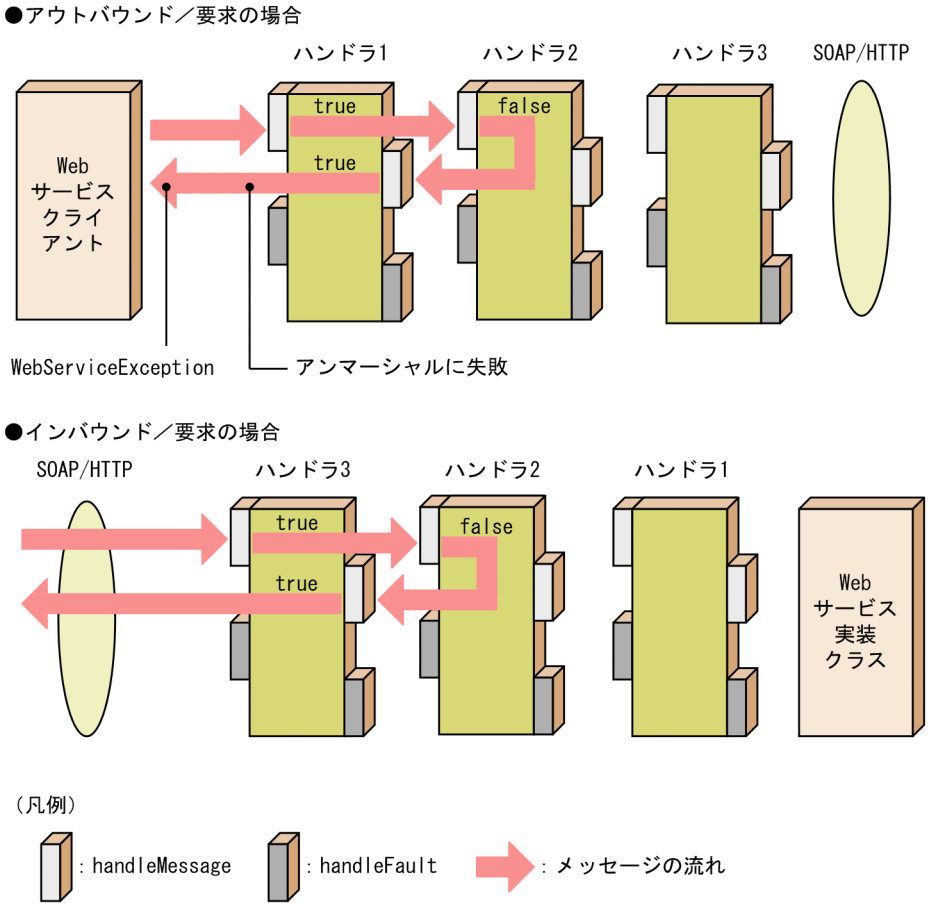
注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

(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 26-6　handleMessage で false を返す場合の処理（要求）



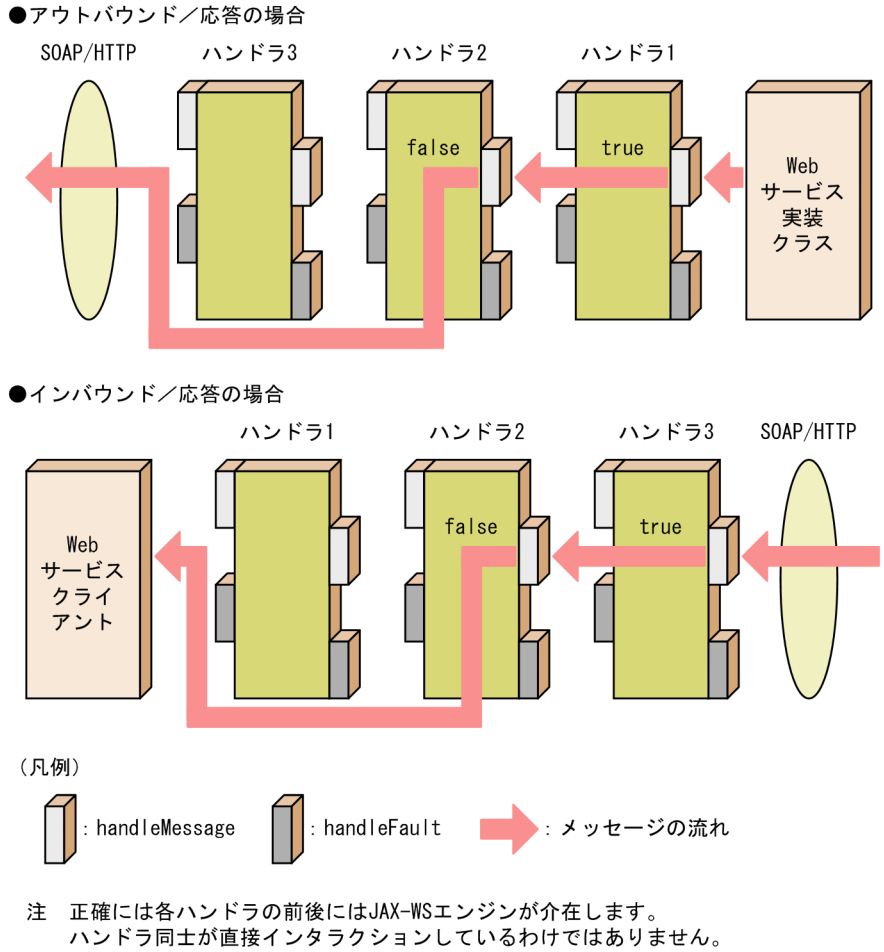
注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

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 26-7　handleMessage で false を返す場合の処理（応答）



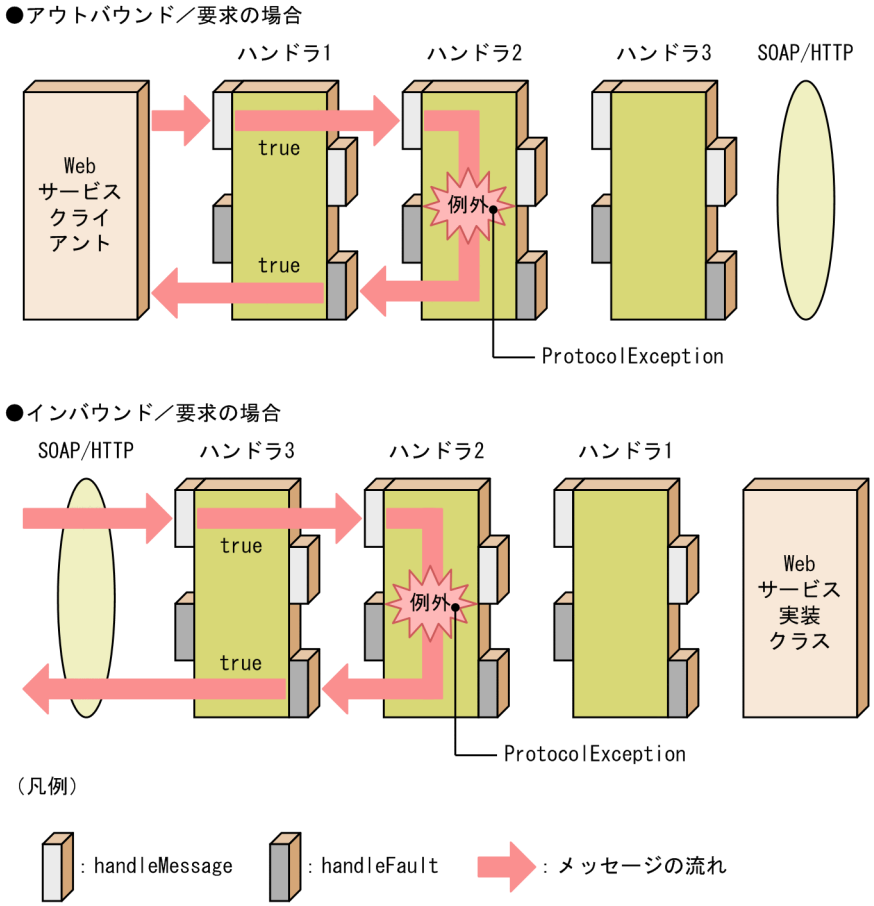
注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

(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 26-8　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。

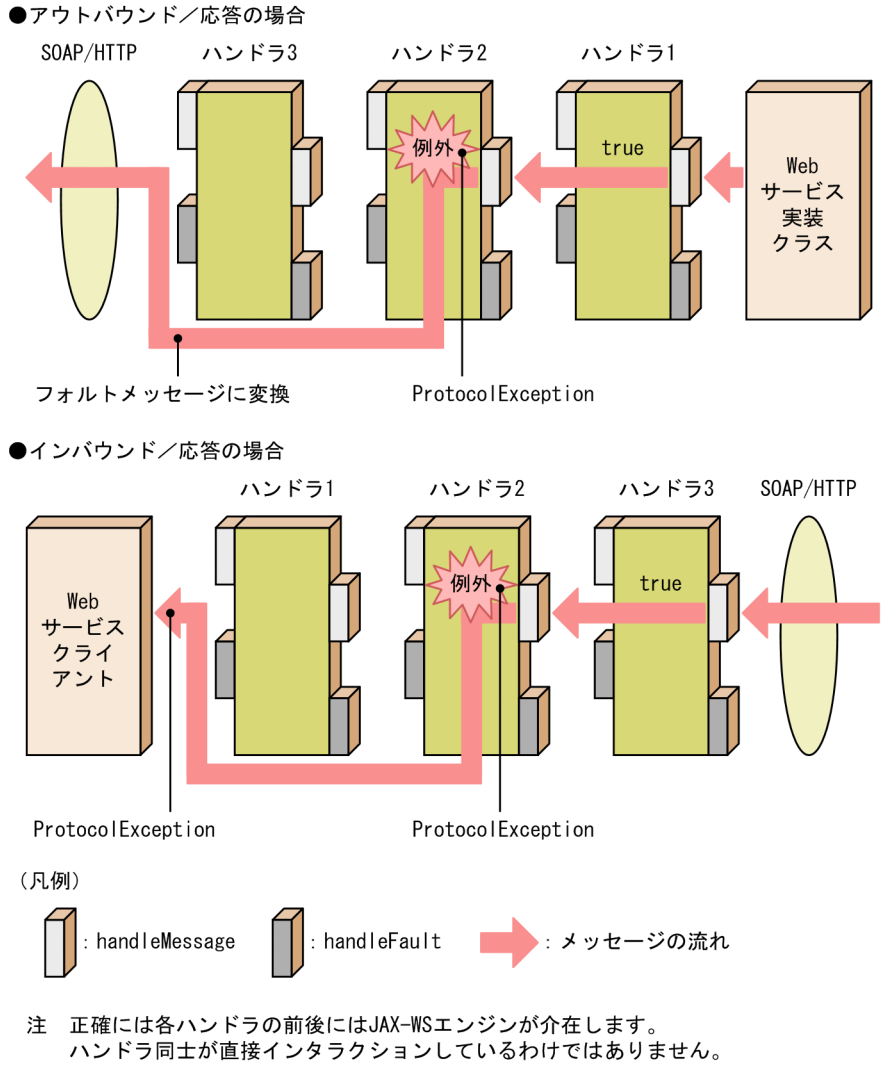
表 26-2　JAX-WS エンジンが生成するフォルトメッセージ（ProtocolException スロー・要求）

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 26-9　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。

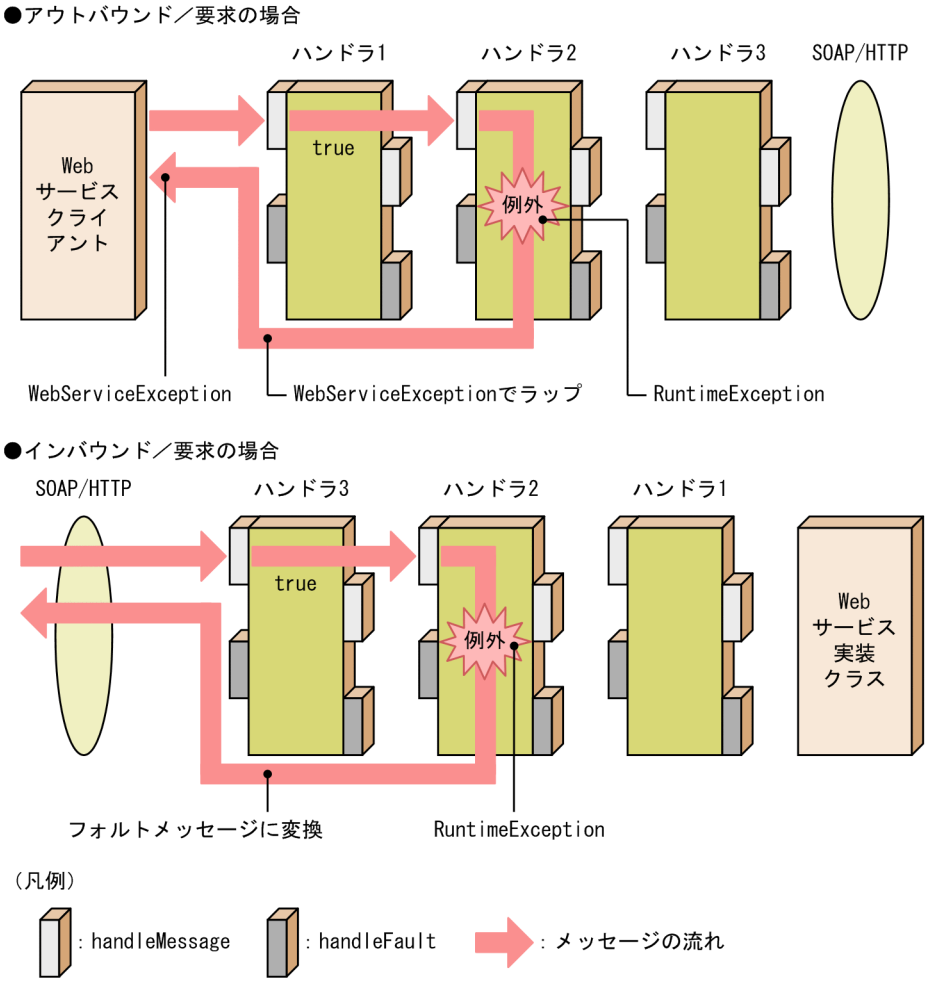
Web サービス側の場合，JAX-WS エンジンは例外を SOAP フォルトに変換して送信します。フォルトメッセージの変換については，「10.4.1(2)　ランタイム例外のバインディング」を参照してください。

(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 26-10　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。

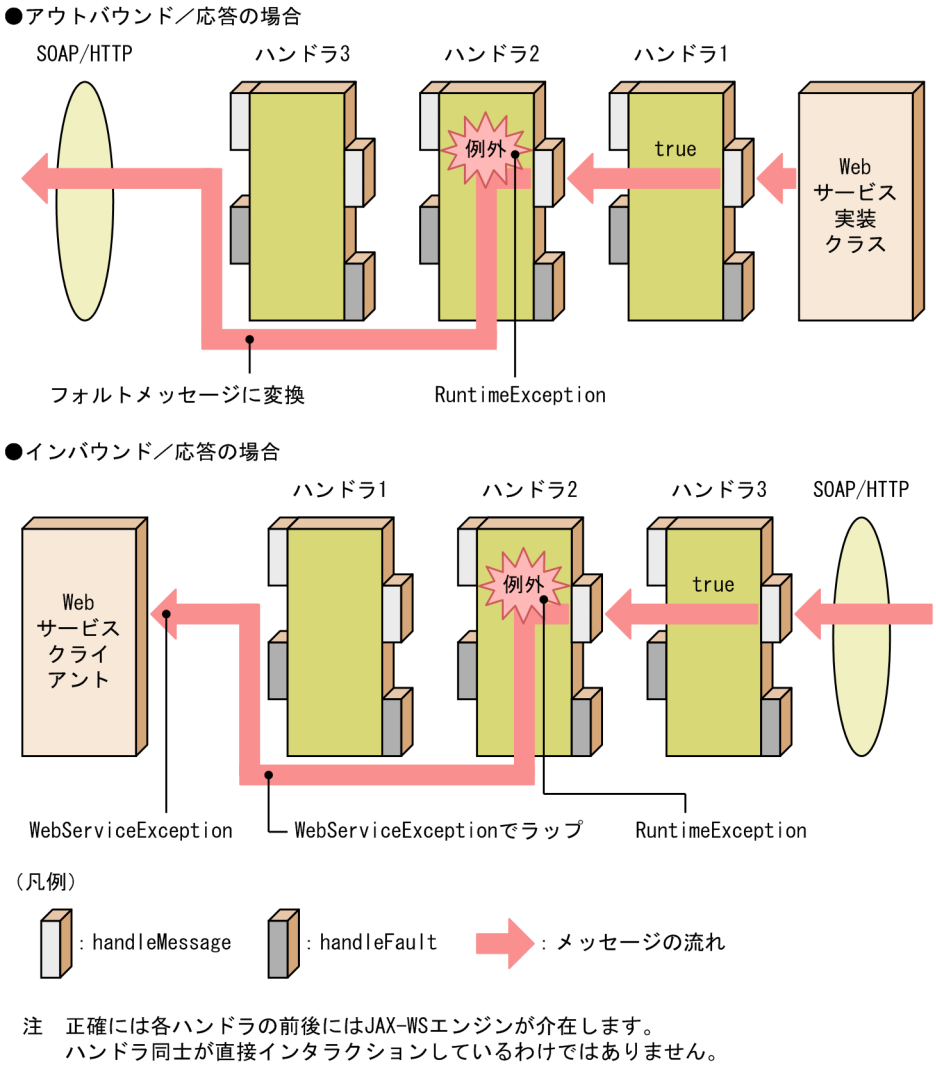
Web サービス側の場合，JAX-WS エンジンは例外を SOAP フォルトに変換して送信します。フォルトメッセージの変換については，「10.4.1(2)　ランタイム例外のバインディング」を参照してください。

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 26-11　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。

26.5.2　handleFault メソッドの処理

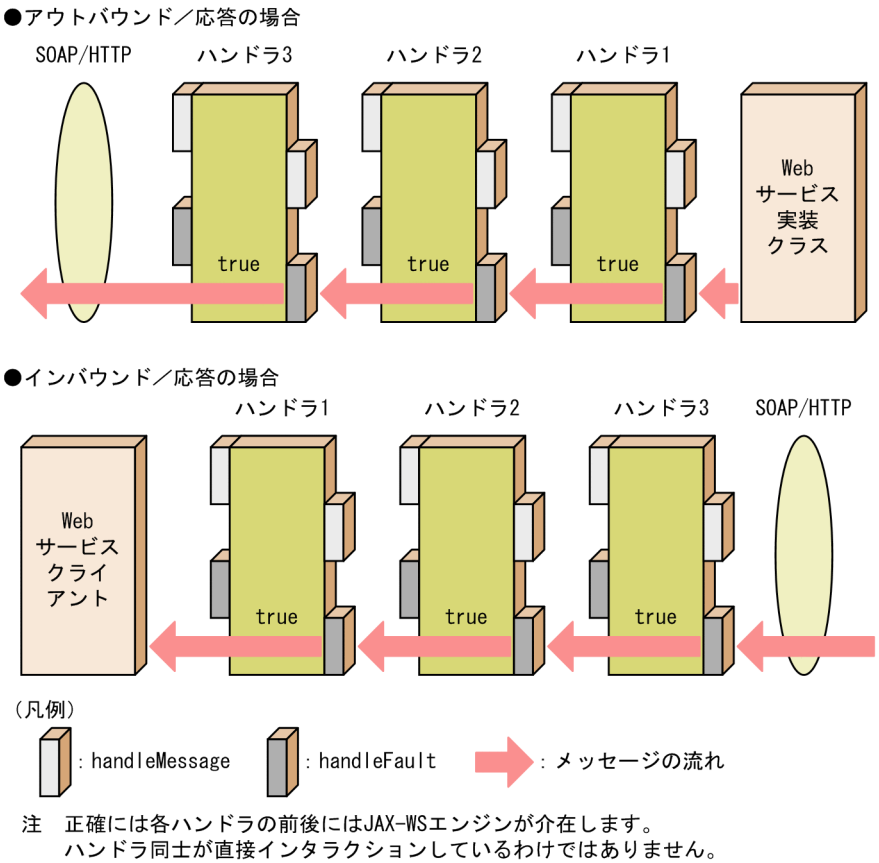
handleMessage メソッドについて，true を返す場合，false を返す場合，および例外をスローする場合の処理について説明します。

（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 26-12　handleFault で true を返す場合の処理（応答）



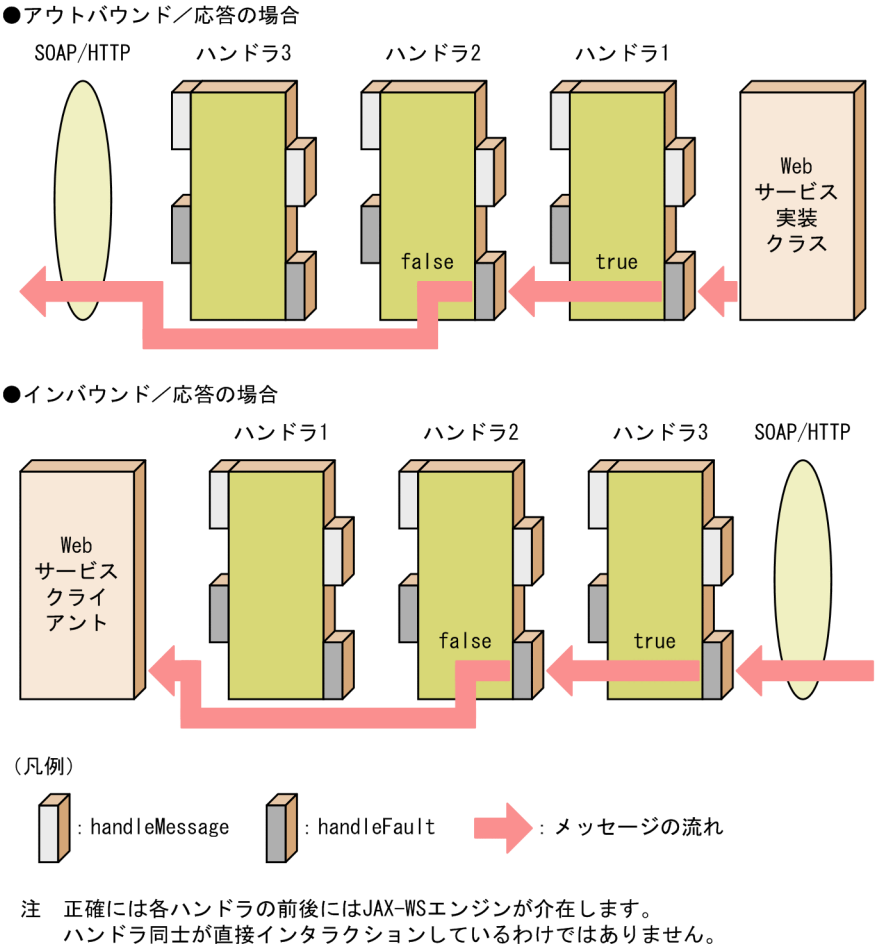
注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

(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 26-13 handleFault で false を返す場合の処理（応答）



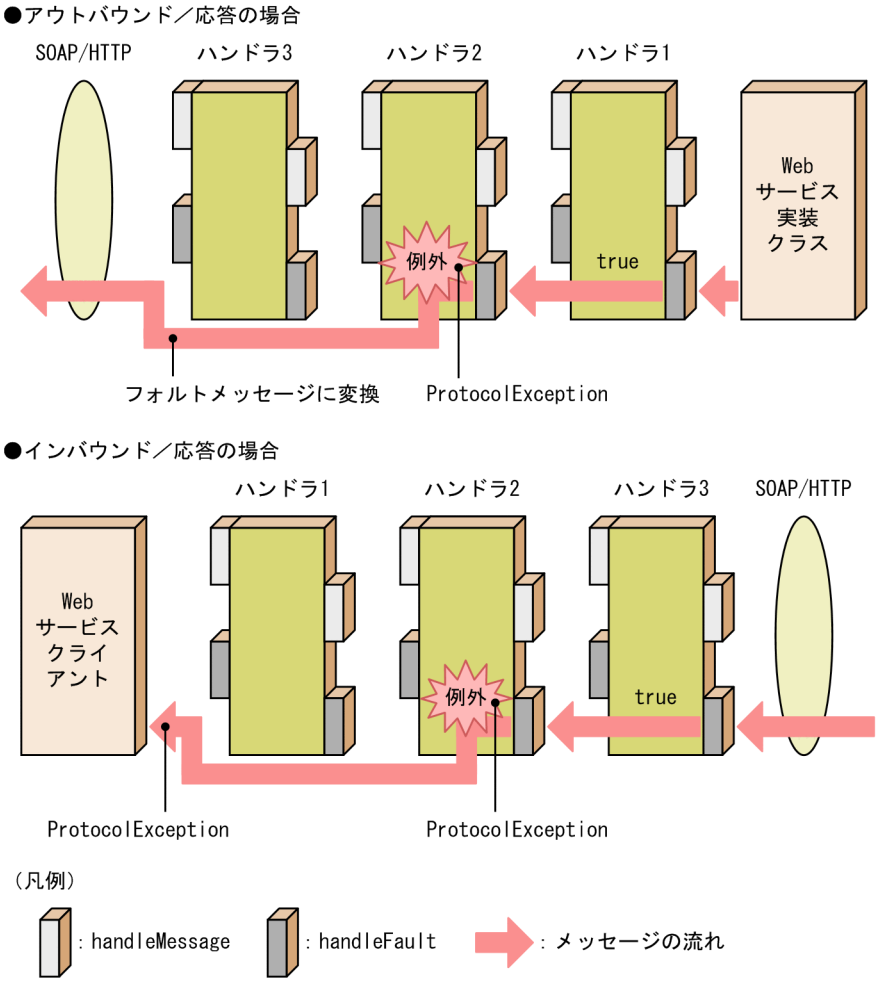
注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

(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 26-14　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。

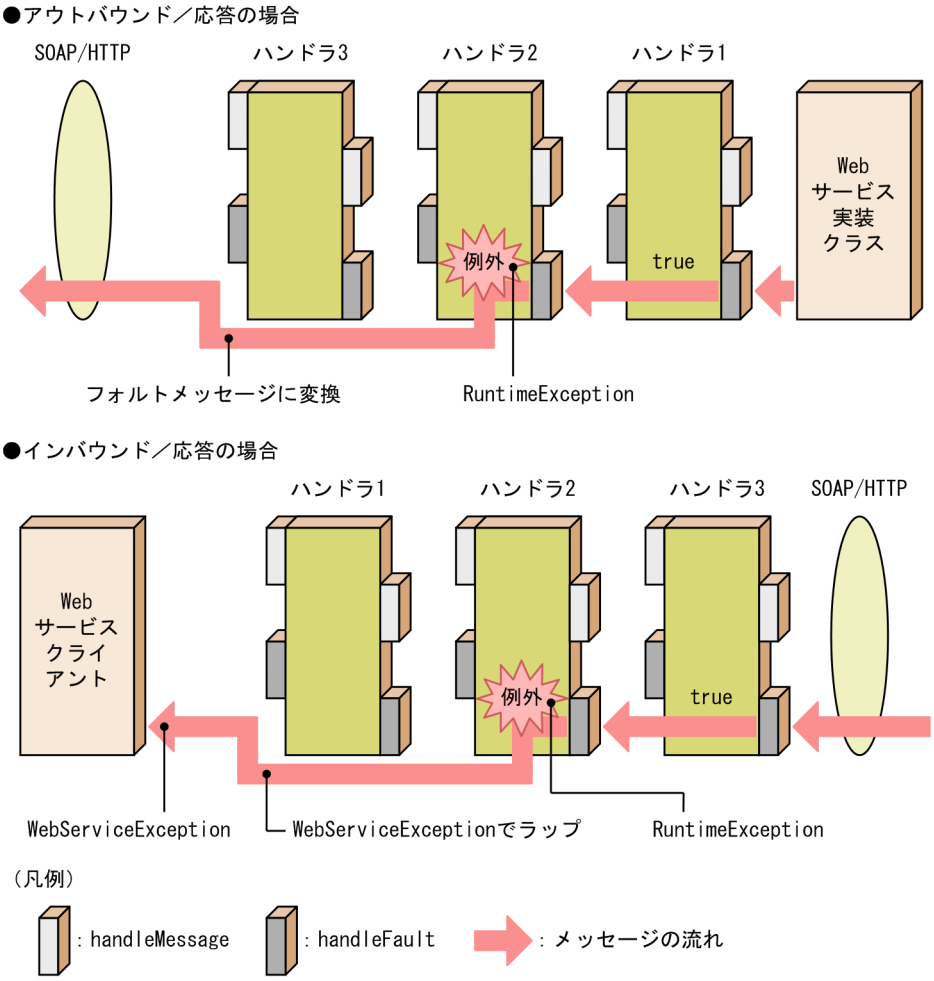
Web サービス側の場合，JAX-WS エンジンは例外を SOAP フォルトに変換し，もともと保持している SOAP フォルトを置き換えます。したがって，実際は新たに生成されたフォルトメッセージを送信します。フォルトメッセージの変換については，「10.4.1(2) ランタイム例外のバインディング」を参照してください。

(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 26-15 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。

26.5.3 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。

26.6 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。

(1) 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。

(2) 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。

26.7 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。

26.7.1 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）

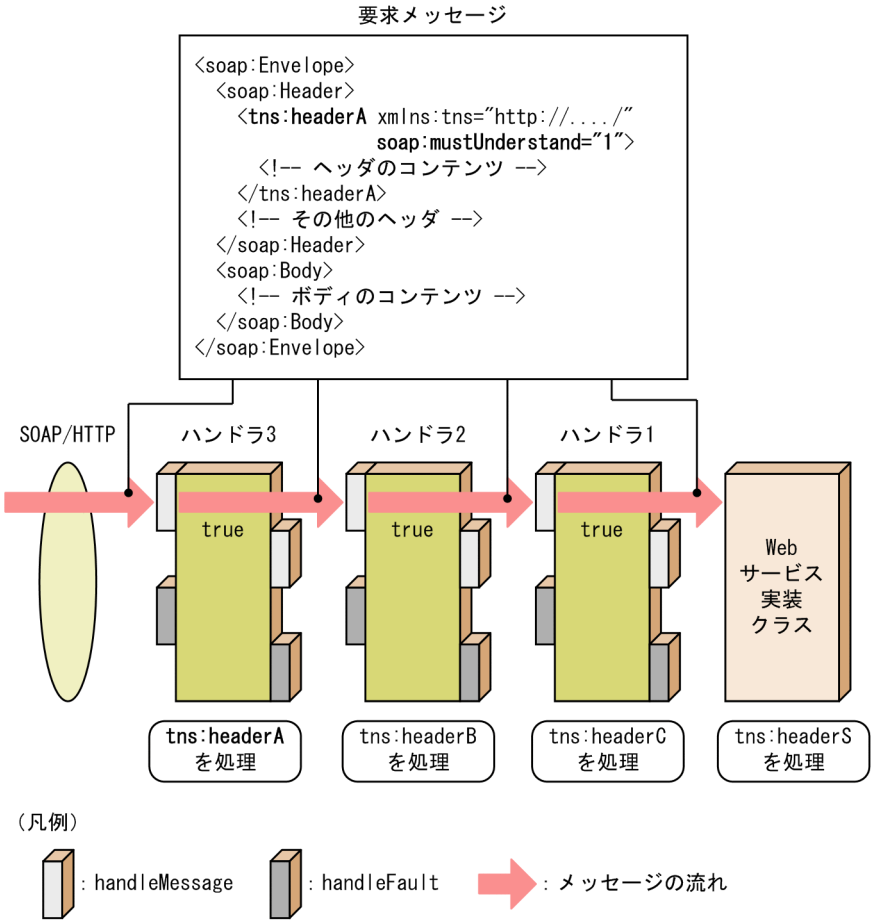
SOAP ヘッダが含まれる SOAP メッセージを受信した場合，SOAP ヘッダの soap:mustUnderstand 属性の設定値によって，ハンドラの処理が異なります。soap:mustUnderstand 属性が "1" の場合，および soap:mustUnderstand 属性が "1" 以外の場合に分けて処理内容を示します。

（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 26-16　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）

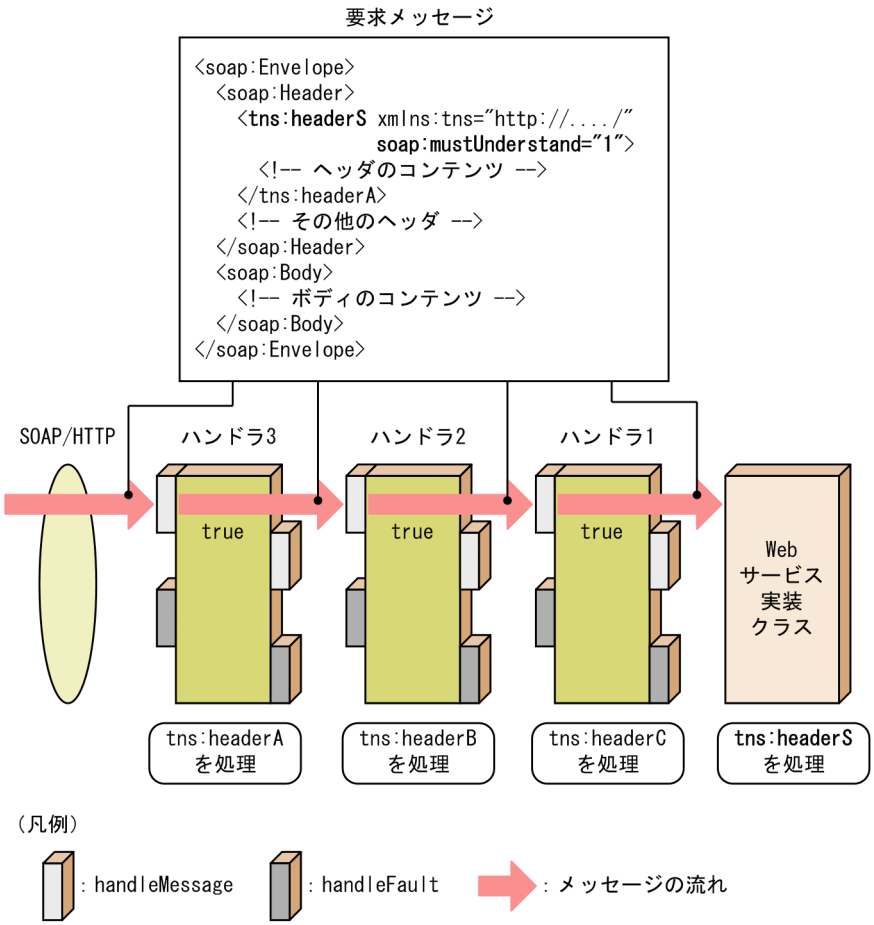


注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

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 26-17　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。

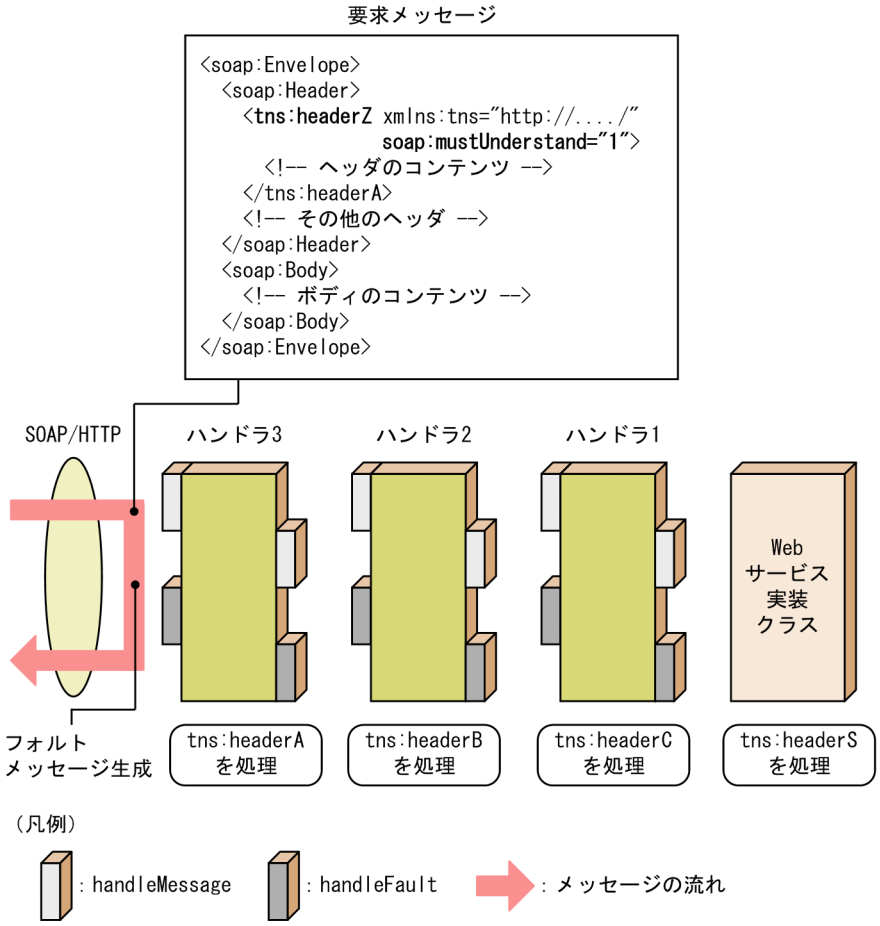
Web サービス実装クラスだけが要素名 tns:headerS を処理できるとした場合も，ハンドラで処理できる場合と同様にすべてのハンドラが呼び出されます。

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 26-18　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）



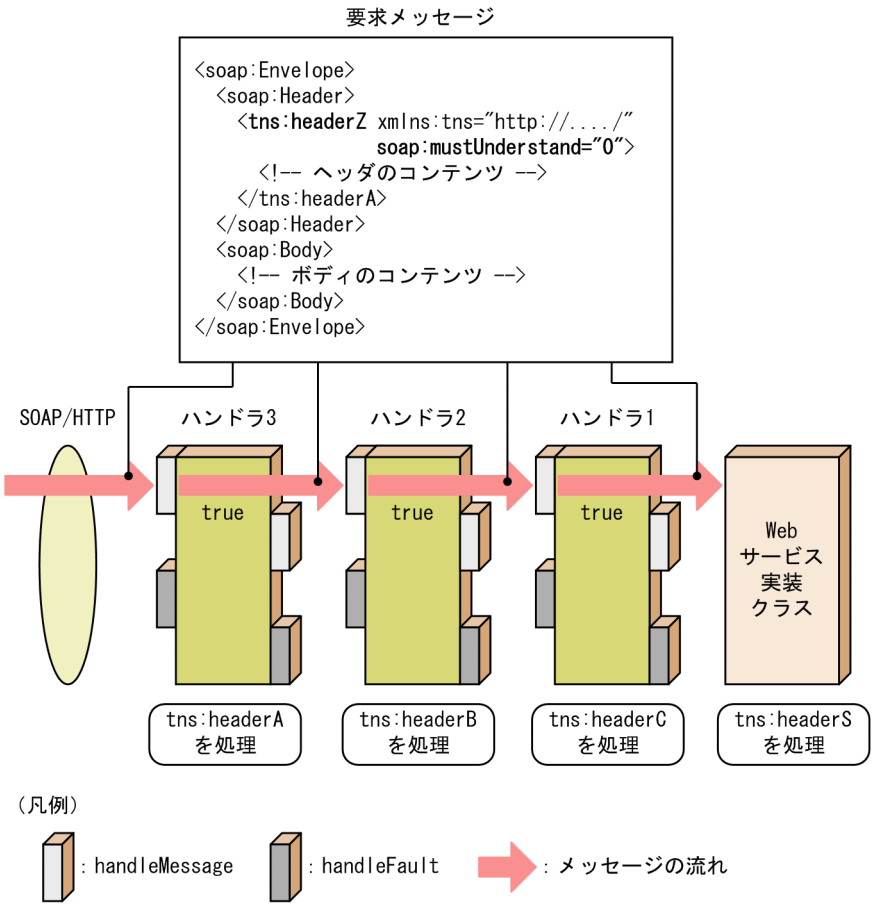
注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

(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 26-19　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。

26.7.2　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）

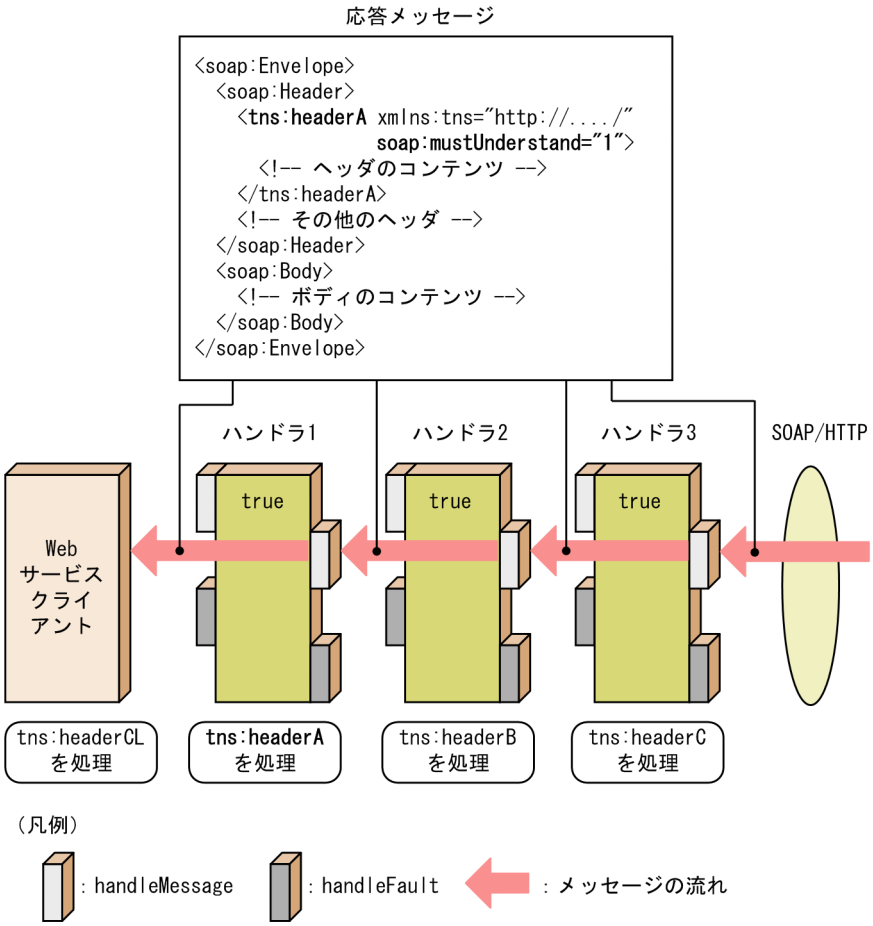
SOAP ヘッダが含まれる SOAP メッセージを受信した場合，SOAP ヘッダの soap:mustUnderstand 属性の設定値によって，ハンドラの処理が異なります。soap:mustUnderstand 属性が "1" の場合，および soap:mustUnderstand 属性が "1" 以外の場合に分けて処理内容を示します。

（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 26-20　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）



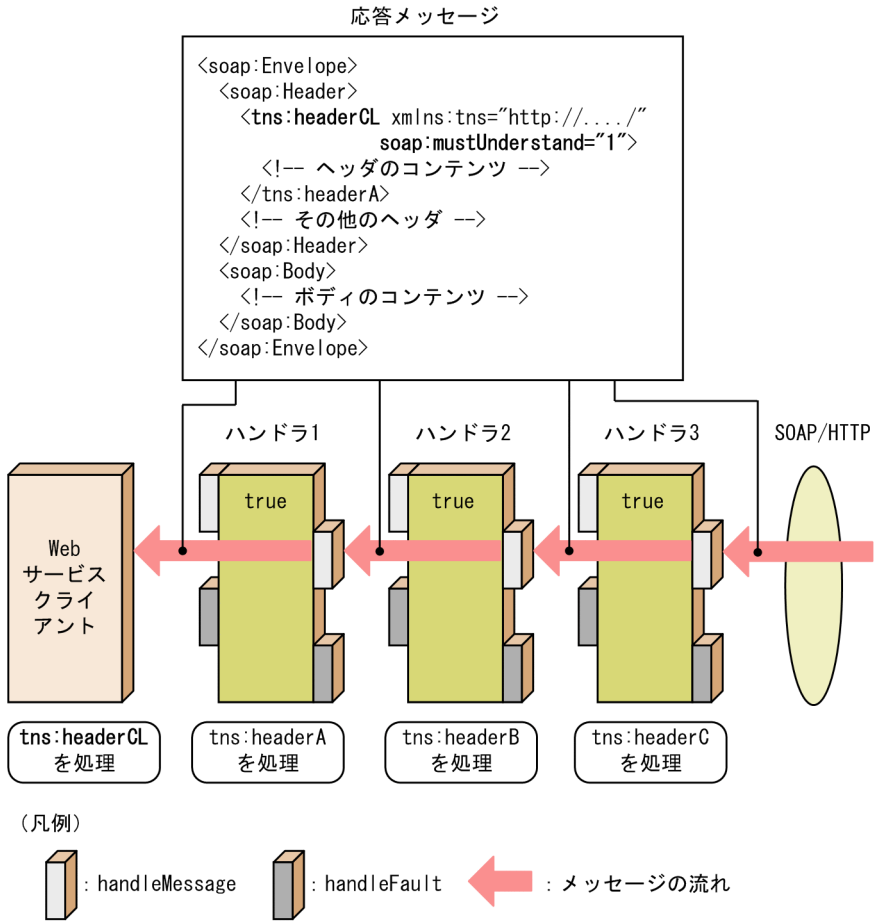
注　正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

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 26-21　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。

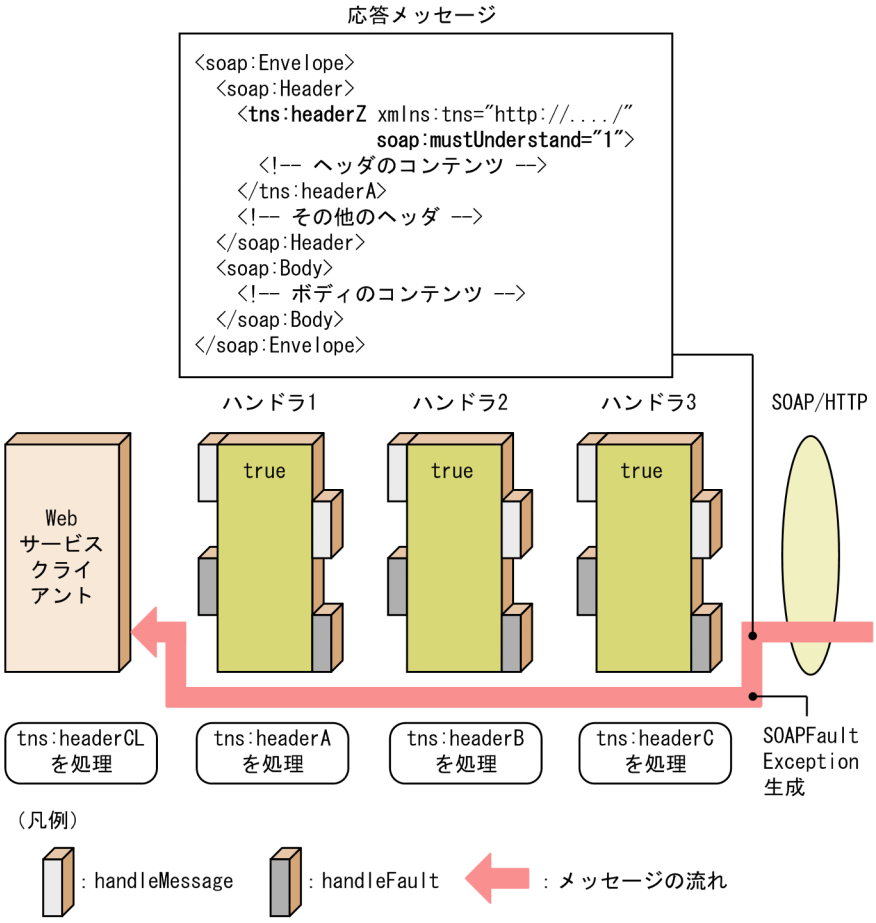
Web サービス実装クラスだけが要素名 tns:headerCL を処理できるとした場合も，ハンドラで処理できる場合と同様にすべてのハンドラが呼び出されます。

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 26-22 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）



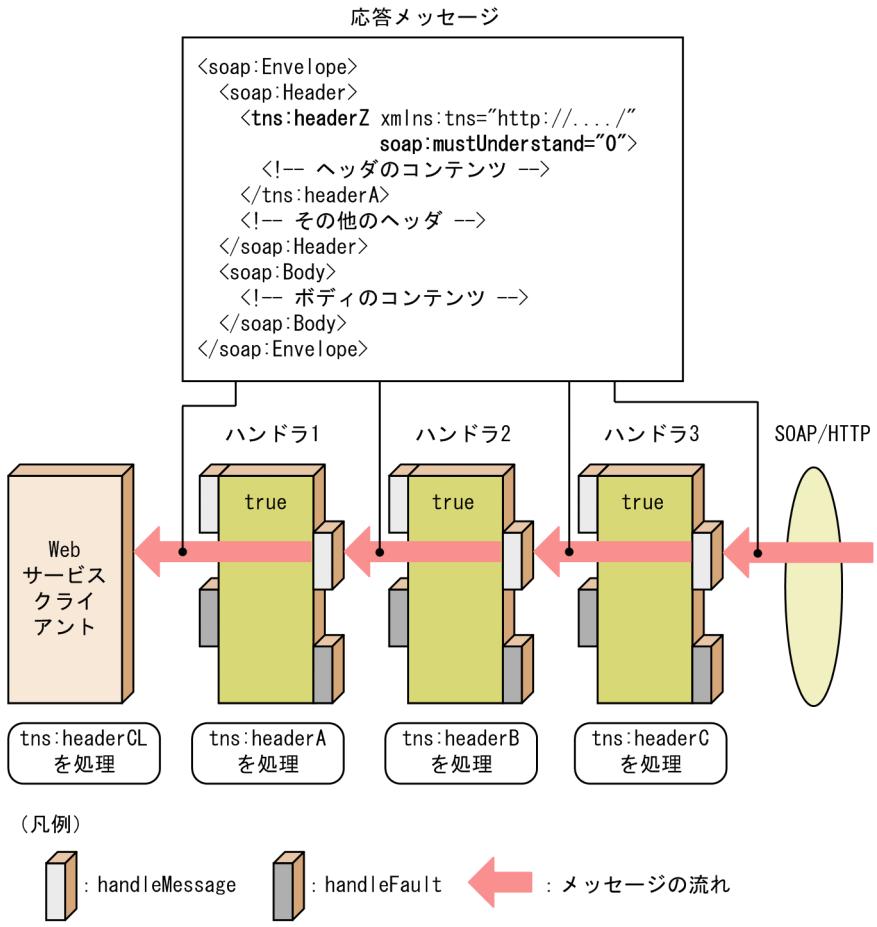
注 正確には各ハンドラの前後にはJAX-WSエンジンが介在します。
ハンドラ同士が直接インタラクションしているわけではありません。

(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 26-23　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。

26.7.3　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, WebParam.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 Cosminexus 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。Cosminexus の JAX-WS 機能での sopa:header 要素の記述については，「14.1.22　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:binding>
    <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"></soap: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「14.1.22　soap:header 要素」を参照してください。

26.8 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。

(1) 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。

(2) 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。

26.9 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。

26.9.1 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，「13.2.3 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，「14.4 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 Cosminexus 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。

26.9.2 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。

(1) ハンドラチェインを設定するコードの追加

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。

使用できる JAX-WS API については，「15. JAX-WS API のサポート範囲」を参照してください。

(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。

27 アドレッシング機能

この章では，Cosminexus 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。

27.1 アドレッシング機能とは

27.2 WSDL の拡張要素と拡張属性

27.3 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

27.4 フォルトメッセージ

27.5 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）

27.6 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）

27.1　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。

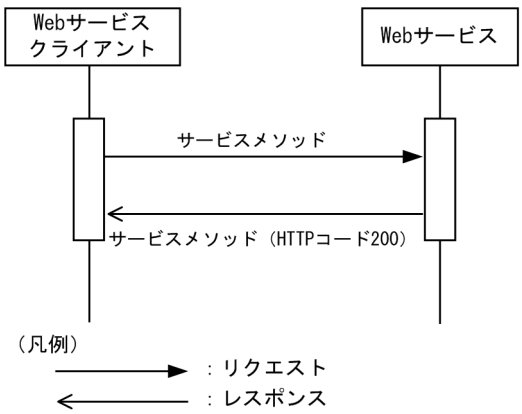
27.1.1　同期通信

(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 27-1　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>
```

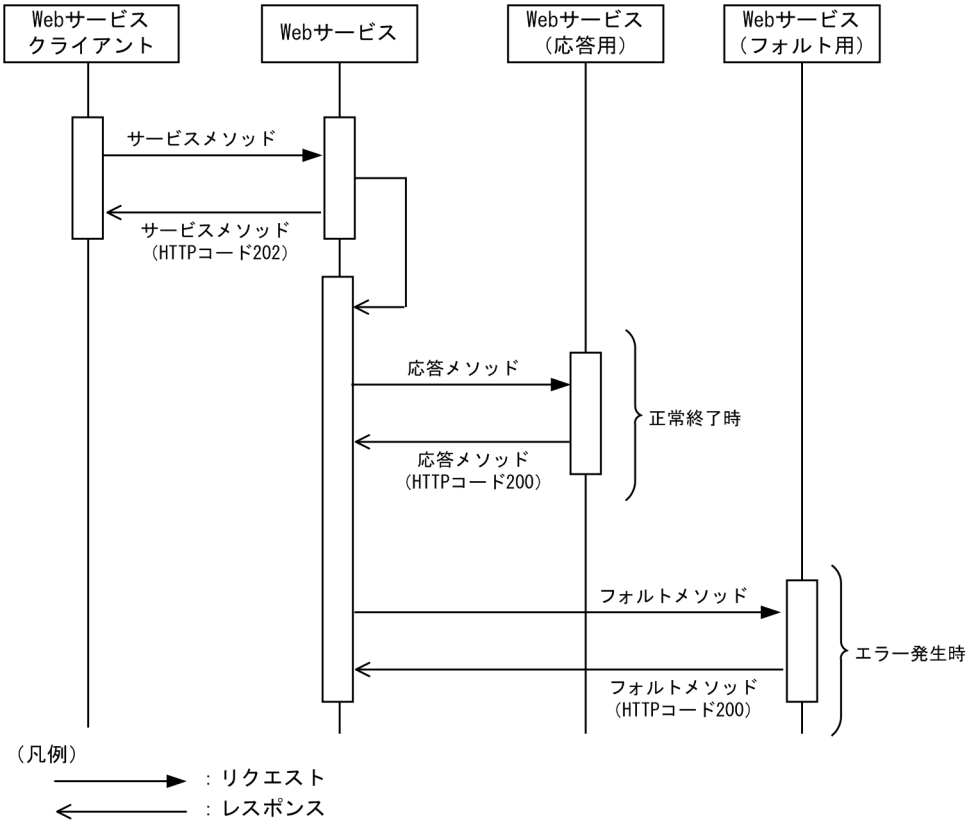
27.1.2 非同期通信

(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 27-2 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。

27.2　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。

27.2.1　WSDL の拡張要素

Cosminexus 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。

● 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）。

● 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，cjwsgen 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。

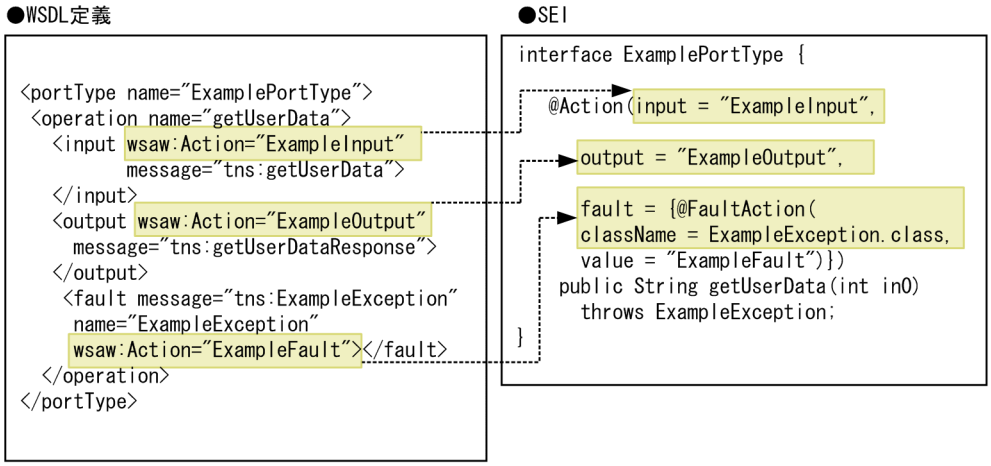
27.2.2 WSDL の拡張属性

Cosminexus 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「13.2.10 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「13.2.12　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 27-3 wsaw:Action 属性を使用した場合のマッピング例



27.3　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，「13.2.1　アノテーション一覧」を参照してください。

27.4 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。

27.4.1 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，Cosminexus 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。

27.4.2 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

27.5 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。

27.5.1 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 27-1 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。

27.5.2 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 27-2 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

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		
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
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。

27.5.3 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 27-3 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，「27.2.1 WSDL の拡張要素」を参照してください。

27.5.4 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，cjwsgen 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。

27.5.5 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。

27.5.6 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 27-4　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。

27.5.7　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。

27.6　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。

27.6.1　メッセージ送受信時の動作

アドレッシング機能を使用する場合，SEI を取得するときに使用する AddressingFeature クラスの生成時に設定した引数によって，メッセージを送受信したときの動作が異なります。

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 27-5　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	
1	true	true	○
2		false	○
3	false	true	×
4		false	×

（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 27-6　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	アドレッシング・ヘッダ	通信
1	true	true	○	成功
2			×	失敗
3		false	○	成功
4			×	成功

項番	AddressingFeature クラス		レスポンスメッセージ受信時の動作	
	enabled	required	アドレッシング・ヘッダ	通信
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。

27.6.2 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，呼び出そうとした Web サービスの WSDL の wsaw:Anonymous 要素に "prohibited" が指定されていると，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。

wsa:Anonymous 要素に "prohibited" が指定されている 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。

27.6.3 wsaw:Action 属性を指定していない場合の動作

WSDL に wsaw:Action 属性が記述されていない場合，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アドレッシング・ヘッダの wsa:Action 要素の値には，WS-Addressing 1.0 仕様で決められているデフォルトの 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 仕様を参照してください。

27.6.4 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。

27.6.5 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。

28 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。

28.1 開発例の構成（SEI 起点・アドレッシング）

28.2 開発例の流れ（SEI 起点・アドレッシング）

28.3 Web サービスの開発例（SEI 起点・アドレッシング）

28.4 デプロイと開始の例（SEI 起点・アドレッシング）

28.5 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）

28.6 Web サービスの実行例（SEI 起点・アドレッシング）

28.1　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 28-1　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
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 28-2　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	「28.3.4　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。「28.3.1　Web サービス実装クラスを作成する」および「28.3.2　Java ソースを生成する」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	「28.3.3　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。「28.3.2　Java ソースを生成する」で使用します。
│ ├ addressing.ear	「28.3.5　EAR ファイルを作成する」で作成します。
│ └ addressing.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。「28.5.1　サービスクラスを生成する」および「28.5.2　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。「28.5.3　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	「28.6.1　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	「28.6.2　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**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。

28.2 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（28.3.1）
2. apt コマンドを実行し，追加の Java ソースを生成する（28.3.2）
3. web.xml を作成する（28.3.3）
4. application.xml を作成する（28.3.4）
5. EAR ファイルを作成する（28.3.5）

デプロイと開始

1. EAR ファイルをデプロイする（28.4.1）
2. Web サービスを開始する（28.4.2）

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（28.5.1）
2. Web サービスクライアントの実装クラスを作成する（28.5.2）
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（28.5.3）

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（28.6.1）
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（28.6.2）
3. Web サービスクライアントを実行する（28.6.3）

28.3　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。

28.3.1　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=SOAPBinding.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。

28.3.2　Java ソースを生成する

apt コマンドを実行して，Web サービス実装クラスから Web サービスの開発に必要な追加の Java ソースを生成します。また，Web サービス実装クラスを含めてコンパイルします。apt コマンドについては，「11.2　apt コマンド」を参照してください。

apt コマンドの実行例を次に示します。

● Windows(x86) の場合

```
> set HNTRLIB2_HOME=<HNTRLib2インストールディレクトリ>
> cd c:¥temp¥jaxws¥works¥addressing¥server¥
> mkdir WEB-INF¥classes¥
> apt -J-Dcosminexus.home="%COSMINEXUS_HOME%" -cp
"%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥client¥lib
¥j2ee-javax.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%
¥jaxp¥lib¥csmjaxp.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar;%HNTRLIB2_H
OME%¥classes¥hntrlib2j.jar;%HNTRLIB2_HOME%¥classes¥hntrlibMj.jar" -d
WEB-INF¥classes¥ -s src src¥com¥sample¥AddNumbers.java
src¥com¥sample¥AddNumbersImpl.java src¥com¥sample¥AddNumbersFault.java
```

● Windows(x64) の場合

```
> set HNTRLIB2_HOME=<HNTRLib2インストールディレクトリ>
> cd c:¥temp¥jaxws¥works¥addressing¥server¥
> mkdir ¥WEB-INF¥classes¥
> apt -factory com.cosminexus.istack.ws.AnnotationProcessorFactoryImpl
-J-Dcosminexus.home="%COSMINEXUS_HOME%" -cp
"%COSMINEXUS_HOME%¥jaxws¥lib¥cjjaxws.jar;%COSMINEXUS_HOME%¥CC¥client¥lib
¥j2ee-javax.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmjaxb.jar;%COSMINEXUS_HOME%
¥jaxp¥lib¥csmjaxp.jar;%COSMINEXUS_HOME%¥jaxp¥lib¥csmstax.jar;%HNTRLIB2_H
OME%¥classes¥hntrlib2j64.jar;%HNTRLIB2_HOME%¥classes¥hntrlibMj64.jar" -d
WEB-INF¥classes¥ -s src src¥com¥sample¥AddNumbers.java
src¥com¥sample¥AddNumbersImpl.java src¥com¥sample¥AddNumbersFault.java
```

<HNTRLib2 インストールディレクトリ > の部分には，次のコマンドの実行結果を指定します。

● Windows(x86) の場合

```
> "%COSMINEXUS_HOME%¥common¥bin¥gethntr2conf.exe" HNTR2INSTDIR
```

● Windows(x64) の場合

```
> "%COSMINEXUS_HOME%¥common¥bin¥gethntr2conf64.exe" HNTR2INSTDIR
```

apt コマンドが正常に終了すると，
c:¥temp¥jaxws¥works¥addressing¥server¥src¥com¥sample¥jaxws¥ 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 28-3　Java ソース生成時の生成物（SEI 起点・アドレッシング）

ファイル名	説明
Add.java	add メソッドに対応するリクエスト bean です。
AddResponse.java	add メソッドに対応するレスポンス bean です。
Add2.java	add2 メソッドに対応するリクエスト bean です。
Add2Response.java	add2 メソッドに対応するレスポンス bean です。
Add3.java	add3 メソッドに対応するリクエスト bean です。
Add3Response.java	add3 メソッドに対応するレスポンス bean です。
AddNumbersFaultBean.java	AddNumbersFault に対応するフォルト bean です。

ファイル名の Add，Add2，Add3，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。

28.3.3　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="2.5" 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_2_5.xsd">
  <description>Sample web service &quot;addressing&quot;</description>
  <display-name>Sample_web_service_addressing</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>
```

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。

28.3.4　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="5" 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_5.xsd">

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

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「Cosminexus 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「5.2.4　application.xml 編集時の注意事項」を参照してください。

28.3.5　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.war .¥WEB-INF
> jar cvf addressing.ear .¥addressing.war .¥META-INF¥application.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.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。

28.4　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。

28.4.1　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.ear
```

cjimportapp コマンドについては，マニュアル「Cosminexus 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「Cosminexus 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。

28.4.2　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
```

cjstartapp コマンドについては，マニュアル「Cosminexus 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「Cosminexus 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。

28.5　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。

28.5.1　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，「11.1 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/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 28-4　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。

ファイル名	説明
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 2.1 仕様の ObjectFactory クラスです。
package-info.java	package-info.java ファイルです。
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。

- 「12.1.2　ポートタイプから SEI 名へのマッピング」
- 「12.1.3　オペレーションからメソッド名へのマッピング」
- 「12.1.4　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）」
- 「12.1.5　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）」

28.5.2　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/
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。

28.5.3　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¥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¥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。

28.6　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。

28.6.1　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=<Cosminexus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=<Cosminexusインストールディレクトリ>
add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF ID>
```

<Cosminexus のインストールディレクトリ > の部分は，Cosminexus 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「Cosminexus 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「15.2　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。

28.6.2　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「Cosminexus 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「15.3　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。

28.6.3　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「Cosminexus 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。

29 障害対策

この章では，Web サービスの運用中に発生した障害の対策について説明します。

29.1　障害の種類と対策
29.2　障害発生時に取得する資料
29.3　ログ
29.4　性能解析トレース（PRF）

29.1　障害の種類と対策

Web サービ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。

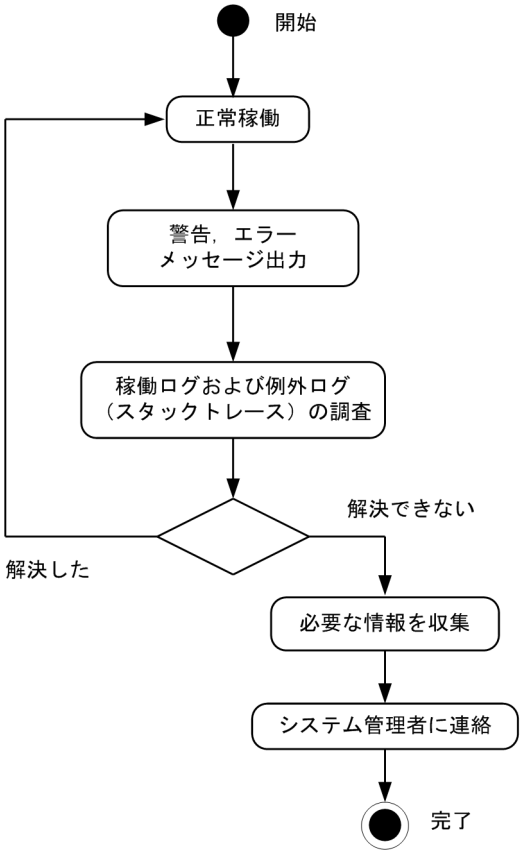
また，プログラムが正常に動作している場合でも，プログラムまたはコンポーネントの処理能力が，期待した性能に届かない場合が考えられます。

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「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「7.10　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。

29.1.1　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 29-1 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，「29.3 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，「29.2 障害発生時に取得する資料」を参照してください。

29.1.2 プログラムが意図したとおりに動作しない場合

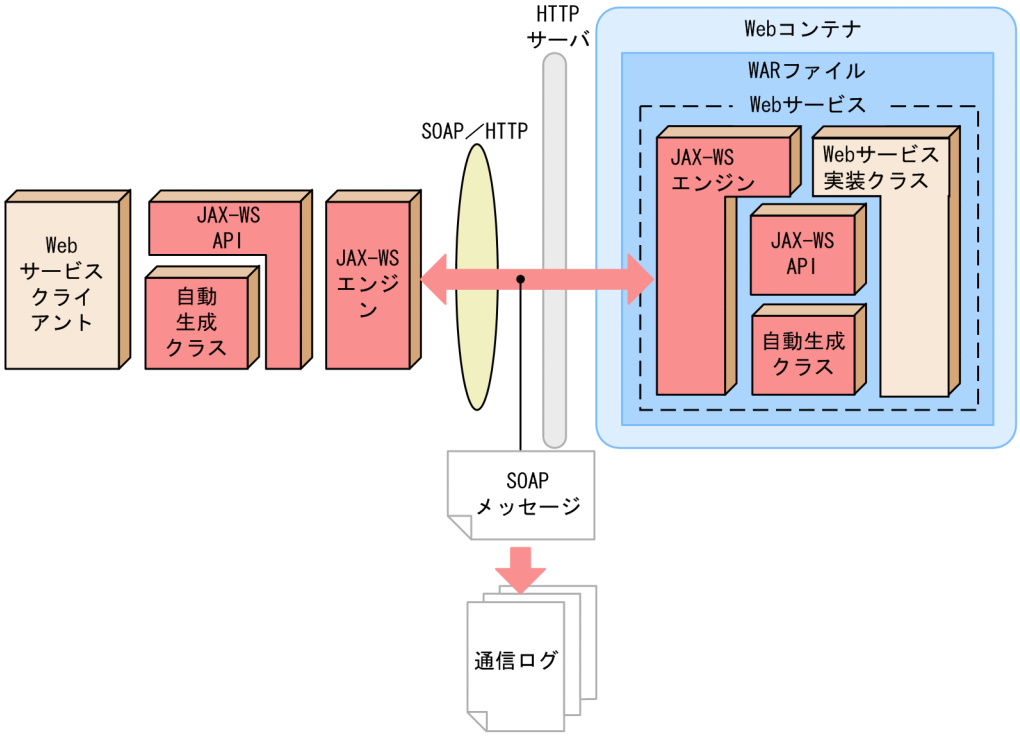
エラーメッセージや警告メッセージが出力されない場合でも，不良などが原因で，作成したプログラムが意図したとおりに動作しないことがあります。この場合，プログラムから Web サービスの実装に対して，意図した SOAP メッセージが送信されていないおそ

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SOAP 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，「29.3.4　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 29-2　通信ログの出力



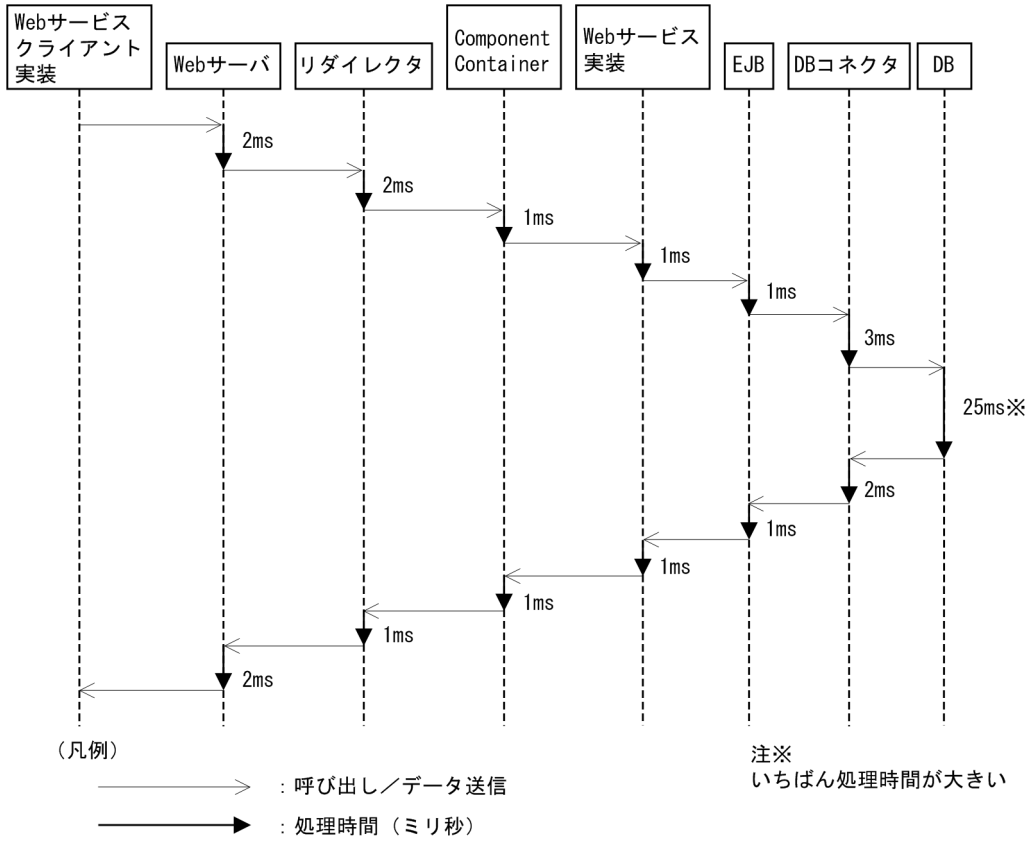
29.1.3　期待した性能が出ない場合

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。Web サービスクライアント側のプログラムから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，「29.4.3　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 29-3　各機能レイヤの処理時間の例



29.2 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）
- 例外ログ（cjwexception?_?.log）
- 保守ログ（cjwmessage?.log）
- 通信ログ（cjwtransport?_?.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）
- 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
- Cosminexus Component Container および Web サーバのログ
- Cosminexus 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 Cosminexus 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

Cosminexus 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「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「10.7.9　資料の取得方法」を参照してください。

29.3　ログ

Web サービスの開発時，および実行時に出力されるログについて説明します。

29.3.1　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 29-1　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 SOAP 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※

通信ログを有効にする場合は，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。

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 29-2　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	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	○	○	○	○
2	コマンドラインインタフェースから起動した Web サービスクライアント	○	○	○	○
3	cjwsimport コマンド	○	○	○	－
4	apt コマンド	○	○	○	－
5	cjwsgen 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。

29.3.2 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 29-3 ログの出力先（J2EE サーバ上で動作する場合）

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	<ejb.server.log.directory>/CJW	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	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「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」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，Cosminexus Component Container の稼働ログ（cjmessage?.log）にログが出力されます。

（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 29-4 ログの出力先（コマンドラインインタフェースを利用する場合）

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
2	例外ログ	<ejbserver.client.log.directory>/ <ejbserver.client.ejb.log>/ <ejbserver.client.log.appid>/CJW	cjwexception?_?.log

項番	ログの種類	出力先ディレクトリ	ファイル名
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「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」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，Cosminexus 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 29-5　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 apt コマンドの場合： cjwapt?.log cjwsgen コマンドの場合：cjwsgen?.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 apt コマンドの場合： cjwaptex?_?.log cjwsgen コマンドの場合： cjwsgenex?_?.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 apt コマンドの場合： cjwapt?.log cjwsgen コマンドの場合：cjwsgen?.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，<Cosminexus 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。

29.3.3　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 29-6　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 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，「29.3.5　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 29-7 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	○	○	○	－
	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または Web サービスによって，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）

通信ログの定義，および出力内容を次の表に示します。

表 29-8 通信ログの出力条件（Web サービスクライアント側）

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。
NONE	送受信メッセージは出力されません。

表 29-9 通信ログの出力条件（Web サービス側）

ログの定義	出力内容
ALL	Web サービ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。
NONE	Web サービ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。

29.3.4 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 29-10 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 けた）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）

項目	出力内容
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」 • apt コマンドの場合：「cjwapt」 • cjwsgen コマンドの場合：「cjwsgen」 • WS-RM 1.2 機能の場合：「wsrm」
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 • 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 29-11　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

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:Body><ns2:jaxWsTest1 xmlns:ns2="http://example.com/
sample"><information>Invocation test.</information><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: Hitachi Web Server
<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/
soap/envelope/"><S:Body><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:jaxWsTest1Response></S:Body></S:Envelope>
```

(3) 通信ログの文字エンコーディング

SOAP メッセージの文字エンコーディングとは関係なく，通信ログには動作定義ファイルの com.cosminexus.jaxws.logger.runtime.transport.encoding 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。

SOAP 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，SOAP 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，SOAP 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の開発者（WSDL の公開先）に通知することをお勧めします。

29.3.5 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。

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.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

com.cosminexus.jaxws.logger.apt.message.level=INFO
com.cosminexus.jaxws.logger.apt.message.file_num=2
com.cosminexus.jaxws.logger.apt.message.file_size=2097152

com.cosminexus.jaxws.logger.apt.exception.level=INFO
com.cosminexus.jaxws.logger.apt.exception.file_num=2
com.cosminexus.jaxws.logger.apt.exception.file_size=16777216

com.cosminexus.jaxws.logger.apt.maintenance.level=ALL
com.cosminexus.jaxws.logger.apt.maintenance.file_num=2
com.cosminexus.jaxws.logger.apt.maintenance.file_size=16777216

com.cosminexus.jaxws.logger.cjwsgen.message.level=INFO
com.cosminexus.jaxws.logger.cjwsgen.message.file_num=2
com.cosminexus.jaxws.logger.cjwsgen.message.file_size=2097152

com.cosminexus.jaxws.logger.cjwsgen.exception.level=INFO
com.cosminexus.jaxws.logger.cjwsgen.exception.file_num=2
com.cosminexus.jaxws.logger.cjwsgen.exception.file_size=16777216

com.cosminexus.jaxws.logger.cjwsgen.maintenance.level=ALL
com.cosminexus.jaxws.logger.cjwsgen.maintenance.file_num=2
com.cosminexus.jaxws.logger.cjwsgen.maintenance.file_size=16777216
```

29.3.6　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 サービスからリクエストを送信し，SEI 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 サービスの SEI で RuntimeException が発生する場合のモデルです。

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／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。
  - com.cosminexus.jaxws.logger.runtime.transport.client_dump
  - com.cosminexus.jaxws.logger.runtime.transport.server_dump

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」を参照してください。

(2) 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 SEI 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 29-12　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] ＝
    [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×正常ケース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×異常ケース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，正常終了するリクエストが 80%，異常終了するリクエストが 20% を占めるとすると，50 リクエスト／分なので，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（起動時の出力量）
＋{0.0KB×18000}（モデル1の全リクエスト処理時の出力量）×0.8（正常ケース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（異常ケース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(起動時の出力量)
＋{3.2KB×18000}（モデル1の全リクエスト処理時の出力量）×0.8（正常ケース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（異常ケース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。

29.4 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。

Cosminexus システムが提供する性能解析トレースについては，マニュアル「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「4.6　性能解析トレース」を参照してください。

29.4.1 性能解析トレースの取得レベル

Cosminexus の JAX-WS 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 29-13　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）	Cosminexus の JAX-WS 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，Cosminexus の JAX-WS 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。

29.4.2 性能解析トレースのトレース出力情報

Cosminexus の JAX-WS 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 29-14　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 AP 情報	ルート AP 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 AP 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。

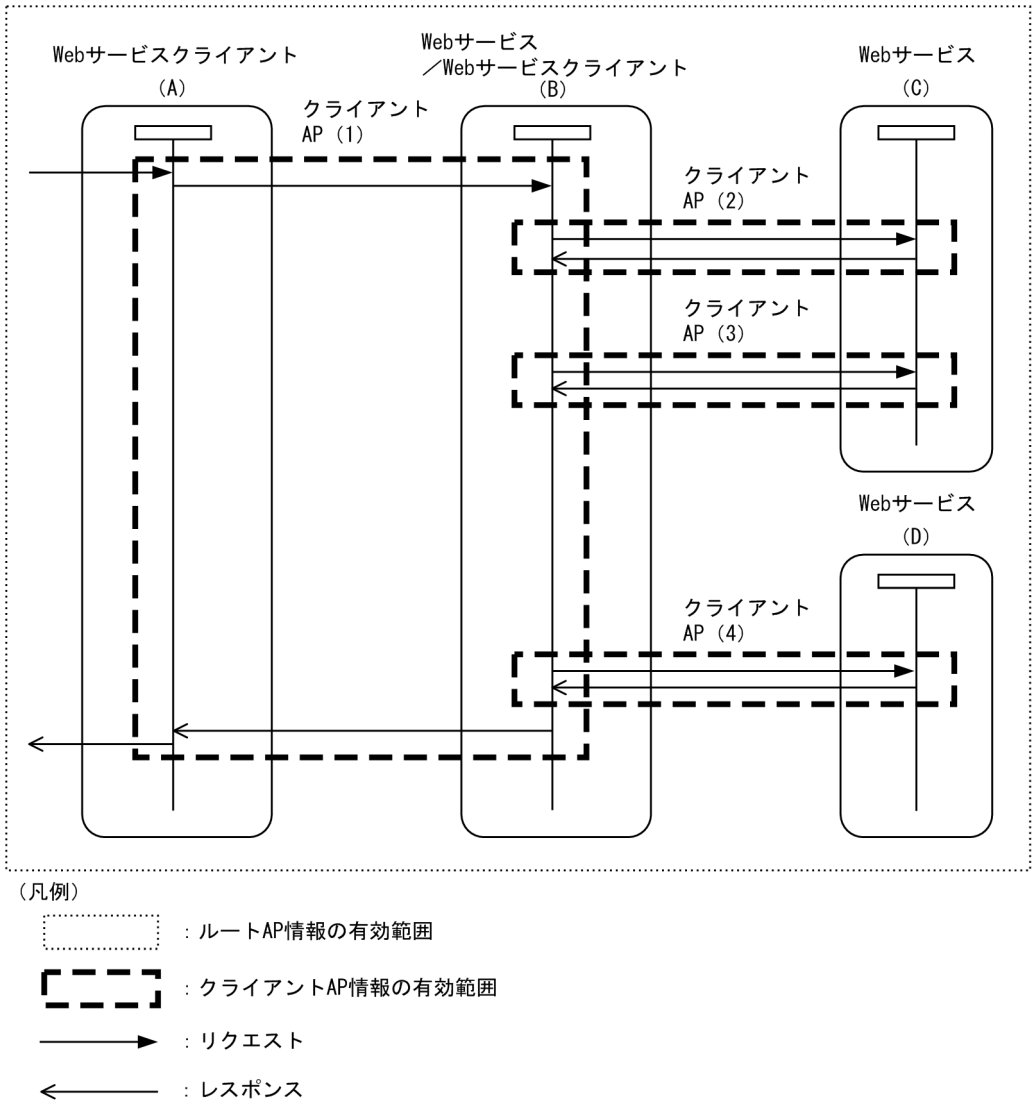
項番	ファイル項目	内容
6	オペレーション名	トレース取得ポイントのメソッド名です。
7	オプション情報	オプション情報です。

ここでは，クライアント AP 情報とルート AP 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 AP 情報とルート AP 情報の有効範囲

次の図を基に，クライアント AP 情報とルート AP 情報の有効範囲について説明します。

図 29-4 クライアント AP 情報とルート AP 情報の有効範囲



ルート AP 情報（通信番号，ルート AP の PID，およびルート AP の動作するマシンの 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。

リクエストを受けた 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 AP 情報（通信番号，クライアント AP の PID，クライアント AP の動作するマシンの IP アドレスから構成される）は，ルート AP 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 AP（1），（2），（3），および（4）は，異なるクライアント AP 情報になります。

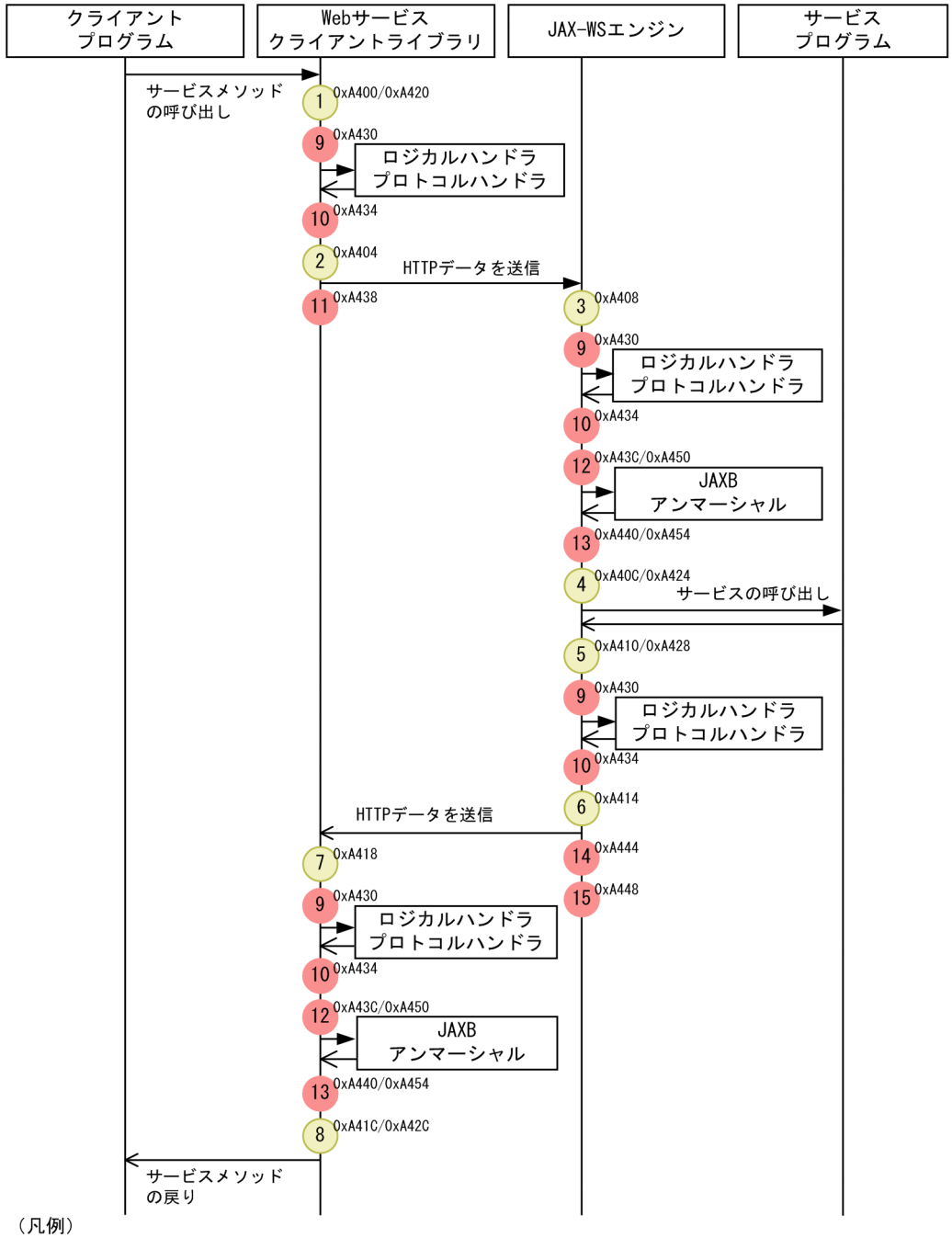
（2） 性能解析トレースのトレース取得ポイント

Cosminexus 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，「29.4.2(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を次の図に示します。

図 29-5　POJO の Web サービス呼び出し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 29-15 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
		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。

表 29-16 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

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
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

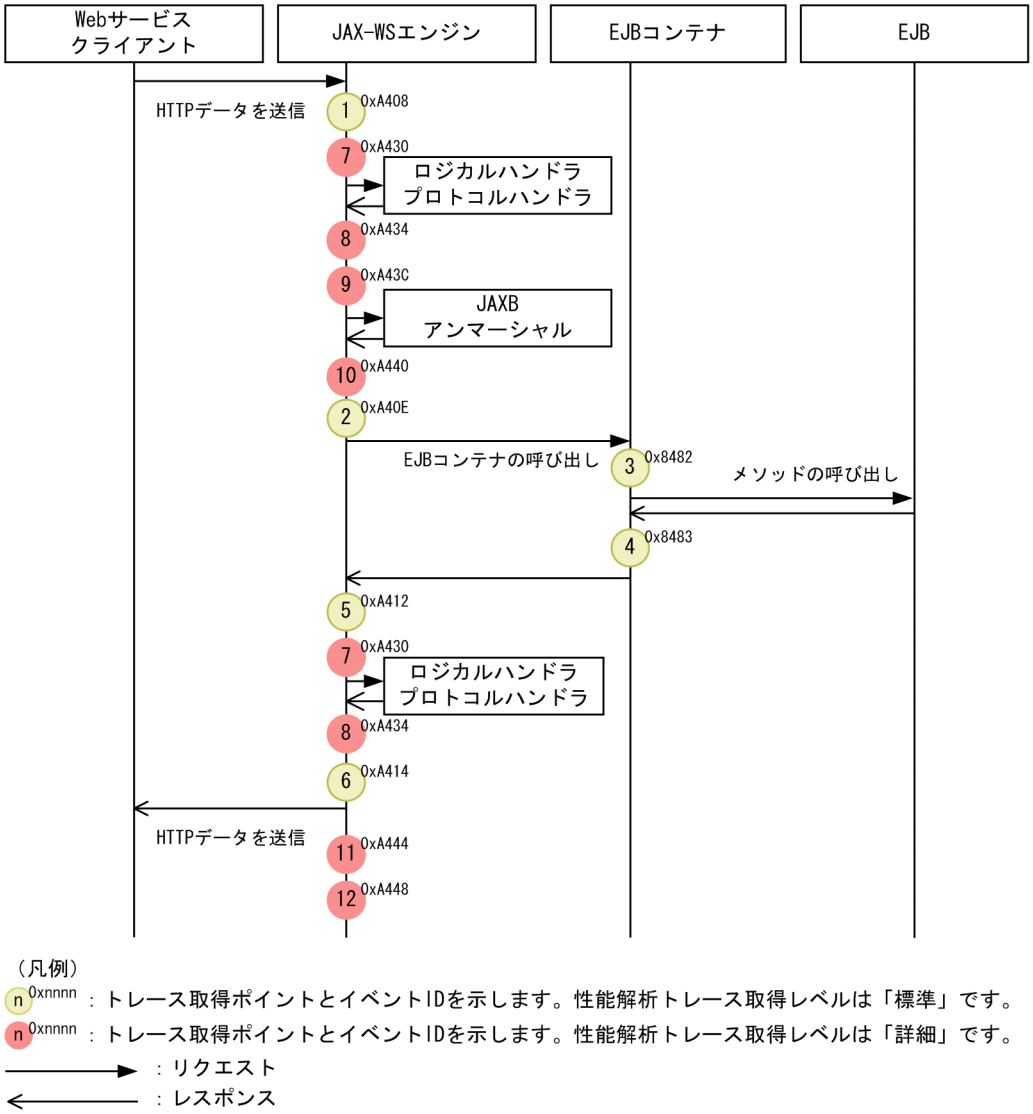
（凡例）

―：オプション情報はありません。

(b) EJB の Web サービス呼び出し時のトレース取得ポイント

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。

図 29-6　EJB の Web サービス呼び出し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 29-17　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

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
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 29-18　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

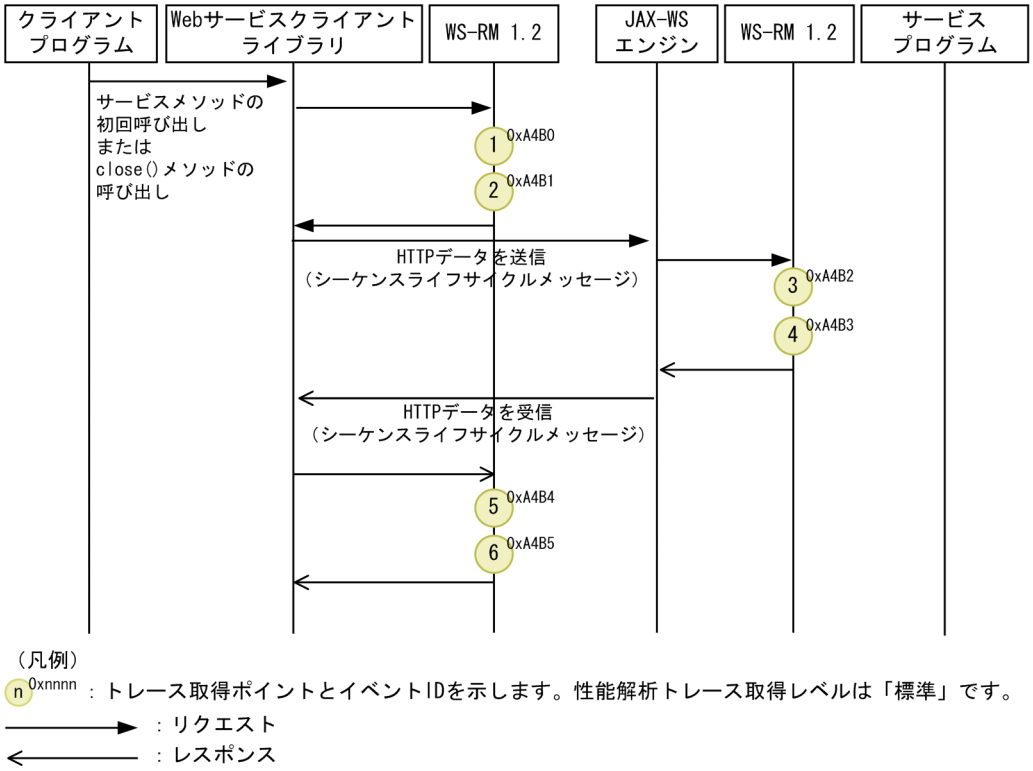
（凡例）

―：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 29-7　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。

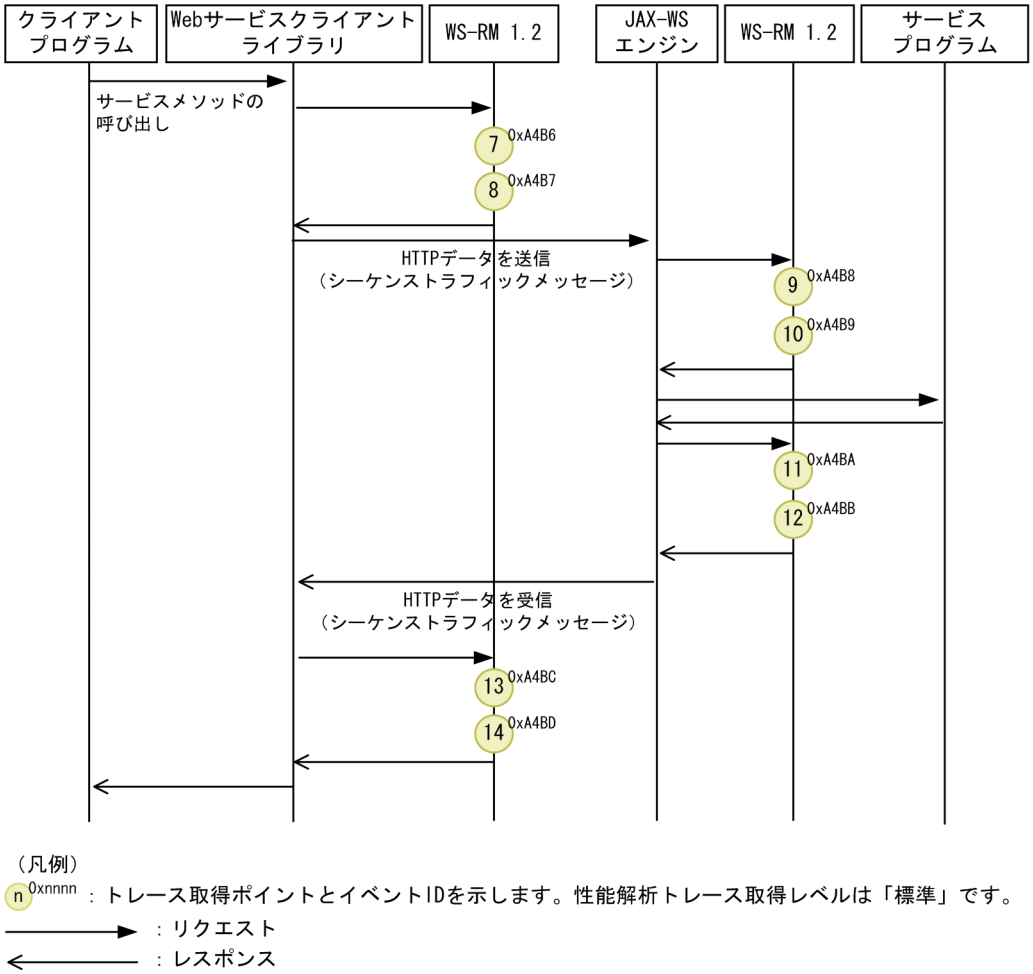
表 29-19　標準レベルでのトレース取得ポイントと出力される情報（WS-RM 1.2 機能のライフサイクルメッセージ）

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]

図中の番号	取得ポイント	出力情報	
		イベント ID	オプション情報
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] • 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 29-8　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



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 29-20　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

図中の番号	取得ポイント	出力情報	
		イベント ID	オプション情報
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

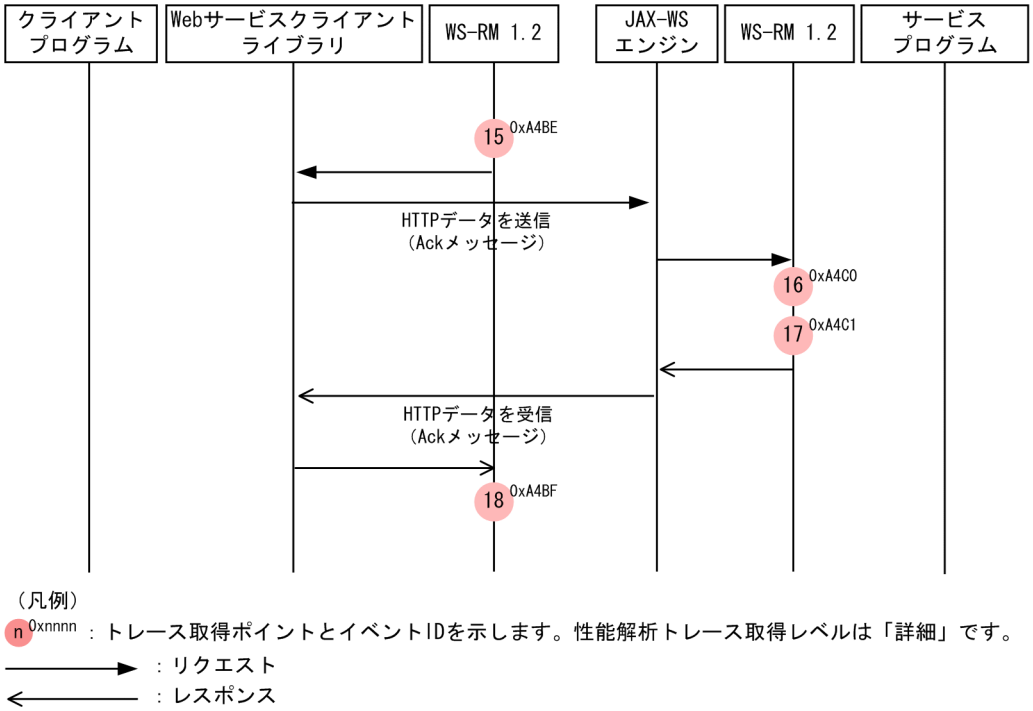
重複したメッセージを受信した場合は出力されません。

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 29-9　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 29-21　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

29.4.3　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。

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，サービスプログラム（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。

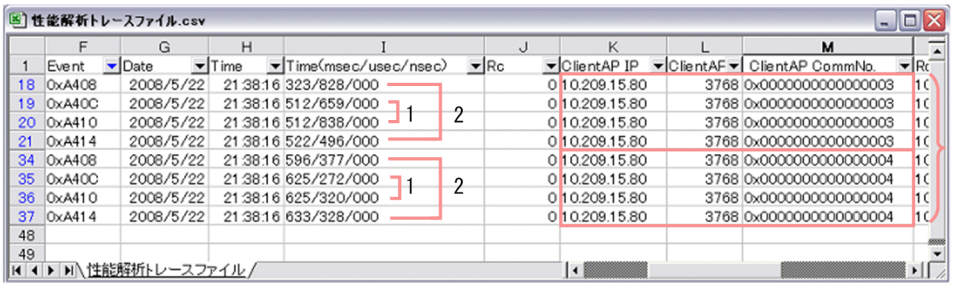
イベント ID「0xA408」，「0xA40C」，「0xA410」，および「0xA414」をキーにして，性能解析トレースファイルをフィルタリングします。各イベント ID に対するトレース取得ポイントを次の表に示します。

表 29-22　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 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「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 29-10　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 AP 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 AP 情報が出力されます。

同一クライアント AP 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，サービスプログラム（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

付録 B　旧バージョンの WS-RM 機能

付録 C　POJO の Web サービスから EJB の Web サービスへの移行

付録 D　JAX-WS エンジンのメモリ使用量の算出

付録 E　このマニュアルの参考情報

付録 F　用語解説

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，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ます。このとき，JAX-WS 2.1 仕様および Cosminexus の JAX-WS 機能のサポート範囲に従って作成してください。

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，および JAX-WS 機能は，Cosminexus 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

Cosminexus をバージョンアップインストールする場合，マニュアル「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「12.1.2　アプリケーションサーバの移行の手順」または「12.2.2　アプリケーションサーバの移行の手順」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，および JAX-WS 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「Cosminexus 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「2.3　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 **エンジンを利用する場合**

JAX-WS 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」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
...
```

● 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

「hitsaaj.jar」と「cjjaxws.jar」の両方のコメントを外した場合，または両方のコメントを付け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。

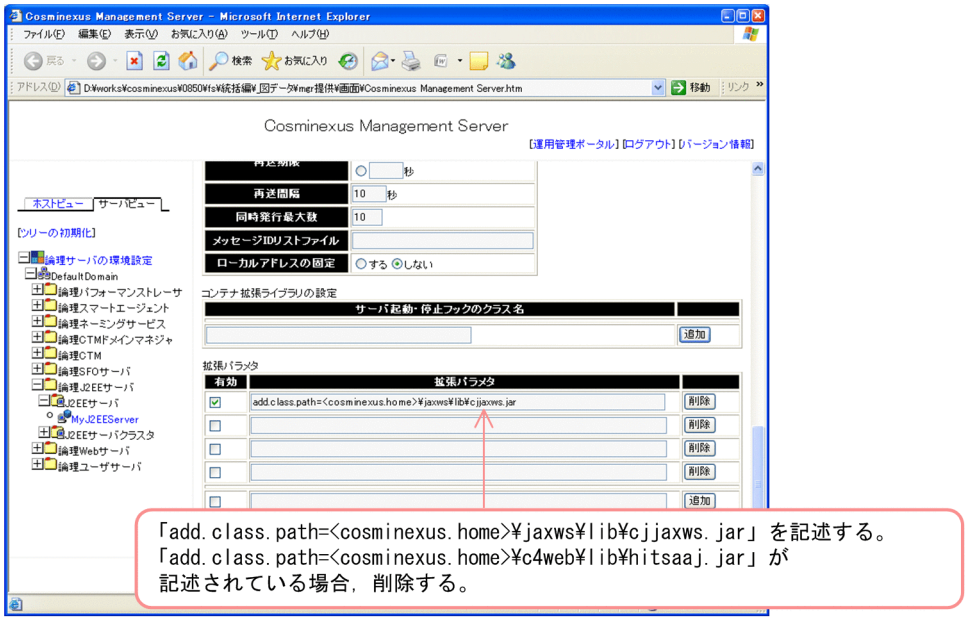
<Cosminexus のインストールディレクトリ >/CC/server/usrconf/ejb/<J2EE サーバ名 >/usrconf.cfg

(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。

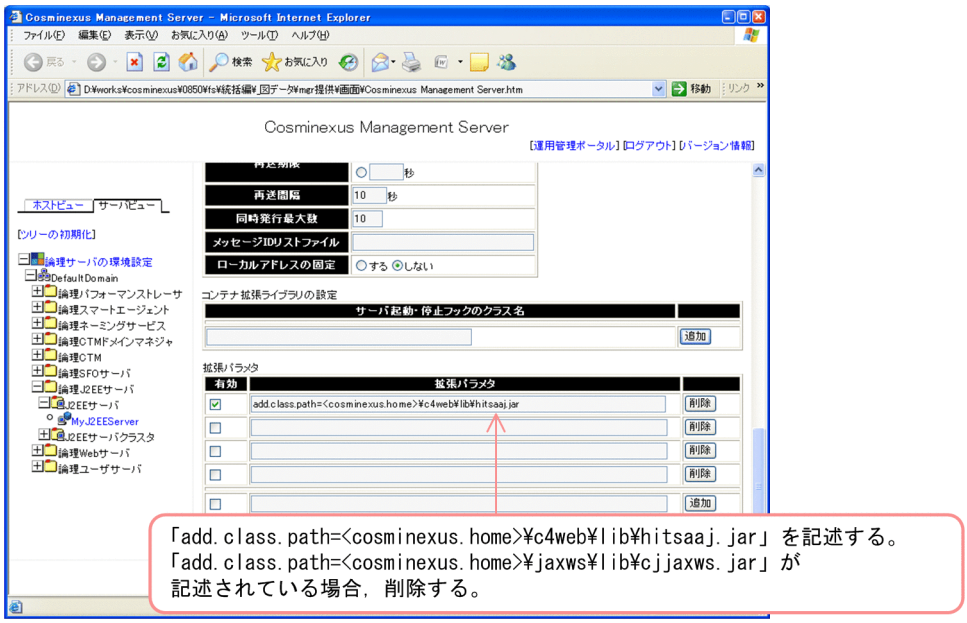
JAX-WS エンジンを利用する場合の設定例を次に示します。

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「Cosminexus 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「10.9.2 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「Cosminexus 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「Cosminexus 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>
```

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		
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。	
14	base64Binary 型				メモリに依存します。
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		
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。

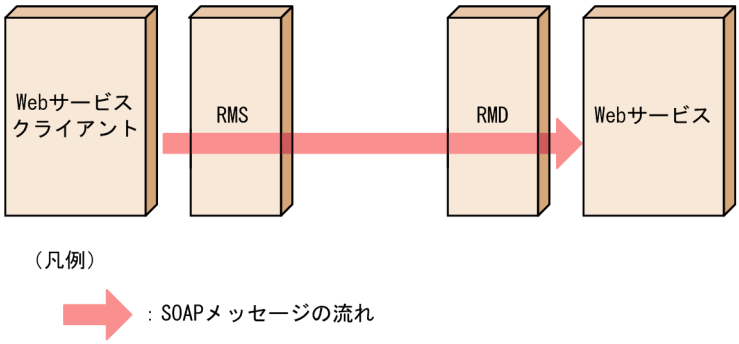
付録 B　旧バージョンの WS-RM 機能

WS-RM 1.1 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。

WS-RM 1.1 機能は，WS-RM 1.2 機能と互換性がありません。新規に WS-RM 機能を利用する場合，WS-RM 1.2 機能を利用することをお勧めします。WS-RM 1.2 機能については，「24.　WS-RM 1.2 機能」を参照してください。

WS-RM 1.1 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。

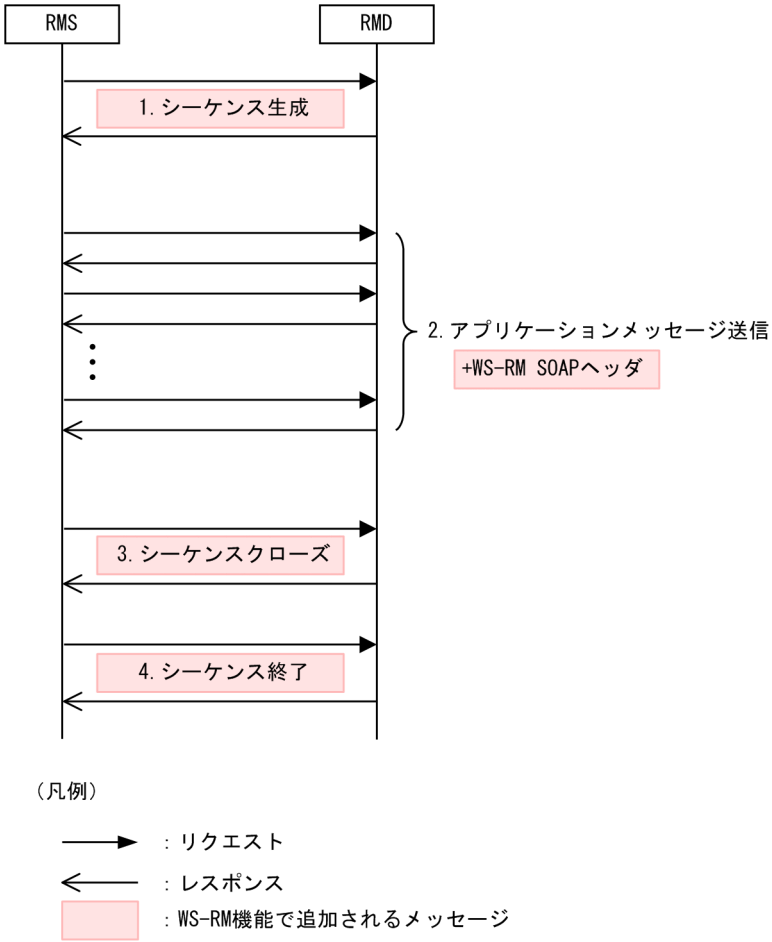
図 B-1　WS-RM 1.1 機能を使用した場合の SOAP メッセージの送受信の流れ



付録 B.1　WS-RM 1.1 機能を使用したメッセージの流れ

WS-RM 1.1 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 B-2　WS-RM 1.1 機能を使用したメッセージの流れ



1. **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 1.1 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。
2. **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。
3. **シーケンスクローズ（任意）**
   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。

4. **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。

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。

付録 B.2　WS-RM 1.1 機能を使用する場合のシステム構成

WS-RM 1.1 機能を使用する場合のシステム構成には，次の種類があります。

- Web サービスと Web サービスクライアントが同じ Cosminexus 上にある場合
- Web サービスと Web サービスクライアントが異なる Cosminexus 上にある場合
- Web サービスが Cosminexus 以外のアプリケーションサーバ上にある場合※

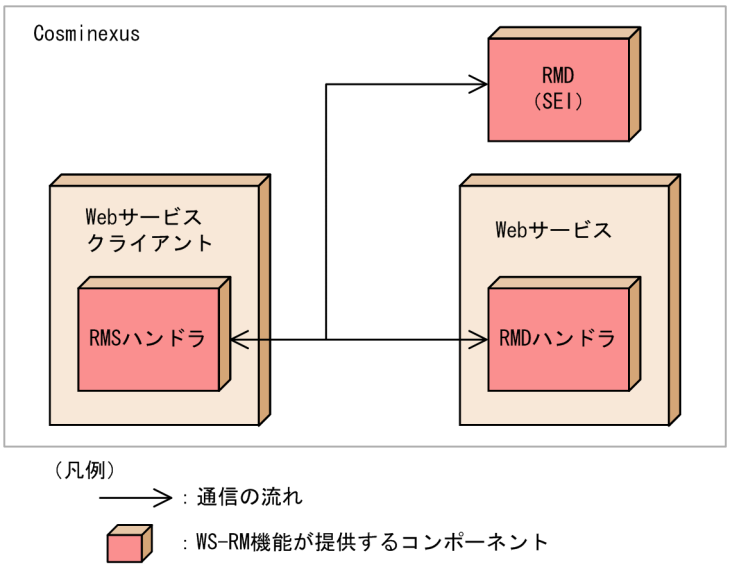
注※
この構成は，Cosminexus 以外のアプリケーションサーバが WS-RM 1.1 仕様に準拠していて，かつ WS-RM 1.1 機能のサポート範囲内で使用する場合だけ使用できます。

なお，Web サービスクライアントが Cosminexus 以外のアプリケーションサーバ上にある構成はサポートしていません。

（1）Web サービスと Web サービスクライアントが同じ Cosminexus 上にある場合

Web サービスと Web サービスクライアントが同じ Cosminexus 上にある場合の構成例を次に示します。

図 B-3　WS-RM 1.1 機能の構成例（Web サービスと Web サービスクライアントが同じ Cosminexus 上にある場合）

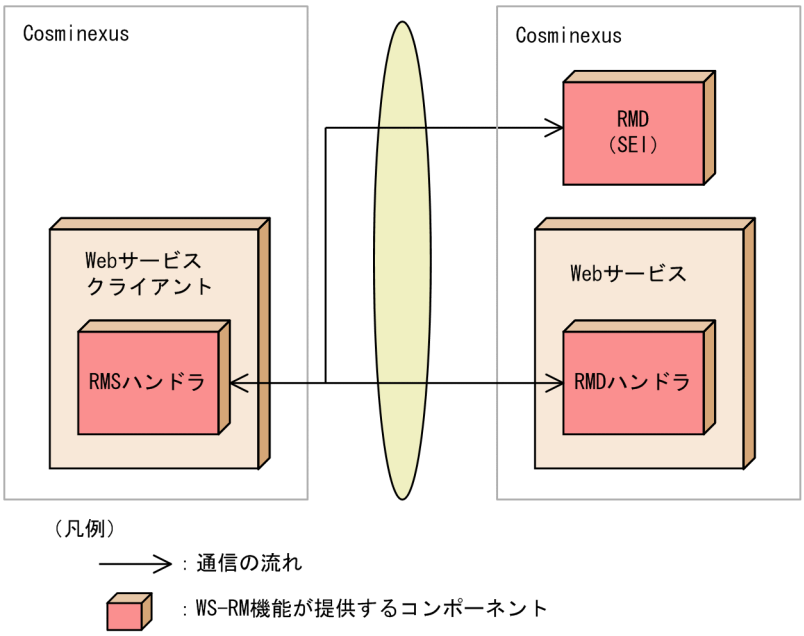


この構成では，Web サービスクライアントと Web サービスが同じ Cosminexus 上で動作します。そのため，RMS ハンドラ，RMD ハンドラ，RMD（SEI）のすべてを同じアプリケーションサーバに配置します。

（2）Web サービスと Web サービスクライアントが異なる Cosminexus 上にある場合

Web サービスと Web サービスクライアントが異なる Cosminexus 上にある場合の構成例を次に示します。

図 B-4　WS-RM 1.1 機能の構成例（Web サービスと Web サービスクライアントが異なる Cosminexus 上にある場合）

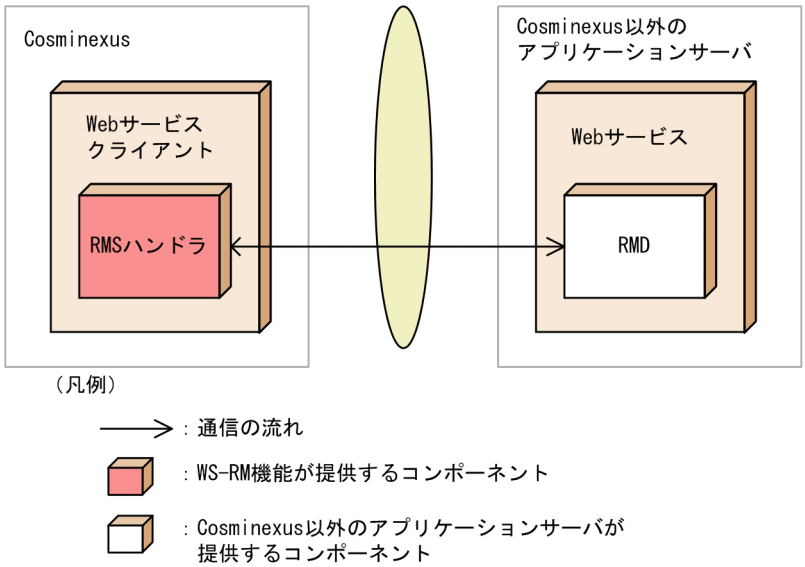


この構成では，Web サービスクライアントと Web サービスが異なる Cosminexus 上で動作します。そのため，Web サービスクライアントが動作する Cosminexus 上に RMS ハンドラを配置し，Web サービスが動作する Cosminexus 上に RMD ハンドラおよび RMD（SEI）を配置します。

（3）Web サービスが Cosminexus 以外のアプリケーションサーバ上にある場合

Web サービスが Cosminexus 以外のアプリケーションサーバ上にある場合の構成例を次に示します。この構成は，Cosminexus 以外のアプリケーションサーバが WS-RM 1.1 仕様に準拠していて，かつ，WS-RM 1.1 機能のサポート範囲内で使用する場合だけ使用できます。

図 B-5　WS-RM 1.1 機能の構成例（Web サービスが Cosminexus 以外のアプリケーションサーバ上にある場合）



この構成では，Web サービスクライアントが Cosminexus 上で動作し，Web サービスが Cosminexus 以外のアプリケーションサーバ上で動作します。そのため，Web サービスクライアントが動作する Cosminexus 上に RMS ハンドラを配置します。RMS ハンドラは，Cosminexus 以外のアプリケーションサーバが提供する RMD と通信します。

付録 B.3　WS-RM 1.1 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，順序制御をする機能です。

ここでは，送達保証の種類および設定方法について説明します。

（1） 送達保証の種類

送達保証の種類および Cosminexus の WS-RM 1.1 機能でのサポート範囲を次の表に示します。

表 B-1　送達保証の種類と Cosminexus の WS-RM 1.1 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	×

項番	種類	動作	RMS の処理	RMD の処理	サポート
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。

(2) 送達保証の設定方法

送達保証は，Web サービスおよび Web サービスクライアントに次のハンドラを追加することで使用できます。

- Web サービスクライアント
  com.cosminexus.wsrm.rms.handler.RMSAtMostOnceHandler
- Web サービス
  com.cosminexus.wsrm.rmd.handler.RMDAtMostOnceHandler

これらのハンドラは，最後のハンドラ（最もネットワーク寄りのハンドラ）として配置してください。

注
RMSAtMostOnceHandler はスレッドセーフではありません。
RMSAtMostOnceHandler オブジェクトや，RMSAtMostOnceHandler オブジェクトを含む HandlerResolver や Service をスレッド間で共有しないでください。

付録 B.4　WS-RM 1.1 機能の API 一覧

WS-RM 1.1 機能の API のインタフェース，クラス，および列挙型の一覧を次の表に示します。

表 B-2　WS-RM 1.1 機能の API のインタフェース，クラス，および列挙型の一覧

項番	インタフェース，クラス，または列挙型名	説明
com.cosminexus.wsrm.rms.api パッケージ		
1	RMSProvider	WS-RM 1.1 機能を使用する場合に最初に呼び出すクラスです。メインの API 群である RMSMessaging インタフェースを取得します。
2	RMSMessaging	WS-RM 1.1 機能の操作を実行するための，クライアントから使用するインタフェースです。シーケンス生成やシーケンス終了などの操作を実行できます。
3	RMSConfigurations	RMS のプロパティをプログラムから動的に設定する場合に使用するクラスです。

項番	インタフェース，クラス，または列挙型名	説明
com.cosminexus.wsrm.common.exception パッケージ		
4	RMException	WS-RM 1.1 機能で使用する例外クラスです。
5	FaultType	例外に対応づけられたフォルトの種類を表す列挙型です。
com.cosminexus.wsrm.rms.handler パッケージ		
6	RMSAtMostOnceHandler	AtMostOnce 送達保証に対応した RMS 機能を提供するハンドラです。
com.cosminexus.wsrm.rmd.handler パッケージ		
7	RMDAtMostOnceHandler	AtMostOnce 送達保証に対応した RMD 機能を提供するハンドラです。
com.cosminexus.wsrm.common.model パッケージ		
8	SequenceStatesEnum	シーケンスの状態を表す列挙型です。

(1) com.cosminexus.wsrm.rms.api.RMSProvider クラス

com.cosminexus.wsrm.rms.api.RMSProvider クラスの目的およびメソッドについて説明します。

(a) 目的

WS-RM 1.1 機能を使用するためには，WS-RM 1.1 機能の API を呼び出す必要があります。WS-RM 1.1 機能を利用するユーザは，最初に com.cosminexus.wsrm.rms.api.RMSProvider クラスを使用して，API を提供する RMSMessaging クラスを取得する必要があります。

(b) メソッド

com.cosminexus.wsrm.rms.api.RMSProvider クラスのメソッドの一覧を次の表に示します。

表 B-3　com.cosminexus.wsrm.rms.api.RMSProvider クラスのメソッドの一覧

項番	メソッド名／説明	
1	public static com.cosminexus.wsrm.rms.api.RMSMessaging getRMSMessaging(Service service) throws com.cosminexus.wsrm.common.RMException	
	説明	RMSMessaging オブジェクトを取得するためのメソッドです。
	引数	javax.xml.ws.Service service: Web サービスとの通信で使用するサービスクラスです。

項番	メソッド名／説明	
	例外	com.cosminexus.wsrm.common.exception.RMException: 次の場合に発生します。 • 引数の service に null を指定した場合 • 処理中にエラーが発生した場合
2	public static com.cosminexus.wsrm.rms.api.RMSMessaging getRMSMessaging (Service service, PortInfo portInfo) throws com.cosminexus.wsrm.common.RMException	
	説明	RMSMessaging オブジェクトを取得するためのメソッドです。
	引数	javax.xml.ws.Service service: Web サービスとの通信で使用するサービスクラスです。 javax.xml.ws.handler.PortInfo portInfo: HandlerResolver オブジェクトから HandlerChain オブジェクトを取得するための PortInfo オブジェクトです。
	例外	com.cosminexus.wsrm.common.exception.RMException: 次の場合に発生します。 • 引数の service に null を指定した場合 • 処理中にエラーが発生した場合
3	public static com.cosminexus.wsrm.rms.api.RMSMessaging getRMSMessaging (Service service, RMSConfigurations rmsConfigurations) throws com.cosminexus.wsrm.common.RMException	
	説明	RMSMessaging オブジェクトを取得するためのメソッドです。
	引数	javax.xml.ws.Service service: Web サービスとの通信で使用するサービスクラスです。 com.cosminexus.wsrm.common.exception.RMSConfigurations rmsConfigurations: プログラムから動的に設定する場合に使用する RMS を設定します。
	例外	com.cosminexus.wsrm.common.exception.RMException: 次の場合に発生します。 • 引数の service に null を指定した場合 • 処理中にエラーが発生した場合
4	public static com.cosminexus.wsrm.rms.api.RMSMessaging getRMSMessaging (Service service, PortInfo portInfo, RMSConfigurations rmsConfigurations) throws com.cosminexus.wsrm.common.RMException	
	説明	RMSMessaging オブジェクトを取得するためのメソッドです。
	引数	javax.xml.ws.Service service: Web サービスとの通信で使用するサービスクラスです。 javax.xml.ws.handler.PortInfo portInfo: HandlerResolver オブジェクトから HandlerChain オブジェクトを取得するための PortInfo オブジェクトです。 com.cosminexus.wsrm.common.exception.RMSConfigurations rmsConfigurations: プログラムから動的に設定する場合に使用する RMS を設定します。
	例外	com.cosminexus.wsrm.common.exception.RMException: 次の場合に発生します。 • 引数の service に null を指定した場合 • 処理中にエラーが発生した場合

(2) com.cosminexus.wsrm.rms.api.RMSMessaging インタフェース

com.cosminexus.wsrm.rms.api.RMSMessaging インタフェースの目的およびメソッドについて説明します。

(a) 目的

シーケンス生成やシーケンス終了など，WS-RM 1.1 機能のメイン API を提供します。

(b) メソッド

com.cosminexus.wsrm.rms.api.RMSMessaging インタフェースのメソッドの一覧を次の表に示します。

表 B-4　com.cosminexus.wsrm.rms.api.RMSMessaging インタフェースのメソッドの一覧

項番	メソッド名／説明	
1	public boolean createSequence() throws com.cosminexus.wsrm.common.exception.RMException	
	説明	シーケンスを生成します。なお，このメソッドを呼び出す前に，RMS 定義ファイルまたは RMSConfigurations クラスを使用して，正しい RMD（SEI）の URL を設定する必要があります。 シーケンス生成に成功した場合は true，失敗した場合は false を返します。
	例外	com.cosminexus.wsrm.common.exception.RMException: 処理中にエラーが発生した場合に発生します。
2	public boolean closeSequence() throws com.cosminexus.wsrm.common.exception.RMException	
	説明	シーケンスをクローズします。 シーケンスのクローズに成功した場合は true，失敗した場合は false を返します。
	例外	com.cosminexus.wsrm.common.exception.RMException: 処理中にエラーが発生した場合に発生します。
3	public boolean terminateSequence() throws com.cosminexus.wsrm.common.exception.RMException	
	説明	シーケンスを終了します。 シーケンスの終了に成功した場合は true，失敗した場合は false を返します。
	例外	com.cosminexus.wsrm.common.exception.RMException: 処理中にエラーが発生した場合に発生します。
4	public long getLastMessageNumber() throws com.cosminexus.wsrm.common.exception.RMException	
	説明	最後に送信したアプリケーションメッセージのメッセージ番号を取得します。
	例外	com.cosminexus.wsrm.common.exception.RMException: 処理中にエラーが発生した場合に発生します。

項番	メソッド名／説明	
5	public boolean getMessageStatus(long messageNumber) throws com.cosminexus.wsrm.common.exception.RMException	
	説明	特定のメッセージ番号を持つメッセージが届いたかどうかを取得します。 メッセージが届いている場合は true，届いていない場合は false を返します。
	引数	long messageNumber: 状態を取得したいメッセージのメッセージ番号です。
	例外	com.cosminexus.wsrm.common.exception.RMException: 処理中にエラーが発生した場合に発生します。
6	public com.cosminexus.wsrm.common.model.SequenceStateEnum getSequenceState() throws com.cosminexus.wsrm.common.exception.RMException	
	説明	使用中のシーケンスの状態を取得します。
	例外	com.cosminexus.wsrm.common.exception.RMException: 処理中にエラーが発生した場合に発生します。
7	public boolean isExpired() throws com.cosminexus.wsrm.common.exception.RMException	
	説明	シーケンスの有効期限を超過しているかどうかを取得します。 有効期限を超過している場合は true，超過していない場合は false を返します。
	例外	com.cosminexus.wsrm.common.exception.RMException: 処理中にエラーが発生した場合に発生します。

(3) com.cosminexus.wsrm.rms.api.RMSConfigurations クラス

com.cosminexus.wsrm.rms.api.RMSConfigurations クラスの目的およびメソッドについて説明します。

(a) 目的

プログラムから RMS を動的に設定するためのクラスです。RMS 定義ファイルより優先度が高いため，com.cosminexus.wsrm.rms.api.RMSConfigurations クラスを使用することで，RMS 定義ファイルの設定を上書きできます。

(b) メソッド

com.cosminexus.wsrm.rms.api.RMSConfigurations クラスのメソッドの一覧を次の表に示します。

表 B-5　com.cosminexus.wsrm.rms.api.RMSConfigurations クラスのメソッドの一覧

項番	メソッド名／説明	
1	public final void setExpiresDuration(long expiresDuration)	
	説明	生成するシーケンスの有効期限を設定します。生成済みのシーケンスの有効期限は変更されません。

項番	メソッド名／説明	
	引数	long expiresDuration: シーケンスの有効期限（秒）です。0 を指定した場合，有効期限は設定されません。
2	public final long getExpiresDuration()	
	説明	設定されているシーケンスの有効期限を取得します。
3	public final void setCwsrmSeiUrl(String wsrmSeiUrl)	
	説明	RMD（SEI）の URL を設定します。
	引数	String wsrmSeiUrl: RMD（SEI）の URL です。
4	public final String getCwsrmSeiUrl()	
	説明	設定されている RMD（SEI）の URL を取得します。

(4) com.cosminexus.wsrm.common.exception.RMException クラス

com.cosminexus.wsrm.common.exception.RMException クラスの目的およびメソッドについて説明します。

(a) 目的

WS-RM 1.1 機能でエラーが発生した場合にスローされる例外クラスです。

(b) メソッド

com.cosminexus.wsrm.common.exception.RMException クラスのメソッドの一覧を次の表に示します。

表 B-6　com.cosminexus.wsrm.common.exception.RMException クラスのメソッドの一覧

項番	メソッド名／説明	
1	public RMException(Throwable throwable)	
	説明	RMException クラスのコンストラクタです。
	引数	Throwable throwable: 原因例外です。
2	public RMException(String errorCode, String errorMessage)	
	説明	RMException クラスのコンストラクタです。
	引数	String errorCode: エラーコードです。 String errorMessage: エラーメッセージです。
3	public RMException(String errorCode, String errorMessage, Throwable throwable)	
	説明	RMException クラスのコンストラクタです。

項番	メソッド名／説明	
	引数	String errorCode: 　エラーコードです。 String errorMessage: 　エラーメッセージです。 Throwable throwable: 　原因例外です。
4	public RMException(String errorCode, String errorMessage, FaultType faultType)	
	説明	RMException クラスのコンストラクタです。
	引数	String errorCode: 　エラーコードです。 String errorMessage: 　エラーメッセージです。 com.cosminexus.wsrm.common.exception.FaultType faultType: 　フォルトの種類です。
5	public RMException(String errorCode, String errorMessage, FaultType faultType, Throwable throwable)	
	説明	RMException クラスのコンストラクタです。
	引数	String errorCode: 　エラーコードです。 String errorMessage: 　エラーメッセージです。 com.cosminexus.wsrm.common.exception.FaultType faultType: 　フォルトの種類です。 Throwable throwable: 　原因例外です。
6	public final com.cosminexus.wsrm.common.exception.FaultType getFaultType()	
	説明	例外に対応づけられているフォルトの種類を取得します。
7	public final String getErrorCode()	
	説明	例外に対応づけられているエラーコードを取得します。
8	public final String getErrorDetails()	
	説明	例外の詳細情報を取得します。

(5) com.cosminexus.wsrm.common.exception.FaultType 列挙型

com.cosminexus.wsrm.common.exception.FaultType 列挙型の目的および要素について説明します。

(a) 目的

RMException クラスに対応づけられたフォルトの種類を表す列挙型です。

(b) 要素

com.cosminexus.wsrm.common.exception.FaultType 列挙型の要素の一覧を次の表に示します。

表 B-7　com.cosminexus.wsrm.common.exception.FaultType 列挙型の要素の一覧

項番	要素名	説明
1	UNKNOWN_SEQUENCE_FAULT	wsrm:UnknownSequence フォルトを受信したことを示します。
2	CREATE_SEQUENCE_REFUSED_FAULT	wsrm:CreateSequenceRefused フォルトを受信したことを示します。
3	SEQUENCE_CLOSED_FAULT	wsrm:SequenceClosed フォルトを受信したことを示します。
4	SEQUENCE_TERMINATED_FAULT	wsrm:SequenceTerminated フォルトを受信したことを示します。
5	WSRM_REQUIRED_FAULT	wsrm:WSRMRequired フォルトを受信したことを示します。

（6） com.cosminexus.wsrm.rms.handler.RMSAtMostOnceHandler クラス

com.cosminexus.wsrm.rms.handler.RMSAtMostOnceHandler クラスの目的およびメソッドについて説明します。

(a) 目的

AtMostOnce 送達保証に対応した RMS ハンドラクラスです。WS-RM 1.1 機能を使用する場合は，com.cosminexus.wsrm.rms.handler.RMSAtMostOnceHandler クラスを Web サービスクライアントの最後のハンドラとして組み込む必要があります。

com.cosminexus.wsrm.rms.handler.RMSAtMostOnceHandler クラスはスレッドセーフではありません。RMSAtMostOnceHandler オブジェクトや，
RMSAtMostOnceHandler オブジェクトを含む HandlerResolver や Service オブジェクトをスレッド間で共有しないでください。

(b) メソッド

com.cosminexus.wsrm.rms.handler.RMSAtMostOnceHandler クラスのメソッドの一覧を次の表に示します。

表 B-8　com.cosminexus.wsrm.rms.handler.RMSAtMostOnceHandler クラスのメソッドの一覧

項番	メソッド名	説明
1	public RMSAtMostOnceHandler()	RMSAtMostOnceHandler クラスのコンストラクタです。

（7） com.cosminexus.wsrm.rmd.handler.RMDAtMostOnceHandler クラス

com.cosminexus.wsrm.rmd.handler.RMDAtMostOnceHandler クラスの目的およびメ

ソッドについて説明します。

(a) 目的

AtMostOnce 送達保証に対応した RMD ハンドラクラスです。WS-RM 1.1 機能を使用する場合は，com.cosminexus.wsrm.rmd.handler.RMDAtMostOnceHandler クラスを Web サービスの最後のハンドラとして組み込む必要があります。

(b) メソッド

com.cosminexus.wsrm.rmd.handler.RMDAtMostOnceHandler クラスのメソッドの一覧を次の表に示します。

表 B-9　com.cosminexus.wsrm.rmd.handler.RMDAtMostOnceHandler クラスのメソッドの一覧

項番	メソッド名	説明
1	public RMDAtMostOnceHandler()	RMDAtMostOnceHandler クラスのコンストラクタです。

(8) com.cosminexus.wsrm.common.model.SequenceStatesEnum 列挙型

com.cosminexus.wsrm.common.model.SequenceStatesEnum 列挙型の目的および要素について説明します。

(a) 目的

シーケンスの状態を表す列挙型です。

(b) 要素

com.cosminexus.wsrm.common.model.SequenceStatesEnum 列挙型の要素の一覧を次の表に示します。

表 B-10　com.cosminexus.wsrm.common.model.SequenceStatesEnum 列挙型の要素の一覧

項番	要素名	説明
1	NONE	シーケンスがないことを示します。
2	CREATING	シーケンスが生成中であることを示します。
3	CREATED	シーケンスが生成されたことを示します。
4	CLOSING	シーケンスがクローズ中であることを示します。
5	CLOSED	シーケンスがクローズされたことを示します。
6	TERMINATING	シーケンスが終了中であることを示します。
7	TERMINATED	シーケンスが終了されたことを示します。

付録 B.5　WS-RM 1.1 機能のプロパティ設定

WS-RM 1.1 機能には，既存の共通定義ファイルに記述する設定と，WS-RM 1.1 機能独自の RMS 定義ファイルに記述する設定があります。

それぞれのファイルに設定するプロパティについて説明します。

(1) 共通定義ファイルに追加するプロパティ

共通定義ファイル（cjwconf.properties）に追加するプロパティを次の表に示します。

表 B-11　共通定義ファイル（cjwconf.properties）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	com.cosminexus.wsrm.rmd.sequence_expire_duration	RMD 側のシーケンスの有効期限を設定します。 RMS が要求した有効期限と，このプロパティに指定した値とを比較して，小さい値をシーケンスの有効期限として採用します。 0 を指定すると，シーケンスの有効期限は設定されません。無期限に使用できます。	秒	0 ～ 3600	300

(2) RMS 定義ファイルに追加するプロパティ

RMS 定義ファイル（cwsrm_rms_conf.properties）に追加するプロパティを次の表に示します。RMS 定義ファイルは，Web サービスクライアント実行時に Web サービスクライアントのクラスパスに含めてください。

表 B-12　RMS 定義ファイル（cwsrm_rms_conf.properties）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	com.cosminexus.wsrm.rms.sequence_expire_duration	RMS 側のシーケンスの有効期限を設定します。シーケンス生成リクエストで RMD に要求する値を指定してください。 0 を指定すると，シーケンスの有効期限は設定されません。無期限に使用できます。	秒	0 ～ 3600	300

項番	プロパティ	説明	単位	範囲	デフォルト値
2	com.cosminexus.wsrm.rms.cwsrm_sei_url	RMD（SEI）の URL を設定します。Web サービスクライアント実行時は，このプロパティを必ず指定してください。 このプロパティには次の値を指定します。 http://<*host*>:<*port*>/cwsrmsei1_1/WSRMSequenceService <*host*> と <*port*> は，RMD（SEI）が動作する環境に合わせて変更してください。	－	－	－

（凡例）
－：該当しません。

付録 B.6　WS-RM 1.1 機能の性能解析トレース（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。

Cosminexus システムが提供する性能解析トレースについては，マニュアル「Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編」の「4.6　性能解析トレース」を参照してください。

(1) 性能解析トレースの取得レベル

Cosminexus の JAX-WS 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 B-13　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）	Cosminexus の JAX-WS 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，Cosminexus の JAX-WS 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。

(2) 性能解析トレースのトレース出力情報

Cosminexus の JAX-WS 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 B-14　性能解析トレースのトレース出力情報

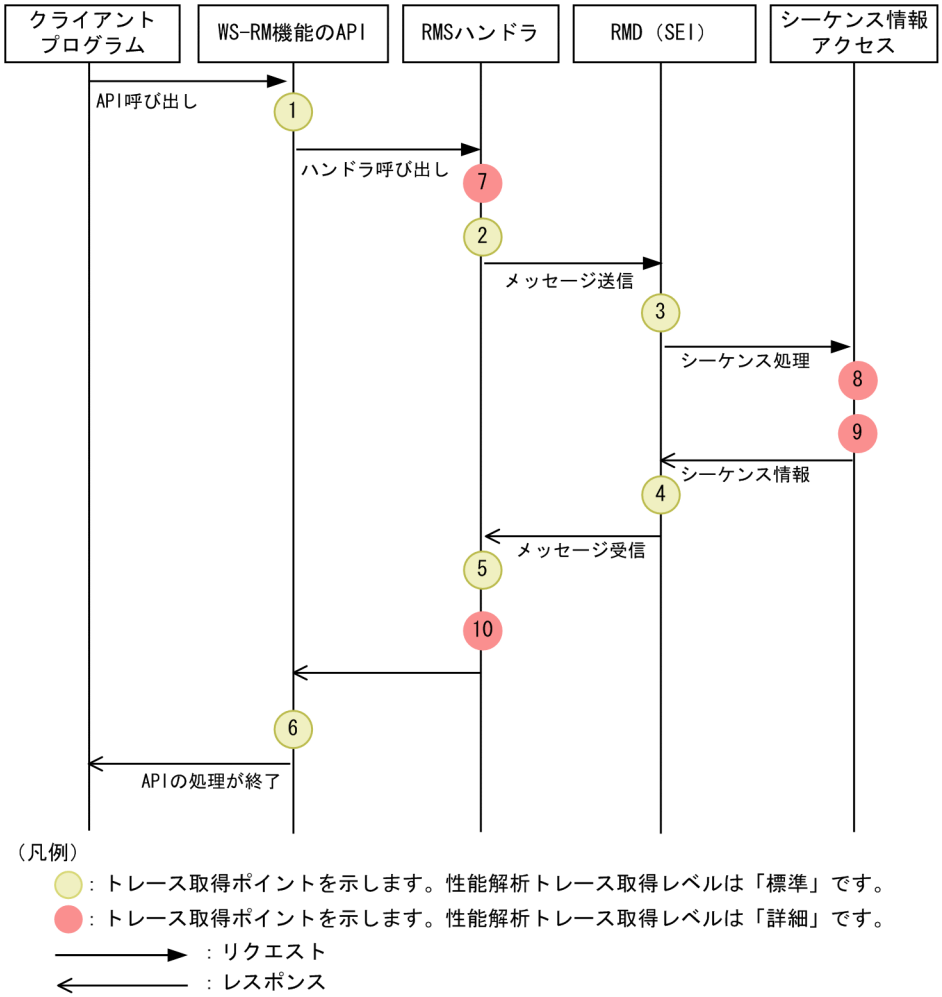
項番	ファイル項目	内容
1	イベント ID	トレース取得ポイントのイベント ID です。
2	クライアント AP 情報	ルート AP 情報の有効範囲内の Web サービスクライアント，Web サービス間で一意の情報です。ただし，WS-RM 1.2 機能の場合，一意ではありません。オプション情報を参照してください。
3	ルート AP 情報	一連のリクエストに対する処理で，一意の情報です。ただし，WS-RM 1.2 機能の場合，一意ではありません。オプション情報を参照してください。
4	リターンコード	トレース取得ポイントの種別（正常終了または異常終了）です。
5	インタフェース名	トレース取得ポイントのクラス名です。
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 AP 情報とルート AP 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。

（3）性能解析トレースのトレース取得ポイント（WS-RM 1.1 機能使用時）

WS-RM 1.1 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 B-6　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



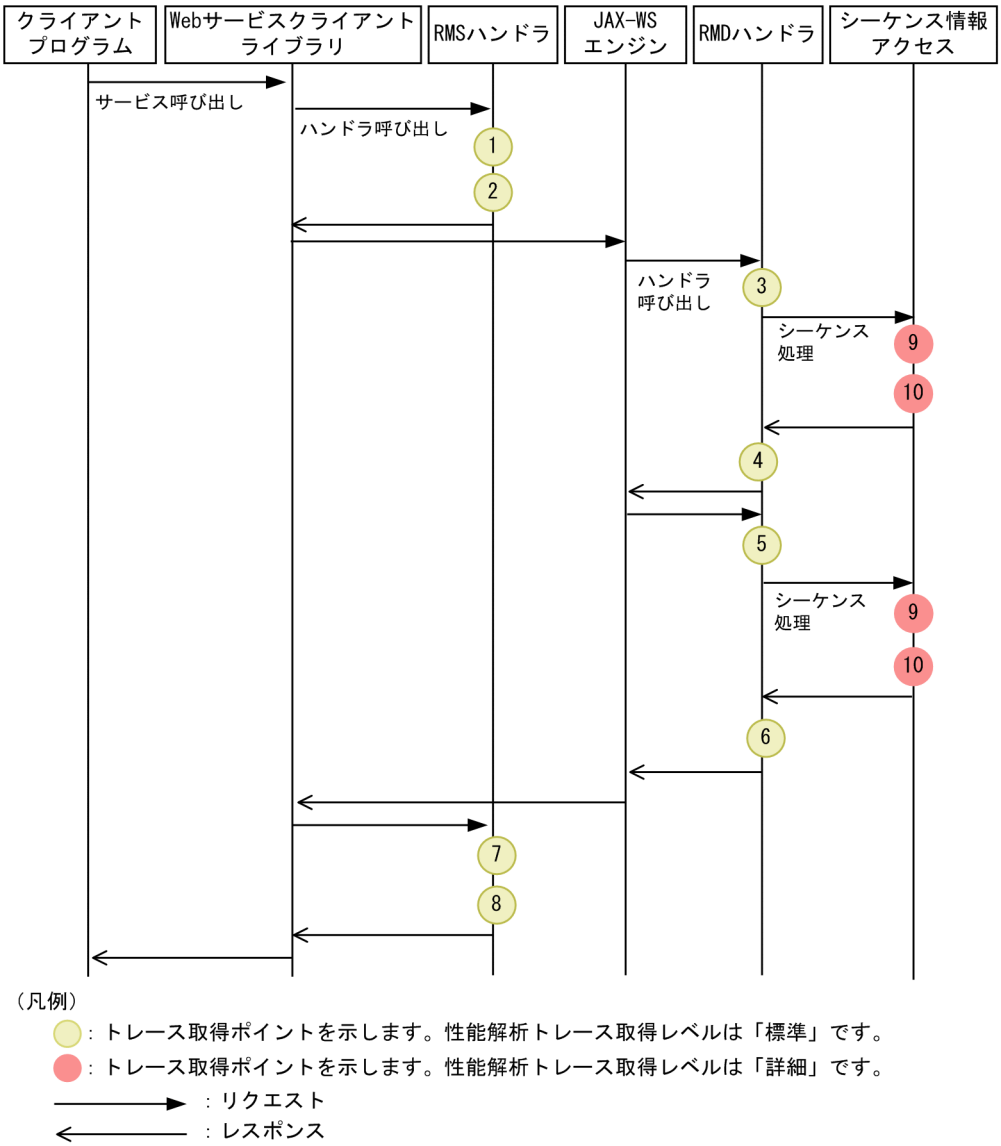
各トレース取得ポイントとイベント ID の対応を次の表に示します。

表 B-15　トレース取得ポイントと出力される情報（WS-RM 1.1 機能のライフサイクルメッセージ）

項番	取得レベル	取得ポイント	出力情報（イベント ID）
1	標準レベル	Web サービスクライアントによるシーケンス生成 API 呼び出し時	0xA4D0
		Web サービスクライアントによるシーケンスクローズ API 呼び出し時	0xA4D1
		Web サービスクライアントによるシーケンス終了 API 呼び出し時	0xA4D2
2		RMS ハンドラが SEI へメッセージを送信する直前	0xA4D3
3		SEI のシーケンス生成処理開始時	0xA4D4
		SEI のシーケンスクローズ処理開始時	0xA4D5
		SEI のシーケンス終了処理開始時	0xA4D6
4		SEI のシーケンス生成処理終了時	0xA4D7
		SEI のシーケンスクローズ処理終了時	0xA4D8
		SEI のシーケンス終了処理終了時	0xA4D9
5		RMS ハンドラが SEI からメッセージを受信した直後	0xA4DA
6		シーケンス生成 API 呼び出し終了時	0xA4DB
		シーケンスクローズ API 呼び出し終了時	0xA4DC
		シーケンス終了 API 呼び出し終了時	0xA4DD
7	詳細レベル	RMS ハンドラのシーケンス生成処理開始時	0xA4DE
		RMS ハンドラのシーケンスクローズ処理開始時	0xA4DF
		RMS ハンドラのシーケンス終了処理開始時	0xA4E0
8		シーケンス情報生成処理開始時	0xA4E1
		シーケンス情報更新処理開始時	0xA4E2
		シーケンス情報削除処理開始時	0xA4E3
		シーケンス情報読み込み処理開始時	0xA4EA
9		シーケンス情報生成処理終了時	0xA4E4
		シーケンス情報更新処理終了時	0xA4E5
		シーケンス情報削除処理終了時	0xA4E6
		シーケンス情報読み込み処理終了時	0xA4EB
10		RMS ハンドラのシーケンス生成処理終了時	0xA4E7
		RMS ハンドラのシーケンスクローズ処理終了時	0xA4E8
		RMS ハンドラのシーケンス終了処理終了時	0xA4E9

WS-RM 1.1 機能のシーケンス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 B-7　シーケンス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 B-16　トレース取得ポイントと出力される情報（WS-RM 1.1 機能のシーケンスメッセージ）

項番	取得レベル	取得ポイント	出力情報（イベント ID）
1	標準レベル	リクエストでの RMS ハンドラ呼び出し開始時	0xA4F0
2		リクエストでの RMS ハンドラ呼び出し終了時	0xA4F1
3		リクエストでの RMD ハンドラ呼び出し開始時	0xA4F2
4		リクエストでの RMD ハンドラ呼び出し終了時	0xA4F3
5		レスポンスでの RMD ハンドラ呼び出し開始時	0xA4F4
6		レスポンスでの RMD ハンドラ呼び出し終了時	0xA4F5
7		レスポンスでの RMS ハンドラ呼び出し開始時	0xA4F6
8		レスポンスでの RMS ハンドラ呼び出し終了時	0xA4F7
9	詳細レベル	シーケンス情報更新処理開始時	0xA4E2
		シーケンス情報削除処理開始時	0xA4E3
		シーケンス情報読み込み処理開始時	0xA4EA
10		シーケンス情報更新処理終了時	0xA4E5
		シーケンス情報削除処理終了時	0xA4E6
		シーケンス情報読み込み処理終了時	0xA4EB

付録 B.7　WS-RM 1.1 仕様のサポート範囲

WS-RM 1.1 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 1.1 仕様の該当個所（章節項）を，小分類は WS-RM 1.1 仕様の該当個所に記載されている内容を示します。

表 B-17　WS-RM 1.1 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	○
		ExactryOnce	×
		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		○

分類				サポート
大分類	小分類			
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		○
		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:Identifire		○
		wsrm:Expires		○
		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		○
		wsrm:Identifire		○
		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:Identifire		○
		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:Identifire		○
		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:Identifire		○
		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:Identifire		○

分類			サポート
大分類	小分類		
		wsrm:MessageNumber	○
3.8	Ack リクエスト※3		○
3.8	Ack リクエスト要素	wsrm:AckRequested	○
		wsrm:Identifire	○
		拡張要素／拡張属性※1	○
3.9	Ack		○
3.9	Ack 要素	wsrm:SequenceAcknowledgement	○
		wsrm:Identifire	○
		wsrm:AcknowledgementRange	○
		wsrm:None	○
		wsrm:Final	○
		wsrm:Nack※4	×
		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※5		△
4.2	wsrm:SequenceTerminated フォルト※6		△
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）

○：Cosminexus の WS-RM 1.1 機能でサポートしています。

×：Cosminexus の WS-RM 1.1 機能でサポートしていません。

△：Cosminexus の WS-RM 1.1 機能でサポートしていますが，一部制限があります。

注※ 1

Cosminexus の WS-RM 1.1 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，anonymous URI だけです。

注※ 3

Ack リクエストは常に Piggy-Backing で送信されます。

注※ 4

Cosminexus の WS-RM 1.1 機能では，受信したメッセージの範囲を wsrm:AcknowledgementRange 要素または wsrm:None 要素で表します。wsrm:Nack 要素は使用しません。wsrm:Nack 要素を含むメッセージを受信した場合，wsrm:Nack 要素は無視されます。

注※ 5

Cosminexus の WS-RM 1.1 機能では，wsrm:SequenceFault フォルトを送信しません。wsrm:FaultCode 要素および wsrm:Detail 要素は，soap:Fault 要素の子要素である detail 要素の子要素となります。なお，wsrm:SequenceFault フォルトを受信した場合は正常に処理されます。

注※ 6

Cosminexus の WS-RM 1.1 機能では，wsrm:SequenceTerminated フォルトではなく wsrm:UnknownSequence フォルトを送信します。なお，wsrm:SequenceTerminated フォルトを受信した場合は正常に処理されます。

付録 C　POJO の Web サービスから EJB の Web サービスへの移行

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 C-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="5" 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_5.xsd">

  <description>Sample application &amp;quot;statelessjava&amp;quot;</description>
  <display-name>Sample_application_statelessjava</display-name>
  <module>
    <ejb>statelessjava.jar</ejb>
  </module>
</application>
```

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 D　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 D.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。

付録 D.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 D.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 D.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 E　このマニュアルの参考情報

このマニュアルを読むに当たっての参考情報を示します。

付録 E.1　関連マニュアル

このマニュアルと関連する Cosminexus のマニュアルを次に示します。

- Cosminexus **アプリケーションサーバ** V8 **概説**（3020-3-U01）
  アプリケーションサーバの概要について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **システム構築・運用ガイド**（3020-3-U04）
  セットアップウィザードおよび Smart Composer 機能を使用したシステムの構築・運用の手順について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **機能解説 保守／移行／互換編**（3020-3-U10）
  アプリケーションサーバで構築したシステムの保守に関する機能，移行情報，および互換用機能について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **運用管理ポータル操作ガイド**（3020-3-U13）
  運用管理ポータルの使用方法について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **リファレンス コマンド編**（3020-3-U14）
  アプリケーションサーバを構築・運用するときに使用するコマンドについて説明しています。
- Cosminexus **アプリケーションサーバ** V8 **リファレンス 定義編 (サーバ定義)**（3020-3-U15）
  アプリケーションサーバを構築・運用するとき，またはアプリケーションを開発するときに使用するファイルのうち，J2EE サーバや Management Server などのサーバの定義に使用するファイルの形式について説明しています。
- Cosminexus **アプリケーションサーバ** V8 **アプリケーション開発ガイド**（3020-3-U25）
  Cosminexus システムで動作させるアプリケーションの開発方法について説明しています。
- Cosminexus **アプリケーションサーバ** V8 Cosminexus XML Processor **ユーザーズガイド**（3020-3-U27）
  Cosminexus XML Processor が提供する XML パーサ・XSLT トランスフォーマの機能，作成方法，および使用方法について説明しています。
- Cosminexus **アプリケーションサーバ** V8 SOAP **アプリケーション開発の手引**（3020-3-U30）
  SOAP アプリケーション開発支援機能および SOAP 通信基盤を使用した SOAP アプリケーションの開発，実行方法について説明しています。
- Cosminexus **アプリケーションサーバ** V8 Web **サービスセキュリティ 使用の手引**（3020-3-U32）
  Cosminexus が提供する Web サービスセキュリティ機能に対応した Web サービスの開発について説明しています。

- Cosminexus アプリケーションサーバ V8 メッセージ 2 KDJE-KDJW 編（3020-3-U42）
  Cosminexus で出力される KDJE から KDJW までのメッセージについて説明しています。

なお，このマニュアルでは，次のマニュアルについて，バージョン番号を省略して表記しています。マニュアルの正式名称とこのマニュアルでの表記を次の表に示します。

正式名称	このマニュアルでの表記
Cosminexus アプリケーションサーバ V8 概説	Cosminexus アプリケーションサーバ 概説
Cosminexus アプリケーションサーバ V8 システム構築・運用ガイド	Cosminexus アプリケーションサーバ システム構築・運用ガイド
Cosminexus アプリケーションサーバ V8 機能解説 保守／移行／互換編	Cosminexus アプリケーションサーバ 機能解説 保守／移行／互換編
Cosminexus アプリケーションサーバ V8 運用管理ポータル操作ガイド	Cosminexus アプリケーションサーバ 運用管理ポータル操作ガイド
Cosminexus アプリケーションサーバ V8 リファレンス コマンド編	Cosminexus アプリケーションサーバ リファレンス コマンド編
Cosminexus アプリケーションサーバ V8 リファレンス 定義編（サーバ定義）	Cosminexus 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）
Cosminexus アプリケーションサーバ V8 アプリケーション開発ガイド	Cosminexus アプリケーションサーバ アプリケーション開発ガイド
Cosminexus アプリケーションサーバ V8 Cosminexus XML Processor ユーザーズガイド	Cosminexus XML Processor ユーザーズガイド
Cosminexus アプリケーションサーバ V8 SOAP アプリケーション開発の手引	Cosminexus 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
Cosminexus アプリケーションサーバ V8 Web サービスセキュリティ 使用の手引	Cosminexus アプリケーションサーバ Web サービスセキュリティ 使用の手引
Cosminexus アプリケーションサーバ V8 メッセージ 2 KDJE-KDJW 編	Cosminexus アプリケーションサーバ メッセージ 2

付録 E.2　このマニュアルでの表記

このマニュアルで使用している表記と，対応する製品名を次に示します。

表記		製品名
Application Server	Application Server Enterprise	uCosminexus Application Server Enterprise
	Application Server Standard	uCosminexus Application Server Standard
Developer	Developer Professional	uCosminexus Developer Professional
	Developer Standard	uCosminexus Developer Standard
IPF		Itanium(R) Processor Family

表記			製品名
UNIX	AIX		AIX 5L V5.3
			AIX V6.1
			AIX V7.1
	HP-UX または HP-UX（IPF）		HP-UX 11i V2（IPF）
			HP-UX 11i V3（IPF）
	Linux	Linux（IPF）	Red Hat Enterprise Linux(R) AS 4（IPF）
			Red Hat Enterprise Linux(R) 5 Advanced Platform（Intel Itanium）
			Red Hat Enterprise Linux(R) 5（Intel Itanium）
		Linux（x86/AMD64 & Intel EM64T）	Red Hat Enterprise Linux(R) AS 4（x86）
			Red Hat Enterprise Linux(R) ES 4（x86）
			Red Hat Enterprise Linux(R) AS 4（AMD64 & Intel EM64T）
			Red Hat Enterprise Linux(R) ES 4（AMD64 & Intel EM64T）
			Red Hat Enterprise Linux(R) 5 Advanced Platform（x86）
			Red Hat Enterprise Linux(R) 5（x86）
			Red Hat Enterprise Linux(R) 5 Advanced Platform（AMD/Intel 64）
			Red Hat Enterprise Linux(R) 5（AMD/Intel 64）
			Red Hat Enterprise Linux(R) Server 6（32-bit x86）
			Red Hat Enterprise Linux(R) Server 6（64-bit x86_64）

このマニュアルで使用している表記と，対応するアプリケーションサーバの機能名を次に示します。

表記	アプリケーションサーバの機能名
Management Server	Cosminexus Management Server
PRF	Cosminexus Performance Tracer
Smart Composer	Cosminexus Smart Composer

このマニュアルで使用している表記と，対応する Java 関連用語を次に示します。

表記	Java 関連用語
EAR	Enterprise Archive

表記	Java 関連用語
EJB または Enterprise JavaBeans	Enterprise JavaBeans™
J2EE または Java 2 Platform, Enterprise Edition	Java™ 2 Platform, Enterprise Edition
JAR	Java™ Archive
Java	Java™
JavaVM または JVM	Java™ Virtual Machine
JAXB	Java™ Architecture for XML Binding
JAXP	Java™ API for XML Processing
JDK	Java™ Development Kit
JSP	JavaServer Pages™
Servlet またはサーブレット	Java™ Servlet
WAR	Web Archive

付録 E.3　英略語

このマニュアルで使用している英略語を次に示します。

英略語	説明
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CSV	Comma Separated Value
DII	Dynamic Invocation Interface
EJB	Enterprise Java Beans
HTTP	Hyper Text Transfer Protocol
MIME	Multipurpose Internet Mail Extensions
OS	Operating System
POJO	Plain Old Java Object または Plain Ordinary Java Object
QName	Qualified Name
RFC	Request For Comments
RMD	Reliable Messaging Destination
RMI	Remote Method Invocation
RMS	Reliable Messaging Source
RPC	Remote Procedure Call
SAAJ	SOAP with Attachments API for Java
SEI	Service Endpoint Interface

英略語	説明
SOAP	Simple Object Access Protocol
UAC	User Account Control
UDDI	Universal Description, Discovery, and Integration
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WS-I	Web Services Interoperability Organization
WSDL	Web Services Description Language
XML	Extensible Markup Language

付録 E.4　KB（キロバイト）などの単位表記について

1KB（キロバイト），1MB（メガバイト），1GB（ギガバイト），1TB（テラバイト）はそれぞれ 1,024 バイト，$1{,}024^2$ バイト，$1{,}024^3$ バイト，$1{,}024^4$ バイトです。

付録 F　用語解説

（英字）

EJB（Enterprise Java Beans）

ビジネスアプリケーション向けに拡張した Java オブジェクトの仕様です。
Cosminexus の Web サービスは，POJO の範囲の仕様でも，EJB の範囲の仕様でも作成できます。
EJB の Web サービスの開発では，EJB プロジェクトを使用します。

JAX-WS API

JAX-WS 2.1 仕様で提供されている API です。ハンドラフレームワークなどの機能を追加する場合に使用します。

JAX-WS エンジン

JAX-WS 2.1 仕様に対応した Web サービスの通信基盤となるエンジンです。Web サービスクライアント側および Web サービス側に配置され，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。

JAX-WS 機能

JAX-WS 2.1 仕様に対応した Web サービスを開発，実行するための機能です。Web サービス開発用のコマンドラインインタフェース，および Web サービス実行用の通信基盤を提供します。

POJO（Plain Old Java Object または Plain Ordinary Java Object）

Java の仕様範囲で作成する Java オブジェクトを示す用語です。
Cosminexus の Web サービスは，POJO の範囲の仕様でも，EJB の範囲の仕様でも作成できます。
POJO の Web サービスの開発では，Web プロジェクトを使用します。

SOAP（Simple Object Access Protocol）

HTTP などの通信プロトコルを下位に持ち，ネットワークを経由したオブジェクト間の通信を規定したプロトコルです。リモートマシン上のオブジェクトに対するアクセスには，XML の技術を利用した SOAP メッセージを利用します。

SOAP アプリケーション

Cosminexus の既存機能である SOAP アプリケーション開発支援機能で開発された Web サービス用のアプリケーションです。
SOAP アプリケーションについては，マニュアル「Cosminexus 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。

SOAP アプリケーション開発支援機能

Cosminexus が提供する，SOAP アプリケーションを開発するための機能です。SOAP アプリケーション開発支援機能では，開発支援コマンドを使用して SOAP アプリケーションを開発できます。
SOAP アプリケーション開発支援機能については，マニュアル「Cosminexus 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。

SOAP エンベロープ

SOAP メッセージの要素です。メッセージのいちばん外側の要素で，SOAP ヘッダと SOAP ボディという子要素を持ちます。

SOAP ヘッダ

SOAP メッセージの要素です。SOAP ヘッダでは，メッセージ処理のあて先，およびメッセージ処理が必須かどうかを指定します。

SOAP ボディ

SOAP メッセージの要素です。SOAP ボディに送信するメッセージの内容を記述します。

SOAP メッセージ

SOAP プロトコルでオブジェクト間の送受信に使用するメッセージです。SOAP メッセージは，SOAP エンベロープ，SOAP ヘッダ，および SOAP ボディという要素で構成されます。

Web サービス

Web 関連の技術を利用して，ネットワークを介して提供されるプログラム（サービス）です。Web サービスの基礎となる技術には，SOAP，WSDL，および UDDI などがあります。
Cosminexus の Web サービスは，次のどちらの仕様範囲でも作成できます。

- POJO
  既存の仕様範囲で Java オブジェクトを作成する方法です。
- EJB
  ビジネスアプリケーション向けに拡張した Java オブジェクトの仕様です。

Web サービスクライアント

Web サービスを利用するクライアントプログラムです。Web サービスの開発では，コマンドで自動生成されたクラスを利用して，Web サービスを呼び出す処理を記述します。

Web サービス実装クラス

Web サービスの実装に当たるプログラムです。Web サービスの開発では，コマンドで自動生成されたクラスを利用して，Web サービスの実装を記述します。

（カ行）

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。

コマンド

このマニュアルでは，Web サービスの開発時に使用する cjwsimport コマンドおよび apt コマンドをコマンドと総称して表記しています。

コンテキストルート

コンテキストのルートパスです。コンテキスト内の Web アプリケーションにアクセスするときに URL 上に指定します。コンテキストパスとも呼びます。

（サ行）

自動生成クラス

cjwsimport コマンドおよび apt コマンドを実行したときに生成される Java クラスを表します。

スケルトン

SOAP サービスに対する静的なインタフェースです。Web サービス側にデプロイし，Web サービスクライアント側との送受信の処理を隠蔽します。

スタブ

Web サービスクライアントからのリクエストを JAX-WS エンジンに渡し，JAX-WS エンジンからの戻り値を返す働きを持つインタフェースです。Web サービスクライアント側にデプロイし，Web サービス側との送受信の処理を隠蔽します。

（タ行）

デプロイ

作成した Java プログラムを利用できる状態にする手続きのことをいいます。EAR ファイルをデプロイすることで，Web サービスとして利用できるようになります。

添付ファイル付き SOAP メッセージ

SOAP ヘッダや SOAP ボディに加えて，任意のファイルが添付された SOAP メッセージです。

動作定義ファイル

共通定義ファイルおよびプロセス別の定義ファイルの総称です。ログ出力やタイムアウトなど，JAX-WS エンジンの動作を設定する場合に定義します。

（ナ行）

名前空間

XML 文書内で同じ名前を持つ要素や属性が，異なる意味を表すために定義する識別子です。

（ハ行）

ハンドラフレームワーク

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。

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必要がある場合に作成する定義ファイルです。

（ヤ行）

ユーザ定義例外

Web サービスの開発者が，Web サービスの処理で独自に実装する例外のことです。必要に応じて Web サービスの処理でスローします。

索引

数字

1 リクエスト当たりのメモリ使用量 797
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） 798

A

action 要素（javax.jws.WebMethod） 359
AddressingFeature 476
API ベースの Web サービスクライアントの開発の流れ 30
application.xml を作成する（SEI 起点） 96
application.xml を作成する（SEI 起点・MTOM/XOP 仕様形式の添付ファイル） 591
application.xml を作成する（SEI 起点・wsi:swaRef 形式の添付ファイル） 553
application.xml を作成する（SEI 起点・アドレッシング） 721
application.xml を作成する（SEI 起点・カスタマイズ） 126
application.xml を作成する（SEI 起点・ストリーミング） 618
application.xml を作成する（WSDL 起点） 81
application.xml を作成する（WSDL 起点・WS-RM 1.2） 648
application.xml を作成する（プロバイダ起点・SAAJ） 159
apt コマンド 263
apt コマンドの生成ファイル一覧 264
apt コマンドの引数 263
AttachmentPart クラス 459

B

binding 属性（wsdl:port 要素） 408

C

childElementName 属性（jaxws:bindings 要素） 323
CID URL スキームの形式 533
cjwsgen コマンド 266
cjwsgen コマンドのオプション一覧 267
cjwsgen コマンドの指定形式 266
cjwsgen コマンドの終了コード 277
cjwsimport コマンド 254
cjwsimport コマンドのオプション一覧 255
cjwsimport コマンドの指定形式 254
cjwsimport コマンドの終了コード 261
cjwsimport コマンドの生成ファイル一覧 259
className 要素（javax.xml.ws.FaultAction） 377
className 要素（javax.xml.ws.RequestWrapper） 379
className 要素（javax.xml.ws.ResponseWrapper） 381
close メソッドの処理 677
com.cosminexus.jaxws.http.proxyPassword 234
com.cosminexus.jaxws.http.proxyUser 233
com.cosminexus.jaxws.https.proxyPassword 235
com.cosminexus.jaxws.https.proxyUser 234
com.cosminexus.wsrm.common.exception.FaultType 列挙型 782
com.cosminexus.wsrm.common.exception.RMException クラス 781
com.cosminexus.wsrm.common.model.SequenceStatesEnum 列挙型 784
com.cosminexus.wsrm.rmd.handler.RMDAtMostOnceHandler クラス 783
com.cosminexus.wsrm.rms.api.RMSConfigurations クラス 780
com.cosminexus.wsrm.rms.api.RMSMessaging インタフェース 779

com.cosminexus.wsrm.rms.api.RMSProvider クラス 777
com.cosminexus.wsrm.rms.handler.RMSAtMostOnceHandler クラス 783
com.sun.xml.ws.developer.StreamingAttachmentFeature クラス 489
com.sun.xml.ws.developer.StreamingAttachment アノテーション 354
com.sun.xml.ws.developer.StreamingDataHandler クラス 491
Conformance への対応 436
cosminexus-jaxws.xml 使用時の設定例 204
cosminexus-jaxws.xml によるカスタマイズ 201
cosminexus-jaxws.xml の書式 201
cosminexus-jaxws.xml のファイル名と格納先 201

D

Detail インタフェース 454
Document Bare スタイルでの MTOM/XOP 仕様形式の添付ファイル 565

E

EAR ファイル 43
EAR ファイルの作成 43
EAR ファイルを作成する（SEI 起点） 97
EAR ファイルを作成する（SEI 起点・MTOM/XOP 仕様形式の添付ファイル） 591
EAR ファイルを作成する（SEI 起点・wsi:swaRef 形式の添付ファイル） 553
EAR ファイルを作成する（SEI 起点・アドレッシング） 721
EAR ファイルを作成する（SEI 起点・カスタマイズ） 127
EAR ファイルを作成する（SEI 起点・ストリーミング） 619
EAR ファイルを作成する（WSDL 起点） 82
EAR ファイルを作成する（WSDL 起点・WS-RM 1.2） 649
EAR ファイルを作成する（プロバイダ起点・SAAJ） 159
EAR ファイルをデプロイする（SEI 起点） 99
EAR ファイルをデプロイする（SEI 起点・MTOM/XOP 仕様形式の添付ファイル） 593
EAR ファイルをデプロイする（SEI 起点・wsi:swaRef 形式の添付ファイル） 555
EAR ファイルをデプロイする（SEI 起点・アドレッシング） 722
EAR ファイルをデプロイする（SEI 起点・カスタマイズ） 128
EAR ファイルをデプロイする（SEI 起点・ストリーミング） 620
EAR ファイルをデプロイする（WSDL 起点） 83
EAR ファイルをデプロイする（WSDL 起点・WS-RM 1.2） 650
EAR ファイルをデプロイする（プロバイダ起点・SAAJ） 160
EJB 3
EJB JAR ファイル 43
EJB JAR ファイルの構成 43
EJB（Enterprise Java Beans）〔用語解説〕 806
EJB の Web サービスに適用する場合の注意事項 661
element 属性（wsdl:part 要素） 395
enabled 要素（javax.xml.ws.soap.Addressing） 383
enabled 要素（javax.xml.ws.soap.MTOM） 384
endpointInterface 要素（javax.jws.WebService） 366
exclude 要素（javax.jws.WebMethod） 358

F

faultBean 要素（javax.xml.ws.WebFault） 386
fault 要素（javax.xml.ws.Action） 374
file 要素（javax.jws.HandlerChain） 356

H

handleFault メソッドの処理 673
handleMessage メソッドの処理 665
header 要素（javax.jws.WebParam） 360
header 要素（javax.jws.WebResult） 362
http.nonProxyHosts 235
http.proxyHost 233
http.proxyPort 233
https.proxyHost 234
https.proxyPort 234
HTTP 応答ヘッダ 508
HTTP ステータスコード 247,508
HTTP ステータスコード（例外をフォルトにバインディングする場合） 217
HTTP ヘッダ（添付ファイルから SOAP メッセージ・MTOM/XOP） 574
HTTP ヘッダ（添付ファイルから SOAP メッセージ・wsi:swaRef 形式） 532
HTTP ヘッダと HTTP ボディの境界文字列（添付ファイルから SOAP メッセージ・wsi:swaRef 形式） 534
HTTP ボディ（添付ファイルから SOAP メッセージ・MTOM/XOP） 574
HTTP ボディ（添付ファイルから SOAP メッセージ・wsi:swaRef 形式） 532

I

implementation 属性（cosminexus-jaxws.xml） 203
input 要素（javax.xml.ws.Action） 374

J

J2EE サーバの移行 764
java.util.List オブジェクト 194
java.util.Map クラス 338
javaee:handler-chains 要素 446
javaee:handler-chain 要素 446
javaee:handler-class 要素 448
javaee:handler-name 要素 447
javaee:handler 要素 447
javaee:soap-header 要素 448
javaee:soap-role 要素 449
JavaVM のプロパティ 233
javax.activation.DataHandler 型に指定できる添付ファイル 525
javax.jws.HandlerChain アノテーション 356
javax.jws.soap.SOAPBinding アノテーション 356
javax.jws.WebMethod アノテーション 357
javax.jws.WebParam アノテーション 359
javax.jws.WebResult アノテーション 362
javax.jws.WebService アノテーション 365
javax.xml.bind.annotation.XmlMimeType アノテーション 368
javax.xml.ws.Action アノテーション 373
javax.xml.ws.BindingProvider インタフェース 478
javax.xml.ws.BindingType アノテーション 375
javax.xml.ws.Dispatch インタフェース 478
javax.xml.ws.EndpointReference クラス 479
javax.xml.ws.FaultAction アノテーション 376
javax.xml.ws.handler.Handler<C extends MessageContext> インタフェース 494
javax.xml.ws.handler.HandlerResolver インタフェース 495
javax.xml.ws.handler.LogicalMessageContext インタフェース 495
javax.xml.ws.handler.MessageContext インタフェース 495
javax.xml.ws.handler.PortInfo インタフェース 496
javax.xml.ws.handler.soap.SOAPHandler<T extends SOAPMessageContext> インタフェース 497
javax.xml.ws.handler.soap.SOAPMessageContext インタフェース 497
javax.xml.ws.Holder<T> クラス 498
javax.xml.ws.LogicalMessage インタフェース 498
javax.xml.ws.ProtocolException クラス 499
javax.xml.ws.Provider インタフェース 487

javax.xml.ws.RequestWrapper アノテーション 377
javax.xml.ws.ResponseWrapper アノテーション 379
javax.xml.ws.ServiceMode アノテーション 381
javax.xml.ws.Service クラス 480
javax.xml.ws.soap.AddressingFeature クラス 499
javax.xml.ws.soap.Addressing アノテーション 382
javax.xml.ws.soap.MTOMFeature クラス 500
javax.xml.ws.soap.MTOM アノテーション 383
javax.xml.ws.soap.SOAPBinding インタフェース 501
javax.xml.ws.soap.SOAPFaultException クラス 502
javax.xml.ws.soap.SOAPFaultException のバインディング 211
javax.xml.ws.WebFault アノテーション 384
javax.xml.ws.WebServiceException クラス 502
javax.xml.ws.WebServiceException のバインディング 210
javax.xml.ws.WebServiceProvider アノテーション 386
javax.xml.ws.wsaddressing.W3CEndpointReferenceBuilder クラス 487
javax.xml.ws.wsaddressing.W3CEndpointReference クラス 486
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（SEI 起点） 103
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（SEI 起点・MTOM/XOP 仕様形式の添付ファイル） 597
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（SEI 起点・wsi:swaRef 形式の添付ファイル） 559
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（SEI 起点・アドレッシング） 728
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（SEI 起点・カスタマイズ） 132
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（SEI 起点・ストリーミング） 624
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（WSDL 起点） 87
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（WSDL 起点・WS-RM 1.2） 655
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（プロバイダ起点・SAAJ） 164
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（SEI 起点） 103
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（SEI 起点・MTOM/XOP 仕様形式の添付ファイル） 597
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（SEI 起点・wsi:swaRef 形式の添付ファイル） 559
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（SEI 起点・アドレッシング） 728
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（SEI 起点・カスタマイズ） 132
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（SEI 起点・ストリーミング） 624
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（WSDL 起点） 87
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（WSDL 起点・WS-RM 1.2） 655
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（プロバイダ起点・SAAJ） 164
Java から WSDL へのデフォルトマッピング 328
Java ソースから WSDL へのマッピング一覧 328

Java ソースから WSDL へのマッピング例 34
Java ソースを生成する（SEI 起点） 94
Java ソースを生成する（SEI 起点・MTOM/XOP 仕様形式の添付ファイル） 589
Java ソースを生成する（SEI 起点・wsi:swaRef 形式の添付ファイル） 551
Java ソースを生成する（SEI 起点・アドレッシング） 719
Java ソースを生成する（SEI 起点・カスタマイズ） 124
Java ソースを生成する（SEI 起点・ストリーミング） 616
Java ソースを生成する（プロバイダ起点・SAAJ） 157
Java のパッケージ名に記述できる文字列 329
Java のラッパ例外クラスからフォルトへのマッピング 342
Java メソッドのオーバーロード 305
Java メソッドのパラメタの条件 337
Java 例外の伝搬 215
JAX-WS 2.1 仕様のサポート範囲 431
JAX-WS API 5
JAX-WS API〔用語解説〕 806
JAX-WS API のインタフェースおよびクラスの種類 474
JAX-WS API のサポート範囲 473
JAX-WS API を使用する場合の実装例 63
JAX-WS エンジン 4
JAX-WS エンジン〔用語解説〕 806
JAX-WS エンジンのサポート範囲（Web サービス側） 184
JAX-WS エンジンの設定 16
JAX-WS エンジンの動作 184
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） 188
JAX-WS エンジンのメモリ使用量の算出 797
JAX-WS 機能〔用語解説〕 806
JAXB アノテーションのサポート 307,348
jaxws:bindings 要素の指定（埋め込みによるバインディング宣言） 310
jaxws:bindings 要素の指定（外部バインディングファイル） 315
jaxws:bindings 要素の属性の指定可否 321
jaxws:enableAsyncMapping 要素（jaxws:bindings 要素） 323
jaxws:enableWrapperStyle 要素（jaxws:bindings 要素） 323
jaxws:enableWrapperStyle 要素の優先順位 314
jaxws:javadoc 要素（jaxws:bindings 要素） 322
jaxws:provider 要素（jax:bindings 要素） 323
jaxwsdd:endpoints 要素（cosminexus-jaxws.xml） 202
jaxwsdd:endpoint 要素（cosminexus-jaxws.xml） 202

L

localName 要素（javax.xml.ws.RequestWrapper） 378
localName 要素（javax.xml.ws.ResponseWrapper） 380
location 属性（soap:address 要素） 415
location 属性（soap12:address 要素） 421
location 属性（wsdl:import 要素） 392,519

M

MessageFactory クラス 460
message 属性（soap:header 要素） 413
message 属性（soap12:header 要素） 418
message 属性（wsdl:fault 要素） 400
message 属性（wsdl:input 要素） 398
message 属性（wsdl:output 要素） 399
MimeHeaders クラス 460
MimeHeader クラス 460
MIME パートの記述順序（MTOM/XOP） 576
MIME パートの記述順序（wsi:swaRef 形式） 535

MIME パートの終端文字列（添付ファイルから SOAP メッセージ・wsi:swaRef 形式） 534
MIME バインディング 302
mode 要素（javax.jws.WebParam） 361
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 563
MTOM/XOP 仕様形式の添付ファイルの使用方法 563
MTOM/XOP仕様形式の添付ファイルの対象 563
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 578
MTOMFeature 476

N

namespace 属性（wsdl:import 要素） 392,519
name 属性（cosminexus-jaxws.xml） 203
name 属性（jaxws:bindings 要素） 322
name 属性（portType 要素） 396
name 属性（soap:fault 要素） 414
name 属性（soap12:fault 要素） 420
name 属性（wsdl:binding 要素） 401
name 属性（wsdl:definitions 要素） 391
name 属性（wsdl:fault 要素） 400,406
name 属性（wsdl:input 要素） 398,404
name 属性（wsdl:message 要素） 394
name 属性（wsdl:operation 要素） 397,403
name 属性（wsdl:output 要素） 399,405
name 属性（wsdl:part 要素） 395
name 属性（wsdl:port 要素） 408
name 属性（wsdl:service 要素） 407
name 要素（javax.jws.WebParam） 360
name 要素（javax.jws.WebResult） 363
name 要素（javax.jws.WebService） 367
name 要素（javax.xml.ws.WebFault） 385
Node インタフェース 454
node 属性の記述形式(jaxws:bindings 要素) 318
non-wrapper スタイルでの MTOM/XOP 仕様形式の添付ファイル（MTOM/XOP） 566

O

operationName 要素（javax.jws.WebMethod） 359
org.jvnet.mimepull.MIMEConfig クラス 492
output 要素（javax.xml.ws.Action） 374

P

parameterStyle 要素（javax.jws.soap.SOAPBinding） 357
partName 要素（javax.jws.WebParam） 360
partName 要素（javax.jws.WebResult） 363
parts 属性（soap:body 要素） 412
parts 属性（soap12:body 要素） 418
part 属性（jaxws:bindings 要素） 323
part 属性（soap:header 要素） 413
part 属性（soap12:header 要素） 419
POJO 3
POJO（Plain Old Java Object または Plain Ordinary Java Object）〔用語解説〕 806
portName 要素（javax.jws.WebService） 367
portName 要素（javax.xml.ws.WebServiceProvider） 387
port 属性（cosminexus-jaxws.xml） 203
PRF デーモンの起動 246

R

required 要素（javax.xml.ws.soap.Addressing） 383
RMD 628,770
RMS 628,770
RMS 定義ファイルに追加するプロパティ 785

S

SAAJ 1.3 仕様のサポート範囲 450
SAAJResult クラス 460
SEI からバインディングへのマッピング 345

SEIからバインディングへのマッピング規則 345
SEI の条件 331
SEI のメソッド名からオペレーションへのマッピング 331
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 332
SEI名およびクラス名の衝突時のマッピング 305
SEI 名からポートタイプへのマッピング 330
SEI 名に記述できる文字列 331
SEI 名の条件 331
SEI を起点とした開発の流れ 23
SEI を起点とした開発の流れ（cjwsgen コマンドを使用する場合） 25
SEI を生成する（WSDL 起点） 78
SEI を生成する（WSDL 起点・WS-RM 1.2） 645
serviceName 要素（javax.jws.WebService） 367
serviceName 要素（javax.xml.ws.WebServiceProvider） 387
servlet-mapping 要素の記述例 38,48
SOAP（Simple Object Access Protocol）〔用語解説〕 806
soap:address 要素 414
soap:binding 要素 409
soap:body 要素 411
soap:fault 要素 413
soap:header 要素 412
soap:mustUnderstand 属性の設定（Web サービス側） 692
soap:mustUnderstand 属性の設定（Web サービスクライアント側） 694
soap:operation 要素 410
soap12:address 要素 420
soap12:binding 要素 416
soap12:body 要素 417
soap12:fault 要素 419
soap12:header 要素 418
soap12:operation 要素 415
soapAction 属性（soap:operation 要素） 411
soapAction 属性（soap12:operation 要素） 415
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 509
SOAPBody インタフェース 455
SOAPElement インタフェース 455
SOAPEnvelope インタフェース 457
SOAPFactory クラス 461
SOAPFault インタフェース 457
SOAPHeaderElement インタフェース 459
SOAPHeader インタフェース 458
SOAPMessage クラス 461
SOAPPart クラス 463
SOAP アプリケーション〔用語解説〕 806
SOAP アプリケーション開発支援機能〔用語解説〕 806
SOAP アプリケーションの移行 764
SOAP エンベロープ〔用語解説〕 807
SOAPトランスポートと転送バインディング 345
SOAP のバージョン選択 242
SOAP バインディング 301
SOAP ハンドラ 662
SOAP ヘッダ〔用語解説〕 807
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） 679
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） 683
SOAP ヘッダの設定方法 687
SOAP ボディ〔用語解説〕 807
SOAP メッセージ〔用語解説〕 807
SOAP メッセージから添付ファイルへのマッピング（MTOM/XOP） 580
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） 539
SOAP ロールとアクタの設定（Web サービス側） 692
SOAP ロールとアクタの設定（Web サービスクライアント側） 694
SSL プロトコルによる接続 238
style 属性（soap:binding 要素） 410
style 属性（soap:operation 要素） 411
style 属性（soap12:binding 要素） 417

style 属性（soap12:operation 要素） 416
style 要素（javax.jws.soap.SOAPBinding） 356

T

targetNamespace 属性（wsdl:definitions 要素） 391
targetNamespace 要素（javax.jws.WebParam） 361
targetNamespace 要素（javax.jws.WebResult） 364
targetNamespace 要素（javax.jws.WebService） 365
targetNamespace 要素（javax.xml.ws.RequestWrapper） 378
targetNamespace 要素（javax.xml.ws.ResponseWrapper） 380
targetNamespace 要素（javax.xml.ws.WebFault） 385
targetNamespace 要素（javax.xml.ws.WebServiceProvider） 386
threshold 要素(javax.xml.ws.soap.MTOM） 384
transport 属性（soap:binding 要素） 410
transport 属性（soap12:binding 要素） 416
type 属性（wsdl:binding 要素） 402

U

UAC（ユーザアカウント制御） 279
url-pattern 属性（cosminexus-jaxws.xml） 203
use 属性（soap:body 要素） 412
use 属性（soap:fault 要素） 414
use 属性（soap:header 要素） 413
use 属性（soap12:body 要素） 417
use 属性（soap12:fault 要素） 420
use 属性（soap12:header 要素） 419
use 要素（javax.jws.soap.SOAPBinding） 357

V

value 要素（javax.xml.bind.annotation.XmlMimeType） 372
value 要素（javax.xml.ws.BindingType） 375
value 要素（javax.xml.ws.FaultAction） 377
value 要素（javax.xml.ws.ServiceMode） 382
version 属性の記述形式（jaxws:bindings 要素） 318

W

WAR ファイル 42
WAR ファイルの構成 42
web.xml の作成 37
web.xml の例 38
web.xml を WAR ファイルに含めない場合の動作 39
web.xml を作成する（SEI 起点） 95
web.xml を作成する（SEI 起点・MTOM/XOP 仕様形式の添付ファイル） 590
web.xml を作成する（SEI 起点・wsi:swaRef 形式の添付ファイル） 552
web.xml を作成する（SEI 起点・アドレッシング） 720
web.xml を作成する（SEI 起点・カスタマイズ） 126
web.xml を作成する（SEI 起点・ストリーミング） 618
web.xml を作成する（WSDL 起点） 80
web.xml を作成する（WSDL 起点・WS-RM 1.2） 647
web.xml を作成する（プロバイダ起点・SAAJ） 158
Web サービス〔用語解説〕 807
Web サービス開発の流れ 18
Webサービス側のフォルトおよび例外の処理 206
Web サービスクライアント〔用語解説〕 807
Web サービスクライアント開発の流れ 29

Web サービスクライアント側のフォルトの処理 214
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 508
Web サービスクライアントの開発例（SEI 起点） 100
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） 556
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） 723
Web サービスクライアントの開発例（SEI 起点・カスタマイズ） 129
Web サービスクライアントの開発例（WSDL 起点） 84
Web サービスクライアントの開発例（WSDL 起点・WS-RM 1.2） 651
Web サービスクライアントの開発例（プロバイダ起点・SAAJ） 161
Web サービスクライアントの形態 13
Web サービスクライアントの実装 51
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（WSDL 起点・WS-RM 1.2） 653
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（SEI 起点） 102
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（SEI 起点・MTOM/XOP 仕様形式の添付ファイル） 595
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（SEI 起点・wsi:swaRef 形式の添付ファイル） 557
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（SEI 起点・アドレッシング） 727
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（SEI 起点・カスタマイズ） 131
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（SEI 起点・ストリーミング） 622
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（WSDL 起点） 86
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（WSDL 起点・WS-RM 1.2） 654
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（プロバイダ起点・SAAJ） 163
Web サービスクライアントの実装クラスを作成する（SEI 起点） 101
Web サービスクライアントの実装クラスを作成する（SEI 起点・MTOM/XOP 仕様形式の添付ファイル） 595
Web サービスクライアントの実装クラスを作成する（SEI 起点・wsi:swaRef 形式の添付ファイル） 557
Web サービスクライアントの実装クラスを作成する（SEI 起点・アドレッシング） 724
Web サービスクライアントの実装クラスを作成する（SEI 起点・カスタマイズ） 130
Web サービスクライアントの実装クラスを作成する（SEI 起点・ストリーミング） 622
Web サービスクライアントの実装クラスを作成する（WSDL 起点） 85
Web サービスクライアントの実装クラスを作成する（WSDL 起点・WS-RM 1.2） 652
Web サービスクライアントの実装クラスを作成する（プロバイダ起点・SAAJ） 161
Web サービスクライアントの実装例（JAX-WS API を使用） 64
Web サービスクライアントの実装例（ディスパッチベース） 61
Web サービスクライアントを実行する（SEI 起点） 103
Web サービスクライアントを実行する（SEI 起点・MTOM/XOP 仕様形式の添付ファイル） 598
Web サービスクライアントを実行する（SEI 起点・wsi:swaRef 形式の添付ファイル） 560
Web サービスクライアントを実行する（SEI 起点・アドレッシング） 729
Web サービスクライアントを実行する（SEI 起点・カスタマイズ） 133

Web サービスクライアントを実行する（SEI 起点・ストリーミング）625
Web サービスクライアントを実行する（WSDL 起点）88
Web サービスクライアントを実行する（WSDL 起点・WS-RM 1.2）656
Web サービスクライアントを実行する（プロバイダ起点・SAAJ）165
Web サービス実装クラス〔用語解説〕807
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 36
Web サービス実装クラスから SEI へのマッピング 329
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 346
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 346
Web サービス実装クラスの条件 329
Web サービス実装クラス名に記述できる文字列 347
Web サービス実装クラス名の条件 346
Web サービス実装クラスをコンパイルする（WSDL 起点）80
Web サービス実装クラスをコンパイルする（WSDL 起点・WS-RM 1.2）647
Web サービス実装クラスを作成する（SEI 起点）93
Web サービス実装クラスを作成する（SEI 起点・MTOM/XOP 仕様形式の添付ファイル）587
Web サービス実装クラスを作成する（SEI 起点・wsi:swaRef 形式の添付ファイル）549
Web サービス実装クラスを作成する（SEI 起点・アドレッシング）717
Web サービス実装クラスを作成する（SEI 起点・カスタマイズ）123
Web サービス実装クラスを作成する（SEI 起点・ストリーミング）614
Web サービス実装クラスを作成する（WSDL 起点）79
Web サービス実装クラスを作成する（WSDL 起点・WS-RM 1.2）646
Web サービスセキュリティハンドラ 660
Web サービスと Web サービスクライアントの形態 11
Web サービスのアンデプロイ 764
Web サービスの開発および実行の前提条件 6
Web サービスの開発の概要 2
Web サービスの開発例（SEI 起点）93
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）587
Web サービスの開発例（SEI 起点・wsi:swaRef 形式の添付ファイル）549
Web サービスの開発例（SEI 起点・アドレッシング）717
Web サービスの開発例（SEI 起点・カスタマイズ）123
Web サービスの開発例（SEI 起点・ストリーミング）614
Web サービスの開発例（WSDL 起点）71
Web サービスの開発例（WSDL 起点・WS-RM 1.2）639
Web サービスの開発例（プロバイダ起点・SAAJ）155
Web サービスの形態 11
Web サービスの実行例（SEI 起点）103
Web サービスの実行例（SEI 起点・wsi:swaRef 形式の添付ファイル）559
Web サービスの実行例（SEI 起点・アドレッシング）728
Web サービスの実行例（SEI 起点・カスタマイズ）132
Web サービスの実行例（WSDL 起点）87
Web サービスの実行例（WSDL 起点・WS-RM 1.2）655
Web サービスの実行例（プロバイダ起点・SAAJ）164
Web サービスの情報表示 229
Web サービスの初期化 231
Web サービスの破棄 232
Web サービスを開始する（SEI 起点）99
Web サービスを開始する（SEI 起点・MTOM/XOP 仕様形式の添付ファイル）593

Web サービスを開始する（SEI 起点・wsi:swaRef 形式の添付ファイル） 555
Web サービスを開始する（SEI 起点・アドレッシング） 722
Web サービスを開始する（SEI 起点・カスタマイズ） 128
Web サービスを開始する（SEI 起点・ストリーミング） 620
Web サービスを開始する（WSDL 起点） 83
Web サービスを開始する（WSDL 起点・WS-RM 1.2） 650
Web サービスを開始する（プロバイダ起点・SAAJ） 160
wrapper 子要素名に記述できる文字列（wrapper スタイル） 290
wrapper スタイルの条件 287
WS-RM 1.1 機能 770
WS-RM 1.1 機能の API 一覧 776
WS-RM 1.1 機能のプロパティ設定 785
WS-RM 1.1 仕様のサポート範囲 791
WS-RM 1.2 機能 628
WS-RM 1.2 仕様のサポート範囲 467
WS-RM Policy 1.2 仕様のサポート範囲 470
WS-RM Policy の追加方法 633
wsdl:binding 要素 400
wsdl:definitions 要素 390
wsdl:documentation 要素 409
wsdl:fault 要素（wsdl:binding 要素の孫要素の場合） 405
wsdl:fault 要素（wsdl:portType 要素の孫要素の場合） 399
wsdl:import 要素 391
wsdl:import 要素の書式 519
wsdl:input 要素（wsdl:binding 要素の孫要素の場合） 403
wsdl:input 要素（wsdl:portType 要素の孫要素の場合） 397
wsdl:message 要素 393
wsdl:operation 要素（wsdl:binding 要素の子要素の場合） 402
wsdl:operation 要素（wsdl:portType 要素の子要素の場合） 396
wsdl:output 要素（wsdl:binding 要素の孫要素の場合） 404
wsdl:output 要素（wsdl:portType 要素の孫要素の場合） 398
wsdl:part 要素 394
wsdl:portType 要素 395
wsdl:port 要素 407
wsdl:service 要素 406
wsdl:types 要素 392
WSDL 1.1 仕様のサポート範囲 390
wsdlLocation 属性の記述形式（jaxws:bindings 要素） 317
wsdlLocation 要素（javax.jws.WebService） 368
wsdlLocation 要素（javax.xml.ws.WebServiceProvider） 387
WSDL インポート機能 515,516
WSDL から Java ソースへのマッピング一覧 282
WSDL から Java ソースへのマッピング例 33
WSDL から Java へのマッピングのカスタマイズ 309
WSDL から添付ファイルの Java 型へのマッピング（wsi:swaRef 形式） 528
WSDL 作成時の注意事項 426
WSDL 定義の階層的なインポート 517
WSDL 定義の再帰的なインポート 518
WSDL の作成 32
WSDL ファイルに WS-RM Policy を追加する（WSDL 起点・WS-RM 1.2） 645
WSDL ファイルを作成する（WSDL 起点） 71
WSDL ファイルを作成する（WSDL 起点・WS-RM 1.2） 639
WSDL を起点とした開発の流れ 20
WSIMPORT_OPTS 環境変数 259

X

XML Schema の出現回数の指定に関する制限 769
XML Schema の制約ファセットに関する制限 769

XML Schema のデータ型の制限 768
XOP 情報セットの形式 578
xsd:schema 要素 421

あ

アーカイブの作成 42
アウトバウンド 658
アドレッシング機能 696
アノテーション一覧 350
アプリケーション起動時のメモリ使用量 797
アプリケーションのクライアントの実行 246
アンマーシャル 4

い

異常終了時の対処（apt コマンド） 265
異常終了時の対処（cjwsgen コマンド） 278
異常終了時の対処（cjwsimport コマンド） 261
インタフェースおよびクラスの一覧表 474
インタフェースおよびクラスのサポート範囲 474
インタフェースおよびクラスの種類 474
インタフェースの透過性 219
インバウンド 658
インポート / インクルードしている場合の注意 227
インポート対象の WSDL 定義の条件 517
インポートできる WSDL 定義 517

う

埋め込みによるバインディング宣言 309
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 320

え

エラーページ 217

お

オーバーロードによる名前衝突 333
オプションの指定有無とファイルの出力先 257
オペレーションとその子要素の記述個数 286
オペレーション名に記述できる文字列 286
オペレーション名の条件 286

か

開発例の構成（SEI 起点） 90
開発例の構成（SEI 起点・EJB の Web サービス） 136
開発例の構成（SEI 起点・MTOM/XOP 仕様形式の添付ファイル） 584
開発例の構成（SEI 起点・wsi:swaRef 形式の添付ファイル） 546
開発例の構成（SEI 起点・アドレッシング） 714
開発例の構成（SEI 起点・カスタマイズ） 120
開発例の構成（SEI 起点・ストリーミング） 610
開発例の構成（WSDL 起点） 68
開発例の構成（WSDL 起点・WS-RM 1.2） 636
開発例の構成（プロバイダ起点・SAAJ） 152
開発例の流れ（SEI 起点） 92
開発例の流れ（SEI 起点・MTOM/XOP 仕様形式の添付ファイル） 586
開発例の流れ（SEI 起点・wsi:swaRef 形式の添付ファイル） 548
開発例の流れ（SEI 起点・アドレッシング） 716
開発例の流れ（SEI 起点・カスタマイズ） 122
開発例の流れ（SEI 起点・ストリーミング） 613
開発例の流れ（WSDL 起点） 70
開発例の流れ（プロバイダ起点・SAAJ） 154
外部バインディングファイルによるカスタマイズ 314
各 MIME パート間の境界文字列（添付ファイルから SOAP メッセージ・wsi:swaRef 形式） 534
稼働ログ 737

き

期待した性能が出ない 734
旧バージョンからの移行 764
旧バージョンで作成した WSDL の互換性 767
共通定義ファイル 169
共通定義ファイル〔用語解説〕 807
共通定義ファイルに追加するプロパティ 785
共通定義ファイルの設定項目 170

く

クライアント API 474
クライアント AP 情報とルート AP 情報の有効範囲 749
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 508
クラスベースのマッピング 296

こ

コア API 474
コマンド 5
コマンド〔用語解説〕 807
コマンドラインインタフェース 279
コマンドラインの実行 245
コマンドライン利用時の設定 245
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 245
コンテキストルート〔用語解説〕 807

さ

サービス API 474
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 302
サービスクラスを生成する（SEI 起点） 100
サービスクラスを生成する（SEI 起点・MTOM/XOP 仕様形式の添付ファイル） 594
サービスクラスを生成する（SEI 起点・wsi:swaRef 形式の添付ファイル） 556
サービスクラスを生成する（SEI 起点・アドレッシング） 723
サービスクラスを生成する（SEI 起点・カスタマイズ） 129
サービスクラスを生成する（SEI 起点・ストリーミング） 621
サービスクラスを生成する（WSDL 起点） 84
サービスクラスを生成する（WSDL 起点・WS-RM 1.2） 651
サービス固有例外の処理 206
サービス名およびポート名に記述できる文字列 304
サービス名およびポート名の条件 304

し

シーケンスライフサイクルメッセージ 630,772
ジェネリクス型の制限 348
ジェネリクスの型削除 347
自動生成クラス 5
自動生成クラス〔用語解説〕 808
障害の種類と対策 732
障害発生時に取得する資料 736
使用できる HTTP メソッド 230
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） 311
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） 319

す

スキーマ型から Java 型へのマッピング 295
スケルトン〔用語解説〕 808
スタブ〔用語解説〕 808
スタブベースの Web サービスクライアントの開発の流れ 29
スタブベースの実装例 51
ストリーミング 600

せ

生成されるラッパ例外クラス 297
性能解析トレースによる性能解析方法 760

性能解析トレースの取得レベル 748
性能解析トレースのトレース出力情報 748
性能解析トレースのトレース取得ポイント 750
前提条件 6

そ

送達保証 775
送達保証の種類 631

た

単位時間当たりのメモリ使用量の算出 799
単位表記 805

つ

通信ログ 737

て

ディスカバリ 5,195
ディスパッチ 5,199
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 30
ディスパッチベースの実装例 61
デプロイ〔用語解説〕 808
デプロイと開始の例（SEI 起点） 99
デプロイと開始の例（SEI 起点・wsi:swaRef 形式の添付ファイル） 555
デプロイと開始の例（SEI 起点・アドレッシング） 722
デプロイと開始の例（SEI 起点・カスタマイズ） 128
デプロイと開始の例（WSDL 起点） 83
デプロイと開始の例（WSDL 起点・WS-RM 1.2） 650
デプロイと開始の例（プロバイダ起点・SAAJ） 160
添付ファイルから SOAP メッセージへのマッピング（MTOM/XOP） 573
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） 531
添付ファイル機能（MTOM/XOP） 562
添付ファイル機能（wsi:swaRef） 522
添付ファイル使用時の WSDL の記述（wsi:swaRef 形式） 526
添付ファイル付き SOAP メッセージ 522
添付ファイル付き SOAP メッセージ〔用語解説〕 808
添付ファイル付き SOAP メッセージの構造 530
添付ファイルデータを取得する方法（wsi:swaRef 形式） 542
添付ファイルに使用できる Java 型 524
添付ファイルの Java インタフェース（MTOM/XOP） 563
添付ファイルの Java インタフェース（wsi:swaRef 形式） 524
添付ファイルの Java 型と WSDL のマッピング（wsi:swaRef 形式） 527
添付ファイルのインスタンスを生成する方法（wsi:swaRef 形式） 540
添付ファイルの拡張子と MIME タイプのマッピング 537
添付ファイルのデータサイズ 538
添付ファイルパートの MIME ヘッダ（添付ファイルから SOAP メッセージ・MTOM/XOP） 575
添付ファイルパートの MIME ヘッダ（添付ファイルから SOAP メッセージ・wsi:swaRef 形式） 533
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（添付ファイルから SOAP メッセージ・wsi:swaRef 形式） 534
添付ファイルパートの MIME ボディ（添付ファイルから SOAP メッセージ・wsi:swaRef 形式） 534
添付ファイルを指定できる個所 524

と

動作環境の切り替え 764
動作定義の優先度 170
動作定義ファイル〔用語解説〕 808
動作定義ファイルの記述規則 169

な

名前空間〔用語解説〕 808
名前空間からパッケージ名へのマッピング 282
名前空間に記述できる文字列 284
名前空間の記述形式 283
名前空間の条件 283
名前衝突時のマッピング 305
名前衝突の優先順位と解決方法 305

は

バージョンアップインストール 764
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） 294
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） 290
パート名に記述できる文字列（non-wrapper スタイル） 294
配列 194
バインディング拡張要素からパラメタへのマッピング 300
パス情報 508
パッケージ名から名前空間へのマッピング 328
パッケージ名のカスタマイズ 310
パッケージ名の条件 328
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） 339
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） 334
パラメタと戻り値の組み合わせ（non-wrapper スタイル） 340
パラメタと戻り値の組み合わせ（wrapper スタイル） 337
パラメタに指定できる Java 型（non-wrapper スタイル） 340
パラメタに指定できる Java 型（wrapper スタイル） 336
ハンドラチェイン設定ファイルの構文 692
ハンドラチェイン設定ファイルのサポート範囲 446
ハンドラチェイン設定ファイルの配置 692
ハンドラチェインの設定（Web サービス側） 691
ハンドラチェインの設定（Web サービスクライアント側） 693
ハンドラチェインの編成 664
ハンドラチェインを設定するコードの追加 693
ハンドラの型 662
ハンドラの初期化と破棄 678
ハンドラの配置 690
ハンドラフレームワーク 658
ハンドラフレームワーク〔用語解説〕 808
ハンドラリゾルバの例 693

ひ

必須オプション（apt コマンド） 263

ふ

フォルト bean へのマッピング 297
フォルトから参照されるメッセージのパートの個数 298
フォルトから例外クラスへのマッピング 297
フォルト名に記述できる文字列 298
フォルト名の条件 298
複数の WSDL 定義のインポート 517
プロキシサーバ経由の接続 233
プログラムが意図したとおりに動作しない 733
プログラムの実行中に異常終了する 732
プロセス別の定義ファイル 169
プロセス別の定義ファイル〔用語解説〕 808
プロセス別の定義ファイルの設定 183
プロトコルハンドラ 662
プロバイダ実装クラスを作成する 155
プロバイダを起点とした開発の流れ 27

へ

ベーシック認証による接続 240

ほ

ポートタイプから SEI 名へのマッピング 284
ポートタイプに記述できる文字列 285
ポートタイプの記述個数 285
ポートタイプ名の条件 284
保守ログ 737

ま

マーシャル 4

め

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 335
メソッド名およびパラメタ名の衝突時のマッピング 307
メソッド名に記述できる文字列 333
メソッド名の条件 333
メタデータの発行 223
メタデータの発行条件 223
メタデータ発行の有効 / 無効 227
メッセージコンテキスト使用時の注意事項 508
メッセージコンテキストの使用 504
メッセージコンテキストのプロパティのサポート範囲 504
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）293
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）287

ゆ

ユーザ定義例外〔用語解説〕 809

ら

ラッパ例外クラスからフォルトへのマッピング規則 343
ラッパ例外クラスの条件 344
ラッパ例外クラス名に記述できる文字列 344
ラッパ例外クラス名の条件 344
ランタイム例外のバインディング 208

り

略語 804

る

ルートパートから添付ファイルへのマッピング（MTOM/XOP）578
ルートパートから添付ファイルへのマッピング（wsi:swaRef 形式）536
ルートパートの MIME ヘッダ（添付ファイルから SOAP メッセージ・MTOM/XOP）574
ルートパートの MIME ヘッダ（添付ファイルから SOAP メッセージ・wsi:swaRef 形式）532
ルートパートの MIME ヘッダと MIME ボディの境界文字列（添付ファイルから SOAP メッセージ・wsi:swaRef 形式）533
ルートパートの MIME ボディ（添付ファイルから SOAP メッセージ・MTOM/XOP）575
ルートパートの MIME ボディ（添付ファイルから SOAP メッセージ・wsi:swaRef 形式）533

れ

例外ログ 737

ろ

ログの重要度 740
ログの出力先 738
ログの出力先（J2EE サーバ上で動作する場合）738
ログの出力先（コマンドラインインタフェースを利用する場合）738
ログの出力先（コマンドを実行する場合）739
ログの出力条件（稼働ログ / 例外ログ / 保守ログ）741

ログの出力条件（通信ログ） 741
ログの種類 737
ログの設定方法 744
ログのフォーマット（稼働ログ / 保守ログ）
742
ログのフォーマット（例外ログ / 通信ログ）
743
ログの見積もり 745
ログの見積もり方法 746
論理ハンドラ 662