HITACHI
Inspire the Next

Cosminexus V11 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

手引・文法書

3021-3-J25-70

前書き

■ 対象製品

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

■ 輸出時の注意

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

なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

■ マイクロソフト製品のスクリーンショットの使用について

マイクロソフトの許可を得て使用しています。

■ 発行

2025 年 4 月　3021-3-J25-70

■ 著作権

変更内容

変更内容(3021-3-J25-70) uCosminexus Application Server 11-60，uCosminexus Client 11-60，uCosminexus Developer 11-60，uCosminexus Service Architect 11-60，uCosminexus Service Platform 11-60

追加・変更内容	変更個所
記載内容は変更なし（リンク情報だけを変更した）。	-

単なる誤字・脱字などはお断りなく訂正しました。

はじめに

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

目次

第 2 編　開発と実行

第 4 編　トラブルシュート

1 Application Server で実現する SOAP アプリケーションの概要

Application Server では，SOAP アプリケーションを開発，実行するための機能を提供しています。

この章では，SOAP アプリケーションを開発，実行するに当たり，前提知識となる Web サービスの概要と SOAP に関する技術について説明します。また，SOAP アプリケーション開発支援機能の特長についても説明します。

なお，このマニュアルでは，JAX-WS 仕様に対応した Web サービスの開発方法については説明していません。JAX-WS 仕様に対応した Web サービスの開発方法については，マニュアル「アプリケーションサーバ Web サービス開発ガイド」を参照してください。

1.1　マニュアルの説明，プレフィクスと名前空間 URI

このマニュアルでは，SOAP および WSDL の技術を利用して開発し，ネットワーク上でサービスを提供するアプリケーションのことを**SOAP アプリケーション**と表記します。また，アプリケーションサーバが提供する SOAP アプリケーション開発用の機能を**SOAP アプリケーション開発支援機能**，SOAP アプリケーション実行用の環境を**SOAP 通信基盤**と表記します。

注意

JAX-WS 仕様に対応した Web サービスを開発する場合，SOAP アプリケーション開発支援機能および SOAP 通信基盤ではなく，JAX-WS 機能（開発用のコマンドラインインタフェースと通信基盤）を使用します。

SOAP アプリケーション開発支援機能および SOAP 通信基盤と，JAX-WS 機能の間に互換性はありません。JAX-WS 機能を使用した Web サービスの開発方法については，マニュアル「アプリケーションサーバ Web サービス開発ガイド」を参照してください。

また，JAX-WS 機能への切り替えに当たっての注意事項については，「1.6　JAX-WS 仕様に対応した Web サービスを開発する場合の注意事項」を参照してください。

このマニュアルの対象となる SOAP アプリケーション開発支援機能，および SOAP 通信基盤は，アプリケーションサーバを構成する次のプログラムプロダクトで提供されています。

SOAP アプリケーション開発支援機能（Component Container）

Developer

Service Architect

SOAP 通信基盤（Component Container）

Application Server

Service Platform

SOAP 通信基盤（Component Container Client）

Client

なお，Cosminexus 08-00 からコネクションプーリングのデフォルト設定が無効になりました。コネクションプーリングを使用する場合，「7.4　コネクションプーリング」を参照し，その内容を十分に理解した上で使用してください。

このマニュアルで使用している表記について説明します。

コーディング例で使用する記号

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。

記号	意味
	（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

コマンドの形式で使用する記号

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，サービスロケーションやファイルなどの可変値を指定することを示します。

Windows の場合のフォルダとパスの表記

このマニュアルでは，Windows，AIX，および Linux で共通の内容の場合，Windows の「フォルダ」を「ディレクトリ」と表記しています。また，「¥」を「/」と表記しています。

Windows の場合，「ディレクトリ」を「フォルダ」に，「/」を「¥」に置き換えてお読みください。

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。

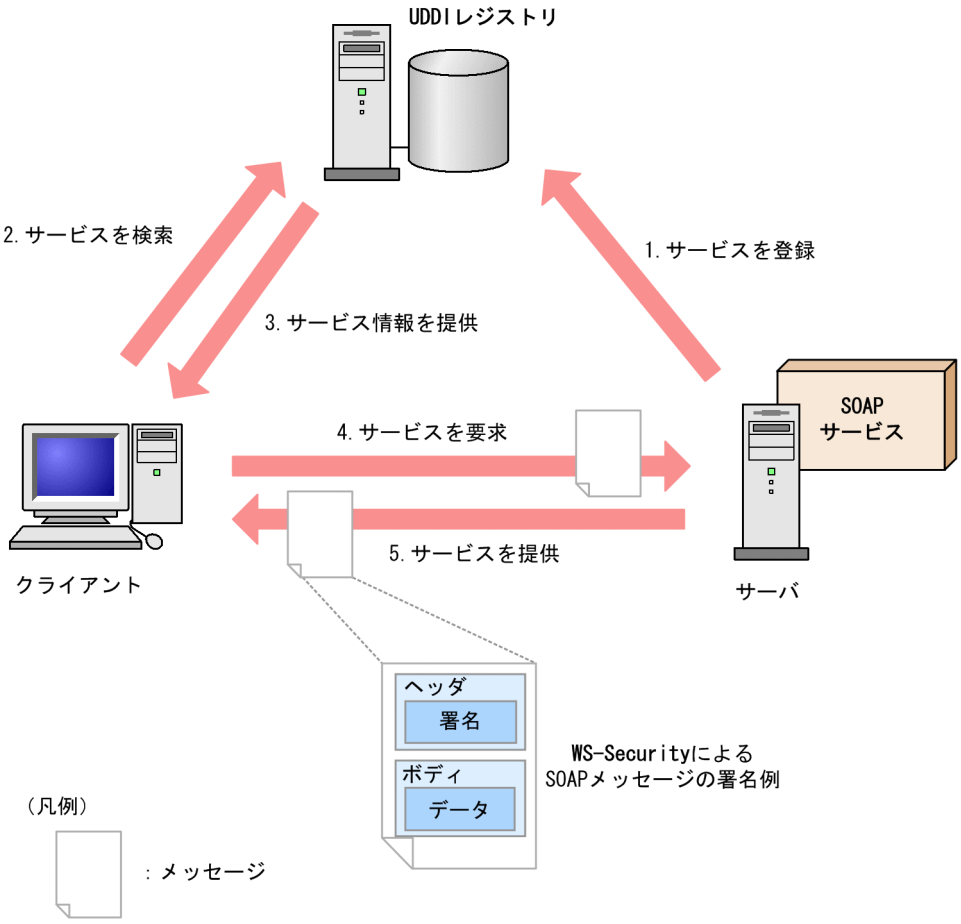
プレフィクス	名前空間 URI
soap	http://schemas.xmlsoap.org/wsdl/soap/
soapenc	http://schemas.xmlsoap.org/soap/encoding/
soapenv	http://schemas.xmlsoap.org/soap/envelope/
wsdl	http://schemas.xmlsoap.org/wsdl/
wsi	http://ws-i.org/profiles/basic/1.1/xsd
xsd	http://www.w3.org/2001/XMLSchema
xsd2000	http://www.w3.org/2000/10/XMLSchema
xsi	http://www.w3.org/2001/XMLSchema-instance

1.2　Web サービスの概要

SOAP アプリケーションの説明に入る前に，まずその上位概念となる **Web サービス**の概要について説明します。

Web サービスは，インターネット上に分散するアプリケーションが提供するサービスを，クライアントに API として提供することを目的としています。Web サービスは，相互接続のためのデータ形式やセキュリティ方式などのインタフェースだけを定義しています。トランスポートプロトコルなどのインタフェース以外については定義していないため，柔軟な相互接続が期待できます。Web サービスに関連する仕様は，基本的に XML がベースになっています。基礎技術としては SOAP，WSDL，UDDI が挙げられます。また，SOAP を対象としたセキュリティ仕様として，WS-Security があります。次に，Web サービスの一例を示します。

図 1-1　Web サービスの一例



Web サービスでは，柔軟にシステム間を相互接続するという概念上，未知のサービスを発見する手段があります。未知のサービスを発見するために，**UDDI レジストリ**を利用します。UDDI レジストリを利用した Web サービスの利用の流れを示します。

1. サービス提供者は，UDDI レジストリに，サービスの内容，アドレスを登録して公開します。
2. サービス利用者は，UDDI レジストリにアクセスし，求めるサービスを検索します。

3. UDDI レジストリは，サービス利用者に，該当するサービスの内容，アドレスを提供します。

4. サービス利用者は，取得したアドレスに接続し，サービスを要求します。

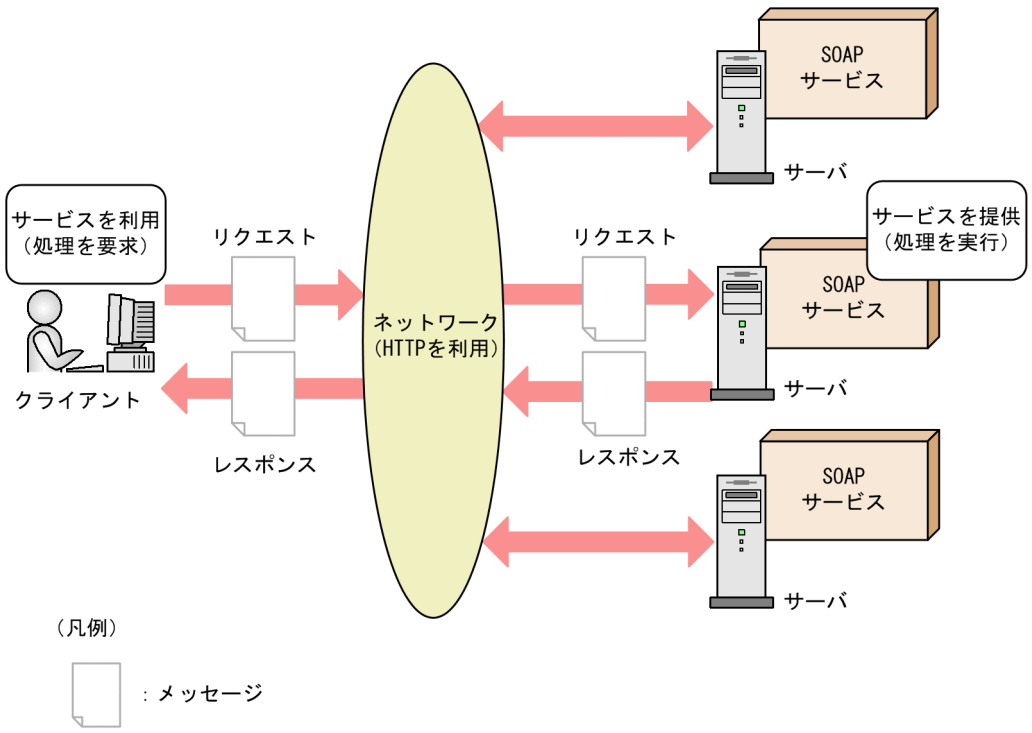
5. サービス提供者は，サービス利用者にサービスを提供します。

サービス利用者とサービス提供者がやり取りする **SOAP メッセージ**は，Web サービスセキュリティの機能を使用することによって，機密情報を保護します。Web サービスセキュリティの機能については，マニュアル「アプリケーションサーバ Web サービスセキュリティ構築ガイド」を参照してください。

1.3　SOAP アプリケーションの概要

SOAP アプリケーションとは，Web の標準技術を利用してネットワーク上でサービスを公開，実行できるアプリケーションのことをいいます。SOAP アプリケーションでは，クライアントとサーバ間のメッセージ交換用のプロトコルに SOAP を利用しています。クライアント側はサーバにメッセージを送信することで，サーバ側で提供される**SOAP サービス**を利用します。次に，SOAP アプリケーションの処理の流れを示します。

図 1-2　SOAP アプリケーションの処理の流れ



リクエストメッセージを受けた SOAP サービスが処理を実行し，処理結果が必要な場合は，レスポンスメッセージをクライアントに返します。

次に，SOAP アプリケーションによるシステム開発の特長について示します。

- 相互接続性の高いシステムを開発できる
- 拡張性の高いシステムが開発できる
- メッセージレベルでセキュアなシステムが開発できる

●相互接続性の高いシステムを開発できる

SOAP ではオブジェクトへのアクセスを実現するために，下位のトランスポート層に Web 標準のプロトコル（HTTP）を利用できます。また，オブジェクトにアクセスするメッセージには，XML の技術を利用します。したがって，Web 標準のプロトコルや XML の技術を利用した既存のシステムを拡張して，相互接続性の高いシステムを開発できます。さらに，開発したシステムを WS-I が定める Basic Profile に適合させることで，他社製品で構築したシステムとの相互接続性をより高めることができま

す。「付録 C　WS-I Basic Profile に適合したシステムを構築するための注意事項」を参考に開発されることをお勧めします。

●拡張性の高いシステムが開発できる

SOAP アプリケーションでは，クライアントとサーバ間でやり取りするメッセージの形式に XML の技術を利用しています。XML 形式のデータは拡張性が高く，データの記述，交換に適しているため，拡張性の高いアプリケーションを開発できます。

●メッセージレベルでセキュアなシステムが開発できる

SOAP アプリケーションでは，従来からの通信路のセキュリティに加え，メッセージレベルでのセキュリティをサポートしています。通信路のセキュリティについては「14.8　FAQ」を参照してください。

メッセージレベルでのセキュリティについては，マニュアル「アプリケーションサーバ Web サービスセキュリティ構築ガイド」を参照してください。

1.4　SOAP アプリケーションを支える技術

SOAP アプリケーションを実現するには，実装するための基本的なプロトコルである SOAP と，実装時のインタフェース定義で使用する WSDL について理解する必要があります。また，公開されているサービスを利用するためには，UDDI の理解も必要です。ここでは，SOAP，WSDL，および UDDI の概要について説明します。

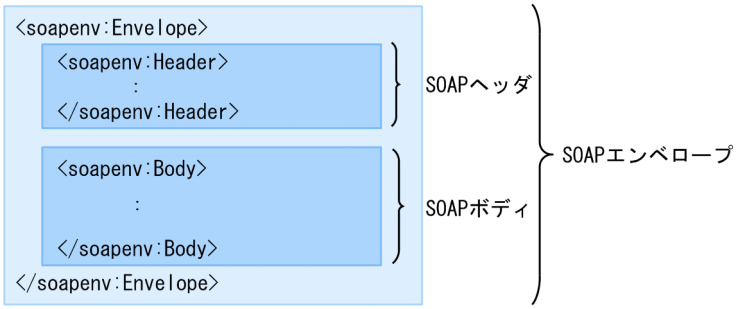
1.4.1　SOAP の概要

SOAP を利用してやり取りするメッセージの構造やメッセージの送受信に関する仕様が，SOAP1.1 仕様として W3C で規定されています。次に，SOAP 1.1 として規定されている仕様の概要について示します。

SOAP メッセージ

クライアントとサーバ間でやり取りするメッセージの構造を規定しています。SOAP メッセージは，SOAP エンベロープと呼ばれる XML ドキュメントの形式の構造を持ち，SOAP ヘッダと SOAP ボディという要素を持ちます。SOAP ボディには，送受信するメッセージを記述します。SOAP メッセージの構造を次に示します。

図 1-3　SOAP メッセージの構造



SOAP 符号化規則

SOAP アプリケーションで使用する変数，配列などのデータを，XML 文書である SOAP メッセージ中にどのように記述するかを規定します。データ型とその XML 記述形式とのマッピングルールです。

SOAP エンコーディング

SOAP 符号化規則の一つで，SOAP1.1 仕様で定義されます。

この符号化規則を使用する場合，encodingStyle 属性には"http://schemas.xmlsoap.org/soap/encoding/"を指定し，use 属性には"encoded"を指定します。広く使われていますが，WS-I Basic Profile 1.0a に対応する場合には推奨されていません。

リテラルエンコーディング

SOAP 符号化規則の一つで，符号化に際して具体的な XML スキーマ定義を参照します。この符号化規則を使用する場合，encodingStyle 属性は指定せず，use 属性には"literal"を指定します。WS-I Basic Profile 1.0a では SOAP エンコーディングではなく，こちらを使用することが推奨されています。

SOAP Fault

SOAP メッセージの処理中に発生したエラーを記述するための SOAP メッセージを表します。SOAP Fault は，SOAP エンジンが作成するものと，SOAP サービスの開発者が作成するものに分けられます。

HTTP バインディング

HTTP を利用した SOAP メッセージの交換について規定しています。ただし，HTTP バインディングは Note での規定となっており，SOAP の仕様として，トランスポートに何を使用するかは規定していません。

RPC 構造

SOAP で RPC を行うための構造化の形式を規定します。

SOAP アプリケーション開発支援機能と SOAP 1.1 のサポート範囲の関係については，「12.1　SOAP 1.1 との対応」を参照してください。

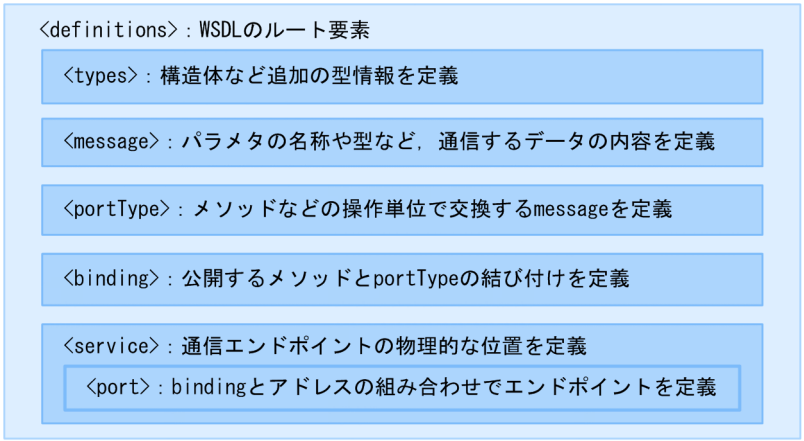
1.4.2　WSDL の概要

WSDL とはインタフェース定義の言語仕様の一つで，開発したプログラムにアクセスするためのインタフェース定義を記述します。

記述形式に XML の技術を採用し，開発したプログラムのオブジェクトが提供する機能，および使用するパラメタを，XML タグの形式で表現できるようにします。WSDL を利用することで，SOAP アプリケーション同士でコミュニケーションを取るときにインタフェース定義を共有できます。SOAP アプリケーション開発支援機能では，SOAP アプリケーション開発支援機能で提供する開発支援コマンドによって WSDL から SOAP アプリケーションに必要なソースコードを生成できます。

次に，WSDL 1.1 で規定されている WSDL の構造を示します。

図 1-4　WSDL の構造



SOAP アプリケーション開発支援機能と WSDL 1.1 のサポート範囲の関係については，「12.2　WSDL 1.1 との対応」を参照してください。

1.4.3 UDDI の概要

UDDI は Web サービスに関する情報を公開したり，検索したりするための API，およびその情報を格納する UDDI レジストリを規定しています。SOAP アプリケーションの情報も UDDI で公開したり，検索したりできます。

次に，UDDI のデータ構造，UDDI の API，および UDDI レジストリについて説明します。

UDDI のデータ構造

UDDI Version 2.0 では五つのデータ構造に関して規定されています。

businessEntity

サービスを公開，管理する主体を記述します。複数の businessService を含みます。

businessService

サービス仕様やサービス内容を記述します。複数の bindingTemplate を含みます。

bindingTemplate

サービスに接続するための情報（エンドポイント）やサービスの構成に関する技術情報を記述します。bindingTemplate は tModel の参照情報だけを含みます。

tModel

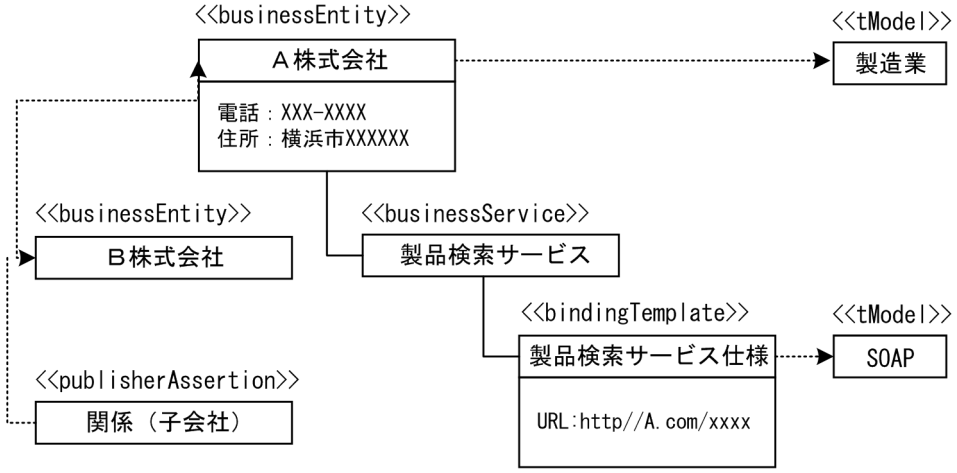
サービスの技術仕様や分類に関する情報を記述します。

publisherAssertion

二つの businessEntity の関係を記述します。

五つのデータ構造の関係を次の図に示します。

図 1-5 UDDI のデータ構造の関係（例）



UDDI の API

UDDI Version 2.0 では，照会 API と発行 API に関して規定されています。

照会 API

データ構造を検索（find）したり，詳細情報を取得（get）したりするための API です。

発行 API

データ構造を登録更新（save）したり，削除（delete）したりするための API です。

UDDI レジストリ

UDDI レジストリは，UDDI のデータ構造を格納管理するためのデータベースであり，UDDI の API を受け付けて処理する機能を持っています。UDDI レジストリは，パブリックとプライベートに分類されます。パブリック UDDI レジストリは，インターネット上に配置され，全世界で同じ UDDI データ構造を共有します。プライベート UDDI レジストリは，企業や組織といった限られた範囲で運用される UDDI レジストリです。

1.5　SOAP アプリケーション開発支援機能の特長

Application Server では，SOAP アプリケーションを開発する機能として，SOAP アプリケーション開発支援機能，SOAP アプリケーションを実行，運用する基盤として SOAP 通信基盤を提供します。SOAP アプリケーション開発支援機能では，RPC 形態およびメッセージング形態の SOAP アプリケーションを開発できます。

RPC 形態の SOAP アプリケーションの概要については，「2.1.1　RPC 形態の SOAP アプリケーション」を参照してください。メッセージング形態の SOAP アプリケーションの概要については，「2.1.2　メッセージング形態の SOAP アプリケーション」を参照してください。

ここでは，SOAP アプリケーション開発支援機能の特長について説明します。

1.5.1　開発支援コマンドを使用して効率良く開発できる

SOAP アプリケーションは開発支援コマンドを使用して開発できます。開発支援コマンドを使用することで，SOAP アプリケーションの開発，実行に必要なファイルやソースコードを自動的に生成できます。

開発支援コマンドが提供する機能を示します。

- **WSDL の生成**
  Java インタフェースおよび Java クラスから，インタフェース定義を記述した WSDL を生成できます。
- **ソースコードの生成**
  WSDL からサーバとクライアントのアクセスに必要なソースコード（スタブとスケルトン）を生成できます。ソースコードを生成することで，複雑な処理を記述する必要はなくなり，業務ロジックの開発に注力できます。
- **サービスデプロイ定義の生成**
  デプロイに関する情報を記述したサービスデプロイ定義を生成できます。

1.5.2　既存の Java プログラムを有効活用できる

SOAP アプリケーションを新規開発するだけでなく，既存の Java プログラムを SOAP アプリケーションとして利用できます。Java アプリケーションや EJB など既存のプログラムを有効活用できるため，SOAP アプリケーション用にプログラムを一から作り直す必要はありません。

1.6　JAX-WS 仕様に対応した Web サービスを開発する場合の注意事項

JAX-WS 仕様に対応した Web サービスを開発する場合，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のではなく，JAX-WS 機能（開発用のコマンドラインインタフェースと通信基盤）に切り替える必要があります。

なお，SOAP アプリケーション開発支援機能および SOAP 通信基盤と，JAX-WS 機能の間に互換性はありません。

JAX-WS 仕様に対応した Web サービスの開発方法については，マニュアル「アプリケーションサーバ Web サービス開発ガイド」を参照してください。

2 SOAP アプリケーションを開発する前に

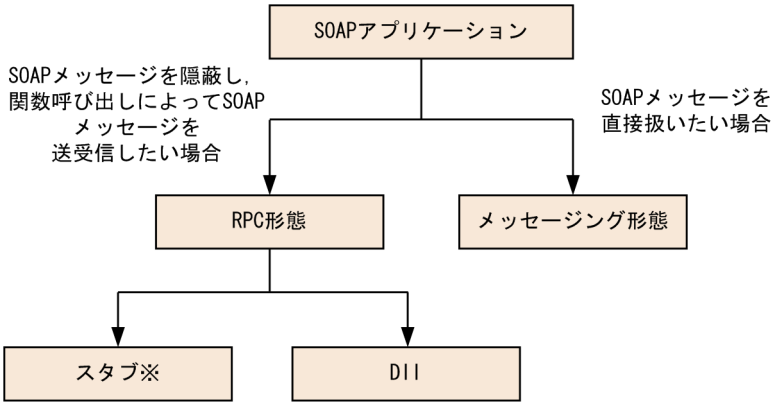
この章では，SOAP アプリケーションを開発する前に必要な SOAP アプリケーションの形態の選択について説明します。また，SOAP アプリケーションの開発，実行で必要な前提環境について説明します。

2.1　SOAP アプリケーションの形態の選択

SOAP アプリケーションでは，サーバとクライアント間のデータの受け渡しに，RPC またはメッセージングのどちらかを利用します。SOAP アプリケーションを開発する前に，どちらの形態を利用するかを決める必要があります。

SOAP アプリケーションの形態，および各形態の利点を次に示します。

図 2-1　SOAP アプリケーションの形態と利点



注※ スタブの場合，サーバ側の実装にEJBを利用できます。

- RPC 形態の SOAP アプリケーション

  RPC 形態の SOAP アプリケーションでは，関数呼び出しによって SOAP メッセージを扱います。サーバとクライアント間の処理に必要な WSDL および Java クラスは，開発支援コマンドで生成できます。

  RPC 形態の SOAP アプリケーションでは，スタブまたは DII（Dynamic Invocation Interface）を使用してクライアント側の処理を実装します。

  また，WSDL のスタイルとして，RPC スタイルまたは DOCUMENT スタイルを選択します。WSDL のスタイルは，構築するシステムに応じて選択してください。RPC スタイルおよび DOCUMENT スタイルについては，「3.3.2　WSDL のスタイルの指定」を参照してください。

- メッセージング形態の SOAP アプリケーション

  メッセージング形態の SOAP アプリケーションでは，SOAP メッセージを直接扱います。サーバとクライアント間の処理は，Java API（SAAJ）を使用して実装します。

2.1.1　RPC 形態の SOAP アプリケーション

RPC 形態の SOAP アプリケーションでは，次のどちらかの方法で SOAP サービスを利用します。

- スタブを使用する
- DII（Dynamic Invocation Interface）を使用する

また，スタブを使用する場合，SOAP アプリケーションのプログラムに EJB を利用することもできます。

ここでは，各方法の概要について説明します。なお，RPC 形態の SOAP アプリケーションを開発するには，WSDL が必要になります。WSDL を使用した開発方法については，「3.1　RPC 形態の SOAP アプリケーション開発の流れ」を参照してください。

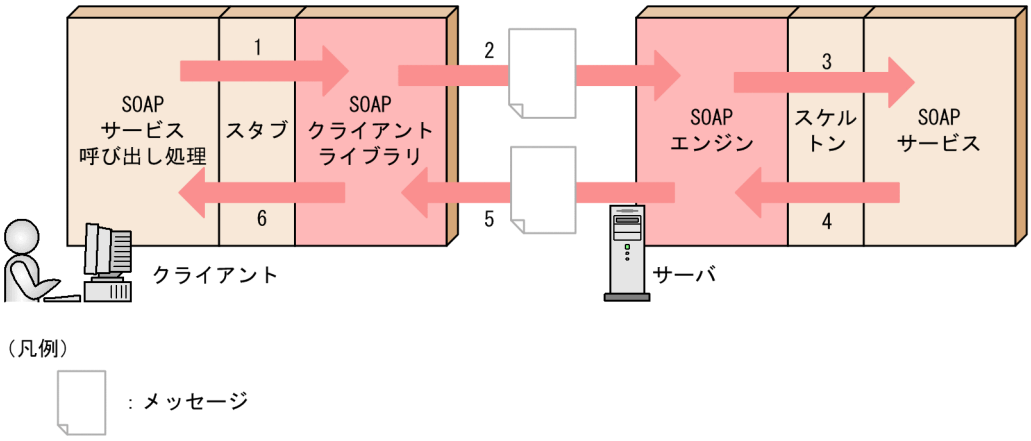
また，RPC 形態の SOAP アプリケーションの開発例については，「4.　RPC 形態の SOAP アプリケーションの開発例」を参照してください。

(1)　スタブを使用する場合

クライアントとサーバ間のインタフェースに関する処理を隠蔽したスタブおよびスケルトンを介して，SOAP メッセージを送受信します。

スタブを使用した RPC 形態の SOAP アプリケーションを実現するには，クライアント側に SOAP サービス呼び出し処理を，サーバ側に SOAP サービスを実装します。クライアント側に実装する SOAP サービス呼び出し処理は，開発支援コマンドによって自動生成されるスタブを使用し，SOAP サービスに渡す引数や戻り値に関する処理を記述することで実装できます。スタブの使用方法については，「4.1.8　クライアント側の処理を実装する」を参照してください。スタブを使用して RPC 形態の SOAP アプリケーションを実現する場合の処理の流れを示します。

図 2-2　RPC 形態の SOAP アプリケーションの処理の流れ（スタブ）



図に沿って，SOAP アプリケーションの処理の流れを説明します。図中に示した番号が次に示す番号に対応しています。

1. SOAP サービス呼び出し処理では，SOAP サービスの URL，メソッド名，および引数を API で指定し，SOAP サービスを呼び出します。
2. SOAP クライアントライブラリは，API で指定された内容から SOAP メッセージを作成し，SOAP エンジンに対してリクエストメッセージを送信します。
3. リクエストメッセージを受信した SOAP エンジンは，SOAP メッセージから呼び出す SOAP サービス，メソッド，および引数を解析して，SOAP サービスを呼び出します。
4. SOAP サービスはパラメタの内容によって処理をして，結果を戻り値として SOAP エンジンに返します。
5. SOAP エンジンは戻り値によって SOAP メッセージを作成し，レスポンスメッセージを SOAP クライアントライブラリに送信します。

6. レスポンスメッセージを受信した SOAP クライアントライブラリは，SOAP メッセージから戻り値を解析し，呼び出し元に返します。

(2)　EJB を利用する場合

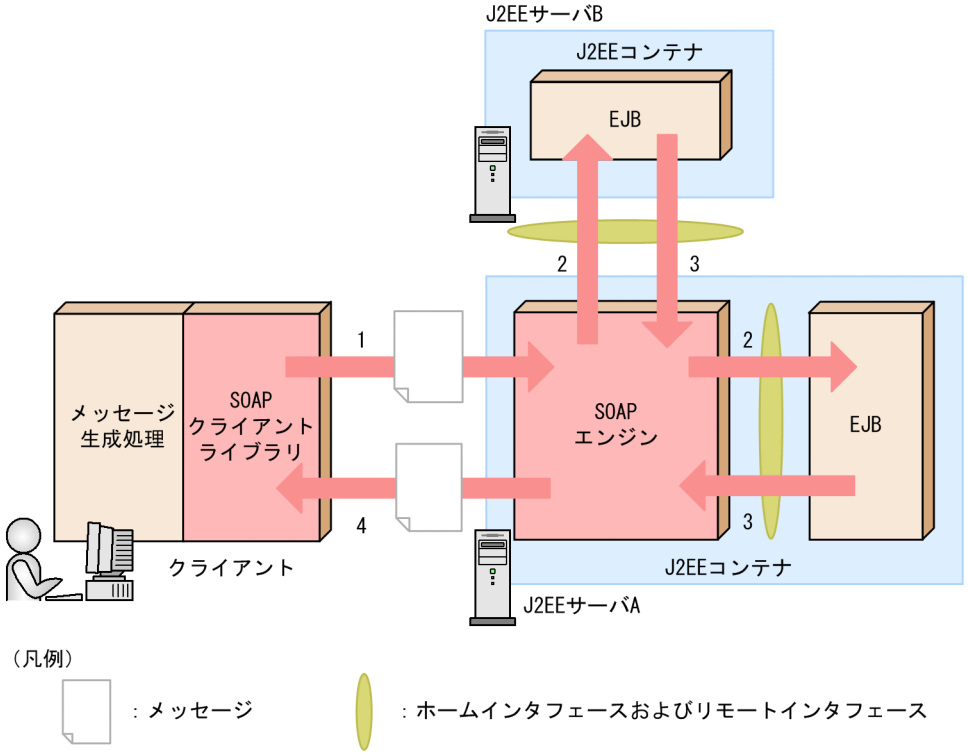
EJB を利用した SOAP アプリケーションは，次の形態に分けられます。

- EJB から SOAP サービスを呼び出す形態
- EJB を SOAP サービスのエンドポイントとする形態

EJB から SOAP サービスを呼び出す形態の処理は，クライアント部分を EJB で実装することを除けば，EJB を利用しない場合と同様です。

EJB を SOAP サービスのエンドポイントとする形態の処理の流れを次に示します。

図 2-3　EJB を SOAP サービスのエンドポイントとする場合の処理の流れ



図に沿って，EJB を SOAP サービスのエンドポイントとする形態の処理を説明します。図中に示した番号が次に示す番号に対応しています。

1. クライアント側で SOAP メッセージを生成し，送信先にリクエストメッセージを送信します。
2. メッセージを受信した SOAP エンジンは，J2EE サーバに配置された EJB を呼び出します。EJB の呼び出しには，ホームインタフェースおよびリモートインタフェースを利用します。SOAP エンジンと同じ J2EE サーバに配置された EJB だけでなく，別の J2EE サーバに配置された EJB を呼び出すこともできます。
3. 呼び出された EJB によって処理を行い，処理結果を SOAP エンジンに返します。

4. SOAP エンジンは，処理結果をクライアントに返します。

SOAP サービスのエンドポイントとして配置できる EJB は，J2EE サーバに配置された，Stateless Session Bean，Stateful Session Bean，および Entity Bean です。

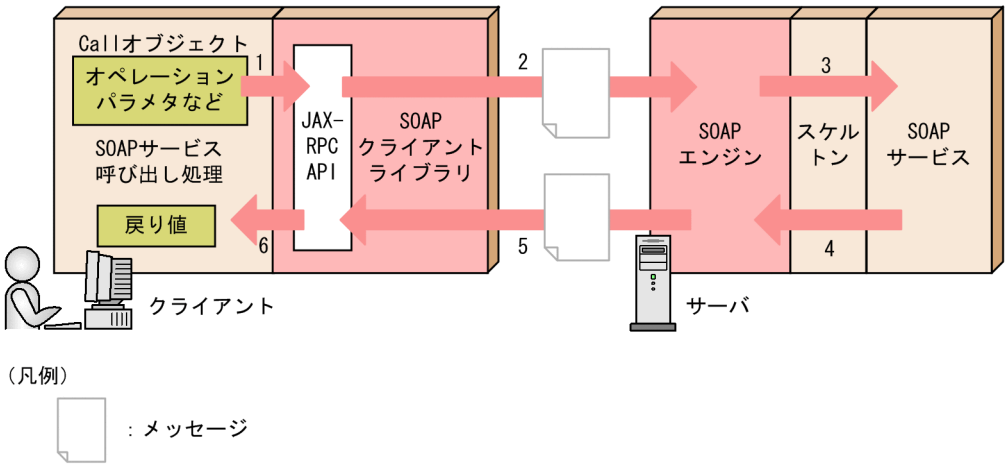
なお，EJB を利用した SOAP アプリケーションの開発例については，「4.3　既存の EJB を利用して開発する場合」を参照してください。EJB から SOAP サービスを呼び出す場合の注意事項については，「3.12 EJB 利用時の注意事項」の「EJB から SOAP サービスを呼び出す場合の注意事項」を参照してください。

(3)　DII を使用する場合

DII とは，サービスインタフェースおよびスタブを使用しないで，JAX-RPC で定義された標準 API で SOAP サービスを呼び出す機能です。スタブを使用しないため，呼び出し先は特定のサービスに固定されることなく，動的に呼び出し先を変更できます。

DII を使用して RPC 形態の SOAP アプリケーションを実現する場合の処理の流れを示します。

図 2-4　RPC 形態の SOAP アプリケーションの処理の流れ（DII）



図に沿って，SOAP アプリケーションの処理の流れを説明します。図中に示した番号が次に示す番号に対応しています。

1. オペレーション名やパラメタなどのサービス呼び出しに必要な情報を Call オブジェクトに入力し，SOAP サービスを呼び出します。サービス呼び出しに必要な情報は，WSDL から読み込んで保持します。
2. SOAP クライアントライブラリ（JAX-RPC API）は，入力した情報をリクエストメッセージに含め，サーバに送信します。
3. リクエストメッセージを受信した SOAP エンジンは，SOAP メッセージから呼び出す SOAP サービス，メソッド，および引数を解析して，SOAP サービスを呼び出します。
4. SOAP サービスはパラメタの内容によって処理をして，結果を戻り値として SOAP エンジンに返します。
5. SOAP エンジンは戻り値によって SOAP メッセージを作成し，レスポンスメッセージを SOAP クライアントライブラリに送信します。

6. レスポンスメッセージを受信した SOAP クライアントライブラリは，SOAP メッセージから戻り値を解析し，呼び出し元に返します。

2.1.2　メッセージング形態の SOAP アプリケーション

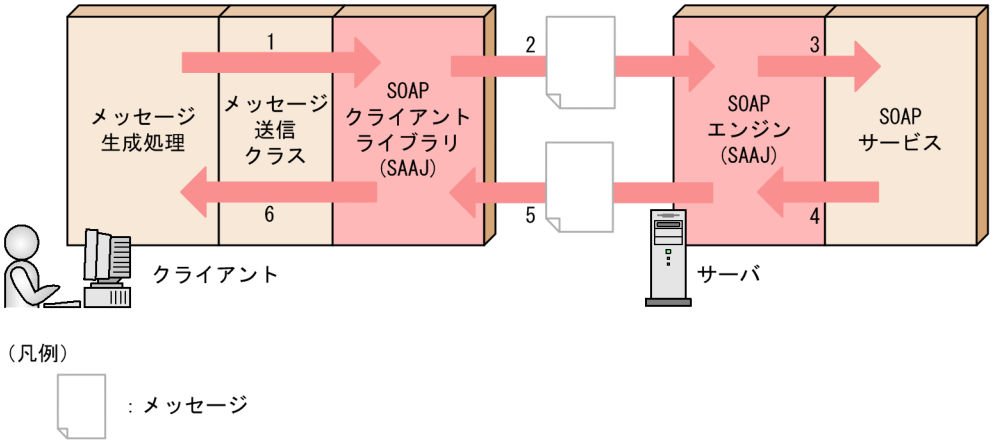
メッセージング形態の SOAP アプリケーションでは，クライアントとサーバ間で任意の XML データを SOAP メッセージとして送受信します。メッセージングでは，SOAP メッセージを直接処理するため，XML 形式で表す複雑なメッセージを扱うことができます。

メッセージング形態の SOAP アプリケーションは，SAAJ を利用して開発できます。なお，メッセージング形態の SOAP アプリケーションでは，WSDL は使用しません。

SAAJ では，メッセージ送信クラスを提供し，クライアントとサーバ間のインタフェースに関する処理を隠蔽するようにしています。

SAAJ を利用した SOAP アプリケーションを実現するには，クライアント側にメッセージ生成処理を，サーバ側に SOAP サービスを実装します。メッセージ生成処理は，SAAJ を使用して，ユーザが開発する必要があります。使用する API については，「12.3　SAAJ 1.2 との対応」を参照してください。次に，SAAJ を使用したメッセージングによる SOAP アプリケーションの処理の流れを示します。

図 2-5　メッセージング形態の SOAP アプリケーションの処理の流れ（SAAJ）



図に沿って，SAAJ を利用した SOAP アプリケーションの処理を説明します。図中に示した番号が次に示す番号に対応しています。

1. メッセージ生成処理では，送信先と送信するメッセージを，提供する API に渡します。送信するメッセージはユーザが SAAJ の API を利用して作成する必要があります。
2. SOAP クライアントライブラリ（SAAJ）は，API で指定されたメッセージを含んだ SOAP メッセージを作成し，送信先にメッセージを送信します。
3. メッセージを受信した SOAP エンジン（SAAJ）は，ユーザ指定部分を解析して取り出し，SOAP サービスを呼び出します。SOAP サービスを実装するメソッドは，あらかじめ登録しておく必要があります。

4. SOAP サービスはメッセージの内容を解析し，処理をします。また，必要があればレスポンスメッセージを作成し，SOAP エンジンに返します。
5. レスポンスメッセージを受け取った場合，SOAP エンジン（SAAJ）は SOAP サービスを実装するメソッドが返したメッセージをクライアントに返信します。
6. SOAP クライアントライブラリ（SAAJ）は，受信した SOAP メッセージの中からユーザ指定部分を解析して取り出し，メッセージを返します。

SAAJ を利用した SOAP アプリケーションの開発例については，「6. メッセージング形態の SOAP アプリケーションの開発例」を参照してください。

2.2 SOAP アプリケーションの前提環境

SOAP アプリケーションの開発および実行時に必要な前提環境について示します。

また，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前に注意が必要なコネクションプーリングについて説明します。

2.2.1 開発時および実行時の前提環境

SOAP アプリケーションは 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 OS および前提ソフトウェアについては，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」を参照してください。

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

また，次の場合には WSDL4J が必要になります。Application Server または Developer をインストールしたあとに，WSDL4J をインストールしてください。

- SOAP アプリケーションを開発する
- DII を使用した RPC 形態の SOAP アプリケーションを実行する

次の場合には WSDL4J をインストールする必要はありません。

- スタブを使用した RPC 形態の SOAP アプリケーションを実行する
- メッセージング形態の SOAP アプリケーションを実行する

2.2.2 .NET Framework を使用する場合の前提環境

Application Server で動作する SOAP サービスは，.NET Framework を使用して開発したクライアントアプリケーションから利用できます。

- **開発時の前提環境**

  .NET Framework を使用して SOAP アプリケーションを開発する場合，次のプログラムを追加してください。

  Microsoft(R) .NET Framework 2.0 Software Development Kit

  開発時に使用できる言語は C#です。

- **実行時の前提環境**

  .NET Framework を使用して開発した SOAP アプリケーションを実行する場合，次のプログラムをクライアント側に追加してください。

Microsoft(R) .NET Framework Version 2.0

2.2.3　コネクションプーリング利用時の注意事項

Application Server 08-00 および Developer 08-00 から，コネクションプーリングのデフォルト設定を無効（使用しない）に変更しています。

コネクションプーリングを使用する場合，「7.4　コネクションプーリング」を参照し，その内容を十分に理解した上で使用してください。

2.3　SOAP アプリケーション開発支援機能および SOAP 通信基盤の設定

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について説明します。

バージョンアップインストールする場合は，「付録 A　SOAP アプリケーションの移行」を参照してください。

2.3.1　J2EE サーバ用オプション定義ファイルの定義内容

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には，J2EE サーバ用オプション定義ファイルで定義します。

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」と「cjjaxrs.jar」の行を無効にします。次の例に従って定義してください。

```
...
add.class.path=<cosminexus.home>¥c4web¥lib¥hitsaaj.jar
#add.class.path=<cosminexus.home>¥jaxws¥lib¥cjjaxws.jar
#add.class.path=<cosminexus.home>¥jaxrs¥lib¥cjjaxrs.jar
...
```

J2EE サーバ用オプション定義ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」を参照してください。

注意事項

「hitsaaj.jar」，「cjjaxws.jar」，および「cjjaxrs.jar」のすべてのコメントを外した場合，またはすべてにコメントを付けた場合の動作は保証されません。

以降では，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。

2.3.2　直接編集する場合

直接編集する場合，次の場所に格納された J2EE サーバ用オプション定義ファイルをテキストエディタで開き，内容を変更します。

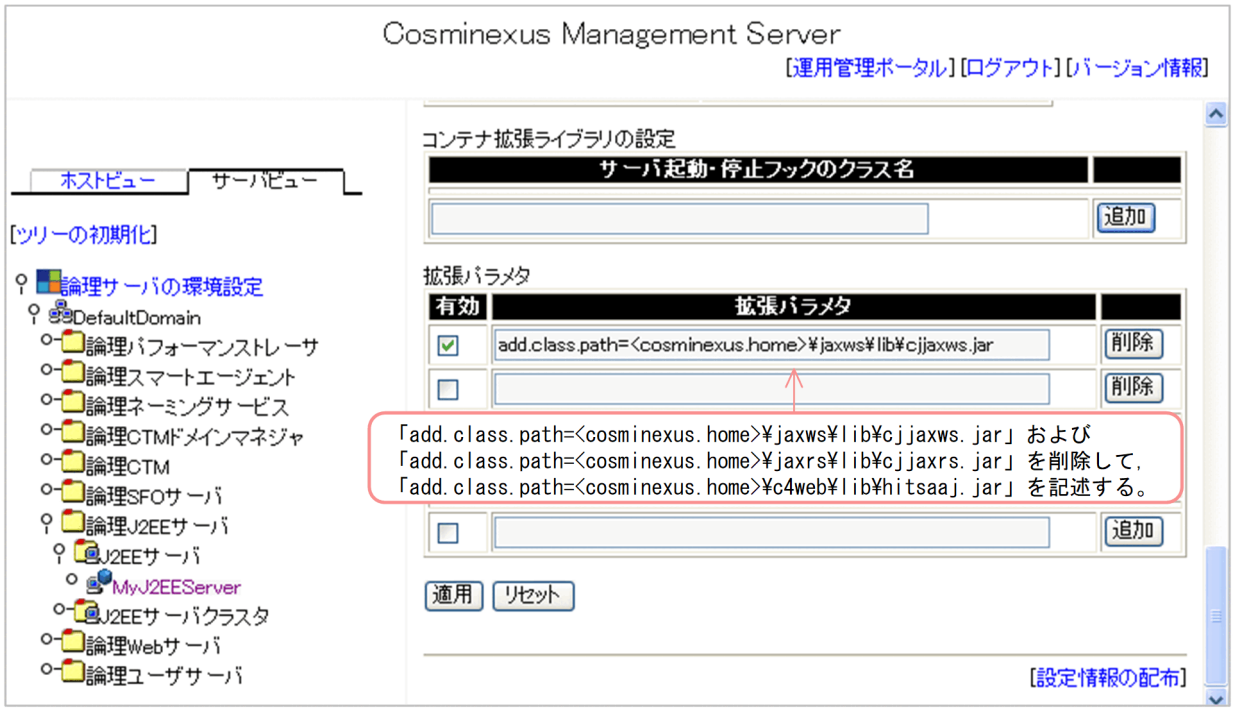
<Application Server のインストールディレクトリ>/CC/server/usrconf/ejb/<J2EE サーバ名>/usrconf.cfg

2.3.3　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。

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 2-6　運用管理ポータルによる SOAP アプリケーション開発支援機能／SOAP 通信基盤の設定例



運用管理ポータルの［J2EE コンテナの設定］画面については，マニュアル「アプリケーションサーバ 運用管理ポータル操作ガイド」を参照してください。

2.3.4　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 機能および簡易構築定義ファイルについては，マニュアル「アプリケーションサーバ システム構築・運用ガイド」を参照してください。

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>
```

3 RPC 形態の SOAP アプリケーションの開発

RPC 形態の SOAP アプリケーションは，開発支援コマンドを使用して効率良く開発できます。

この章では，RPC 形態の SOAP アプリケーション開発の流れ，および各開発工程で必要な作業について説明します。また，document/literal に対応した SOAP アプリケーションを開発する方法および例外処理を実装する方法についても説明します。

3.1　RPC 形態の SOAP アプリケーション開発の流れ

RPC 形態の SOAP アプリケーションは，開発支援コマンドで WSDL や Java クラスなどを生成しながら開発できます。

ここでは，次に示す場合の開発の流れを示します。

- SOAP アプリケーションを新規に開発する場合
- 既存の Java クラスを利用して SOAP アプリケーションを開発する場合
- 既存の EJB を利用して SOAP アプリケーションを開発する場合
- 添付ファイルを使用して SOAP アプリケーションを開発する場合
- DII を使用して SOAP アプリケーションを開発する場合
- .NET Framework を使用して SOAP アプリケーションのクライアントを開発する場合

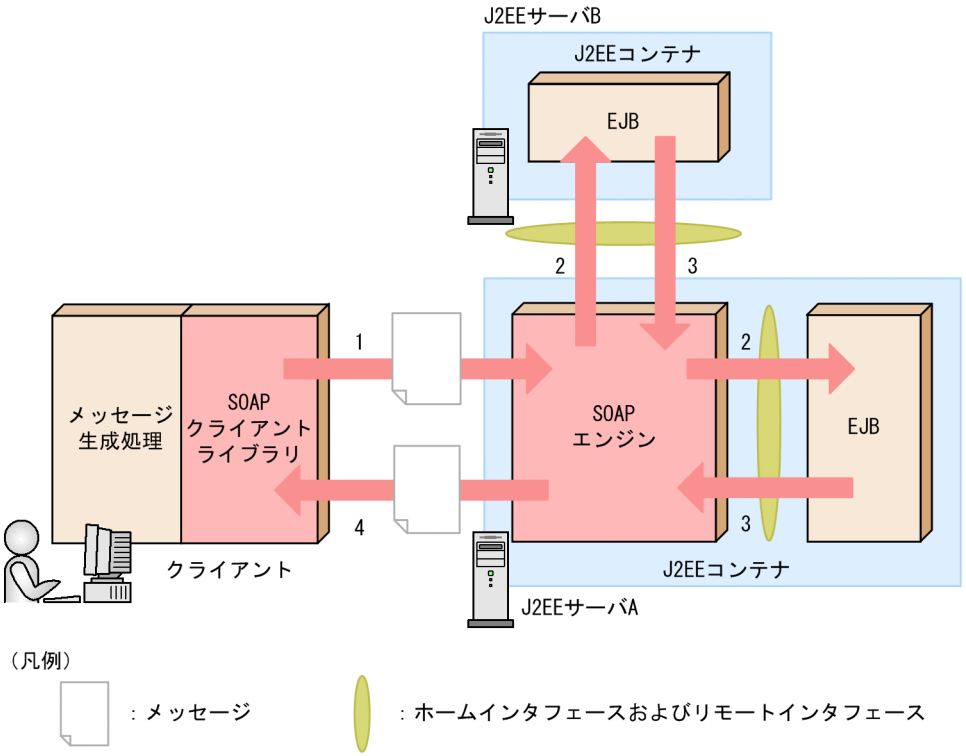
JAX-WS 仕様に対応した Web サービスの開発方法については，マニュアル「アプリケーションサーバ Web サービス開発ガイド」を参照してください。

開発支援コマンドについては，「9.　開発支援コマンド」を参照してください。

3.1.1　SOAP アプリケーションを新規に開発する場合

SOAP アプリケーションを新規に開発する場合の流れを示します。

図 3-1　WSDL を使用した SOAP アプリケーションの新規開発



SOAP アプリケーションを新規に開発する場合，Java インタフェース（リモートインタフェース）から WSDL を生成し，生成した WSDL からサーバおよびクライアントに必要な次のファイルを生成します。

- サーバで使用する Java クラス（スケルトン）およびサービスデプロイ定義
- クライアントで使用する Java クラス（スタブ）

SOAP アプリケーションを新規に開発する場合の手順を示します。括弧内の数字は，開発例の参照先を示します。

1. Java インタフェース（リモートインタフェース）を作成する（4.1.2）

   SOAP アプリケーションの WSDL を作成するために，Java インタフェース，およびユーザ定義のデータ型クラスを作成します。

2. WSDL を生成する（4.1.3）

   作成した Java インタフェースおよびユーザ定義のデータ型クラスから，SOAP アプリケーションの WSDL を生成します。

3. スケルトンおよびサービスデプロイ定義を生成する（4.1.4）

   生成された WSDL からサーバ側に必要なソースコードであるスケルトンを生成します。同時に，生成したソースコードを SOAP アプリケーションとして利用するためのサービスデプロイ定義を生成します。

4. サーバ側の処理を実装する（4.1.5）

   スケルトン生成時に生成された実装クラスに，SOAP サービスの実装を記述します。

5. スタブを生成する（4.1.7）

WSDL からクライアント側に必要なソースコードであるスタブを生成します。

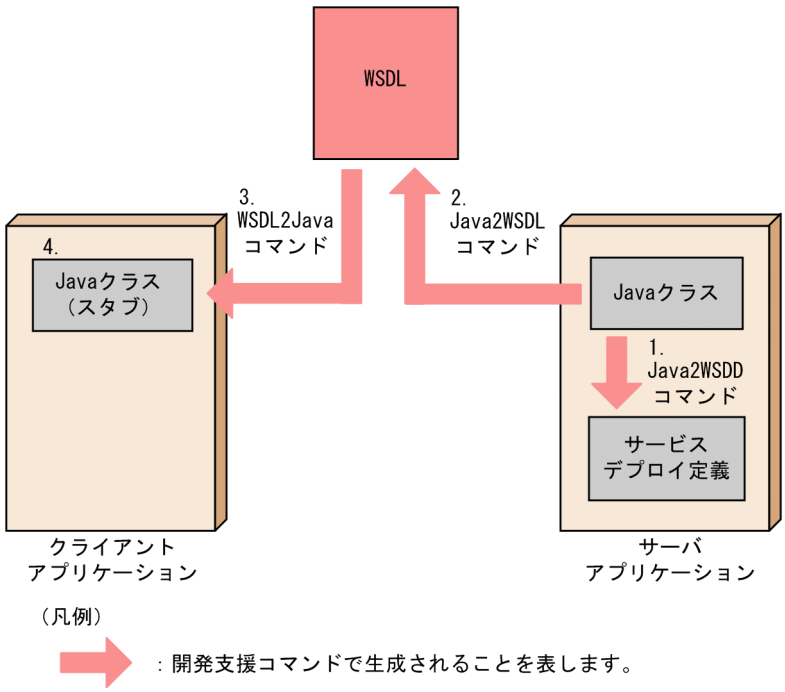
6. クライアント側の処理を実装する（4.1.8）

SOAP サービスを利用するクライアント側の処理を実装します。

3.1.2　既存の Java クラスを利用して SOAP アプリケーションを開発する場合

既存の Java クラスを利用して SOAP アプリケーションを開発する場合の流れを示します。

図 3–2　既存の Java クラスを利用した SOAP アプリケーションの開発



既存の Java クラスから WSDL およびサービスデプロイ定義を生成します。生成した WSDL からクライアントで使用する Java クラス（スタブ）を生成します。

既存の Java クラスを利用して SOAP アプリケーションを開発する場合の手順を示します。括弧内の数字は，開発例の参照先を示します。

1. サービスデプロイ定義を生成する（4.2.2）

既存の Java クラスを SOAP アプリケーションとして利用するためのサービスデプロイ定義を生成します。

2. WSDL を生成する（4.2.4）

既存の Java クラスを SOAP サービスとして公開するための WSDL を生成します。

3. スタブを生成する（4.2.5）

生成された WSDL からクライアント側に必要なソースコードであるスタブを生成します。

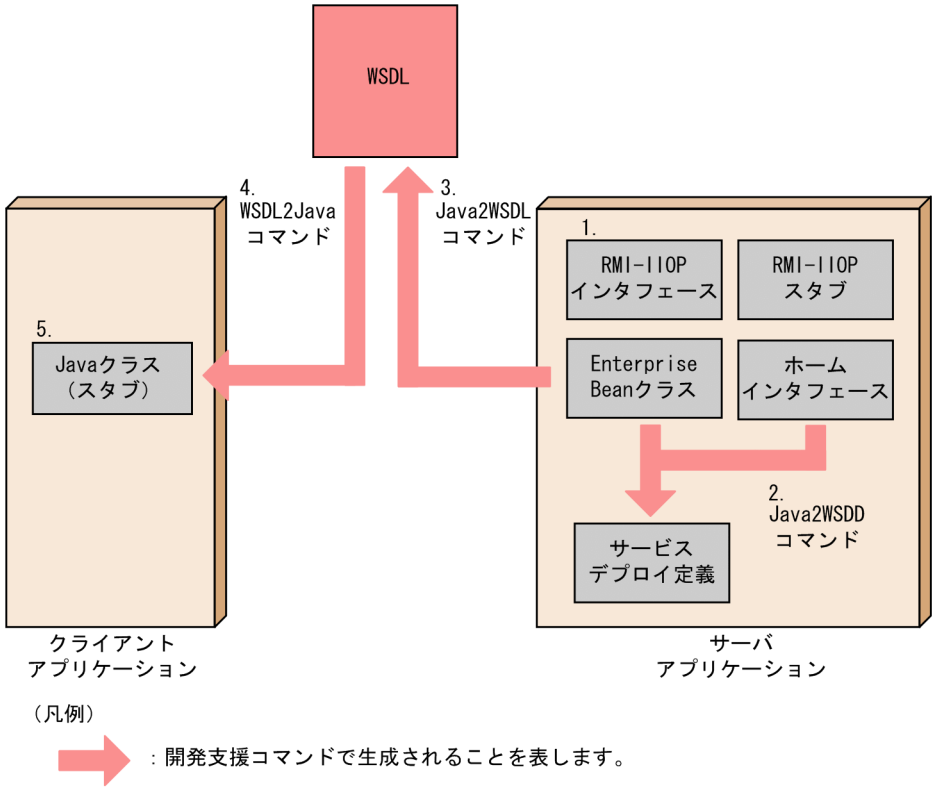
4. クライアント側の処理を実装する（4.2.6）

SOAP サービスを利用するクライアント側の処理を実装します。

3.1.3　既存の EJB を利用して SOAP アプリケーションを開発する場合

既存の EJB を利用して SOAP アプリケーションを開発する場合の流れを示します。

図 3-3　既存の EJB を利用した SOAP アプリケーションの開発



既存の EJB から WSDL およびサービスデプロイ定義を生成します。生成した WSDL からクライアントで使用する Java クラス（スタブ）を生成します。

既存の EJB を利用して SOAP アプリケーションを開発する場合の手順を示します。括弧内の数字は，開発例の参照先を示します。

1. EJB 呼び出し環境を設定する（4.3.2）

J2EE アプリケーションの RMI-IIOP インタフェースおよび RMI-IIOP スタブを取得します。

2. サービスデプロイ定義を生成する（4.3.3）

Enterprise Bean クラスとホームインタフェースから，既存の Java クラスを SOAP アプリケーションとして利用するためのサービスデプロイ定義を生成します。

3. WSDL を生成する（4.3.5）

既存の EJB を SOAP サービスとして公開するための WSDL を生成します。

4. スタブを生成する（4.3.6）

生成された WSDL からクライアント側に必要なソースコードであるスタブを生成します。

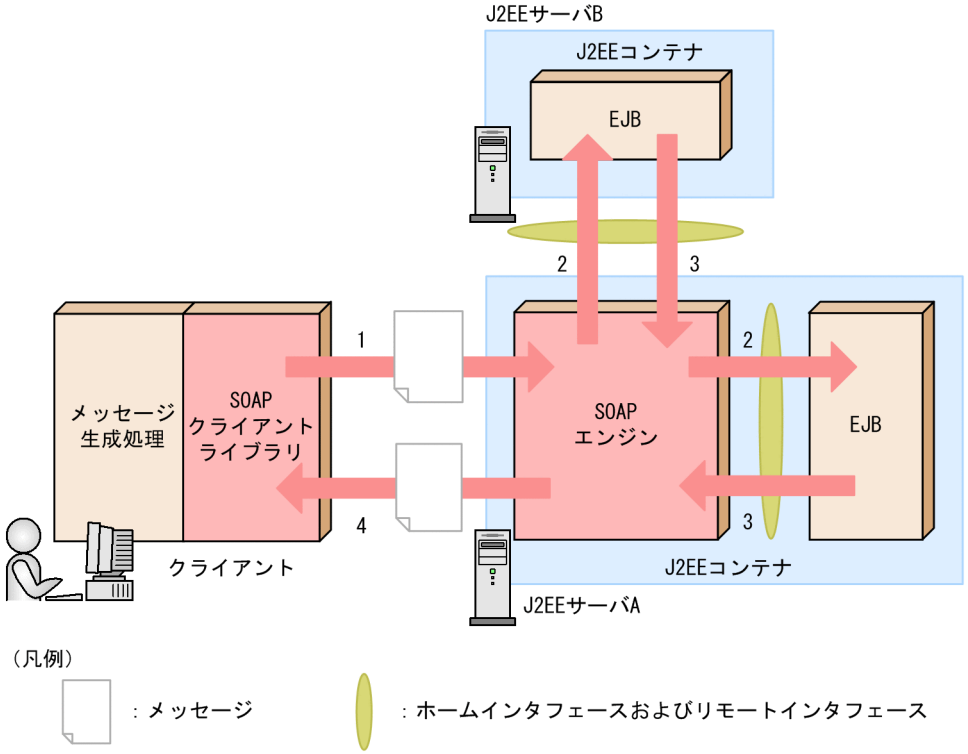
5. クライアント側の処理を実装する（4.3.7）

SOAP サービスを利用するクライアント側の処理を実装します。

3.1.4　添付ファイルを使用して SOAP アプリケーションを開発する場合

添付ファイルを使用した SOAP アプリケーションを開発する場合の流れを示します。

図 3-4　添付ファイルを使用した SOAP アプリケーションの開発



添付ファイルを使用した SOAP アプリケーションを開発する場合，Java インタフェース（リモートインタフェース）から WSDL を生成し，生成した WSDL からサーバおよびクライアントに必要な次のファイルを生成します。

- サーバで使用する Java クラス（スケルトン）およびサービスデプロイ定義
- クライアントで使用する Java クラス（スタブ）

添付ファイルを使用した SOAP アプリケーションを開発する場合の手順を示します。括弧内の数字は，開発例の参照先を示します。

1. Java インタフェース（リモートインタフェース）を作成する（4.4.2）

SOAP アプリケーションの WSDL を作成するために，Java インタフェース，およびユーザ定義のデータ型クラスを作成します。

2. WSDL を生成する（4.4.3）

作成した Java インタフェースおよびユーザ定義のデータ型クラスから，SOAP アプリケーションの WSDL を生成します。

3. スケルトンおよびサービスデプロイ定義を生成する（4.4.4）

生成された WSDL からサーバ側に必要なソースコードであるスケルトンを生成します。同時に，生成したソースコードを SOAP アプリケーションとして利用するためのサービスデプロイ定義を生成します。

4. サーバ側の処理を実装する（4.4.5）

スケルトン生成時に生成された実装クラスに，SOAP サービスの実装を記述します。

5. スタブを生成する（4.4.7）

WSDL からクライアント側に必要なソースコードであるスタブを生成します。

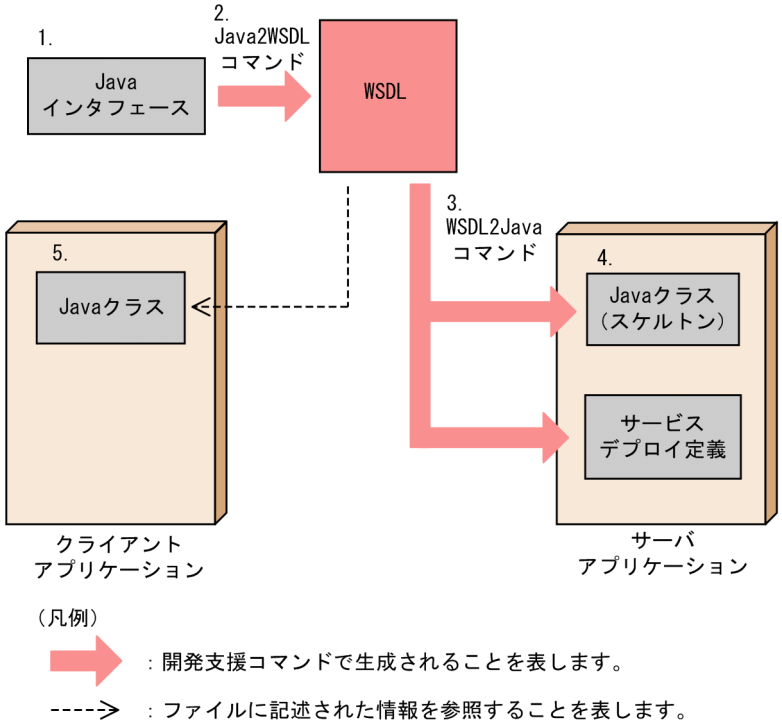
6. クライアント側の処理を実装する（4.4.8）

SOAP サービスを利用するクライアント側の処理を実装します。

3.1.5　DII を使用して SOAP アプリケーションを開発する場合

DII を使用した SOAP アプリケーションを開発する場合の流れを示します。

図 3-5　DII を使用した SOAP アプリケーションの開発



Java インタフェースから WSDL を生成し，生成した WSDL からサーバで使用する Java クラス（スケルトン）およびサービスデプロイ定義を生成します。DII を使用する場合，スタブは使用しません。SOAP サービスの呼び出しに必要な情報を WSDL ファイルから取得します。

DII を使用した SOAP アプリケーションを開発する場合の手順を示します。括弧内の数字は，開発例の参照先を示します。

1. Java インタフェース（リモートインタフェース）を作成する（4.1.2）
   SOAP アプリケーションの WSDL を作成するために，Java インタフェース，およびユーザ定義のデータ型クラスを作成します。

2. WSDL を生成する（4.1.3）
   作成した Java インタフェースから，SOAP アプリケーションの WSDL を生成します。

3. スケルトンおよびサービスデプロイ定義を生成する（4.1.4）
   生成された WSDL からサーバ側に必要なソースコードであるスケルトンを生成します。

4. サーバ側の処理を実装する（4.1.5）
   スケルトンの生成と同時に生成された実装クラスに，SOAP サービスの実装を記述します。

5. DII のクライアント側の処理を実装する（3.8.2）
   SOAP サービスを利用するクライアント側の処理を実装します。

3.1.6 .NET Framework を使用して SOAP アプリケーションのクライアントを開発する場合

.NET Framework を使用して SOAP アプリケーションを開発する場合の流れは，新規に開発する場合，既存の Java クラスを利用する場合，および既存の EJB を利用する場合と同じです。

ただし，クライアント側に必要なソースコード（スタブ）を生成するときに，.NET Framework SDK が提供する wsdl コマンドを使用する必要があります。

3.2　Java インタフェースの作成と Java クラスの利用

SOAP アプリケーションを新規に開発する場合，最初に WSDL の生成に必要な Java インタフェースを作成します。既存の Java クラスまたは既存の EJB を利用して SOAP アプリケーションを開発する場合，最初に Java クラスまたは EJB が利用できる状態かどうかを確認します。

SOAP アプリケーションを新規に開発する場合，3.2.1～3.2.6 に記載された内容に従って Java インタフェースを作成してください。

既存の Java クラスおよび EJB を利用して SOAP アプリケーションを開発する場合は，「3.2.7　既存の Java クラスおよび EJB の利用」を参照してください。

3.2.1　Java インタフェース作成の概要

SOAP アプリケーションの新規開発時に作成する Java インタフェース（リモートインタフェース）の例を示します。

```
package com.sample;

public interface Sample extends java.rmi.Remote{

    public java.lang.String test( int od )
      throws java.rmi.RemoteException, com.sample.UserDefinedException;
}
```

リモートインタフェースのメソッドのパラメタや戻り値には，ユーザ定義のデータ型クラスを使用できます。

なお，リモートインタフェースとユーザ定義のデータ型クラスは，Java2WSDL コマンドで WSDL を生成するために作成します。SOAP アプリケーションで使用するのは，WSDL2Java コマンドで生成したリモートインタフェースおよびユーザ定義のデータ型クラスであることに注意してください。

3.2.2　リモートインタフェース作成時の規則

リモートインタフェースは，次に示す規則に従って作成してください。

表 3-1　リモートインタフェースの作成規則

項目	規則
継承関係	java.rmi.Remote インタフェースを継承します。
パッケージ名	• 半角英数字（大文字／小文字ともに使用できます）およびアンダースコア（"_"）の範囲で Java 言語の文法規則に従って記述してください。 • デフォルトパッケージは使用できません。

項目	規則
インタフェース名	半角英数字（大文字／小文字ともに使用できます）およびアンダースコア（"_"）の範囲でJava 言語の文法規則に従って記述してください。
メソッド名	
メソッドのパラメタ名	
メソッドのパラメタと戻り値の型	「3.2.3　リモートインタフェースで使用できる型とクラス」を参照してください。
メソッドの例外	• java.rmi.RemoteException を定義してください。※ • ユーザ定義例外を定義することもできます。ユーザ定義例外を定義する場合は，「3.2.4 ユーザ定義のデータ型クラス作成時の規則」を参照してください。

注※

throws 節に java.rmi.RemoteException を定義していない場合，Java2WSDL コマンドによって KDCCC0228-W のメッセージが出力され，処理は続行されます。また，throws 節に java.rmi.RemoteException およびユーザ定義例外のどちらでもない例外を定義した場合，Java2WSDL コマンドによって KDCCC0229-W のメッセージが出力されます。この例外は wsdl:fault 要素にマッピングされません。

3.2.3　リモートインタフェースで使用できる型とクラス

リモートインタフェースのメソッドのパラメタや戻り値には，次に示す型およびクラスを使用できます。また，配列を使用することもできます。

- プリミティブ型（int や double など）
- ラッパークラス（java.lang.Integer や java.lang.Double など）
- Java 標準クラス（java.lang.String や java.util.Calendar など）
- Application Server 独自の型（org.apache.axis.types.NegativeInteger など）
- Holder クラス（javax.xml.rpc.holders.IntHolder や javax.xml.rpc.holders.DoubleHolder など）※1
- ユーザ定義のデータ型クラス※2

注※1

Holder クラスは用途に制限があります。Holder クラスの制限については，「3.2.5　Holder クラスによる INOUT パラメタの格納」を参照してください。

注※2

ユーザ定義のデータ型クラスについては「3.2.4　ユーザ定義のデータ型クラス作成時の規則」を参照してください。

使用できる型とクラスの一覧については「11.3　Java クラスから WSDL を生成する場合のデータ型の関係」を参照してください。

3.2.4　ユーザ定義のデータ型クラス作成時の規則

リモートインタフェースのメソッドのパラメタや戻り値には，ユーザ定義のデータ型クラスを使用できます。DII でユーザ定義のデータ型クラスを使用する場合は，「3.8.3　DII でのユーザ定義のデータ型クラスの使用」を参照してください。

ユーザ定義のデータ型クラスは，次に示す規則に従って作成してください。

表 3-2　ユーザ定義のデータ型クラスの作成規則

項目	規則
継承関係	java.lang.Object クラスまたはほかのユーザ定義のデータ型クラスを継承します。※1
パッケージ名	• 半角英数字（大文字／小文字ともに使用できます）およびアンダースコア（"_"）の範囲で Java 言語の文法規則に従って記述してください。 • デフォルトパッケージは使用できません。
クラス名	半角英数字（大文字／小文字ともに使用できます）およびアンダースコア（"_"）の範囲で Java 言語の文法規則に従って記述してください。
フィールド（プロパティ）名	
メソッド（アクセサ）名	
メソッド（アクセサ）のパラメタ名	
フィールド（プロパティ）の型	「3.2.3　リモートインタフェースで使用できる型とクラス」を参照してください。
メソッド（アクセサ）のパラメタと戻り値の型	
コンストラクタ	public であるデフォルトコンストラクタを定義してください。
プロパティおよびアクセサの関係※2	• プロパティとアクセサ（getter および setter）は必ず一組になるように定義してください。 • プロパティ以外のフィールドやアクセサ以外のメソッドは，定義しないでください。

注※1

java.util.Calendar のように直接 java.lang.Object を継承している場合でも，次に示す Java 標準クラスおよび Application Server の実装クラスをスーパークラスとして利用することはできません。

Java 標準クラス：J2SE 5.0 仕様に対応した次のパッケージ名で始まるクラスは利用できません。

・java
・javax
・org.ietf
・org.omg
・org.w3c
・org.xml

Application Server の実装クラス：Application Server が提供する次のパッケージ名で始まるクラスは利用できません。

・com.cosminexus.c4web
・com.cosminexus.cws
・com.cosminexus.xml

・com.cosminexus.wss
・org.apache.axis
・org.apache.commons.discovery
・com.ibm.wsdl

注※2

プロパティとはユーザ定義のデータ型クラスで保持する情報のことです。また，アクセサとはプロパティの値を取得したり設定したりするためのメソッドのことです。アクセサのうち，取得するメソッドを getter，設定するメソッドを setter と呼びます。getter と setter はプロパティと関係づけて一組定義してください。

getter および setter を作成するときの規則を次に示します。

表 3-3　getter の作成規則

項目	規則
メソッド（アクセサ）名	"get" ＋ 先頭を大文字にしたプロパティ名を定義してください。
パラメタ	定義しないでください。
戻り値	プロパティの値を返してください。

表 3-4　setter の作成規則

項目	規則
メソッド（アクセサ）名	"set" ＋ 先頭を大文字にしたプロパティ名を定義してください。
パラメタ	プロパティと同じ型のパラメタを一つだけ定義してください。パラメタ名は任意です。
戻り値	void としてください。

3.2.5　Holder クラスによる INOUT パラメタの格納

リモートインタフェースのメソッドのパラメタに，INOUT パラメタを格納するために Holder クラスを使用できます。リモートインタフェースのメソッドのパラメタ以外では Holder クラスは使用できません。

Holder クラスの生成例，および使用できない Holder クラスの用途の例を示します。

(1)　Holder クラスの生成例

- Holder クラスを使用したリモートインタフェース

```
package localhost;

public interface HolderTest extends java.rmi.Remote{

    public void test( javax.xml.rpc.holders.StringHolder inout ) throws java.rmi.RemoteEx
ception;
}
```

- リモートインタフェースから，Java2WSDL コマンドによって生成された WSDL の一部

```
<wsdl:definitions targetNamespace="http://localhost" ... >
...
 <wsdl:message name="testRequest">
  <wsdl:part name="in0" type="xsd:string"/>
 </wsdl:message>
 <wsdl:message name="testResponse">
  <wsdl:part name="in0" type="xsd:string"/>
 </wsdl:message>
 <wsdl:portType name="HolderTest">
  <wsdl:operation name="test" parameterOrder="in0">
   <wsdl:input message="intf:testRequest" name="testRequest"/>
   <wsdl:output message="intf:testResponse" name="testResponse"/>
  </wsdl:operation>
 </wsdl:portType>
...
</wsdl:definitions>
```

(2) 使用できない Holder クラスの用途の例

Holder クラスは，リモートインタフェースのメソッドのパラメタだけに使用できます。メソッドの戻り値，配列の要素，およびユーザ定義のデータ型クラスのフィールドでは使用できません。

次に示す例を含む，リモートインタフェースのメソッドのパラメタ以外の用途に使用しないでください。

- メソッドの戻り値

  （例）Java インタフェース

```
package localhost;

public interface HolderTest extends java.rmi.Remote{

    public javax.xml.rpc.holders.StringHolder test() throws java.rmi.RemoteException;
}
```

- 配列の要素

  （例）Java インタフェース

```
package localhost;

public interface HolderTest extends java.rmi.Remote{

    public void test( javax.xml.rpc.holders.StringHolder[] inout ) throws java.rmi.Remote
Exception;
}
```

- ユーザ定義のデータ型クラスのフィールド

  （例）Java インタフェース

```
package localhost;

public interface HolderTest extends java.rmi.Remote{
```

```
    public void test( localhost.UserData data ) throws java.rmi.RemoteException;
}
```

（例）上記のユーザ定義のデータ型クラス

```
package localhost;

public class UserData implements java.io.Serializable {
    public javax.xml.rpc.holders.StringHolder value;
}
```

このような使い方をした場合，開発支援コマンドが KDCCC0012-E のエラーメッセージを出力して終了したり※1，通信時に C4Fault 例外※2，または RemoteException 例外※3 が発生したりすることがあります。

注※1

メソッドの戻り値に Holder クラスを使用した場合が該当します。

注※2

Holder クラスを注※1 を除く用途に使用して SOAP アプリケーションを開発し，スタブを使用して実行した場合が該当します。

注※3

Holder クラスを注※1 を除く用途に使用して SOAP アプリケーションを開発し，DII を使用して実行した場合が該当します。

3.2.6 リモートインタフェースおよびユーザ定義のデータ型クラスでの配列の使用

リモートインタフェースのメソッドのパラメタ，戻り値，またはこれらに指定するユーザ定義のデータ型クラス内で配列を使用できます。

配列を使用した場合，Java2WSDL コマンドおよび WSDL2Java コマンド実行時に，"SequenceOf"で始まるデータ型クラスが生成されます。生成例を次に示します。

- リモートインタフェース

```
package localhost;
public interface ArrayBank extends java.rmi.Remote{
    public void sentArrayData( java.lang.String[] arrayData ) throws java.rmi.RemoteExcep
tion;
}
```

- リモートインタフェースから，Java2WSDL コマンドによって生成された WSDL の一部

```
<wsdl:definitions xmlns:intf="http://localhost" targetNamespace="http://localhost" ... >
...
 <wsdl:types>
  <schema targetNamespace="http://localhost" xmlns="http://www.w3.org/2001/XMLSchema">
```

```
   <complexType name="SequenceOf_xsd_string">
    <sequence>
     <element maxOccurs="unbounded" minOccurs="0" name="item" type="xsd:string"/>
    </sequence>
   </complexType>
  </schema>
 </wsdl:types>
 <wsdl:message name="sentArrayDataRequest">
  <wsdl:part name="in0" type="intf:SequenceOf_xsd_string"/>
 </wsdl:message>
 <wsdl:message name="sentArrayDataResponse">
 </wsdl:message>
 <wsdl:portType name="ArrayBank">
  <wsdl:operation name="sentArrayData" parameterOrder="in0">
   <wsdl:input message="intf:sentArrayDataRequest" name="sentArrayDataRequest"/>
   <wsdl:output message="intf:sentArrayDataResponse" name="sentArrayDataResponse"/>
  </wsdl:operation>
 </wsdl:portType>
...
</wsdl:definitions>
```

- 上記の WSDL から，WSDL2Java コマンドによって生成されたリモートインタフェース

```
package localhost;

public interface ArrayBank extends java.rmi.Remote {
    public void sentArrayData(localhost.SequenceOf_xsd_string in0) throws java.rmi.Remote
Exception;
}
```

- localhost.SequenceOf_xsd_string クラスの一部

```
package localhost;

public class SequenceOf_xsd_string  implements java.io.Serializable {
    private java.lang.String[] item;

    public SequenceOf_xsd_string() {
    }

    public SequenceOf_xsd_string(
           java.lang.String[] item) {
           this.item = item;
    }

    public java.lang.String[] getItem() {
        return item;
    }

    public void setItem(java.lang.String[] item) {
        this.item = item;
    }
...
}
```

例えば，sentArrayData メソッドに対して，{ "data1", "data2" }という配列を与える場合，次のように SequenceOf_xsd_string インスタンスを生成して呼び出します。

```
ArrayBankServiceLocator locator = new ArrayBankServiceLocator();
ArrayBank port = locator.getArrayBank();
SequenceOf_xsd_string in0 = new SequenceOf_xsd_string();
in0.setItem( new String[]{ "data1", "data2" } );
port.sentArrayData( in0 );
```

なお，配列を使用する場合，その配列が空である（要素数が 0 である）ことと，その配列が null であることを区別することはできません。配列が空である場合も，配列が null であるものとして扱われます。

また，例えば，sentArrayData メソッドの第 1 引数を in0 として，次のように生成した SequenceOf_xsd_string インスタンスをクライアント側の実装で与えた場合，SOAP サービス側の実装のメソッド内で第 1 引数から取得できる in0 に対し，getItem メソッドを呼び出した結果はどちらも null です。

- 空の配列を与える

```
SequenceOf_xsd_string in0 = new SequenceOf_xsd_string();
in0.setItem( new String[]{} );
```

- null を与える

```
SequenceOf_xsd_string in0 = new SequenceOf_xsd_string();
in0.setItem( null );
```

3.2.7　既存の Java クラスおよび EJB の利用

既存の Java クラスおよび既存の EJB を利用して開発する場合，利用する次のプログラムが 3.2.2～3.2.6 に記載された規則に従っているか確認してください。

- Java クラス（Java クラスを利用して開発する場合）
- Bean クラス（EJB を利用して開発する場合）
- ホームインタフェース（EJB を利用して開発する場合）
- Java クラスまたは EJB から参照するリモートインタフェースおよびユーザ定義のデータ型クラス

なお，EJB を利用して開発する場合は，Holder クラスは使用できません。

3.3 WSDL の生成と定義

RPC 形態の SOAP アプリケーションの開発で必要な WSDL は，Java2WSDL コマンドで生成するか，手動で定義します。

ここでは，WSDL を生成する場合および WSDL を定義する場合に，留意が必要な点について説明します。

3.3.1 サービスロケーションの指定

Java2WSDL コマンドで WSDL を生成するときに，サービスロケーションを指定します（service 要素の address location 属性）。サービスロケーションの指定例を示します。

http://localhost:8080/RPCSampleService/services/UserInfo

指定例を基に，サービスロケーションの指定内容を示します。

http
: 通信プロトコルを指定します。「http」または「https」を指定します。

localhost:8080
: サービスが存在するホスト名（または IP アドレス），およびポート番号を指定します。半角英数字および次の記号を使用できます。

  - _ . ~ @ : % ! $ & ' ( ) * + , ; = [ ]

RPCSampleService
: 呼び出す SOAP サービスのコンテキストルート名を指定します。半角英数字および次の記号を使用できます。

  - _ . ~ /

services
: SOAP エンジンが提供するサーブレットパス名を指定します。半角英数字および次の記号を使用できます。

  - _ . ~ /

UserInfo
: 呼び出すサービスの名称を指定します。半角英数字および次の記号を使用できます。

  - _ . ~ /

3.3.2 WSDL のスタイルの指定

Java2WSDL コマンドで WSDL を生成するときに，WSDL のスタイルを指定します。WSDL のスタイルは「RPC」または「DOCUMENT」を指定します。

RPC

SOAP 1.1 仕様の Section 7.1 に従い，RPC の操作名と同じ名前を持つ要素で送信するデータ（パラメタまたは戻り値）をラップし，SOAP ボディ要素の子要素として送信する形態です。

DOCUMENT

送信するデータを SOAP ボディ要素の子要素としてそのまま送信する形態です。使用する XML ドキュメントは，エンコードスタイルまたは具体的なスキーマで定義されたルールに従っている必要があります。

また，WSDL のスタイルの指定に合わせて use 属性を指定します。use 属性は，WSDL の message 要素の part 要素がエンコード規則（例：SOAP encoding）を使用してエンコードされているかどうか，または part 要素が message 要素の具体的スキーマを定義しているかどうかを表します。use 属性は，「LITERAL」または「ENCODED」を指定します。

LITERAL

スキーマを使用する場合に指定します。

ENCODED

エンコード規則を使用する場合に指定します。

WSDL のスタイルおよび use 属性は，Java2WSDL コマンドのオプションで指定します。オプションの指定値の組み合わせとサポート範囲を次の表に示します。

表 3-5　Java2WSDL コマンドのオプション指定値の組み合わせとサポート範囲

スタイルの指定値	use 属性の指定値	サポート
RPC	LITERAL	○（デフォルト値）
RPC	ENCODED	○
DOCUMENT	LITERAL	○
DOCUMENT	ENCODED	×

（凡例）

○：サポートされます。

×：サポートされません。

指定できるオプションの指定値の組み合わせ，特徴，および WS-I Basic Profile の推奨パターンを次の表に示します。

表 3-6　Java2WSDL コマンドのオプション指定値の組み合わせ，特徴，WS-I Basic Profile の推奨パターン

指定値の組み合わせ	特徴	WS-I Basic Profile の推奨パターン
RPC／LITERAL （デフォルト値）	SOAP メッセージ内に型情報が含まれません。指定値を省略した場合に設定されるデフォルト値です。	○

指定値の組み合わせ	特徴	WS-I Basic Profile の推奨パターン
RPC／ENCODED	SOAP メッセージ内に型情報が含まれるため，メソッド引数が多くなると，メッセージのサイズが大きくなります。	×
DOCUMENT／LITERAL	SOAP メッセージ内に型情報が含まれません。	○

（凡例）

○：WS-I Basic Profile で推奨されている組み合わせです。

×：WS-I Basic Profile で推奨されていない組み合わせです。

WSDL のスタイルおよび use 属性の指定方法については，「9.1　Java2WSDL コマンド（WSDL の生成）」を参照してください。

3.3.3　生成される WSDL の例

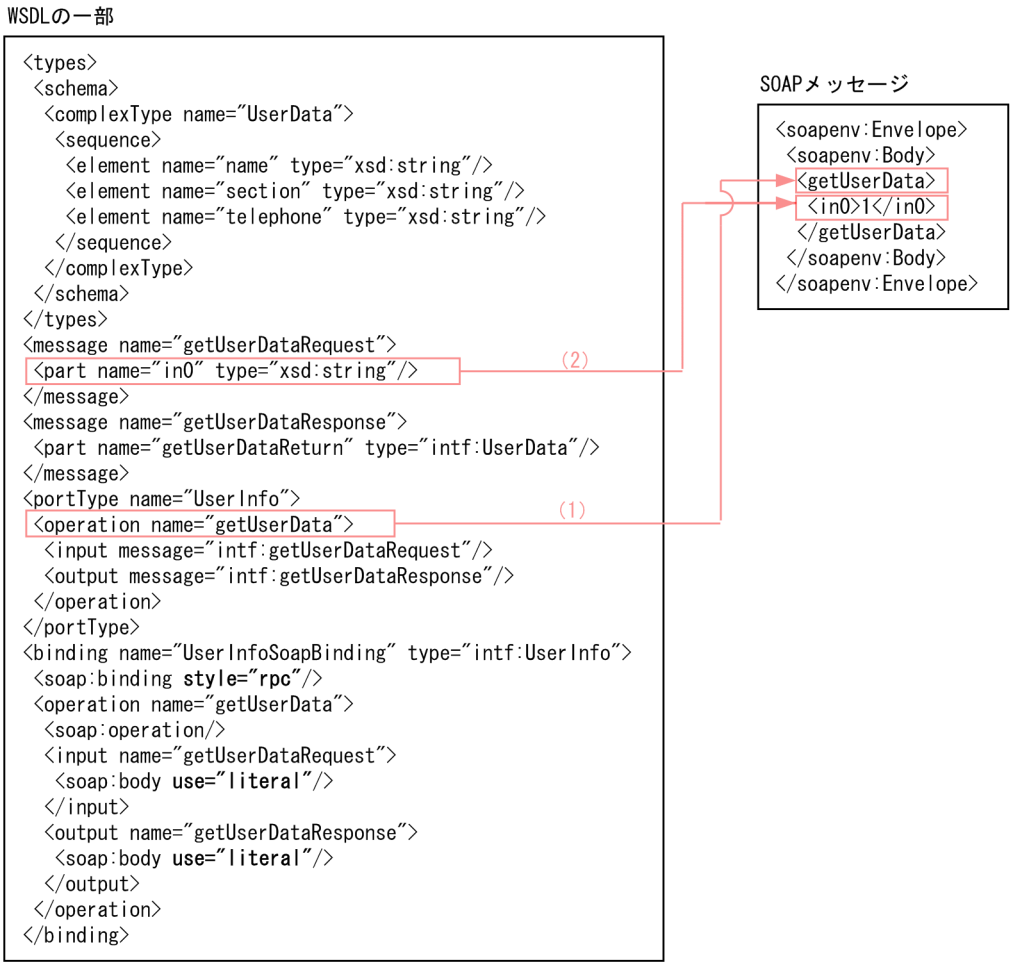
次の Java メソッドに対して，Java2WSDL コマンドを実行したときに生成される WSDL の例を示します。

```
public UserData getUserData(String user_no);
```

(1)　RPC／LITERAL を指定した場合

WSDL のスタイルで「RPC」，use 属性で「LITERAL」を指定した場合に生成される WSDL の一部と SOAP メッセージの例を示します。

図 3-6　RPC／LITERAL を指定した場合の WSDL の生成例



WSDL の operation 要素の name 属性値が，SOAP メッセージのボディ直下の要素に対応します（図の(1)）。WSDL の part 要素が，SOAP メッセージのオペレーション名を表す要素（<getUserData>）の子要素に対応します（図の（2））。

また，WSDL の binding 要素以下の style 属性が"rpc"，use 属性が"literal"になります。

(2)　RPC／ENCODED を指定した場合

WSDL のスタイルで「RPC」，use 属性で「ENCODED」を指定した場合に生成される WSDL の一部と SOAP メッセージの例を示します。

図 3-7　RPC／ENCODED を指定した場合の WSDL の生成例

WSDLの一部

```
<types>
  <schema>
   <complexType name="UserData">
    <sequence>
     <element name="name" type="xsd:string"/>
     <element name="section" type="xsd:string"/>
     <element name="telephone" type="xsd:string"/>
    </sequence>
   </complexType>
  </schema>
</types>
<message name="getUserDataRequest">
 <part name="in0" type="xsd:string"/>
</message>
<message name="getUserDataResponse">
 <part name="getUserDataReturn" type="intf:UserData"/>
</message>
<portType name="UserInfo">
 <operation name="getUserData">
  <input message="intf:getUserDataRequest"/>
  <output message="intf:getUserDataResponse"/>
 </operation>
</portType>
<binding name="UserInfoSoapBinding" type="intf:UserInfo">
 <soap:binding style="rpc"/>
 <operation name="getUserData">
  <soap:operation/>
  <input name="getUserDataRequest">
   <soap:body encodingStyle="http://schemas.xmlsoap.org/
soap/encoding/" use="encoded"/>
  </input>
  <output name="getUserDataResponse">
   <soap:body encodingStyle="http://schemas.xmlsoap.org/
soap/encoding/" use="encoded"/>
  </output>
 </operation>
</binding>
```

(2)

(1)

SOAPメッセージ

```
<soapenv:Envelope>
 <soapenv:Body>
  <getUserData>
   <in0 xsi:type="xsd:
string">1</in0>
  </getUserData>
 </soapenv:Body>
</soapenv:Envelope>
```

WSDL の operation 要素の name 属性値が，SOAP メッセージのボディ直下の要素に対応します（図の(1)）。WSDL の part 要素が，SOAP メッセージのオペレーション名を表す要素（<getUserData>）の子要素に対応します（図の（2））。SOAP メッセージにはエンコード情報（xsi:type="xsd:string"）が含まれます。

また，WSDL の binding 要素以下の style 属性が"rpc"，use 属性が"encoded"になります。

(3)　DOCUMENT／LITERAL を指定した場合

WSDL のスタイルで「DOCUMENT」，use 属性で「LITERAL」を指定した場合に生成される WSDL の一部と SOAP メッセージの例を示します。

図 3-8　DOCUMENT／LITERAL を指定した場合の WSDL の生成例

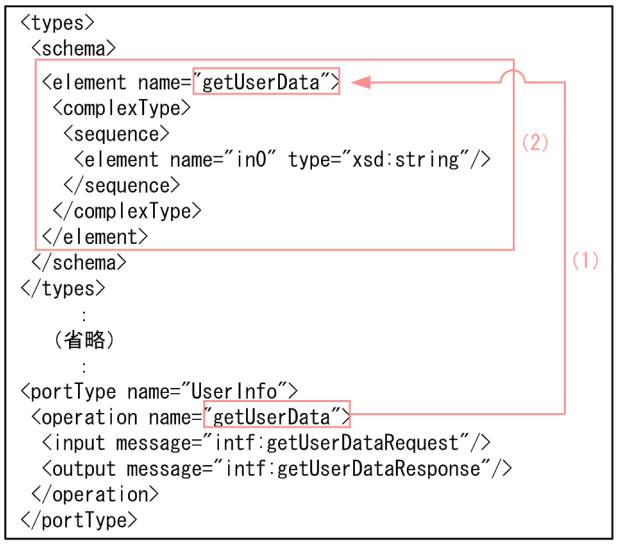
WSDL の part 要素の element 属性で参照している要素（図の（1））が，SOAP メッセージのボディの子要素に対応します（図の（2））。

また，WSDL の binding 要素以下の style 属性が"document"，use 属性が"literal"になります。

wrapped 形式の WSDL

Java2WSDL コマンド実行時に，DOCUMENT／LITERAL を指定して生成された WSDL では，オペレーションの引数が operation 要素の name 属性の値から命名された element 要素（次の図の（1））でラップされます（次の図の（2））。このような形式の WSDL を wrapped 形式と呼びます。

図 3-9　wrapped 形式の WSDL の例



nonwrapped 形式の WSDL

wrapped 形式でない DOCUMENT スタイルの WSDL を nonwrapped 形式と呼びます。

3.3.4　WSDL の構文

WSDL の要素の記述規則，および記述例を示します。WSDL を定義する場合は，この項に記載された内容，および「12.2　WSDL 1.1 との対応」に記載された内容に注意してください。

(1)　XML Schema の URI

XML Schema の URI は「http://www.w3.org/2001/XMLSchema」を使用します。対象となる XML Schema の URI は，WSDL（インポート，インクルードされたファイルを含む）に記述されたすべての URI となります。

(2)　wsdl:types 要素

wsdl:types 要素は，SOAP メッセージで使用する型に関する情報を定義する要素です。wsdl:types 要素は，次に示す規則に従って記述してください。

- wsdl:definitions 要素の子要素として記述してください。
- 記述できる個数は 0 個または 1 個です。2 個以上は記述できません。
- wsdl:documentation 要素および wsdl:import 要素よりもあとに記述してください。
- wsdl:documentation 要素および wsdl:import 要素を除く，ほかのすべての要素よりも前に記述してください。

(3) xsd:schema 要素

xsd:schema 要素は，XML Schema を記述する要素です。xsd:schema 要素は，wsdl:types 要素の子要素として記述してください。

xsd:schema 要素の子要素には，xsd:import 要素および xsd:include 要素を定義できます。

- xsd:import 要素

  xsd:import 要素は，XML Schema をインポートする場合に定義する要素です。xsd:import 要素の書式や構文については，「3.3.6　XML Schema のインポート」を参照してください。

- xsd:include 要素

  xsd:include 要素は，XML Schema をインクルードする場合に定義する要素です。xsd:include 要素の書式や構文については，「3.3.7　XML Schema のインクルード」を参照してください。

(4) wsdl:import 要素

wsdl:import 要素は，WSDL をインポートする場合に定義する要素です。wsdl:import 要素の書式や構文については，「3.3.8　WSDL のインポート」を参照してください。

(5) wsdl:part 要素

wsdl:part 要素は，SOAP メッセージの内容を構成する要素です。wsdl:part 要素は，次に示す規則に従って記述してください。

- IN パラメタ，OUT パラメタ，および INOUT パラメタの wsdl:part 要素を合計した数は 254 個までです。255 個以上は記述できません。

(6) 拡張要素

拡張要素（extensibility element）は次に示す要素にだけ記述できます。

- soap:binding 要素
- soap:operation 要素
- soap:body 要素
- soap:fault 要素
- soap:address 要素

(7) soap:binding 要素

soap:binding 要素は，SOAP バインディングを定義する要素です。soap:binding 要素は，次に示す規則に従って記述してください。

- wsdl:binding 要素の子要素として記述してください。

- 必ず 1 個記述してください。2 個以上は記述できません。
- transport 属性には HTTP バインディングを示す"http://schemas.xmlsoap.org/soap/http"を指定してください。

(8) wsdl:operation 要素

wsdl:portType 要素の子要素である wsdl:operation 要素と，wsdl:binding 要素の子要素である wsdl:operation 要素は，一対一になるように記述してください。

(9) soap:operation 要素

soap:operation 要素は，SOAP バインディングでのオペレーションの情報を定義する要素です。soap:operation 要素は，次に示す規則に従って記述してください。

- wsdl:binding 要素の子要素である wsdl:operation 要素の子要素として記述してください。
- 必ず 1 個記述してください。2 個以上は記述できません。

(10) soap:body 要素

soap:body 要素は，SOAP メッセージの soap:body 要素以下のメッセージを定義する要素です。soap:body 要素は，次に示す規則に従って記述してください。

- wsdl:binding 要素の子要素である wsdl:input 要素または wsdl:output 要素の子要素として記述してください。
- 必ず 1 個記述してください。2 個以上は記述できません。

なお，parts 属性を記述することはできますが，無視されます。

(11) wsdl:fault 要素

wsdl:fault 要素は fault を定義する要素です。wsdl:fault 要素は，次に示す規則に従って記述してください。

- wsdl:portType 要素および wsdl:binding 要素の子要素として記述してください。
- 同じ name 属性の値を持つ wsdl:fault 要素を複数記述しないでください。

(12) soap:fault 要素

soap:fault 要素は，SOAP メッセージの soap:fault 要素に含まれる detail 子要素以下のメッセージを定義する要素です。soap:fault 要素は，次に示す規則に従って記述してください。

- wsdl:binding 要素の子要素である wsdl:fault 要素の子要素として記述してください。
- 必ず 1 個記述してください。2 個以上は記述できません。

(13)　wsdl:service 要素

wsdl:service 要素は，SOAP サービスを定義する要素です。一つの WSDL に複数の SOAP サービスを対応させることはできません。wsdl:service 要素は，次に示す規則に従って記述してください。

- 必ず 1 個記述してください。2 個以上は記述できません。

3.3.5　WSDL 定義で使用できる文字

WSDL 定義では，RFC3986 で規定されている非予約文字と日本語を使用できます。ここでは，WSDL 定義で使用できる文字，および使用する文字に関する注意事項を示します。

WSDL ファイル名で使用できない文字については，「3.3.9(3)　WSDL ファイル名に使用できない文字」を参照してください。

(1)　使用できる文字

document/literal に対応した SOAP アプリケーションでは，WSDL 定義に日本語を使用できます。WSDL のスタイルごとに，使用できる文字を示します。

(a)　「DOCUMENT／LITERAL」の場合

半角英数字（A～Z，a～z，0～9），アンダースコア（_），および次に示す文字を使用できます。

- 全角ひらがな，全角かたかな
- 全角ギリシャ文字，全角ロシア文字など
- 繰り返し符号（「ゝ」，「ゞ」，「々」）
- 漢字（JIS X 0208 の第一水準，第二水準に含まれる範囲）

WSDL のデータ型が anyURI の属性については，次の文字も使用できます。

- ハイフン（-）
- ピリオド（.）
- チルダ（~）

日本語のサポートについては，「12.2.1(2)　WSDL 1.1 仕様での日本語のサポート範囲」を参照してください。なお，DII を使用する場合，日本語は使用できません。

(b)　「RPC／LITERAL」または「RPC／ENCODED」の場合

半角英数字（A～Z，a～z，0～9）およびアンダースコア（_）を使用できます。WSDL のデータ型が anyURI の属性については，次の文字も使用できます。

- ハイフン（-）

- ピリオド（.）
- チルダ（~）

(2)　日本語で定義する場合の注意

WSDL 定義に日本語を使用する場合の注意事項について説明します。

(a)　wsdl:port 要素の name 属性を日本語で定義する場合

wsdl:port 要素の name 属性の値と，soap:address 要素の location 属性に指定された URL パスの末尾の値は一致させる必要があります。ただし，wsdl:port 要素の name 属性を日本語で定義する場合，同じ値をパーセントエンコードして soap:address 要素の location 属性の URL パスに定義する必要があります（クライアントアプリケーションで動的に URL を指定する場合は除きます）。

パーセントエンコードするときに使用できる符号化形式は UTF-8 です。また，パーセントエンコードした文字の 16 進コード（%xx）の A から F は，大文字を使用してください。

次に，wsdl:port 要素の name 属性を日本語で定義する場合の例を示します。

```
...
<wsdl:service name="UserInfoService">
 <wsdl:port binding="intf:UserInfoSoapBinding" name="日本語">
   <soap:address location="http://localhost:8080/RPCSampleService/services/%E6%97%A5%E6%9C%AC%E8%AA%9E"/>
  </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

soap:address 要素の location 属性に誤った値を定義した場合，エラーとなり通信できません。

(b)　anyURI の属性に対して日本語で定義する場合

WSDL のデータ型が anyURI の属性に対して，日本語で値を定義する場合は，パーセントエンコードする必要があります。

パーセントエンコードするときに使用できる符号化形式は UTF-8 です。また，パーセントエンコードした文字の 16 進コード（%xx）の A から F は，大文字を使用してください。

anyURI の属性を持つ WSDL の要素については，「12.2.1(2)　WSDL 1.1 仕様での日本語のサポート範囲」を参照してください。

(3)　WSDL 定義での記号の扱い

WSDL 定義で次の記号を名称に使用している場合，ソースコード生成時に記号が削除されます。そのため，記号を使用して名称を重複しないようにしている場合でも，ソースコード生成時には名称が重複して，コンパイルエラーになることがあります。

ソースコード生成時に削除される記号

- ハイフン（-）
- ピリオド（.）
- コロン（:）

（例）

WSDL 定義での名称：Section-Name

ソースコードでの名称：SectionName

(4) 先頭文字についての注意

WSDL 定義で使用する名称の先頭文字は「a～z」，「A～Z」，または「_」（アンダースコア）で始まるように定義してください。それ以外の名称を先頭文字に使用すると，サービス呼び出し時にエラー（KDCCF0006-E）が発生することがあります。

(5) 同一データ型の変数名の定義に関する注意

WSDL 定義でデータ型を定義する場合，同一データ型内の変数名として，"A"と"a"または"A1"と"a1"のように，大文字小文字だけが異なる変数名を定義すると，ソースコード生成後のコンパイルでエラーになることがあります。

大文字小文字だけが異なるような変数名は定義しないでください。

(6) Java の予約語を定義した場合の注意

WSDL 定義で使用する名称に，Java の予約語を定義すると，ソースコード生成時に名称の前に「_」（アンダースコア）が付加されます。

名称の前に「_」（アンダースコア）が付加される Java の予約語を次に示します。

```
”abstract”,  ”assert”,     ”boolean”,   ”break”,        ”byte”,
”case”,      ”catch”,      ”char”,      ”class”,        ”const”,
”continue”,  ”default”,    ”do”,        ”double”,       ”else”,
”enum”,      ”extends”,    ”false”,     ”final”,        ”finally”,
”float”,     ”for”,        ”goto”,      ”if”,           ”implements”,
”import”,    ”instanceof”, ”int”,       ”interface”,    ”long”,
”native”,    ”new”,        ”null”,      ”package”,      ”private”,
”protected”, ”public”,     ”return”,    ”short”,        ”static”,
”strictfp”,  ”super”,      ”switch”,    ”synchronized”, ”this”,
”throw”,     ”throws”,     ”transient”, ”true”,         ”try”,
”void”,      ”volatile”,   ”while”
```

（例）

WSDL 定義での名称：boolean

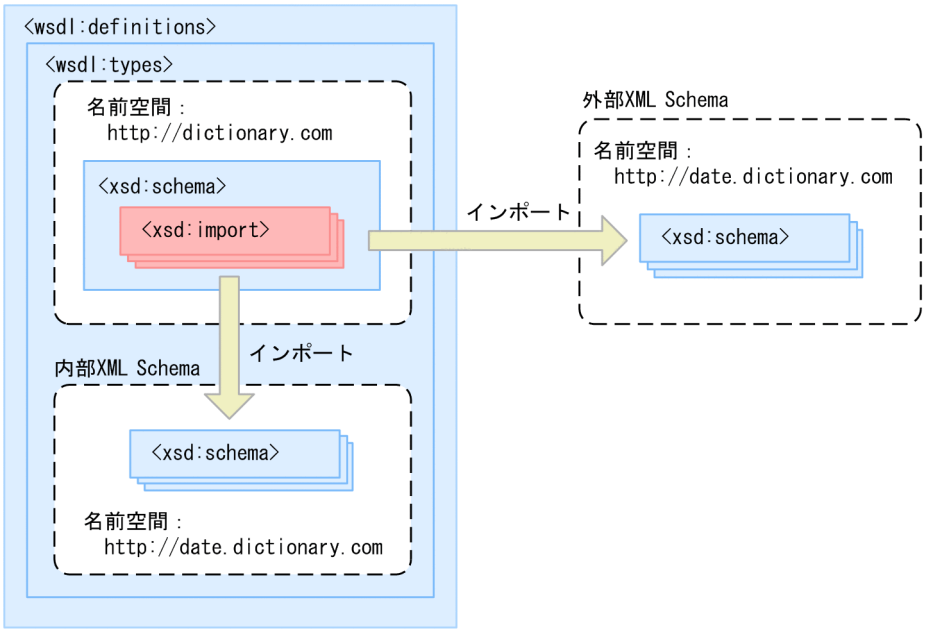
ソースコードでの名称：_boolean

3.3.6　XML Schema のインポート

xsd:import 要素を定義することで，同じ WSDL 内の XML Schema，および別ファイルの XML Schema をインポートできます。XML Schema のインポートでは，異なる名前空間に属する XML Schema をインポートできます。

XML Schema のインポートの概念を次の図に示します。

図 3-10　XML Schema のインポート



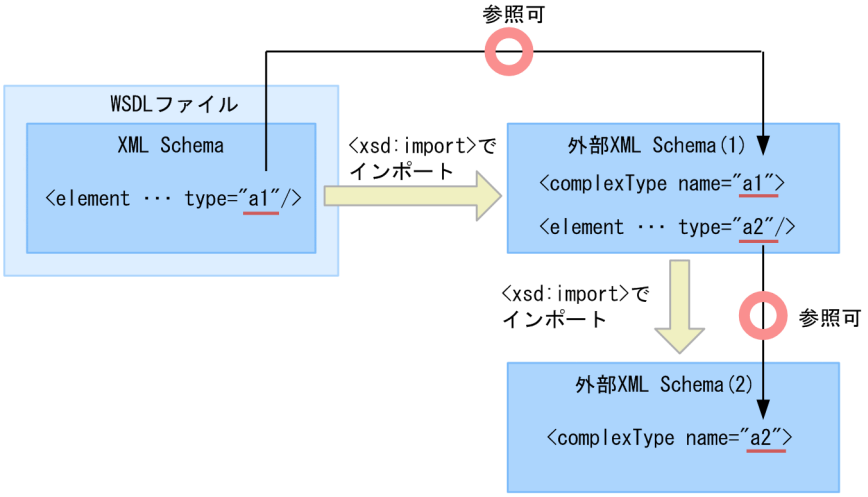
なお，名前空間「http://www.w3.org/2001/XMLSchema」はインポートしないでください。

(1)　xsd:import 要素の有効範囲

XML Schema をインポートすることで，インポート対象の XML Schema の要素および属性を参照できます。XML Schema の型定義を参照する場合，同じファイル内に参照する型定義がなくても，型定義を記述した XML Schema のファイルをインポートすれば参照できます。

XML Schema の要素および属性を参照できる場合の例を示します。

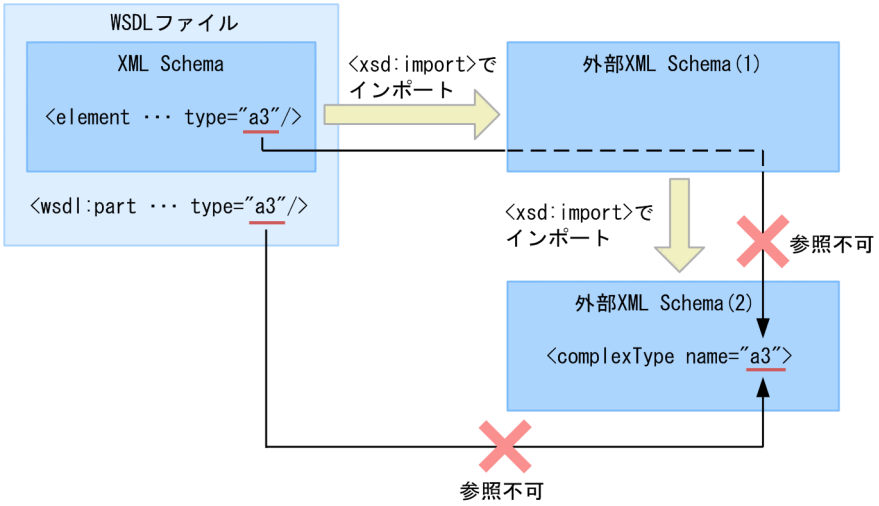
図 3-11　XML Schema の要素および属性を参照できる場合（インポート）



対象となる XML Schema をインポートしない場合は，ほかの名前空間に属する XML Schema の要素および属性を参照できません。また，別のファイルを介して，間接的に XML Schema の要素および属性を参照することはできません。

XML Schema の要素および属性を参照できない場合の例を示します。

図 3-12　XML Schema の要素および属性を参照できない場合（インポート）



(2)　xsd:import 要素の書式

xsd:import 要素の書式と属性を示します。

書式

```
<xsd:import namespace="インポートする名前空間のURI"
            [schemaLocation="インポート対象のXML Schemaのファイル名"]/>
```

注　[　]は定義を省略できることを表します。

属性

- namespace

  インポート対象の XML Schema が属する名前空間の URI を定義します。この属性の定義は必須です。

- schemaLocation

  インポート対象の XML Schema のファイル名を定義します。同じ WSDL ファイル内の XML Schema をインポートする場合，および次の URI をインポートする場合は，この属性の定義を省略します。

  ・http://schemas.xmlsoap.org/soap/encoding/

  ・http://www.w3.org/2003/05/soap-encoding

  ・http://xml.apache.org/xml-soap

  ・http://ws-i.org/profiles/basic/1.1/xsd

  schemaLocation 属性に定義できるのは相対パスだけです。したがって，絶対パスや，http://やftp://などのプロトコルで始まる URI は定義できません。また，相対パスにショートカットを含めることはできません。

相対パスに指定する文字に関する注意

- 相対パスに指定する文字の使用可否を次の表に示します。次の表にない文字を使用する場合は，パーセントエンコードする必要があります。

  表 3-7　相対パスに指定する文字の使用可否

分類	文字
そのまま使用できる文字 （指定時にパーセントエンコード不要）	半角英数字（A～Z，a～z，0～9）および次の文字 - . _ ~ /
使用できない文字 （パーセントエンコードしても指定不可）	半角スペースおよび次の文字 ? : * % | < > " [ ] & +

- 「#」はパーセントエンコードすると指定できます。ただし，ファイル名の先頭には指定できません。
- 相対パスに指定した文字は，大文字と小文字は区別されません。
- パーセントエンコードした文字の 16 進コード（%xx）の A から F は，大文字を使用してください。
- パーセントエンコードする場合，正しくパーセントエンコードされているか確認してください。不完全な文字列は指定できません。

(3)　xsd:import 要素の構文

xsd:import 要素の構文，および定義時の注意事項を示します。

(a)　xsd:import 要素の記述位置

xsd:import 要素は，次に示す規則に従って記述してください。

- xsd:schema 要素の子要素として記述してください。
- xsd:annotation 要素よりもあとに記述してください。
- xsd:annotation 要素および xsd:include 要素を除く，ほかのすべての要素よりも前に記述してください。

(b) インポート対象の XML Schema の定義

インポート対象の XML Schema では，ルート要素を xsd:schema 要素にしてください。また，インポート対象の xsd:schema 要素の targetNamespace 属性は，次に示す規則に従って記述してください。

- インポート元の xsd:import 要素の namespace 属性と同じ値を定義してください。
- インポート元の targetNamespace 属性と異なる値を定義してください。

(c) 複数の XML Schema のインポート

xsd:schema の子要素に xsd:import 要素を複数行記述することで，複数の XML Schema をインポートできます。

schemaLocation 属性の値が異なれば，namespace 属性の値がそれぞれ同じ場合でも，異なる場合でもインポートできます。schemaLocation 属性の値が同じ場合は，どちらもインポートできません（schemaLocation 属性の値が同じ xsd:import 要素は複数行記述できません）。schemaLocation 属性を省略する場合は，namespace 属性の値が同じ xsd:import 要素を複数行記述できません。

(4) xsd:import 要素の定義例

別ファイルの XML Schema をインポートする場合，および同じ WSDL 内の XML Schema をインポートする場合の定義例を示します。

(a) 別ファイルの XML Schema をインポートする場合

- DataInfo.wsdl（インポート元）

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://dictionary.com"
 xmlns:intf="http://dictionary.com"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <wsdl:types>
    <xsd:schema targetNamespace="http://dictionary.com">

      <xsd:import namespace="http://date.dictionary.com" schemaLocation="Date.xsd"/>
      ...
    </xsd:schema>
  </wsdl:types>
  ...
</wsdl:definitions>
```

- Date.xsd（インポート対象）

```
<?xml version="1.0" ?>
<xsd:schema targetNamespace="http://date.dictionary.com"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="DateInfo">
    <xsd:sequence>
      <xsd:element name="fromDate" nillable="true" type="xsd:string"/>
      <xsd:element name="toDate" nillable="true" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

(b)　同じ WSDL 内の XML Schema をインポートする場合

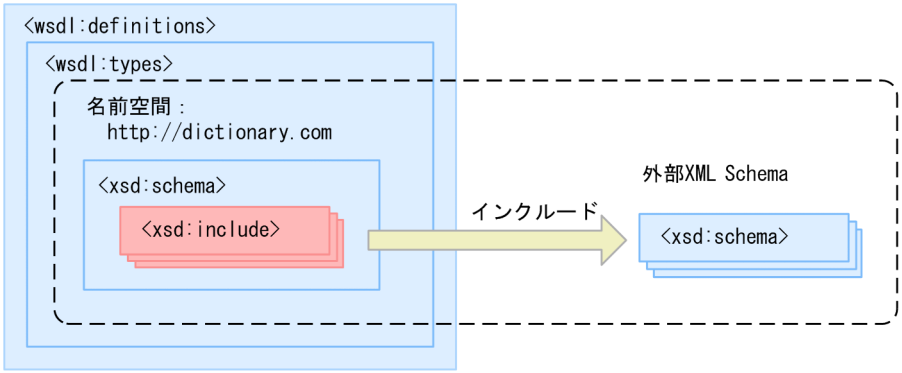
```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://dictionary.com"
 xmlns:intf="http://dictionary.com"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <wsdl:types>
    <xsd:schema targetNamespace="http://dictionary.com"
     xmlns:tns2="http://date.dictionary.com"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema">

      <xsd:import namespace="http://date.dictionary.com"/>

      <xsd:complexType name="Data">
        <xsd:sequence>
          <xsd:element name="date" nillable="true" type="tns2:DateInfo"/>
          <xsd:element name="title" nillable="true" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>

    <xsd:schema targetNamespace="http://date.dictionary.com"
     xmlns="http://www.w3.org/2001/XMLSchema">
      <xsd:complexType name="DateInfo">
        <xsd:sequence>
          <xsd:element name="fromDate" nillable="true" type="xsd:string"/>
          <xsd:element name="toDate" nillable="true" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </wsdl:types>
  ...
</wsdl:definitions>
```

3.3.7　XML Schema のインクルード

xsd:include 要素を定義することで，同じ名前空間に属する XML Schema をインクルードできます。

XML Schema のインクルードの概念を次の図に示します。

図 3-13　XML Schema のインクルード

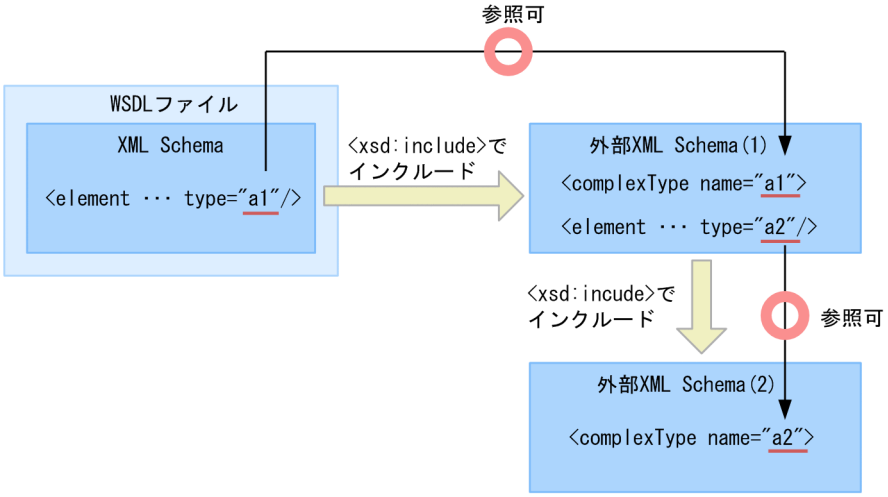


(1)　xsd:include 要素の有効範囲

XML Schema をインクルードすることで，インクルード対象の XML Schema の要素および属性を参照できます。XML Schema の型定義を参照する場合，同じファイル内に参照する型定義がなくても，型定義を記述した XML Schema のファイルをインクルードすれば参照できます。

XML Schema の要素および属性を参照できる場合の例を示します。

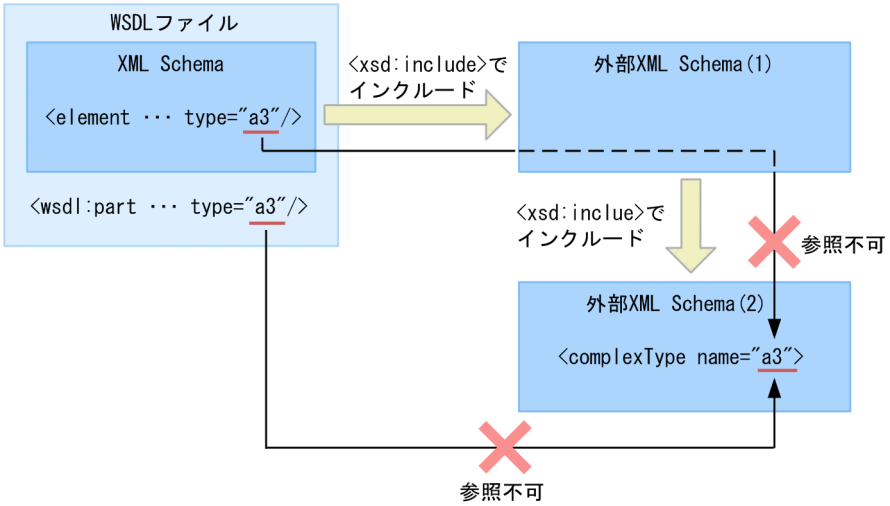
図 3-14　XML Schema の要素および属性を参照できる場合（インクルード）



対象となる XML Schema をインクルードしない場合は，別のファイルに記述された XML Schema の要素および属性を参照できません。また，別のファイルを介して，間接的に XML Schema の要素および属性を参照することはできません。

XML Schema の要素および属性を参照できない場合の例を示します。

図 3-15　XML Schema の要素および属性を参照できない場合（インクルード）



(2)　xsd:include 要素の書式

xsd:include 要素の書式と属性を示します。

書式

```
<xsd:include schemaLocation="インクルード対象のXML Schemaのファイル名"/>
```

属性

- schemaLocation

  インクルード対象の XML Schema のファイル名を定義します。この属性の定義は必須です。

  schemaLocation 属性に定義できるのは相対パスだけです。したがって，絶対パスや，http://や ftp://などのプロトコルで始まる URI は定義できません。また，相対パスにショートカットを含めることはできません。

  相対パスを記述するときは，「3.3.6(2)　xsd:import 要素の書式」に記載された「相対パスに指定する文字に関する注意」の内容に注意してください。

(3)　xsd:include 要素の構文

xsd:include 要素の構文，および定義時の注意事項を示します。

(a)　xsd:include 要素の記述位置

xsd:include 要素は，次に示す規則に従って記述してください。

- xsd:schema 要素の子要素として記述してください。
- xsd:annotation 要素よりもあとに記述してください。
- xsd:annotation 要素および xsd:import 要素を除く，ほかのすべての要素よりも前に記述してください。

(b) インクルード対象の XML Schema の定義

インクルード対象の XML Schema では，ルート要素を xsd:schema 要素にしてください。また，インクルード対象の xsd:schema 要素の targetNamespace 属性には，インクルード元の targetNamespace 属性と同じ値を定義してください。

(c) 複数の XML Schema のインクルード

xsd:schema の子要素に xsd:include 要素を複数行記述することで，複数の XML Schema をインクルードできます。ただし，ファイル名（schemaLocation 属性の値）が重複した場合はインクルードできません。

(4) xsd:include 要素の定義例

XML Schema をインクルードする場合の定義例を示します。

- DataInfo.wsdl（インクルード元）

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://dictionary.com"
 xmlns:intf="http://dictionary.com"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <wsdl:types>
    <xsd:schema targetNamespace="http://dictionary.com">

      ...
      <xsd:include schemaLocation="Data.xsd"/>
      ...

    </xsd:schema>
  </wsdl:types>

  ...

</wsdl:definitions>
```

- Data.xsd（インクルード対象）

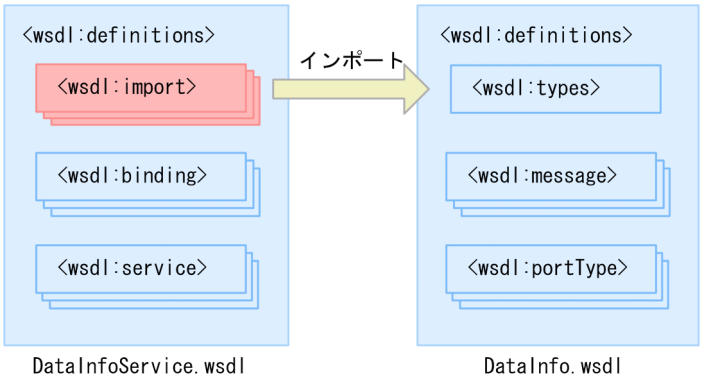
```
<?xml version="1.0" ?>
<xsd:schema targetNamespace="http://dictionary.com"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="Data">
    <xsd:sequence>
      <xsd:element name="date" nillable="true" type="xsd:string"/>
      <xsd:element name="title" nillable="true" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

3.3.8　WSDL のインポート

wsdl:import 要素を定義することで，ほかの WSDL ファイルをインポートできます。インポートできるのは，インポート元とインポート対象とを合わせて 2 階層までです。インポート対象に，さらに wsdl:import 要素を定義することはできません。

WSDL のインポートの概念を次の図に示します。

図 3-16　WSDL のインポート



(1)　wsdl:import 要素の書式

wsdl:import 要素の書式と属性を示します。

書式

```
<wsdl:import namespace="インポート対象の名前空間のURI"
             location="インポート対象のWSDLのファイル名"/>
```

属性

- namespace

  インポート対象の名前空間の URI を定義します。この属性の定義は必須です。

- location

  インポート対象の WSDL のファイル名を定義します。この属性の定義は必須です。

  location 属性に定義できるのは相対パスだけです。したがって，絶対パスや，http://や ftp://などのプロトコルで始まる URI は定義できません。また，相対パスにショートカットを含めることはできません。

  相対パスを記述するときは，「3.3.6(2)　xsd:import 要素の書式」に記載された「相対パスに指定する文字に関する注意」の内容に注意してください。

(2)　wsdl:import 要素の構文

wsdl:import 要素の構文，および定義時の注意事項を示します。

(a) WSDL をインポートする場合の要素の組み合わせ

WSDL をインポートする場合，インポート元またはインポート対象のどちらかの WSDL に，wsdl:binding 要素を定義する必要があります。次の表に，WSDL をインポートする場合に定義が必要な要素の組み合わせを示します。

表 3-8 WSDL をインポートする場合の要素の組み合わせ

インポート元	インポート対象
wsdl:binding wsdl:service	wsdl:types※ wsdl:message wsdl:portType
wsdl:service	wsdl:types※ wsdl:message wsdl:portType wsdl:binding

注※

wsdl:types 要素を省略する場合は不要です。

(b) wsdl:import 要素の記述位置

wsdl:import 要素は，次に示す規則に従って記述してください。

- wsdl:definitions 要素の子要素として記述してください。
- wsdl:documentation 要素よりもあとに記述してください。
- wsdl:documentation 要素を除く，ほかのすべての要素よりも前に記述してください。

(c) インポート対象の WSDL の定義

インポート対象の WSDL では，ルート要素を wsdl:definitions 要素にしてください。また，インポート対象の wsdl:definitions 要素の targetNamespace 属性は，次に示す規則に従って記述してください。

- インポート元の wsdl:import 要素の namespace 属性と同じ値を定義してください。
- インポート元の targetNamespace 属性と異なる値を定義してください。

(d) 複数の WSDL のインポート

wsdl:definitions の子要素に wsdl:import 要素を複数行記述することで，複数の WSDL をインポートできます。ただし，インポート対象の WSDL で名前空間（namespace 属性の値）またはファイル名（location 属性の値）のどちらかが重複した場合は，インポートできません。

(3) wsdl:import 要素の定義例

WSDL をインポートする場合の定義例を示します。

- DataInfoService.wsdl（インポート元）

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://service.dictionary.com"
 xmlns:intf="http://dictionary.com"
 xmlns:svc="http://service.dictionary.com"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <wsdl:import namespace="http://dictionary.com" location="DataInfo.wsdl"/>

  <wsdl:binding name="DataInfoSoapBinding" type="intf:DataInfo">
    ...
  </wsdl:binding>

  <wsdl:service name="DataInfoService">
    ...
  </wsdl:service>
</wsdl:definitions>
```

- DataInfo.wsdl（インポート対象）

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://dictionary.com"
 xmlns:intf="http://dictionary.com"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <wsdl:types>
    ...
  </wsdl:types>

  <wsdl:message name="getDataResponse">
    ...
  </wsdl:message>

  <wsdl:portType name="DataInfo">
    ...
  </wsdl:portType>
</wsdl:definitions>
```

3.3.9　WSDL の生成時および定義時の注意事項

WSDL を生成および定義するときに注意が必要な内容について説明します。

(1)　ソースコード生成時の WSDL 検証機能の実行について

WSDL2Java コマンドを実行してソースコードを生成するときは，WSDL 検証機能を有効にしてください。WSDL 検証機能は，コマンド実行時に-C オプションを指定することで有効にできます。

WSDL 検証機能については，「9.2　WSDL2Java コマンド（ソースコードの生成）」を参照してください。

(2) Java インタフェースのメソッドのパラメタ名と生成される WSDL の対応

Java2WSDL コマンドを実行して Java インタフェースから WSDL を生成する場合，Java インタフェースのメソッドのパラメタ名は，WSDL に「inXX」（XX は整数）の名称でマッピングされます。

Java インタフェースおよび生成される WSDL の例を示します。

- Java インタフェース

```
package localhost;

public interface UserInfo extends java.rmi.Remote {
    public String getUserData(String user_no1, String user_no2) throws java.rmi.RemoteExc
eption;
}
```

- 生成される WSDL

```
...
 <wsdl:message name="getUserDataResponse">
 </wsdl:message>
 <wsdl:message name="getUserDataRequest">
  <wsdl:part name="in0" type="xsd:string"/>
  <wsdl:part name="in1" type="xsd:string"/>
 </wsdl:message>
 <wsdl:portType name="UserInfo">
  <wsdl:operation name="getUserData" parameterOrder="in0 in1">
   <wsdl:input message="intf:getUserDataRequest" name="getUserDataRequest"/>
   <wsdl:output message="intf:getUserDataResponse" name="getUserDataResponse"/>
  </wsdl:operation>
...
```

(3) WSDL ファイル名に使用できない文字

次に示す文字は，WSDL のファイル名に使用できません。

? : * # % | < > "

(4) WSDL を手動で定義する場合の注意

WSDL を手動で定義する場合は，Application Server でサポートしている WSDL 1.1 仕様の範囲で記述してください。Application Server でサポートしている WSDL 1.1 仕様の範囲については「12.2 WSDL 1.1 との対応」を参照してください。

(5) extension 要素を使用した場合のメッセージ形式について

次に示す例のように，多段的に extension base を使用した場合，多段的な complexType 要素の一部が上位要素の属性と見なした SOAP メッセージが送信されます。

- WSDL 定義

```
<complexType name="arrayList">
  <complexContent>
    <extension base="tns:list">
      <sequence />
    </extension>
  </complexContent>
</complexType>
<complexType name="list">
  <complexContent>
    <extension base="tns:collection">
      <sequence />
    </extension>
  </complexContent>
</complexType>
<complexType name="collection">
  <complexContent>
    <restriction base="soapenc:Array">
      <attribute ref="soapenc:arrayType"
                 wsdl:arrayType="anyType[]" />
    </restriction>
  </complexContent>
</complexType>
```

- SOAP メッセージ

```
<ns1:checkArrayList
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:ns1="http://testdata">
  <in xsi:type="ns1:ArrayList">
    <collection
        xsi:type="soapenc:Array"
        soapenc:arrayType="soapenc:string[2]"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
      <item xsi:type="xsd:string">testdata</item>
      <item xsi:type="xsd:int">1</item>
    </collection>
  </in>
</ns1:checkArrayList>
```

(6) 他社クライアントおよび他社サービスと通信する場合のデータ型に関する注意

他社クライアントまたは他社サービスと通信するための WSDL 定義を作成する場合，双方で扱えるデータ型を WSDL 定義に記述してください。

SOAP 通信基盤で扱えるデータ型については，「12.2.3　XML Schema のデータ型のサポート範囲」を参照してください。他社クライアントまたは他社サービスで扱えるデータ型については，各製品のマニュアルを参照してください。

(7) 派生データ型の定義に関する注意

データ型を派生させるために使用する，制限（restriction），ユニオン（union），およびリスト（list）の中で，SOAP 通信基盤で有効になるのは制限（restriction）の列挙（enumeration）だけです。WSDL 定義に派生データ型を定義する場合，または WSDL 定義からソースを生成する場合に，列挙（enumeration）以外は使用しないでください。列挙（enumeration）以外を使用した場合，指定値が有効にならなかったり，生成されたソースがコンパイルできなかったりする場合があります。

(8) nillable 属性に"false"を指定した場合の動作について

WSDL 定義で nillable 属性に"false"を指定した変数（データ型）に null を設定しても，SOAP メッセージの送受信時エラーにはなりません。その場合，他社製品と通信できない場合があります。null を指定した場合の SOAP メッセージの例を次に示します。

```
<変数名 xsi:nil="true"/>
```

(9) WSDL2Java コマンド実行時の KDCCC0289-W メッセージについて

Java2WSDL コマンドを実行して生成された WSDL ファイル定義では，名前空間 URI「http://schemas.xmlsoap.org/soap/encoding/」に属する型を参照していても，その名前空間 URI「http://schemas.xmlsoap.org/soap/encoding/」のインポート文が記述されません。そのため，WSDL2Java コマンド実行時に，Java2WSDL コマンドを使用して生成された WSDL ファイルを指定すると，KDCCC0289-W メッセージが出力されます。

WSDL2Java コマンドを実行したときに KDCCC0289-W メッセージが出力されても，名前空間 URI「http://schemas.xmlsoap.org/soap/encoding/」のインポート文が記述されている場合と同様に，正常に処理されます。

KDCCC0289-W メッセージは，次のすべての条件が重なる場合に出力されます。

- Java2WSDL コマンドの入力である Java インタフェースで，次に示す Java クラスを使用している。

  <この現象の原因となる Java クラス名と WSDL 内のデータ型の対応>

  （Java クラス名）→（WSDL 内データ型）

  java.lang.String → soapenc:string

  java.lang.Boolean → soapenc:boolean

  java.lang.Double → soapenc:double

  java.lang.Float → soapenc:float

  java.lang.Integer → soapenc:int

  java.math.BigInteger → soapenc:integer

  java.math.BigDecimal → soapenc:decimal

  java.lang.Long → soapenc:long

  java.lang.Short → soapenc:short

java.lang.Byte → soapenc:byte

byte[] → soapenc:base64

注

プレフィクス soapenc は，「http://schemas.xmlsoap.org/soap/encoding/」です。

- Java2WSDL コマンド実行時に，-T オプションに"1.2"を指定して WSDL ファイルを生成する。
- 上記で生成した WSDL ファイルを指定して WSDL2Java コマンドを実行する。

KDCCC0289-W メッセージを出力させないためには，WSDL ファイルを手動で編集し，名前空間 URI「http://schemas.xmlsoap.org/soap/encoding/」のインポート文を追加してください。WSDL2Java コマンドを実行するときには，編集した WSDL ファイルを指定してください。

(10) WSDL と SOAP サービスの対応

一つの WSDL に複数の SOAP サービスを対応させることはできません。

3.4　document/literal に対応した SOAP アプリケーションの開発

document/literal は，スキーマで定義された形式の XML ドキュメントを SOAP ボディの子要素としてそのまま送受信するインタフェース形態です。Application Server では，document/literal に対応した SOAP アプリケーションの開発を支援する機能として，次の機能を提供しています。

- WSDL の生成
  Java インタフェースから document/literal に対応した WSDL を生成します。
- ソースコードの生成
  document/literal に対応した WSDL からソースコードを生成します。
- サービスデプロイ定義の生成
  document/literal に対応したサービスデプロイ定義ファイルを生成します。

ここでは，各機能の使用方法，マッピング，および注意事項について説明します。

3.4.1　document/literal 使用時の WSDL の生成

document/literal に対応した WSDL の生成方法，および Java インタフェースと WSDL のマッピングについて説明します。

(1)　WSDL の生成

document/literal に対応した WSDL は，Java2WSDL コマンドで生成できます。Java2WSDL コマンドの-z オプションで「DOCUMENT」，-u オプションで「LITERAL」を指定します。

Java2WSDL コマンドの使用方法については，「9.1　Java2WSDL コマンド（WSDL の生成）」を参照してください。

注意事項

DOCUMENT スタイルを使用する場合，メソッドをオーバーロードしている Java インタフェースを指定することはできません。

DOCUMENT スタイルでは，Java インタフェースのメソッド名を要素名とする要素が WSDL に定義され，その要素によって引数および戻り値がラップされます。したがって，Java インタフェースでメソッドをオーバーロードしている場合，同一名称のメソッドが複数存在することになり，WSDL にも同一要素名の要素が多重定義されてしまいます。

Java2WSDL コマンドで指定できない Java インタフェースを指定した場合は，エラーメッセージが出力され，異常終了となります。

(2)　Java2WSDL コマンドで生成される WSDL

document/literal 使用時の Java インタフェースと WSDL のマッピング，および生成される WSDL の詳細仕様について説明します。

(a)　Java インタフェースと WSDL のマッピング

次に示す WSDL の生成例を基に，document/literal 使用時の Java インタフェースと WSDL のマッピングについて説明します。

図 3-17　Java インタフェースと WSDL のマッピング（document/literal）

Java インタフェースと WSDL のマッピング

- wsdl:types 要素内の xsd:element 要素に，対象メソッドの「メソッド名」が要素名として定義されます。また，「メソッド名」要素の子要素に，対象メソッドの引数が定義されます（図中の（1）および（2））。
- wsdl:types 要素内の xsd:element 要素に，対象メソッドの「メソッド名」＋「Response」が要素名として定義されます。また，「メソッド名」＋「Response」要素によって，対象メソッドの戻り

値および引数で与えられた Holder クラスがラップされます。なお，対象メソッドの戻り値は，「メソッド名」＋「Return」要素の子要素として定義されます（図中の（3）および（4））。

- 対象メソッドの引数および戻り値をラップした要素が，wsdl:message 要素の子要素である wsdl:part 要素の element 属性で参照されます（図中の（5）および（6））。

Java2WSDL コマンドで document/literal に対応した WSDL を生成する場合，常に wrapped 形式の WSDL が生成されます。

なお，次のどちらかが該当する場合，子要素に空の complexType 要素（<complexType/>）を持つ「メソッド名」要素および「メソッド名」＋「Response」要素が生成されます。

- 対象メソッドの引数がない場合
- 戻り値が void で，引数に Holder クラスが含まれない場合

(b) 生成される WSDL の定義内容

次の表に，Java2WSDL コマンドで document/literal に対応した WSDL を生成した場合の定義内容を示します。

表 3-9 Java2WSDL コマンドで生成された WSDL の定義内容（document/literal）

WSDL 中の要素			定義内容
wsdl:types 要素	xsd:schema 要素	xsd:schema 要素	elementFormDefault 属性の値は"qualified"が定義されます。
		リクエストメッセージに対応する xsd:element 要素	• name 属性で"メソッド名"が定義されます。 • 子要素として xsd:complexType 要素が定義されます。xsd:complexType 要素では，メソッドの引数を表す要素が xsd:sequence で並べられます。 • xsd:sequence 下の要素の並び順は，メソッドシグネチャの引数の順序どおりとなります。 • xsd:sequence 下の要素の要素名は"in"＋"通し番号"※1 で，型は引数に対応する型が定義されます。ただし Holder クラスの場合は内部に持つ値の型となります。 • メソッドの引数がない場合は子要素として空の xsd:complexType 要素が定義されます。
		レスポンスメッセージに対応する xsd:element 要素	• name 属性で"メソッド名"＋"Response"が定義されます。 • 子要素として xsd:complexType 要素が定義されます。xsd:complexType 要素では，メソッドの戻り値および引数で与えられた Holder クラスを表す要素が xsd:sequence で並べられます。 • xsd:sequence 下の要素の並び順は，1 番目が戻り値で，その後はメソッドシグネチャの引数の順序どおりとなります。

WSDL 中の要素			定義内容
			• xsd:sequence 下の戻り値を表す要素の要素名は"メソッド名"+"Return"で，型は戻り値に対応する型が定義されます。 • xsd:sequence 下の Holder クラスを表す要素は，リクエストメッセージの対応する要素と同じ要素名で同じ型※2 が定義されます。 メソッドの戻り値が void，かつメソッドの引数に Holder クラスが含まれない場合は子要素として空の xsd:complexType 要素が定義されます。
リクエストメッセージに対応する wsdl:message 要素※3	wsdl:part 要素		element 属性で xsd:schema 要素下で定義されている，name 属性が"メソッド名"の要素が定義されます。
レスポンスメッセージに対応する wsdl:message 要素※3	wsdl:part 要素		element 属性で xsd:schema 要素下で定義されている name 属性が"メソッド名"+"Response"の要素が定義されます。
wsdl:binding 要素	soap:binding 要素		style 属性で"document"が定義されます。
	wsdl:input 要素／wsdl:output 要素	soap:body 要素	• use 属性で"literal"が定義されます。 • namespace 属性は存在しません。

注※1

1 番目の引数から n 番目の引数まで，in0，in1，…，in（n-1）となります。

注※2

例えば 2 番目の引数と 4 番目の引数が Holder クラスの場合は，in1 と in3 が xsd:sequence 下に含まれます。

注※3

子要素として part 要素が一つ定義されます。

(3) Java ソースコードと WSDL のマッピング例

Java インタフェース，Java クラス，および document/literal に対応した WSDL の生成例を示します。

- Java2WSDL コマンドに指定する Java インタフェース

```
package localhost;

public interface UserInfo {
    public UserData getUserData(String in0, int in1);
}
```

- Java2WSDL コマンドに指定する Java インタフェースから利用されるクラスの定義

```
package localhost;
public class UserData {
    private String name;
    private String section;
```

```
    private String telephone;

    public UserData() {
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getSection() {
        return section;
    }
    public void setSection(String section) {
        this.section = section;
    }
    public String getTelephone() {
        return telephone;
    }
    public void setTelephone(String telephone) {
        this.telephone = telephone;
    }
```

- 生成された document/literal の WSDL

```
<wsdl:definitions targetNamespace="http://localhost" xmlns:intf="http://localhost" xmlns:
soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/
" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types>
  <schema elementFormDefault="qualified" targetNamespace="http://localhost" xmlns="http:/
/www.w3.org/2001/XMLSchema">
   <element name="getUserData">
    <complexType>
     <sequence>
      <element name="in0" type="xsd:string"/>
      <element name="in1" type="xsd:int"/>
     </sequence>
    </complexType>
   </element>
   <element name="getUserDataResponse">
    <complexType>
     <sequence>
      <element name="getUserDataReturn" type="intf:UserData"/>
     </sequence>
    </complexType>
   </element>
   <complexType name="UserData">
    <sequence>
     <element name="name" nillable="true" type="xsd:string"/>
     <element name="section" nillable="true" type="xsd:string"/>
     <element name="telephone" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
  </schema>
 </wsdl:types>
 <wsdl:message name="getUserDataRequest">
  <wsdl:part element="intf:getUserData" name="parameters"/>
 </wsdl:message>
```

```
 <wsdl:message name="getUserDataResponse">
  <wsdl:part element="intf:getUserDataResponse" name="parameters"/>
 </wsdl:message>
 <wsdl:portType name="UserInfo">
  <wsdl:operation name="getUserData">
   <wsdl:input message="intf:getUserDataRequest" name="getUserDataRequest"/>
   <wsdl:output message="intf:getUserDataResponse" name="getUserDataResponse"/>
  </wsdl:operation>
 </wsdl:portType>
 <wsdl:binding name="UserInfoSoapBinding" type="intf:UserInfo">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="getUserData">
   <soap:operation soapAction=""/>
   <wsdl:input name="getUserDataRequest">
    <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output name="getUserDataResponse">
    <soap:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
 </wsdl:binding>
 <wsdl:service name="UserInfoService">
  <wsdl:port binding="intf:UserInfoSoapBinding" name="UserInfo">
   <soap:address location="http://localhost/RPCSampleService/services/UserInfo"/>
  </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

3.4.2　document/literal 使用時のソースコードの生成

document/literal に対応した WSDL からソースコードを生成する方法，および WSDL とソースコードのマッピングについて説明します。また，wrapped 形式の WSDL から，ソースコードを生成したときの要素展開規則について説明します。

(1)　ソースコードの生成

document/literal 使用時のソースコードは WSDL2Java コマンドで生成できます。WSDL2Java コマンド実行時に，document/literal 使用時のソースコードの生成に対応した WSDL を指定すると，クライアントアプリケーションおよびサーバアプリケーションに必要なソースコードが生成されます。

WSDL2Java コマンドの使用方法については，「9.2　WSDL2Java コマンド（ソースコードの生成）」を参照してください。

注意事項

DOCUMENT スタイルを使用する場合，次の構造を持つ WSDL を指定することはできません。

- 一つの wsdl:message に対応する wsdl:part 要素が二つ以上ある。
- wsdl:part 要素で element 属性ではなく，type 属性を使用している。
- WSDL 1.1 のスキーマに沿っていない。

WSDL2Java コマンドで指定できない WSDL を指定した場合，エラーメッセージが出力され，異常終了となります。

(2) WSDL と Java ソースコードのマッピング

document/literal に対応した WSDL からソースコードを生成する場合に，注意が必要なマッピング規則について説明します。

(a) WSDL の style 属性の指定とオペレーションの style の設定

soap:binding 要素の style 属性および soap:operation 要素の style 属性の指定と，オペレーションの style に設定される値の関係を次の表に示します。

表 3-10 style 属性の指定とオペレーションの style の設定値

style 属性の指定※		オペレーションの style の設定値
soap:binding 要素	soap:operation 要素	
○	○	soap:operation 要素の style 属性の値が設定されます。
○	×	soap:binding 要素の style 属性の値が設定されます。
×	○	soap:operation 要素の style 属性の値が設定されます。
×	×	"document"が設定されます。

（凡例）

○：指定されていることを示します。

×：指定されていない，または無効な値が指定されていることを示します。

注※

style 属性の有効値は"rpc"または"document"です。

注意事項

サービスデプロイ定義の構造上，オペレーションごとに style 属性を設定することはできません。したがって，WSDL の wsdl:binding 要素中に複数の soap:operation を記述する場合，style 属性はすべて同じ値を指定する必要があります。異なる style 属性を指定した場合，WSDL2Java コマンド実行時に KDCCC0266-E のエラーが出力され，異常終了となります。

(b) wsdl:part 要素から参照されている xsd:element 要素の値と Java ソースコードの関係

リクエストメッセージおよびレスポンスメッセージの wsdl:part 要素から参照されている xsd:element 要素の値と，Java ソースコードの関係を次の表に示します。

表 3-11　xsd:element 要素の値と Java のマッピング

項番	WSDL 中の項目	条件	Java へのマッピング
1	リクエストメッセージの wsdl:part 要素から参照されている xsd:element 要素	• 型が complexType の場合。 • name 属性の値が対応する wsdl:operation 要素の name 属性の値と一致する場合。	リクエストメッセージの要素の型およびレスポンスメッセージの要素の型に対応する JavaBeans クラスは生成されません。complexType の内部に持つ要素が展開され，それぞれがメソッドの引数および戻り値として設定されます。
2	レスポンスメッセージの wsdl:part 要素から参照されている xsd:element 要素	• 型が complexType の場合。 • name 属性の値が対応する wsdl:operation 要素の name 属性＋ Response の値と一致する場合。	
3	項番 1 および 2 以外の場合		リクエストメッセージの要素の型およびレスポンスメッセージの要素の型に対応する JavaBeans クラスがそれぞれ生成され，メソッドの引数および戻り値として設定されます。

一つの wsdl:portType 要素内に複数の wsdl:operation 要素が定義されている場合，一つの wsdl:operation 要素でも項番 3 の条件を満たせば，すべての wsdl:operation に対して項番 3 のマッピング規則が適用されます。

次に，complexType が JavaBeans クラスにマッピングされない例，およびマッピングされる例を示します。

図 3-18　complexType が JavaBeans クラスにマッピングされない例

WSDLの一部

```
<wsdl:types>
 <schema ･･･>
  <element name="getUserData">
   <complexType>
    <sequence>
     <element name="in0" type="xsd:string"/>
     <element name="in1" type="xsd:int"/>
    </sequence>
   </complexType>
  </element>
  <element name="getUserDataResponse">
   <complexType>
    <sequence>
     <element name="getUserDataReturn"  type="intf:UserData"/>
    </sequence>
   </complexType>
  </element>
    <complexType name="UserData">
     <!--省略-->
    </complexType>
 </schema>
<wsdl:types>
<wsdl:message name="getUserDataRequest">
 <wsdl:part element="intf:getUserDataElement"  name="parameters"/>
</wsdl:message>
<wsdl:message name="getUserDataResponse">
 <wsdl:part element="intf:getUserDataResponse" name="parameters"/>
</wsdl:message>
<wsdl:portType name="UserInfo">
 <wsdl:operation name="getUserData">
  <wsdl:input message="intf:getUserDataRequest"/>
  <wsdl:output message="intf:getUserDataResponse"/>
 </wsdl:operation>
</wsdl:portType>
```

＋”Response”

<element>の子要素は
<complexType>

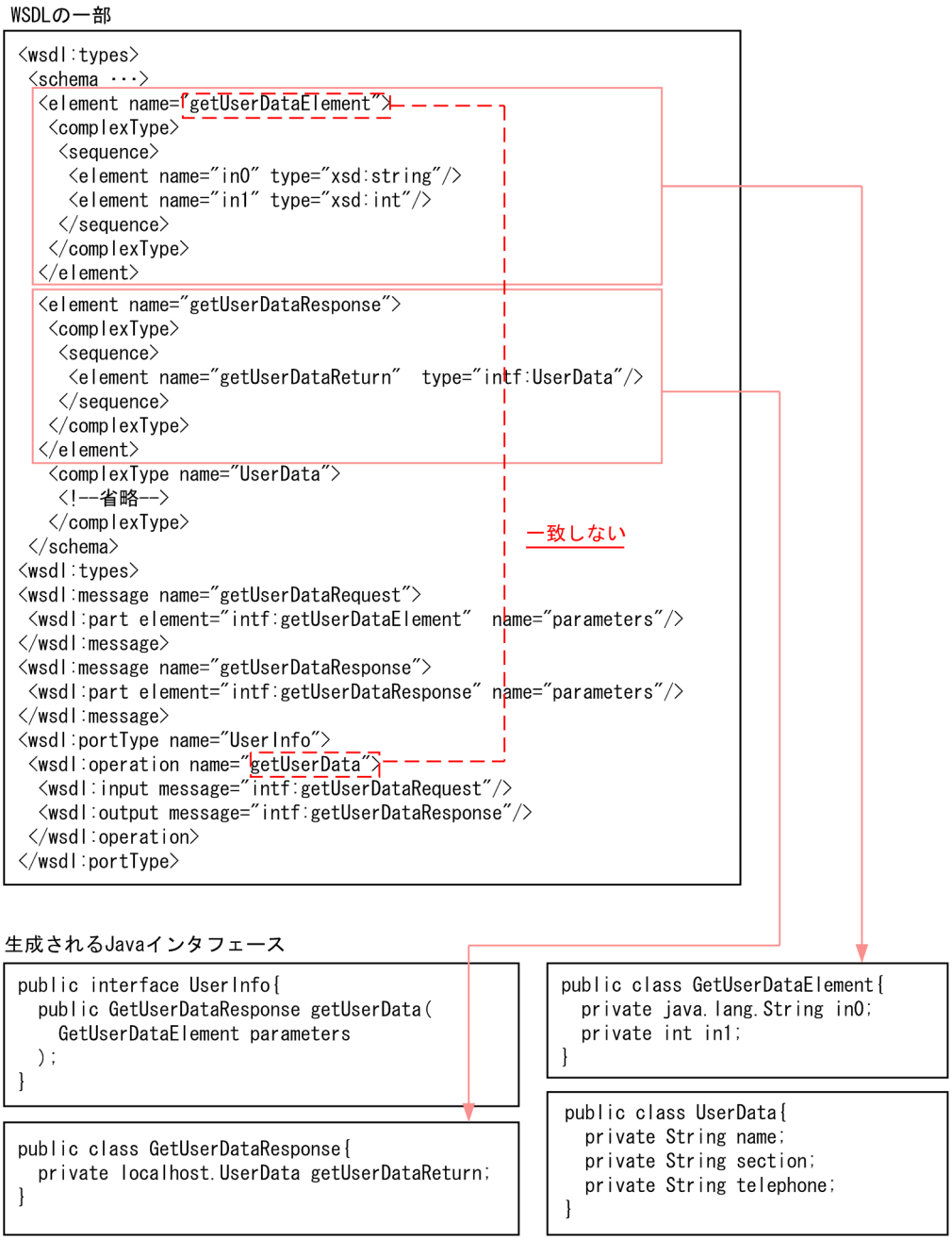
一致する

生成されるJavaインタフェース

```
public interface UserInfo{
  public UserData getUserData(String in0, int in1);
}
```

```
public class UserData{
  private String name;
  private String section;
  private String telephone;
}
```

図 3-19　complexType が JavaBeans クラスにマッピングされる例



(3)　wrapped 形式の要素展開規則

wrapped 形式の要素展開規則を次の表に示します。展開した各子要素を part と見なし，IN 属性のものは WSDL の型から単純に Java の型にマッピングされ，OUT 属性および INOUT 属性のものは WSDL の型から Holder クラスにマッピングされます。なお，レスポンスメッセージに INOUT 属性でない要素が 2 個以上存在する場合は，どれが戻り値か判断できないため，INOUT 属性以外の要素は OUT 属性となります。

表 3-12　wrapped 形式の要素展開規則

<table>
<tr><th rowspan="2">リクエストメッセージの子要素※1 数</th><th colspan="3">レスポンスメッセージの子要素※2 数</th></tr>
<tr><th>0</th><th>1</th><th>2 以上</th></tr>
<tr><td>0</td><td>[引数]
なし
[戻り値]
void</td><td>[引数]
なし
[戻り値]
レスポンスメッセージの子要素の型</td><td>[引数]
レスポンスメッセージの各子要素の型に対応する Holder クラスが，要素の順序どおりに引数として定義されます。
[戻り値]
void</td></tr>
<tr><td rowspan="3">1 以上</td><td rowspan="3">[引数]
リクエストメッセージの各子要素の型
[戻り値]
void</td><td colspan="2">レスポンスメッセージの子要素数－共通要素※3 である子要素数=0 の場合
[引数]
次の項目の順序で引数が定義されます。
• リクエストメッセージの各子要素の型
• レスポンスメッセージの各子要素の型に対応する Holder クラス
[戻り値]
void</td></tr>
<tr><td colspan="2">レスポンスメッセージの子要素数－共通要素※3 である子要素数=1 の場合
[引数]
次の項目が引数として定義されます。
• 共通要素以外のリクエストメッセージの子要素の型
• 共通要素の型に対応する Holder クラス
引数の順序はリクエストメッセージの子要素の順序どおりとなります。
[戻り値]
共通要素以外のレスポンスメッセージの子要素の型</td></tr>
<tr><td colspan="2">レスポンスメッセージの子要素数－共通要素※3 である子要素数>1 の場合
[引数]
次の項目が引数として定義されます。
• 共通要素以外のリクエストメッセージの子要素の型
• 共通要素の型に対応する Holder クラス
• 共通要素以外のレスポンスメッセージの子要素の型に対応する Holder クラス
引数の順序は，まずリクエストメッセージの子要素の順序どおりに定義され，続いて共通要素以外のレスポンスメッセージの子要素の順序となります。
[戻り値]
void</td></tr>
</table>

注※1

リクエストメッセージの子要素とは，WSDL の中で，リクエストメッセージに対応する part 要素から参照されている element の子要素を指します。

注※2

レスポンスメッセージの子要素とは，WSDL の中で，レスポンスメッセージに対応する part 要素から参照されている element の子要素を指します。

注※3

共通要素とは，リクエストメッセージの子要素と，レスポンスメッセージの子要素で，要素名と型が一致する要素を指します。

注意事項

- レスポンスメッセージの子要素は，戻り値，INOUT 属性，OUT 属性の順序で定義してください。また，INOUT 属性の場合，リクエストメッセージの子要素とレスポンスメッセージの子要素は同じ順序で定義してください。異なる順序で定義した場合，レスポンスメッセージの子要素の定義とは異なる順序で，応答時の SOAP メッセージを送受信します。
- IN 属性，OUT 属性，および INOUT 属性の要素を合計した数は 254 個までです。255 個以上は定義できません。合計で 255 個以上定義した場合，不正なソースが生成されます。

リクエストメッセージの子要素数およびレスポンスメッセージの子要素数が，それぞれ 1 以上となる次の場合の例を示します。

- レスポンスメッセージの子要素数－共通要素数の子要素数=0 の場合
- レスポンスメッセージの子要素数－共通要素数の子要素数=1 の場合
- レスポンスメッセージの子要素数－共通要素数の子要素数>1 の場合

図 3-20　レスポンスメッセージの子要素数－共通要素数の子要素数=0 の場合

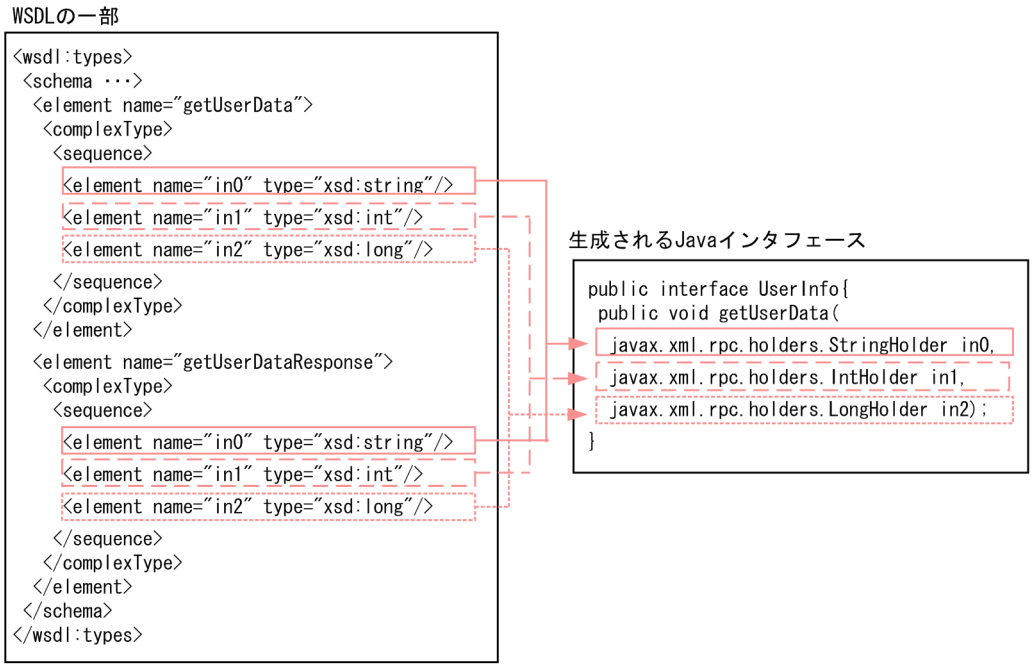


図 3-21　レスポンスメッセージの子要素数－共通要素数の子要素数=1 の場合

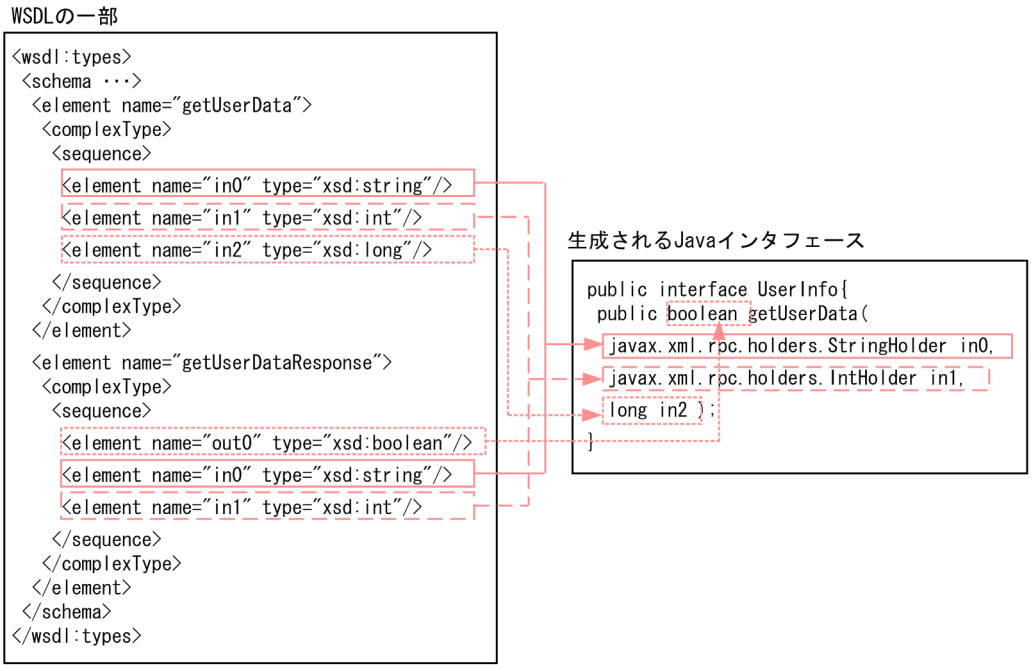
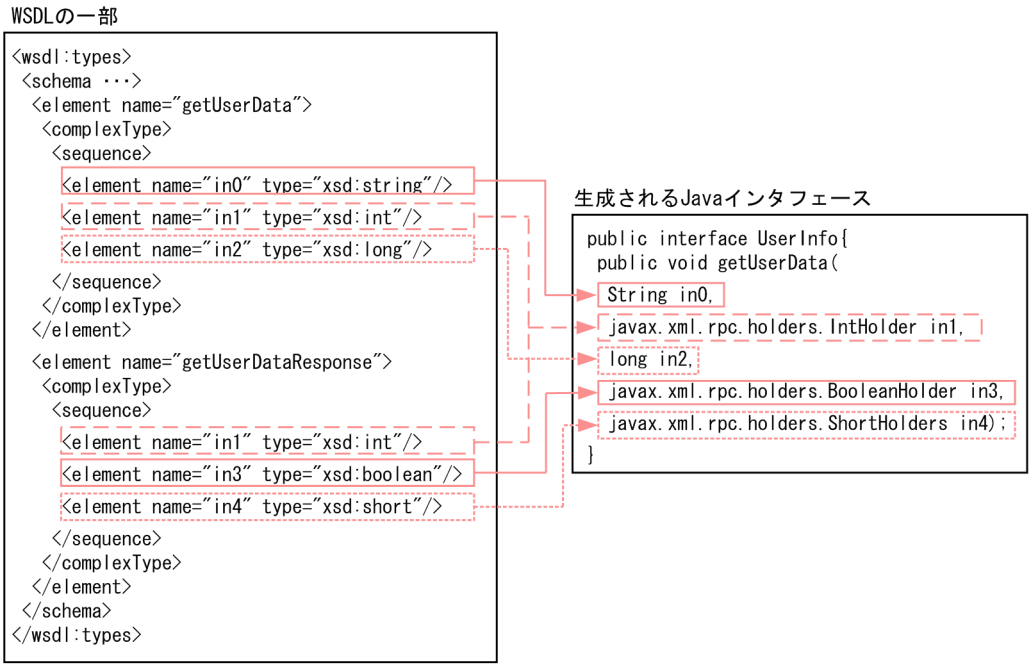


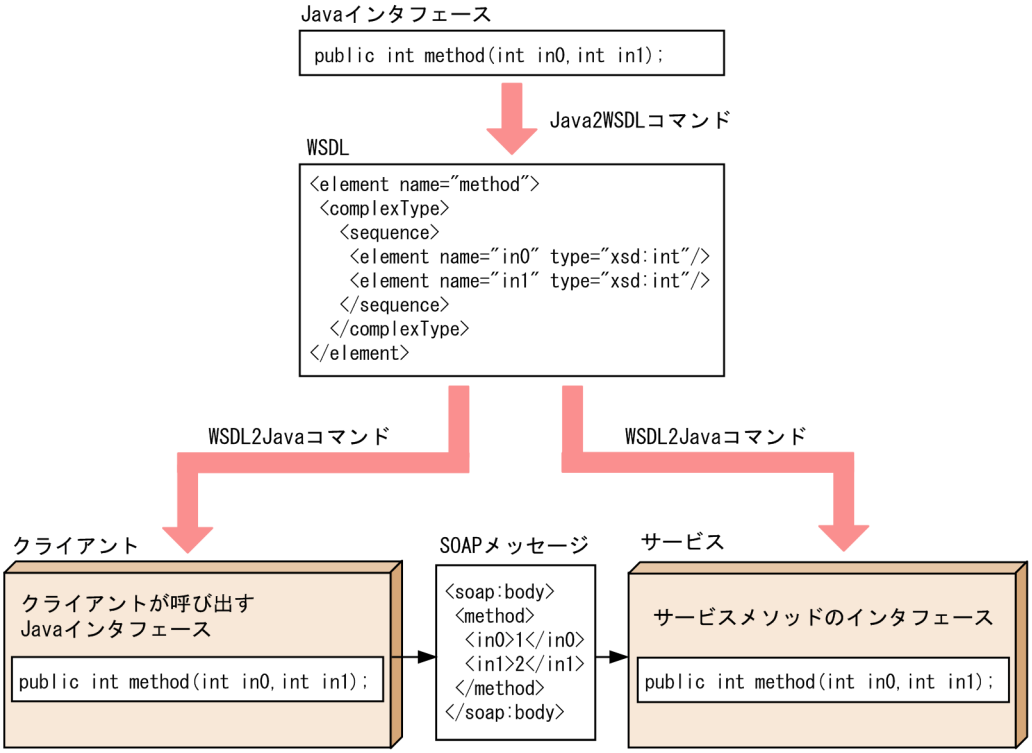
図 3-22　レスポンスメッセージの子要素数－共通要素数の子要素数>1 の場合



document/literal 対応のソースコードから，Java2WSDL コマンドによって生成された WSDL は，オペレーション名と同一名称の要素で引数がラップされます。また，生成された WSDL は，メソッドの引数および戻り値をラップする要素が定義されます。この WSDL に対して WSDL2Java コマンドを実行した場合，ラップした要素が分解されて元の引数，戻り値を持つインタフェースとなります。

document/literal 使用時に生成される WSDL，および生成されるソースコードの関係を次の図に示します。

図 3-23　WSDL 生成と Java ソースコード生成の関係（document/literal）



ただし，一部の型については，WSDL2Java コマンド実行後に異なる型で生成される場合があります。例えば，元となる Java インタフェースで使用している型が配列でも，生成された Java インタフェースでは，配列をラップしたクラスになるといった場合があります。

3.4.3　document/literal 使用時のサービスデプロイ定義の生成

document/literal に対応したサービスデプロイ定義の生成方法について説明します。

document/literal に対応したサービスデプロイ定義は Java2WSDD コマンドで生成できます。
Java2WSDD コマンドの-z オプションで「DOCUMENT」，-u オプションで「LITERAL」を指定します。

Java2WSDD コマンドの使用方法については，「9.3　Java2WSDD コマンド（サービスデプロイ定義の生成）」を参照してください。

注意事項

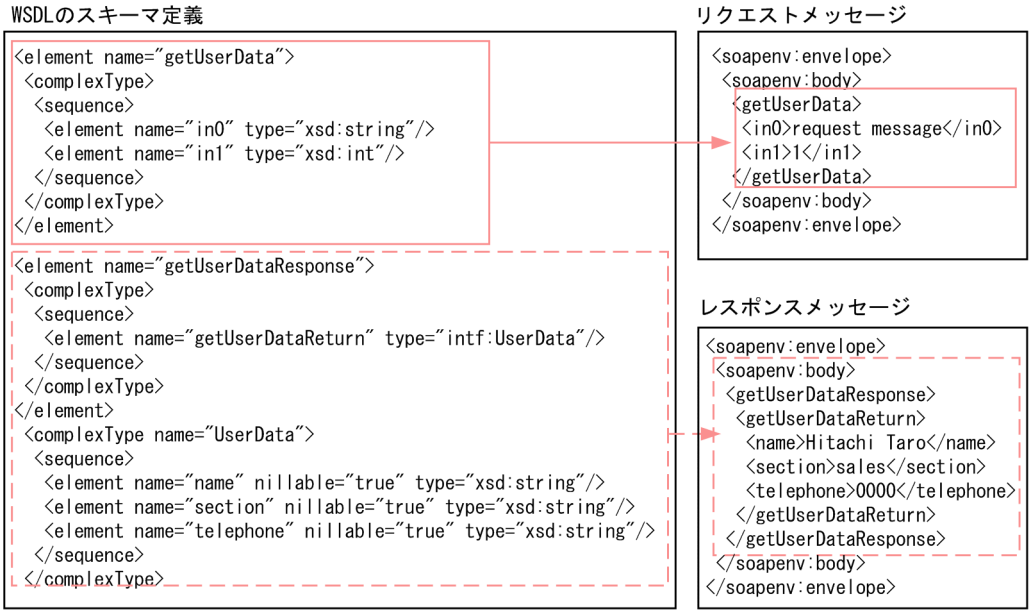
DOCUMENT スタイルを使用する場合，メソッドのオーバーロードをしている Java インタフェースを指定することはできません。

Java2WSDD コマンドで指定できない Java インタフェースを指定した場合は，エラーメッセージが出力され，異常終了となります。

3.4.4　document/literal 使用時の SOAP メッセージの送信

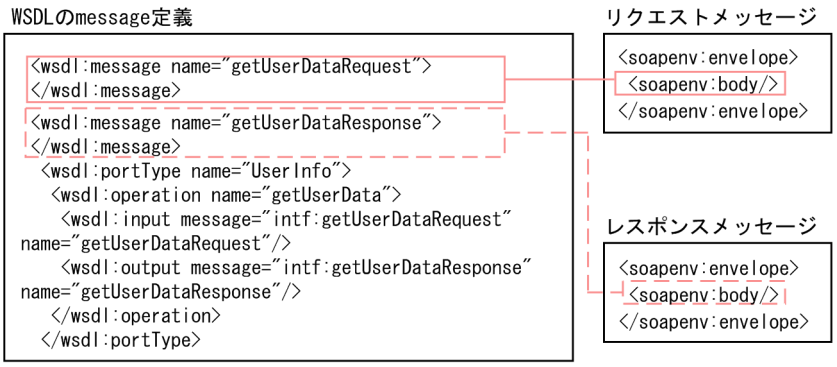
document/literal に対応した SOAP アプリケーションでは，WSDL の wsdl:types 要素で定義したスキーマに従った XML ドキュメントが，SOAP ボディの子要素として格納されます。次の図に，WSDL のスキーマ定義と，リクエストメッセージおよびレスポンスメッセージの対応を示します。

図 3-24　WSDL のスキーマ定義と SOAP メッセージ（document/literal）



また，document/literal に対応した WSDL で wsdl:part が存在しない場合は，次の図に示すとおり，子要素を持たない空の SOAP ボディ要素が送信されます。

図 3-25　part 要素がない場合の WSDL のスキーマ定義と SOAP メッセージ（document/literal）



3.5　RPC 形態の添付ファイルの使用

RPC 形態の SOAP アプリケーションでは，SOAP サービスを呼び出すときの引数や戻り値を利用して，添付ファイル付き SOAP メッセージを送受信できます。例えば，javax.activation.DataHandler クラスを使用して，引数に指定した XML 文書やイメージデータを送受信できます。

RPC 形態の添付ファイルは，document/literal に対応した SOAP アプリケーションで使用できます。また，RPC 形態の添付ファイルは，WS-I Attachments Profile - Version 1.0 仕様に準拠し，wsi:swaRef 形式の WSDL を使用します。

ここでは，RPC 形態で添付ファイル付き SOAP メッセージを送受信する方法について説明します。

3.5.1　添付できるファイル

RPC 形態では，SOAP メッセージで添付できるファイルの種類に制限はありません。ここでは，添付ファイルのサイズおよび添付ファイルの個数の上限値を示します。また，添付ファイルに対応する MIME タイプについて説明します。

(1)　添付ファイルのサイズの上限値

添付できるファイルサイズは，デフォルトで 100MB（104,857,600 バイト）までです。この値は次に示す動作定義ファイルで変更できます。

サーバ定義ファイル

- c4web.attachment.<識別子>.send_max_attachment_size
- c4web.attachment.<識別子>.receive_max_attachment_size

クライアント定義ファイル

- c4web.attachment.send_max_attachment_size
- c4web.attachment.receive_max_attachment_size

サーバ定義ファイルについては，「10.2　サーバ定義ファイルの設定」を参照してください。クライアント定義ファイルについては，「10.3　クライアント定義ファイルの設定」を参照してください。

なお，上限値を超える添付ファイルを送信しようとした場合，SOAP サービスの呼び出し時に，SOAP Fault または例外が発生します。

(2)　添付ファイルの個数の上限値

添付できるファイル個数は，デフォルトで 100 個までです。この値は次に示す動作定義ファイルで変更できます。

サーバ定義ファイル

- c4web.attachment.<識別子>.send_max_attachment_count
- c4web.attachment.<識別子>.receive_max_attachment_count

クライアント定義ファイル

- c4web.attachment.send_max_attachment_count
- c4web.attachment.receive_max_attachment_count

サーバ定義ファイルについては，「10.2　サーバ定義ファイルの設定」を参照してください。クライアント定義ファイルについては，「10.3　クライアント定義ファイルの設定」を参照してください。

なお，上限値を超える添付ファイルを送信しようとした場合，SOAP サービスの呼び出し時に，SOAP Fault または例外が発生します。

(3)　添付ファイルに対応する MIME タイプ

添付したファイルに対応する MIME タイプは，添付したファイルの拡張子によって決まります。次の表に，添付ファイルの拡張子とデフォルトで設定される MIME タイプの対応を示します。

表 3-13　添付ファイルの拡張子と MIME タイプの対応（RPC）

添付ファイルとして設定したファイルの拡張子	設定される MIME タイプ
html，htm	text/html
txt，text	text/plain
gif，GIF	image/gif
ief	image/ief
jpeg，jpg，jpe，JPG	image/jpeg
tiff，tif	image/tiff
xwd	image/x-xwindowdump
ai，eps，ps	application/postscript
rtf	application/rtf
tex	application/x-tex
texinfo，texi	application/x-texinfo
t，tr，roff	application/x-troff
au	audio/basic
midi，mid	audio/midi
aifc	audio/x-aifc
aif，aiff	audio/x-aiff

添付ファイルとして設定したファイルの拡張子	設定される MIME タイプ
wav	audio/x-wav
mpeg，mpg，mpe	video/mpeg
qt，mov	video/quicktime
avi	video/x-msvideo

3.5.2　添付ファイルに使用できる Java 型

添付ファイルに使用できる Java 型，および Java インタフェースで指定できる個所について説明します。

(1)　使用できる Java 型

Java インタフェースに指定する Java 型と，添付ファイルとしての使用可否を次の表に示します。

表 3–14　添付ファイルに使用できる Java 型（RPC）

項番	Java インタフェースで指定する Java 型	添付ファイルとしての使用可否
1	javax.activation.DataHandler	○※1
2	org.apache.axis.holders.DataHandlerHolder	○
3	javax.activation.DataHandler の配列型	○
4	org.apache.axis.holders.DataHandlerHolder の配列型	×※2
5	javax.activation.DataHandler を継承したデータ型	×※3

（凡例）

○：使用できます。

×：使用できません。

注※1

javax.activation.DataHandler は，JavaBeans Activation Framework（JAF）の型であるため，JAF を使用して任意の MIME タイプのファイルを添付できます。

注※2

org.apache.axis.holders.DataHandlerHolder の配列型を使用した場合，javax.activation.DataHandler の配列型としてマッピングされます。

注※3

javax.activation.DataHandler を継承したデータ型を使用した場合，Java2WSDL コマンドは anyType 型，Java2WSDD コマンドはユーザ定義のデータ型クラスとして扱います。

(2)　Java インタフェースでの指定個所

添付ファイルとして使用できる Java 型と，指定できる個所の対応を次の表に示します。

表 3-15　Java インタフェースでの指定個所と添付ファイルで指定できる Java 型の対応（RPC）

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		org.apache.axis.holders.DataHandlerHolder	○
3		javax.activation.DataHandler の配列型	○
4	メソッド戻り値	javax.activation.DataHandler	○
5		org.apache.axis.holders.DataHandlerHolder	×※1
6		javax.activation.DataHandler の配列型	○
7	ユーザ定義のデータ型クラスのフィールド	javax.activation.DataHandler	○
8		org.apache.axis.holders.DataHandlerHolder	×※2
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	×※3
11		org.apache.axis.holders.DataHandlerHolder	×※3
12		javax.activation.DataHandler の配列型	×※3

（凡例）

○：指定できます。

×：指定できません。

注※1

Holder クラスをメソッドの戻り値に指定した場合，Java2WSDL コマンドまたは Java2WSDD コマンドの実行時にエラーとなり，処理が終了します。

注※2

Holder クラスをユーザ定義のデータ型クラスのフィールドに指定した場合，通信時に C4Fault 例外が発生します。

注※3

ユーザ定義例外のフィールドに添付ファイルとして使用できる Java 型を指定した場合，Java2WSDL コマンドまたは Java2WSDD コマンドの実行時にエラーとなり，処理が終了します。

3.5.3　添付ファイルの Java インスタンス生成とデータ取得

添付ファイルの Java 型である javax.activation.DataHandler は，JavaBeans Activation Framework（JAF）の型です。そのため，JAF の仕様に従って添付ファイルを利用できます。

添付ファイルのインスタンス生成方法とデータ取得方法について説明します。

(1)　インスタンスの生成方法

添付ファイルのオブジェクトである javax.activation.DataHandler オブジェクトの生成方法を説明します。

まず，javax.activation.DataHandler クラスのコンストラクタを次の表に示します。

表 3-16　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。ただし，指定できる型は，JAF の仕様でサポートしている範囲の型となります。 第 2 引数 オブジェクトの MIME タイプです。JAF の仕様に従って，第 1 引数で指定したオブジェクトに合わせて指定してください。

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 オブジェクトの生成方法が異なります。生成方法の例を次に示します。

(a)　ファイルを添付ファイルとして送信する場合

ファイルを添付ファイルとして送信する場合の，javax.activation.DataHandler オブジェクトの生成方法を次に示します。なお，ファイルは存在するファイルを指定してください。

1. 添付ファイルのファイルパスを引数に指定し，javax.activation.FileDataSource オブジェクトを生成します。

```
javax.activation.FileDataSource fdSource = new javax.activation.FileDataSource("C:¥¥sample.jpg");
```

2. javax.activation.FileDataSource オブジェクトを引数に指定し，javax.activation.DataHandler オブジェクトを生成します。

```
javax.activation.DataHandler dhandler = new javax.activation.DataHandler(fdSource);
```

(b)　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

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，javax.activation.DataHandler オブジェクトの生成方法を次に示します。

1. 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. java.lang.String オブジェクト，および MIME タイプを引数に指定し，javax.activation.DataHandler オブジェクトを生成します。

java.lang.String オブジェクトは，charset パラメタに指定された文字コードでエンコードされます。

```
javax.activation.DataHandler dhandler = new javax.activation.DataHandler(attachments, ”te
xt/plain; charset=UTF-8”);
```

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 タイプおよび 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。javax.activation.DataHandler(Object, String) コンストラクタの第 2 引数の記述形式を次に示します。

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

なお，charset パラメタを指定する場合は，次の点に注意してください。

- charset パラメタを指定しないと，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てしまいます。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は，必ず charset パラメタを指定してください。
- charset パラメタは，添付ファイルとして String オブジェクトを送信する場合だけ指定できます。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 パラメタの指定値（文字コード）については，次の点に注意してください。

- 大文字／小文字は区別されません。
- 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。

(2) 添付ファイルのデータ取得方法

添付ファイルのデータ取得方法を次に示します。なお，次の指定例に示す「dhandler」は，受信した javax.activation.DataHandler オブジェクトです。

(a) 添付ファイルを受信して java.io.InputStream オブジェクトとして取得する場合

1. 添付ファイルのデータを java.io.InputStream として取得します。

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();
```

(b) 添付ファイルを受信して javax.activation.DataSource オブジェクトとして取得する場合

1. javax.activation.DataSource オブジェクトを取得します。

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();
```

(c) 添付ファイルを受信して java.lang.String オブジェクトとして取得する場合

添付ファイルを受信して java.lang.String オブジェクトとして取得する場合の方法を次に示します。

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()メソッドを実行して，添付ファイルの MIME タイプおよび文字コードを取得します。

   ```
   ava.lang.String mimetype = dhandler.getContentType();
   ```

   ここでは，mimetype の内容として「text/plain; charset=UTF-8」が取得されます。

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;
   ```

3.5.4 添付ファイル使用時の WSDL の生成

添付ファイルに対応した WSDL は，Java2WSDL コマンドで生成できます。Java2WSDL コマンドの-z オプションで「DOCUMENT」，-u オプションで「LITERAL」を指定します。

添付ファイルに対応した Java 型を定義した Java インタフェースと，WSDL のマッピング例を示します。

図 3-26　Java インタフェースと WSDL のマッピング例（RPC）

Javaインタフェース

```
public interface UserInfo extends java.rmi.Remote{
     public UserData getUserData(java.lang.String in0 ,
                                 javax.activation.DataHandler in1 )
throws java.rmi.RemoteException;
}
```

Javaクラス

```
public class UserData implements java.io.Serializable {
    private java.lang.String message;
    private java.lang.String name;
    private java.lang.String section;
}
```

(1)

生成されたWSDLの一部

```
 <wsdl:definitions targetNamespace="http://localhost"
 xmlns:wsi="http://ws-i.org/profiles/basic/1.1/xsd" ... >        (2)
  <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">                                 (3)
     <complexType>
      <sequence>
       <element name="in0" type="xsd:string"/>
       <element name="in1" type="wsi:swaRef"/>
      </sequence>
     </complexType>
    </element>
    <complexType name="UserData">
     <sequence>
      <element name="message" nillable="true" type="xsd:string"/>
      <element name="name" nillable="true" type="xsd:string"/>
      <element name="section" nillable="true" type="xsd:string"/>
     </sequence>
    </complexType>
    <element name="getUserDataResponse">
     <complexType>
      <sequence>
       <element name="getUserDataReturn" type="intf:UserData"/>
      </sequence>
     </complexType>
    </element>
   </schema>
  </wsdl:types>
```

添付ファイルの Java 型は，wsi:swaRef 型にマッピングされます（図の（1））。wsdl:definitions 要素で，wsi の名前空間が宣言されます（図の（2））。wsi:swaRef 型を指定している XML Schema で wsi の名前空間がインポートされます（図の（3））。

3.5.5　添付ファイル使用時のソースコードの生成

添付ファイルに対応したソースコードは，WSDL2Java コマンドで生成できます。WSDL2Java コマンドの-z オプションで「DOCUMENT」，-u オプションで「LITERAL」を指定します。

注意事項

- WSDL 定義では，MIME 拡張要素（mime:multipartRelated，mime:part，mime:content，mime:mimeXml 要素）は使用できません。
- ユーザ定義例外で添付ファイルは使用できません。wsdl:fault 要素内のユーザ定義例外に対応する wsdl:types 要素内の element 要素の型に，添付ファイルが定義されている場合，WSDL2Java コマンド実行時にエラーとなり，処理が終了します。このとき，ソースコードは生成されません。
- WSDL 定義内の wsi:swaRef 型を指定している XML スキーマで，wsi の名前空間「http://ws-i.org/profiles/basic/1.1/xsd」をインポートしていない場合，警告メッセージが出力され，処理は続行します。
- WSDL では，wsi:swaRef 型は xsd:element 内の要素として指定できます。xsd:attribute 内の要素として指定した場合，WSDL2Java コマンド実行時にエラーとなります。

WSDL とソースコードのマッピング例を示します。

図 3-27　WSDL とソースコードのマッピング例（RPC）

WSDLの一部

```
<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>
   <complexType name="UserData">
    <sequence>
     <element name="message" nillable="true" type="xsd:string"/>
     <element name="name" nillable="true" type="xsd:string"/>
     <element name="section" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <element name="getUserDataResponse">
    <complexType>
     <sequence>
      <element name="getUserDataReturn" type="intf:UserData"/>
     </sequence>
    </complexType>
   </element>
  </schema>
 </wsdl:types>
```

(1)

Javaインタフェース

```
public interface UserInfo extends java.rmi.Remote{
     public UserData getUserData(java.lang.String in0 ,
                                 javax.activation.DataHandler in1 )
throws java.rmi.RemoteException;
}
```

WSDL 定義の wsi:swaRef 型は添付ファイルの Java 型へマッピングされます（図の（1））。

3.6　例外処理の実装

この節では，RPC 形態の SOAP アプリケーションで実装する例外処理について説明します。

SOAP アプリケーションでは，Java 例外，SOAP Fault，および WSDL Fault に関する処理を実装できます。ここでは，Java 例外の種類を示し，Java 例外と WSDL Fault，および Java 例外と SOAP Fault のマッピングについて説明します。

3.6.1　Java 例外の種類

SOAP アプリケーションで使用できる Java 例外の種類を示します。

ユーザ定義例外

ユーザ定義例外とは，サービスメソッドの throws 節で必須の java.rmi.RemoteException に追加して定義できる例外です。java.lang.Exception またはほかのユーザ定義例外を直接継承して作成します。

C4Fault

C4Fault（com.cosminexus.cws.service.exception.C4Fault）は，Application Server 独自の例外です。SOAP Fault を表します。

SOAPFaultException

SOAPFaultException（javax.xml.rpc.soap.SOAPFaultException）は，JAX-RPC 1.1 規定の例外です。SOAP Fault を表します。

RemoteException

RemoteException（java.rmi.RemoteException）は，リモートメソッドの呼び出しで発生する通信に関する例外です。サービスメソッドの throws 節では，RemoteException を定義する必要があります。

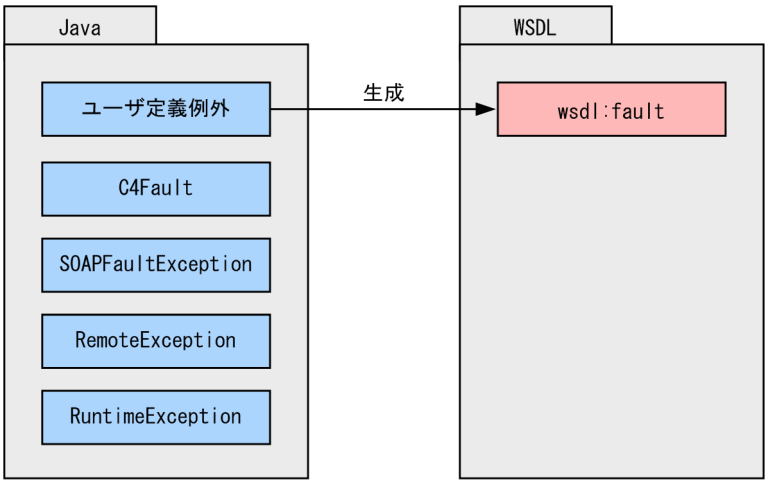
RuntimeException

RuntimeException（java.lang.RuntimeException）は，アプリケーション実行時に発生するランタイム例外です。

3.6.2　Java 例外から WSDL へのマッピング

Java2WSDL コマンドによって，Java 例外に対応する WSDL を生成できます。Java 例外と WSDL のマッピングを次の図に示します。

図 3-28　Java 例外と WSDL のマッピング（Java2WSDL コマンド実行時）



Java 例外のうち，ユーザ定義例外が WSDL Fault（wsdl:fault 要素）にマッピングされます。そのほかの Java 例外については，マッピングされません。

(1)　ユーザ定義例外の定義

ユーザ定義例外は，ユーザ定義のデータ型クラスと同様に作成します。ユーザ定義のデータ型クラスの作成方法については，「4.1.2　Java インタフェースを作成する」および「4.2.1　SOAP アプリケーションを設計する」を参照してください。ただし，ユーザ定義例外は java.lang.Exception，またはほかのユーザ定義例外を直接継承しなければなりません。

java.lang.Exception を除くほかの Java 標準例外※1，および Application Server の実装クラス※2 をスーパークラスとして利用することはできません。

java.sql.SQLException のように直接 java.lang.Exception を継承している場合でも，Java 標準例外※1，および Application Server の実装クラス※2 自体をユーザ定義例外として利用することはできません。

注※1

J2SE 5.0 仕様に対応した次のパッケージ名で始まるクラスは利用できません。

- java
- javax
- org.ietf
- org.omg
- org.w3c
- org.xml

注※2

Application Server が提供する次のパッケージ名で始まるクラスは利用できません。

- com.cosminexus.c4web

- com.cosminexus.cws
- com.cosminexus.xml
- com.cosminexus.wss
- org.apache.axis
- org.apache.commons.discovery
- com.ibm.wsdl

なお，既存の Java クラスおよび EJB を利用する場合も，ここで示した内容に従って定義する必要があります。

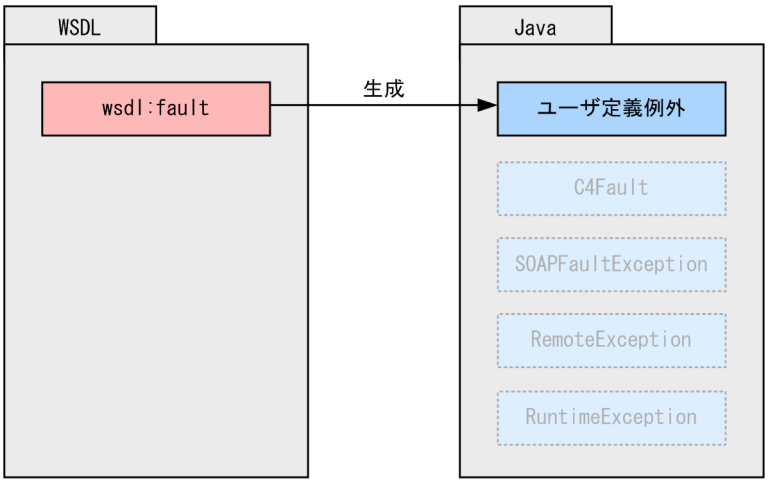
(2) リモートインタフェースの定義

メソッドの throws 節にユーザ定義例外を定義します。リモートインタフェースの例については，「4.1.2 Java インタフェースを作成する」を参照してください。

3.6.3 WSDL から Java 例外へのマッピング

WSDL2Java コマンドによって，WSDL Fault に対応するソースコードを生成できます。WSDL と Java 例外のマッピングを次の図に示します。

図 3-29 WSDL と Java 例外のマッピング（WSDL2Java コマンド実行時）



WSDL Fault（wsdl:fault 要素）とユーザ定義例外のマッピングは可逆ではありません。

(1) wsdl:part 要素の属性

WSDL の wsdl:fault 要素から参照されている wsdl:message 要素は，wsdl:part 要素を一つ持ちます。wsdl:part 要素が複数存在する場合，WSDL2Java コマンドによって KDCCC0239-E のメッセージが出力されます。Java ソースコードは生成されません。

wsdl:part 要素の element 属性が参照できる要素の型，および type 属性が参照できる型は複合型です。単純型および配列は直接参照できません。単純型または配列を利用する場合は，複合型の子要素として定義してください。

単純型または配列を直接参照するように定義した場合は，WSDL2Java コマンドによって，KDCCC0238-E のメッセージが出力されます。Java ソースコードは生成されません。

(2) WSDL Fault からマッピングされるユーザ定義例外の宣言

WSDL2Java コマンド実行時に，WSDL Fault からマッピングされるユーザ定義例外の宣言を示します。

表 3-17 WSDL Fault からマッピングされるユーザ定義例外の宣言

Java 文法	ユーザ定義例外の宣言
パッケージ名※1	xsd:schema 要素の targetNamespace 属性値※2
クラス修飾子	"public"
例外名	xsd:complexType 要素の name 属性値
スーパークラス	java.lang.Exception ただし，xsd:extension 要素を利用して派生している場合は，派生元の xsd:complexType 要素に対応する例外

注※1

「3.6.2(1) ユーザ定義例外の定義」で示した利用できないパッケージ名の場合，KDCCC0231-E のメッセージが出力され，ソースコードは生成されません。

注※2

WSDL2Java コマンドの-N オプション，-f オプション，および-p オプションで名前空間とパッケージ名のマッピングを変更できます。

(3) リモートインタフェースの定義

リモートインタフェースのメソッドの throws 節に，必須の java.rmi.RemoteException に加えて，ユーザ定義例外を定義します。

(4) xsd:complexType 要素からユーザ定義例外へのマッピング

xsd:complexType 要素の子要素から，ユーザ定義例外のフィールド，メソッド，コンストラクタへのマッピングは，ユーザ定義のデータ型クラスのマッピング規則に従います。

(5) soap:fault 要素の定義

wsdl:fault 要素は，soap:fault 要素を一つ持ちます。wsdl:fault 要素に soap:fault 要素が存在しない場合，WSDL2Java コマンドによって KDCCC0012-E エラーが出力されます。Java ソースコードは生成されません。wsdl:fault 要素に soap:fault 要素が複数存在する場合，WSDL2Java コマンドによって KDCCC0247-E エラーが出力されます。Java ソースコードは生成されません。

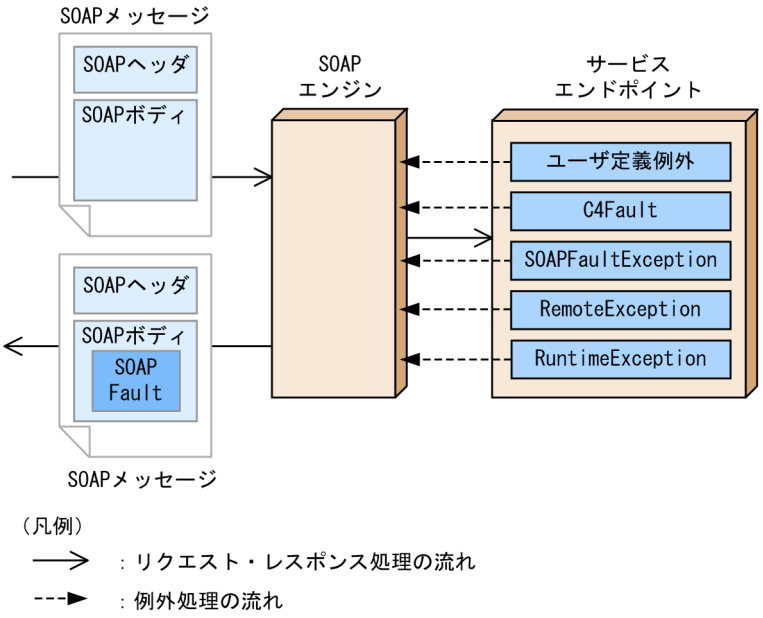
なお，soap:fault 要素の name 属性は，WSDL 1.1 仕様では必須ですが，Application Server では必須ではありません。name 属性が定義されていない場合，処理が続行されます。また，name 属性が定義されている場合でも，name 属性の値は利用されません。

3.6.4　Java 例外から SOAP Fault へのマッピング

SOAP アプリケーションの実行中に例外が発生した場合の処理の流れと，Java 例外と SOAP Fault のマッピングについて説明します。

サービスエンドポイントが例外をスローしたときの処理を次の図に示します。

図 3-30　例外をスローしたときの処理の流れ



Java 例外から SOAP Fault へのマッピング規則を次の表に示します。

表 3-18　Java 例外から SOAP Fault へのマッピング規則

項番	Java 例外	SOAP Fault			
		faultcode	faultstring	faultactor	detail
1	ユーザ定義例外	message part のメッセージ型指定属性が参照する QName	KDCCP0015-E のメッセージ	なし	wsdl:fault 要素に対応するスキーマインスタンス
2	ユーザ定義例外を継承する例外※1	スーパークラスのユーザ定義例外に対応する faultcode	KDCCP0015-E のメッセージ	なし	スーパークラスのユーザ定義例外に対応する detail
3	C4Fault	例外が保持する faultcode	例外が保持する faultstring	例外が保持する faultactor	例外が保持する detail

項番	Java 例外	SOAP Fault			
		faultcode	faultstring	faultactor	detail
4	SOAPFaultException	例外が保持する faultcode	例外が保持する faultstring	例外が保持する faultactor	例外が保持する detail
5	RemoteException	c4web:Server.UserServiceException	KDCCP0007-E のメッセージ	なし	なし
6	RuntimeException	c4web:Server.UserServiceException	KDCCP0007-E のメッセージ	なし	なし
7	項番 1～6 以外※2	c4web:Server.UserServiceException	KDCCP0007-E のメッセージ	なし	なし

注※1

サービスメソッドの throws 節で定義されているユーザ定義例外を継承する例外が該当します。ユーザ定義例外を継承する例外は，throws 節で定義されていないため，WSDL Fault とは対応しません。スーパークラスであるユーザ定義例外として扱われます。

注※2

サービスメソッドの throws 節に，java.rmi.RemoteException およびユーザ定義例外のどちらでもない例外を定義した場合に該当します。

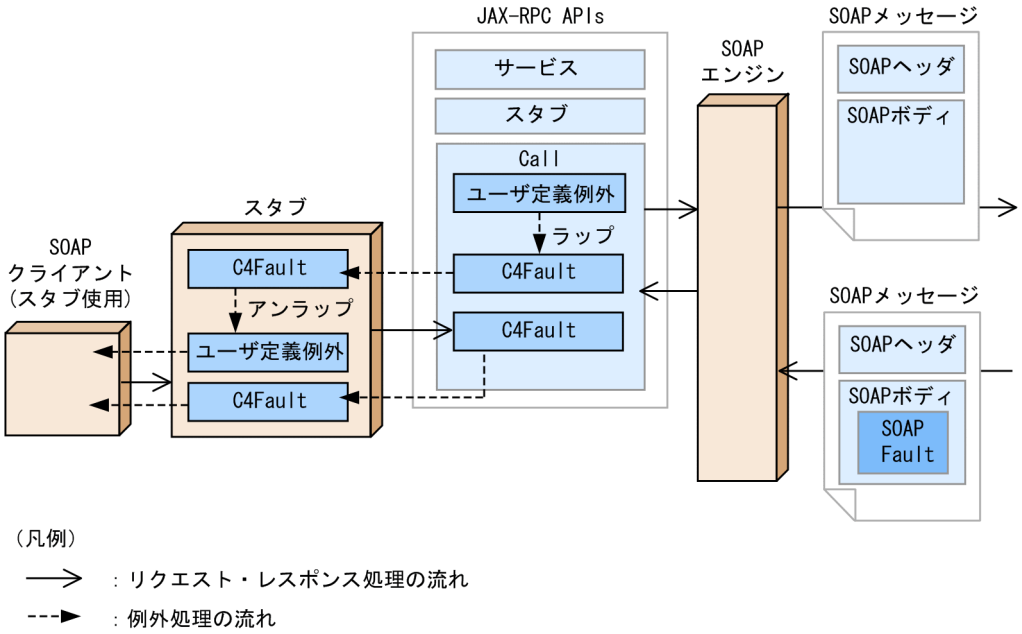
3.6.5 SOAP Fault から Java 例外へのマッピング

SOAP アプリケーションの実行中に SOAP Fault を受信した場合の処理の流れと，SOAP Fault と Java 例外のマッピングについて説明します。

(1) スタブを使用した場合

スタブを使用した場合に，SOAP Fault を受信したときの処理の流れを次の図に示します。

図 3-31　SOAP Fault 受信時の処理の流れ（スタブ）



SOAP Fault を受信したときの処理を次の表に示します。

表 3-19　SOAP Fault 受信時の処理（スタブ）

項番	SOAP Fault	SOAP クライアント
1	WSDL Fault に対応するインスタンス	ユーザ定義例外が取得されます。
2	WSDL Fault に対応しないインスタンス	C4Fault が取得されます。

SOAP Fault から Java 例外へのマッピング規則を次の表に示します。

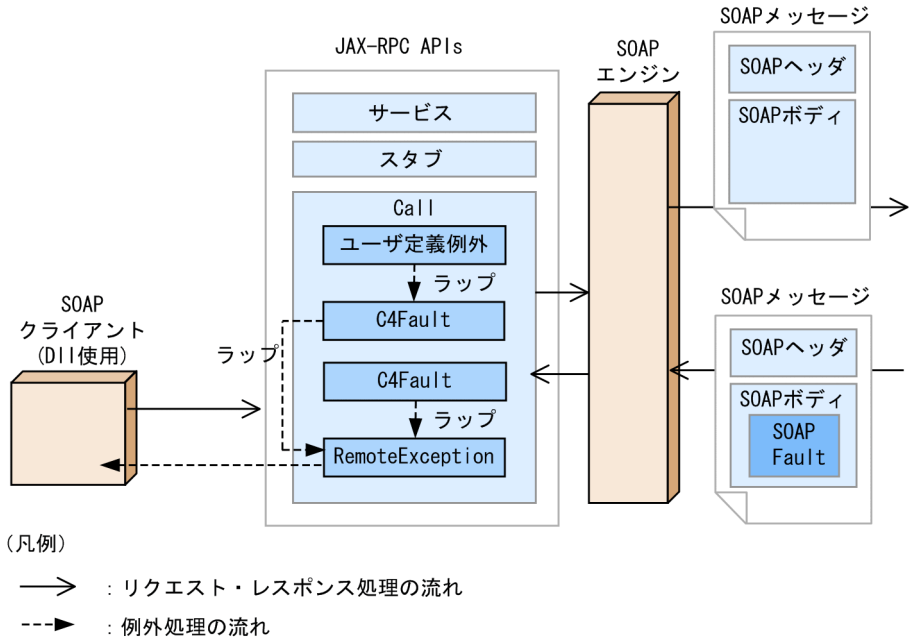
表 3-20　SOAP Fault から Java 例外へのマッピング規則（スタブ）

SOAP Fault の要素	ユーザ定義例外の場合	C4Fault の場合
faultcode	マッピングされません。	C4Fault の FaultCode に設定されます。
faultstring	マッピングされません。	C4Fault の FaultString に設定されます。
faultactor	マッピングされません。	C4Fault の FaultActor に設定されます。
detail	wsdl:fault 要素で指定しているスキーマに従って，ユーザ定義例外へマッピングされます。	C4Fault の FaultDetail に設定されます。

(2)　DII を使用した場合

DII を使用した場合に，SOAP Fault を受信したときの処理の流れを次の図に示します。

図 3-32　SOAP Fault 受信時の処理の流れ（DII）



SOAP Fault を受信したときの処理を次の表に示します。

表 3-21　SOAP Fault 受信時の処理（DII）

項番	SOAP Fault	SOAP クライアント
1	WSDL Fault に対応するインスタンス	1. RemoteException から C4Fault がアンラップされます。 2. C4Fault からユーザ定義例外がアンラップされます。 3. ユーザ定義例外が取得されます。
2	WSDL Fault に対応しないインスタンス	C4Fault をラップした RemoteException が取得されます。

C4Fault をラップした RemoteException の内容を示します。

表 3-22　C4Fault をラップした RemoteException（DII）

RemoteException	内容
詳細メッセージ	KDCCP3008-E An attempt to execute the invoke method has failed.(Class name = <クラス名>, Method signature = <メソッドシグネチャ>, reason = <理由>)
原因例外	C4Fault

3.7　クライアントの開発

SOAP アプリケーションを新規に開発する場合，および既存の Java クラスを利用して開発する場合，クライアントは WSDL2Java コマンドで生成したスタブを使用して実装します。また，SOAP アプリケーションのクライアントでは，Management クラスと ClientID クラスの実装が必要になります。

ここでは，スタブの使用方法，および Management クラスと ClientID クラスの使用方法について説明します。

なお，DII を使用して SOAP アプリケーションを開発する場合のクライアントについては，「3.8　DII を使用したクライアントの開発」を参照してください。

3.7.1　スタブの使用

生成されるスタブの内容，およびスタブを使用して SOAP サービスを呼び出す場合の実装について説明します。

(1)　スタブの内容

WSDL2Java コマンドで生成するファイルのうち，クライアントでは次に示すファイル（スタブ）を使用します。

表 3-23　クライアントで利用するファイル（スタブ）

ファイル名	内容
<サービス名>Locator.java	サービスクラスです。SOAP サービスへの接続情報を保持しています。
<ポートタイプ名>.java	リモートインタフェースです。SOAP サービスで利用できるメソッドを定義しています。
<データ型名>.java	ユーザ定義のデータ型クラスです。メソッドのパラメタや，パラメタのクラスで参照されるデータ型を定義しています。必要に応じて使用します。

サービス名は WSDL の wsdl:service 要素の name 属性の値が設定されます。ポートタイプ名は WSDL の wsdl:portType 要素の name 属性の値が設定されます。また，データ型名はユーザ定義のデータ型の名前に応じて設定されます。

- サービスクラス

  SOAP サービスへの接続先（endpoint）情報を参照および設定できます。このクラスでは次のメソッドを提供します。

  表 3-24　サービスクラスのメソッド

メソッド名	機能説明
get<ポート名>Address	SOAP サービスへの接続先情報を返します。

メソッド名	機能説明
	戻り値：java.lang.String
get<ポート名>	リモートインタフェースへのオブジェクトポインタを返します。 戻り値：リモートインタフェースへのオブジェクトポインタ
get<ポート名>(java.net.URL portAddress)	指定された SOAP サービスへの接続先情報を使用して，リモートインタフェースへのオブジェクトポインタを返します。 戻り値：リモートインタフェースへのオブジェクトポインタ

ポート名は WSDL の wsdl:port 要素の name 属性の値が設定されます。

注意事項

生成されたスタブの内容は変更しないでください。

- リモートインタフェース

メソッドを呼び出すことによって，SOAP サービスのオペレーションを呼び出すことができます。

(2)　サービス呼び出し処理の実装

スタブを使用して SOAP サービスを呼び出す処理を実装します。次の例では，サービスクラスの例を UserInfoServiceLocator.java，リモートインタフェースの例を UserInfo.java としています。

1. クライアントの開始処理を実行し，クライアント識別子を取得します。

Management クラスおよび ClientID クラスを使用します。なお，呼び出しごとにこの操作を実行する必要はありません。

```
（例）
  ClientID cltID = Management.initializeClient();
```

Management クラスおよび ClientID クラスの使用方法については，「3.7.2　Management クラスと ClientID クラスの使用」を参照してください。Management クラスおよび ClientID クラスの API 仕様については，「13.　SOAP 通信基盤が提供する API」を参照してください。

2. クライアント識別子を現在のスレッドに関連づけます。

```
（例）
  Management.connectClientIDtoCurrentThread(cltID);
```

3. サービスクラスのオブジェクトを生成します。

```
（例）
  UserInfoServiceLocator uis = new UserInfoServiceLocator();
```

4. サービスクラスのオブジェクトを使用して，リモートインタフェースのオブジェクトを生成します。

なお，生成，取得したリモートインタフェースのインスタンスは，複数のスレッドで共有できません。

```
（例）
  UserInfo ui = uis.getUserInfo();
```

SOAP サービスへの接続先は，WSDL の wsdl:service 要素の soap:address 子要素の location 属性となります。クライアントプログラム内で SOAP サービスへの接続先情報を取得する場合は，次のように記述します。

```
（例）
  String url = uis.getUserInfoAddress();
```

また，クライアントプログラム内で SOAP サービスへの接続先を変更する場合は，次のように記述します。

```
（例）
   java.net.URL endpoint = new java.net.URL("http://hostname:8080/WebApp1/services/UserInfo");
   UserInfo ui = uis.getUserInfo(endpoint);
```

5. リモートインタフェースのオブジェクトのメソッドを呼び出して，SOAP アプリケーションのオペレーションを実行します。

```
（例）
  String name = ui.getName();
```

実装例の全体は，「4.1.8　クライアント側の処理を実装する」を参照してください。

なお，DeployScope が「Session」となっている SOAP サービスを呼び出すクライアントプログラムを実装する場合，SOAP サービスの Java インタフェースのインスタンスは，JSP やサーブレットが呼び出されるたびに，ローカル変数に生成してください。初期化や呼び出し時に，インスタンスを生成後，それをメンバ変数に退避して，複数のスレッドで共有すると，DeployScope の「Session」が有効にならない場合があります。なお，DeployScope が「Session」の場合は，クライアントプログラムからの呼び出し単位で，セッションが切り替わります。

このセッションは，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」で記述している「J2EE サーバ間のセッション情報の引き継ぎ」には対応していません。

3.7.2　Management クラスと ClientID クラスの使用

Management クラスは，トレースファイルおよびアプリケーションログの初期化処理と終了処理，クライアント識別子とスレッドの関連づけを行うクラスです。また，**ClientID クラス**はクライアント識別子を表すクラスです。SOAP 通信基盤でクライアントを開発する場合は，必ず Management クラスと ClientID クラスを使用してください。

なお，サーバから別の SOAP アプリケーションを呼び出すような場合（サーバが別のサーバに対するクライアントとなる場合）は，このクラスのメソッドを使用する必要はありません。

Management クラスおよび ClientID クラスの API 仕様については，「13.　SOAP 通信基盤が提供する API」を参照してください。

ここでは，Management クラスおよび ClientID クラスのメソッドを呼び出すタイミングについて説明します。

(1) コマンドラインを利用する場合

コマンドラインから SOAP アプリケーションを利用する場合，Management クラスのメソッドは次の表に示すタイミングで呼び出します。

表 3-25 Management クラスのメソッド呼び出しのタイミング（コマンドライン利用時）

メソッド名	呼び出し位置
initializeClient	起動時に 1 回だけ呼び出します。
connectClientIDtoCurrentThread	スタブが提供するメソッドを呼び出す前に 1 回だけ呼び出します。マルチスレッド環境の場合は，各スレッドの開始時に 1 回だけ呼び出します。
disconnectClientIDtoCurrentThread	スタブが提供するメソッドを呼び出したあとに 1 回だけ呼び出します。マルチスレッド環境の場合は，各スレッドの終了時に 1 回だけ呼び出します。
finalizeClient	終了時に 1 回だけ呼び出します。

(2) サーブレットを利用する場合

サーブレットから SOAP アプリケーションを利用する場合，Management クラスのメソッドは次の表に示すタイミングで呼び出します。

表 3-26 Management クラスのメソッド呼び出しのタイミング（サーブレット利用時）

メソッド名	呼び出し位置
initializeClient	init メソッドで 1 回だけ呼び出します。
connectClientIDtoCurrentThread	スタブが提供するメソッドを呼び出す前に 1 回だけ呼び出します。
disconnectClientIDtoCurrentThread	スタブが提供するメソッドを呼び出したあとに 1 回だけ呼び出します。マルチスレッド環境の場合は，各スレッドの終了時に 1 回だけ呼び出します。
finalizeClient	destroy メソッドで 1 回だけ呼び出します。

(3) EJB を利用する場合

EJB から SOAP アプリケーションを利用する場合，Management クラスのメソッドは次の表に示すタイミングで呼び出します。

表 3-27 Management クラスのメソッド呼び出しのタイミング（EJB 利用時）

メソッド名	呼び出し位置
initializeClient	ejbCreate メソッドで 1 回だけ呼び出します。
connectClientIDtoCurrentThread	スタブが提供するメソッドを呼び出す前に 1 回だけ呼び出します。

メソッド名	呼び出し位置
disconnectClientIDtoCurrentThread	スタブが提供するメソッドを呼び出したあとに 1 回だけ呼び出します。マルチスレッド環境の場合は，各スレッドの終了時に 1 回だけ呼び出します。
finalizeClient	ejbRemove メソッドで 1 回だけ呼び出します。

3.8　DII を使用したクライアントの開発

DII を使用した SOAP アプリケーションを開発するには，WSDL に必要な情報を定義して，クライアント側の処理を実装する必要があります。

ここでは，DII を使用して SOAP アプリケーションのクライアントを開発する方法について説明します。

3.8.1　DII の使用時に WSDL に必要な情報

DII を使用した SOAP アプリケーションでは，WSDL を読み込んで必要な情報を保持して，SOAP サービスのオペレーションを呼び出します。

DII を使用する場合に，WSDL に必要な情報を示します。WSDL の例については，「4.5　DII を使用して開発する場合」を参照してください。

- WSDL のサービス名
  名前空間 URI…wsdl:service 要素の名前空間 URI
  ローカル名…wsdl:service 要素の name 属性
- WSDL の URL※
- WSDL のポート名
  名前空間 URI…wsdl:port 要素の名前空間 URI
  ローカル名…wsdl:port 要素の name 属性
- WSDL のオペレーション名
  名前空間 URI…wsdl:operation 要素の名前空間 URI
  ローカル名…wsdl:operation 要素の name 属性
- オペレーションのパラメタ名

  wsdl:part 要素で type 属性を使用している場合
  　パラメタ名…wsdl:part 要素の name 属性
  　パラメタの型…wsdl:part 要素の type 属性

  wsdl:part 要素で element 属性を使用している場合
  　パラメタ名…wsdl:part 要素の element 属性で参照している要素の name 属性
  　パラメタの型…wsdl:part 要素の element 属性で参照している要素の type 属性

  wrapped 形式の WSDL の場合
  　パラメタ名…ラップされた要素の name 属性
  　パラメタの型…ラップされた要素の type 属性
- オペレーションの戻り値

wsdl:part 要素で type 属性を使用している場合

戻り値の型…wsdl:part 要素の type 属性

wsdl:part 要素で element 属性を使用している場合

戻り値の型…wsdl:part 要素の element 属性で参照している要素の type 属性

wrapped 形式の WSDL の場合

戻り値の型…ラップされた要素の type 属性

- XML の複合型

  複合型の名前空間 URI…xsd:complexType 要素の名前空間 URI

  複合型の名前…xsd:complexType 要素の name 属性

  複合型に定義した子要素の名前…xsd:element 要素の name 属性

  複合型に定義した子要素の型…xsd:element 要素の type 属性

- XML の配列

  maxOccurs 属性に unbounded を指定した要素の場合

  配列の名前…xsd:element 要素の name 属性

  配列の要素型…xsd:element 要素の type 属性

  wsdl:arrayType 属性を利用して soapenc:Array を制限した複合型の場合

  配列の名前…xsd:complexType 要素の name 属性

  配列の要素型…xsd:attribute 要素の wsdl:arrayType 属性

注※

WSDL の URL には，ローカルファイルへのパスを設定してください。なお，Windows 環境では大文字と小文字は区別されません。UNIX 環境では大文字と小文字が区別されます。

3.8.2　DII のクライアント側の処理を実装する

DII を使用する場合，クライアント側に Service オブジェクト，Call オブジェクト，およびパラメタを生成する処理を実装します。また，SOAP サービスを呼び出す処理を実装します。クライアント側の処理の実装例については，「4.5　DII を使用して開発する場合」を参照してください。

(1)　Service オブジェクトの生成

Service オブジェクトは次の手順で生成します。

1. ServiceFactory#newInstance メソッドを使用して，ServiceFactory オブジェクトを生成します。
2. WSDL の URL を表す URL オブジェクトを生成します。引数に WSDL の URL 文字列を指定します。
3. サービスの QName オブジェクトを生成します。wsdl:service 要素の名前空間 URI を名前空間 URI として，wsdl:service 要素の name 属性をローカル名として，引数に指定します。

4. ServiceFactory#createService(URL wsdlDocumentation, QName serviceName)メソッドを使用して，Service インスタンスを生成します。引数に 2.および 3.で生成したオブジェクトを指定します。

   URL には，ローカルファイルへのパスを設定する必要があります。Windows 環境では大文字と小文字は区別されません。UNIX 環境では大文字と小文字が区別されます。ローカルファイル以外のパスを設定した場合は，javax.xml.rpc.ServiceException 例外がスローされます。

   SOAP クライアントライブラリは，Service インスタンス生成時に，WSDL を読み込み，Service オブジェクトは WSDL の情報を保持します。

(2) Call オブジェクトの生成

Call オブジェクトは次の手順で生成します。

1. ポートの QName オブジェクトを生成します。wsdl:port 要素の名前空間 URI を名前空間 URI として，wsdl:port 要素の name 属性をローカル名として，引数に指定します。

2. オペレーションの QName オブジェクトを生成します。wsdl:operation 要素の名前空間 URI を名前空間 URI として，wsdl:operation 要素の name 属性をローカル名として，引数に指定します。

3. Service#createCall(QName portName, QName operationName)メソッドを使用して，Call インスタンスを生成します。引数に 1.および 2.で生成したオブジェクトを指定します。

   ここで生成された Call オブジェクトは，Service オブジェクトが保持する WSDL の情報を保持します。

(3) プロパティの設定

必要に応じて，Call#setProperty メソッドを使用してプロパティを設定します。設定できるプロパティについては，「13.3 C4Property クラス（実行時オプションの設定）」の setProperty メソッドの説明を参照してください。

(4) パラメタの生成

パラメタは次の手順で生成します。メソッドのパラメタには，「12.2.6 DII 使用時のサポート範囲」に示す Java データ型を指定できます。

1. (2)で生成した Call オブジェクトを com.cosminexus.cws.xml.rpc.Call 型へキャストします。

   com.cosminexus.cws.xml.rpc.Call インタフェースの getParameterClassByName(String paramName)メソッドの引数に，オペレーションのパラメタのローカル名を指定し，パラメタの Java 型の Class オブジェクトを取得します。

2. パラメタの Java 型のオブジェクトを生成します。

   パラメタでプリミティブ型および Java 標準クラスを使用する場合

   1.取得したパラメタの Java 型が，WSDL のパラメタのデータ型に対応する Java 型と一致するか確認します。

   2.Java 型が一致した場合，Java 型のオブジェクトを生成します。

パラメタでユーザ定義のデータ型クラスを使用する場合

1.XML の複合型から完全修飾 Java クラス名を生成します。クラス名は複合型の名前空間 URI と複合型の名前から生成します。

2.取得したパラメタの Java 型のクラス名が，生成したクラス名と一致するか確認します。

3.クラス名が一致した場合，取得したパラメタの Java 型の Class オブジェクトから，java.lang.Class#newInstance()メソッドを使用してオブジェクトを生成します。

4.複合型に定義した子要素の名前と型から，ユーザ定義のデータ型クラスのアクセサの名前を取得します。リフレクションを使用してアクセサを呼び出し，ユーザ定義のデータ型クラスのプロパティを設定します。なお，アクセサについては，「3.8.3(2)　WSDL とユーザ定義のデータ型クラスのマッピング」を参照してください。

パラメタで配列を使用する場合

1.取得したパラメタの Java 型が，XML の配列に対応する Java 配列型と一致するか確認します。

2.Java 型が一致した場合，リフレクションを使用して配列を生成します。

(5)　SOAP サービスの呼び出し

SOAP サービスは次の手順で呼び出します。

1. (4)で生成したパラメタを要素として持つ Object 配列を生成します。
2. Call#invoke(Object[] inputParams)メソッドを使用して，サービスを呼び出します。引数には，1.で生成した Object 配列を指定します。
3. 戻り値を取得します。
4. RemoteException がスローされた場合は，必要な処理を行います。

(6)　戻り値の処理

戻り値は次の手順で処理します。メソッドの戻り値には，「12.2.6　DII 使用時のサポート範囲」に示す Java データ型を指定できます。

1. 戻り値のオブジェクトから java.lang.Object#getClass()メソッドを使用して，戻り値の Java 型の Class オブジェクトを取得します。
2. 戻り値のオブジェクトから値を取得します。

戻り値でプリミティブ型および Java 標準クラスを使用する場合

1.取得した戻り値の Java 型が，WSDL の戻り値のデータ型に対応する Java 型と一致するか確認します。

2.Java 型が一致した場合，リフレクションを使用して戻り値のオブジェクトから値を取得します。

戻り値でユーザ定義のデータ型クラスを使用する場合

1.XML の複合型から完全修飾 Java クラス名を生成します。クラス名は複合型の名前空間 URI と複合型の名前から生成します。

2.取得した戻り値の Java 型のクラス名が，生成したクラス名と一致するか確認します。

3.クラス名が一致した場合，複合型に定義した子要素の名前と型から，ユーザ定義のデータ型クラスのアクセサの名前を取得します。リフレクションを使用してアクセサを呼び出し，ユーザ定義のデータ型クラスのプロパティを取得します。なお，アクセサについては，「3.8.3(2)　WSDL とユーザ定義のデータ型クラスのマッピング」を参照してください。

戻り値で配列を使用する場合

1.取得した戻り値の Java 型が，XML の配列に対応する Java 配列型と一致するか確認します。

2.Java 型が一致した場合，リフレクションを使用して配列の要素を生成します。

(7)　例外の処理

次の手順で例外を処理します。例外には，ユーザ定義例外クラスを指定できます。

1. RemoteException から C4Fault 例外を取得します。

2. C4Fault 例外からユーザ定義例外を取得します。

C4Fault 例外にユーザ定義例外が含まれる場合

リフレクションを使用してユーザ定義例外の情報を取得します。取得方法は，「3.8.2(6)　戻り値の処理」の 2.「戻り値でユーザ定義のデータ型クラスを使用する場合」を参照してください。

C4Fault 例外にユーザ定義例外が含まれない場合

C4Fault 例外から SOAP Fault の情報を取得します。

3.8.3　DII でのユーザ定義のデータ型クラスの使用

ユーザ定義のデータ型クラスとは，任意のデータ型を持つデータ型です。DII でユーザ定義のデータ型クラスを使用する場合，データ数やデータの型は，WSDL の複合型として自由に定義できます。

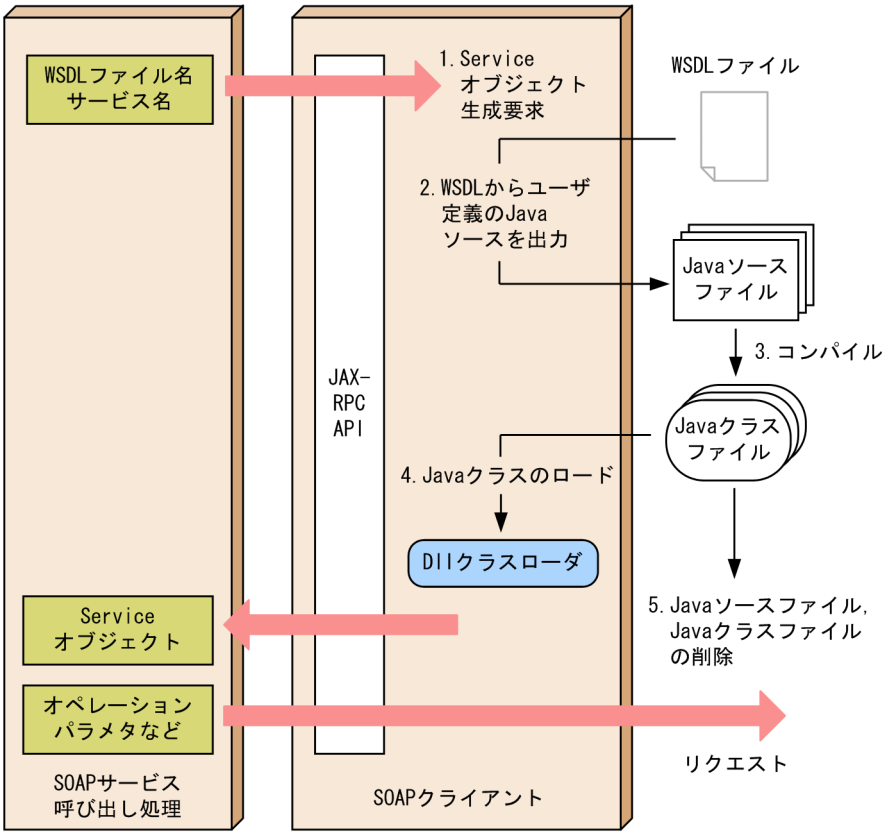
ここでは，DII でユーザ定義のデータ型クラスを使用する方法について説明します。

(1)　処理の流れ

ユーザ定義のデータ型クラスを使用する場合，必要な情報を WSDL の複合型として定義することで，対応する Java クラスを生成できます。

ユーザ定義のデータ型クラスは，次の図に示す流れで使用します。

図 3-33　ユーザ定義のデータ型クラスの使用の流れ（DII）



1. 複合型が定義されている WSDL ファイル名と利用するサービス名を ServiceFactory#createService メソッドの引数に指定し，Service オブジェクトを呼び出します。
2. Service オブジェクトの生成時に WSDL ファイルから Java ソースファイルが生成されます。
3. 2.と同時に Java クラスにコンパイルされます。
4. 3.でコンパイルされた Java クラスは，DII クラスローダによって読み込まれ，使用できる状態になります。
5. Java ソースおよびクラスファイルは，クラスローダに読み込まれたあとに削除されます。

(2)　WSDL とユーザ定義のデータ型クラスのマッピング

WSDL とユーザ定義のデータ型クラスのマッピング規則，および getter／setter のマッピング規則を示します。

表 3-28　WSDL とユーザ定義のデータ型クラスのマッピング規則（DII）

要素	属性	ユーザ定義のデータ型クラス	備考
xsd:schema	targetNamespace	パッケージ名	－

要素			属性	ユーザ定義のデータ型クラス	備考
└	xsd:complexType		name	クラス名	－
				コンストラクタ	public である引数なしコンストラクタが定義されます。
	├	xsd:element	name	フィールド（プロパティ）名	－
				メソッド（アクセサ）のパラメタ名	－
			type	フィールド（プロパティ）の型	対応する型については，「12.2.6　DII 使用時のサポート範囲」を参照してください。
				メソッド（アクセサ）のパラメタと戻り値の型	
	└	xsd:complexContent	－	継承	java.lang.Object クラス，またはほかのユーザ定義のデータ型クラスを継承して定義されます。
		└ xsd:extension	base		

（凡例）

－：特にありません。

表 3-29　getter のマッピング規則（DII）

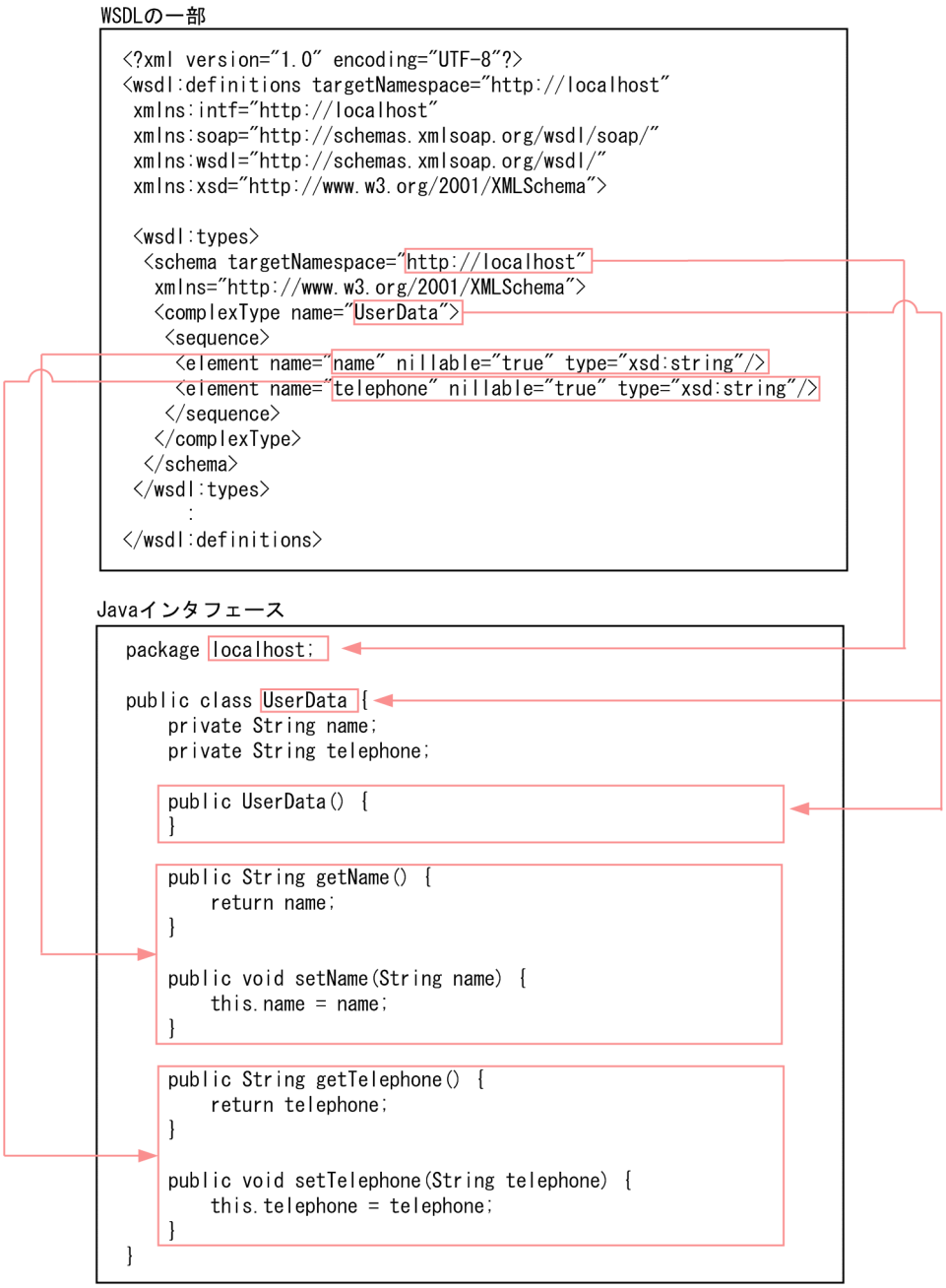
要素	属性	ユーザ定義のデータ型クラス	備考
xsd:element	neme	メソッド（アクセサ）名	"get"+先頭を大文字にした名称で出力されます。戻り値が boolean の場合は，"is"+先頭を大文字にした名称で出力されます。
	type	戻り値	プロパティの値が返されます。

表 3-30　setter のマッピング規則（DII）

要素	属性	ユーザ定義のデータ型クラス	備考
xsd:element	neme	メソッド（アクセサ）名	"set"+先頭を大文字にしたプロパティ名で出力されます。
	type	パラメタ	プロパティと同じ型のパラメタが一つだけ出力されます。

WSDL とユーザ定義のデータ型クラスのマッピング例を次に示します。

図 3-34　WSDL とユーザ定義のデータ型クラスのマッピング例（DII）



(3)　Java ソース生成時の出力先ディレクトリ

Java ソースファイルおよびクラスファイルの出力先ディレクトリを次に示します。

Windows の場合

<Application Server のインストールディレクトリ>/c4web/dii/temp

UNIX の場合

/opt/Cosminexus/c4web/dii/temp

Java ソースファイルおよびクラスファイルの出力先ディレクトリは，サーバ定義ファイルまたはクライアント定義ファイルの次のプロパティで変更できます。

サーバ定義ファイル

c4web.application.<識別子>.dii_temp_directory

クライアント定義ファイル

c4web.application.dii_temp_directory

サーバ定義ファイルについては，「10.2　サーバ定義ファイルの設定」を参照してください。クライアント定義ファイルについては，「10.3　クライアント定義ファイルの設定」を参照してください。

(4)　ユーザ定義のデータ型クラスの Class オブジェクトを取得する方法

ユーザ定義のデータ型クラスを使用する場合，リフレクションを使用してプロパティの値を取得したり設定したりします。リフレクションで必要なユーザ定義のデータ型クラスの Class オブジェクトは，次の方法で取得します。

表 3-31　ユーザ定義のデータ型クラスの Class オブジェクトを取得する方法

項番	ユーザ定義のデータ型クラスの使用個所	Class オブジェクトの取得方法
1	パラメタ	com.cosminexus.cws.xml.rpc.Call インタフェースの getParameterClassByName メソッドの引数に，パラメタのローカル名を指定して取得します。
2	戻り値	java.lang.Object クラスの getClass メソッドを使用して，戻り値のオブジェクトから取得します。
3	ユーザ定義のデータ型クラスのプロパティ	java.lang.Class クラスの getMethods メソッドを使用して，アクセサの java.lang.reflect.Method オブジェクトを取得します。アクセサによって，取得方法が次のように異なります。 **アクセサが getter の場合** java.lang.reflect.Method インタフェースの getReturnType メソッドを使用して，プロパティの Class オブジェクトを取得します。 **アクセサが setter の場合** java.lang.reflect.Method インタフェースの getParameterTypes メソッドを使用して，プロパティの Class オブジェクトを取得します。
4	配列の要素型	java.lang.Class クラスの getComponentType メソッドを使用して，配列クラスから取得します。

注

java.lang.Class クラスの forName メソッドの引数に，ユーザ定義のデータ型クラスの完全修飾名を指定して，ユーザ定義のデータ型クラスの Class オブジェクトを取得することはできません。

3.8.4　WSDL のキャッシュ

DII では SOAP サービスを呼び出すときに，Service オブジェクトを生成します。Service オブジェクトを生成するたびに WSDL ファイルを読み込むと，処理性能が低下してしまいます。そこで，WSDL をキャッシュし，再利用することで処理性能を向上させます。

JavaVM 稼働中であればキャッシュした WSDL が参照されます。ただし,キャッシュ後に WSDL ファイルを変更した場合は，その情報は反映されません。キャッシュ後に WSDL を変更した場合は，JavaVM を再度開始してください。

3.8.5　WSDL 解析と Java ソース生成のタイムアウト

DII では WSDL の解析および Java ソースの生成に時間が掛かると，タイムアウトが発生し処理が終了します。タイムアウトのデフォルトは 45 秒です。タイムアウト値は，サーバ定義ファイルまたはクライアント定義ファイルの次のプロパティでデフォルト値から変更できます。

サーバ定義ファイル

c4web.application.<識別子>.dii_wsdl_parse_timeout

クライアント定義ファイル

c4web.application.dii_wsdl_parse_timeout

サーバ定義ファイルについては，「10.2　サーバ定義ファイルの設定」を参照してください。クライアント定義ファイルについては，「10.3　クライアント定義ファイルの設定」を参照してください。

3.9　アーカイブ（WAR ファイル）の作成

SOAP サービスを J2EE サーバにデプロイするには，アーカイブ（WAR ファイル）を作成する必要があります。WAR ファイルの作成方法については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」を参照してください。

ここでは，SOAP アプリケーションの開発に必要な内容について説明します。

3.9.1　サービスデプロイ定義ファイル（server-config.xml）の格納

WSDL2Java コマンドまたは Java2WSDD コマンドで生成したサービスデプロイ定義ファイル（server-config.xml）は，必ず WEB-INF ディレクトリの直下に格納してください。

3.9.2　DD（web.xml）の記述

servlet 要素および servlet-mapping 要素には，次のように記述してください。

表 3-32　<servlet>の設定項目

項番	<servlet-name>	<display-name>	<servlet-class>	<load-on-startup>
1	任意の文字列	任意の文字列	com.cosminexus.cws.transport.http.c4webServlet	指定しません。
2	任意の文字列	任意の文字列	com.cosminexus.cws.transport.http.AdminServlet	100

表 3-33　<servlet-mapping>の設定項目

<url-pattern>	<servlet-name>
/servlet/c4webServlet	<servlet>の設定項目の項番 1 で設定したサーブレット名
/services/*	<servlet>の設定項目の項番 1 で設定したサーブレット名
/servlet/AdminServlet	<servlet>の設定項目の項番 2 で設定したサーブレット名

- DD（web.xml）での設定項目とサービスロケーションの関係について

  「サーブレットクラス」が"com.cosminexus.cws.transport.http.c4webServlet"の場合，「URL パターン（/services/*）」がサービスロケーションの一部になります（サービスロケーションが http://localhost:8080/WebApp1/services/UserInfo の場合，services 部分）。サービスロケーションの名称に合わせて，URL パターンを変更してください。サービスロケーションについては「3.3.1　サービスロケーションの指定」を参照してください。

3.9.3　アーカイブ（WAR ファイル）作成時の注意事項

一つのアーカイブ（WAR ファイル）には，一つの WSDL が対応します。複数の WSDL から生成した SOAP サービスを，同一のアーカイブ（WAR ファイル）に含めることはできません。

3.10　.NET Framework に対応した SOAP アプリケーションの開発

SOAP アプリケーション開発支援機能では，.NET Framework で開発したクライアントアプリケーションを利用する SOAP アプリケーションを開発できます。

ここでは，.NET Framework を使用して開発する場合の前提条件，および SOAP アプリケーションで使用するファイルの生成について説明します。

3.10.1　.NET Framework 使用時の前提条件

.NET Framework を使用して開発する場合の前提条件を示します。

(1)　インタフェース形態

.NET Framework を使用する場合，document/literal で開発してください。また，wrapped 形式の WSDL を使用してください。

wrapped 形式については，「3.3.3(3)　DOCUMENT／LITERAL を指定した場合」を参照してください。

(2)　WSDL のサポート範囲

WSDL のサポート範囲については，「12.2.1　WSDL 1.1 仕様のサポート範囲」を参照してください。
XML Schema のサポート範囲については，「12.2.7　.NET Framework 使用時のサポート範囲」を参照してください。

.NET Framework を使用して開発する場合，soapencoding 型は使用できません。

(3)　通信プロトコルについて

.NET Framework を使用して開発する場合，通信プロトコルに IPv6 は使用できません。

3.10.2　.NET Framework 使用時の WSDL の生成

.NET Framework に対応した WSDL は，Java2WSDL コマンドで生成できます。コマンド実行時は，次のオプション値を指定してください。

表 3-34　.NET Framework 使用時の Java2WSDL コマンドの指定値

オプション	値	説明
-T	1.1	TypeMappingVersion を表す-T オプションに，1.1 を指定するか，このオプションの指定を省略してください。

オプション	値	説明
-z	DOCUMENT	WSDL のバインディングスタイルを表す-z オプションに，DOCUMENT を指定してください。
-u	LITERAL	use 属性を表す-u オプションに，LITERAL を指定してください。

3.10.3 .NET Framework 使用時のソースコードの生成

.NET Framework を使用して開発するクライアントアプリケーションに必要なソースコード（スタブ）は，.NET Framework SDK が提供する wsdl コマンドで生成してください。

3.10.4 .NET Framework 使用時のサービスデプロイ定義の生成

.NET Framework を使用して開発したクライアントアプリケーションを利用する場合のサービスデプロイ定義は，Java2WSDD コマンドで生成できます。コマンド実行時は，次のオプション値を指定してください。

表 3-35 .NET Framework 使用時の Java2WSDD コマンドの指定値

オプション	値	説明
-T	1.1	TypeMappingVersion を表す-T オプションに，1.1 を指定するか，このオプションの指定を省略してください。
-z	DOCUMENT	WSDL のバインディングスタイルを表す-z オプションに，DOCUMENT を指定してください。
-u	LITERAL	use 属性を表す-u オプションに，LITERAL を指定してください。

3.10.5 .NET Framework 使用時の注意事項

.NET Framework で開発したクライアントアプリケーションから，SOAP アプリケーション開発支援機能で開発したサービスを呼び出す場合，メソッドの引数に null オブジェクトを指定しないでください。また，サービスからの戻り値に null オブジェクトを指定しないでください。メソッドの引数または戻り値に null オブジェクトを指定すると，次の現象が発生する場合があります。

- 不正な SOAP メッセージが生成される。
- 対象オブジェクトに不正な値が設定される。

3.11　RPC 形態の SOAP アプリケーション開発時の注意事項

RPC 形態の SOAP アプリケーション開発時の注意事項について説明します。EJB を利用している場合は，「3.12　EJB 利用時の注意事項」もあわせて参照してください。

3.11.1　SOAP アプリケーションのサービス名に関する注意

RPC 形態，メッセージング形態，および EJB 形態の SOAP サービスで，同一のサービス名は指定しないでください。同一のサービス名の SOAP サービスがデプロイされていると，予期しない動作をするおそれがあります。

3.11.2　WSDL 定義と Holder クラスの対応について

WSDL 中の<output>要素に記述されているデータ型は，ソース生成時に Holder クラスになります。

例えば，string(http://www.w3.org/2001/XMLSchema)を<output>要素に記述した場合，javax.xml.rpc.holders.StringHolder クラスが生成されます。

string を例に，Holder クラスの使用例を次に示します。

- Holder クラスから値を取得する場合

```
public void get(javax.xml.rpc.holders.StringHolder inout0)
{
    java.lang.String value = inout0.value;
}
```

- Holder クラスに値を格納する場合

```
public void set(javax.xml.rpc.holders.StringHolder out0)
{
    out0.value = new java.lang.String(”test”);
}
```

3.11.3　anyType 型のデータ型の送信に関する注意

WSDL 定義でデータ型に anyType を使用した場合，実行時オプションのデータ型定義オプション（send_xsi_types）の設定に関係なく，送信する SOAP メッセージ中にデータ型が含まれます。SOAP 通信基盤を他社の SOAP 製品と接続する場合は SOAP メッセージ中にデータ型を含めるようにしてください。anyType 型のデータを受信する場合に，SOAP メッセージ中にデータ型が含まれていないと C4Fault 例外が発生する場合があります。

3.11.4 GET プロトコルを使用した WSDL ファイルの取得に関する注意

SOAP 通信基盤では，SOAP サービスのエンドポイント URL に「?WSDL」を付与した URL に対して HTTP GET プロトコルを使用しても，WSDL ファイルを取得できません。HTTP GET プロトコルを使用して WSDL ファイルを取得できるようにする場合は，SOAP サービスを含む WAR ファイルに WSDL ファイルを格納し，該当するファイルを表す URL に対して HTTP GET プロトコルを使用するようにしてください。

3.11.5 LITERAL 指定時の SOAP メッセージに関する注意

use 属性に LITERAL，そして TypeMappingVersion に 1.2 を指定して生成された WSDL または WSDD を使用して通信する場合，SOAP メッセージ中の要素に xsi:type 属性が付与されることがあります。

3.11.6 ENCODED 指定時の SOAP メッセージに関する注意

use 属性に ENCODED，そして TypeMappingVersion に 1.1 を指定して生成された WSDL または WSDD を使用して通信する場合，SOAP メッセージ中の要素に SOAP エンコーディングのデータ型情報が付与されることがあります。

3.11.7 リテラルエンコーディングと多重参照オプション（do_multirefs）に関する注意

SOAP 符号化規則にリテラルエンコーディングを使用した SOAP メッセージを送受信する場合，多重参照オプションに true を設定しても有効になりません。多階層となるような SOAP メッセージが送受信できなくなることがあります。多階層となるような SOAP メッセージを送受信したい場合は，SOAP 符号化規則に SOAP エンコーディングを使用してください。

3.11.8 DeployScope を「Session」にした SOAP アプリケーションに関する注意

DeployScope を「Session」にした SOAP アプリケーションを開発する場合，SOAP 通信基盤に対してセッション終了を要求するサービスメソッドを用意できます。このサービスメソッド内で SOAP 通信基盤が提供する「C4Session の invalidate メソッド」を呼び出すことで，SOAP 通信基盤に対してセッション終了を要求できます。クライアント側の処理では，セッション終了を希望するタイミングでこのサービスメソッドを呼び出してください。

SOAP アプリケーションのインタフェースの例を次に示します。

```
package localhost;

public interface UserInfo extends java.rmi.Remote {
    // サービスメソッド
    localhost.UserData getUserData(java.lang.String user_no) throws java.rmi.RemoteException
;

    // セッション終了用サービスメソッド
    public void endSession() throws java.rmi.RemoteException;
}
```

サービス側の処理の実装例を次に示します。

```
package localhost;

public class UserInfoSoapBindingImpl implements localhost.UserInfo {
    // サービスメソッド
    public localhost.UserData getUserData(java.lang.String user_No) throws java.rmi.RemoteEx
ception {
        // サービスメソッド実装を記述
    }

    // セッション終了用サービスメソッド
    public void endSession() throws java.rmi.RemoteException {
        // セッション終了時の処理を記述

        // セッション終了要求を通知するAPIを使用する
        try {
            C4Session session = C4Session.getInstance();
            if (session != null) {
                session.invalidate();
            }
        }
        catch (java.lang.RuntimeException e) {
            // エラー処理
        }
    }
}
```

クライアント側の処理の実装例を次に示します。

```
package localhost;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import com.cosminexus.cws.management.*;

public class RPCSampleClient extends HttpServlet
{

    ...

    // サービスのインタフェースクラスのオブジェクトを生成
```

```
    UserInfoServiceLocator uis = new UserInfoServiceLocator();

    // スタブのインタフェースクラスのオブジェクトを生成
    UserInfo ui = uis.getUserInfo();

    // サービスメソッドを呼び出す
    UserData ud1 = ui.getUserData(strUserNo1); // セッション開始
    UserData ud2 = ui.getUserData(strUserNo2); // セッション継続

    // セッション終了用サービスメソッドを呼び出す
    ui.endSession(); // セッション終了要求

    ...
}
```

3.11.9　クッキーについて

Web サーバから受け取ったクッキーは，SOAP サービスのインタフェースクラスのインスタンスで保持されます。クッキーを利用する場合，SOAP サービスのメソッドを呼び出すたびに，SOAP サービスのインタフェースクラスのインスタンスを破棄することなく，継続して使用してください。

3.11.10　クッキーの有効期限について

expires 属性で設定したクッキーの有効期限に達しても，SOAP クライアントライブラリは保持しているクッキーを削除しません。期限切れのクッキーを受け取っても処理できるようにサーバで対処してください。

3.11.11　WSDL で定義された part アクセサ要素が存在しない SOAP メッセージに関する注意

SOAP アプリケーション開発支援機能で開発した SOAP クライアントおよび SOAP サービス以外と通信する場合，WSDL で定義された part アクセサ要素が存在しない SOAP メッセージを送信させないでください。

RPC 形態のアプリケーションで，WSDL で定義された part アクセサ要素が存在しない SOAP メッセージを受信した場合，SOAP アプリケーションは，パラメタとして null を受け取ります。SOAP サービスが SOAP フォルトを返信する，または SOAP クライアントが例外を受け取ることはありません。

3.12　EJB 利用時の注意事項

SOAP アプリケーションのプログラムに，EJB を利用する場合の注意事項について説明します。

3.12.1　JNDI 名前空間の指定について

JNDI 名前空間に指定する Enterprise Bean の名称は，次のどれかの形式で指定してください。

- JNDI 名前空間の直接指定

  （例）HITACHI_EJB/SERVERS/MyServer/EJB/UserInfo/UserInfoBean※1
- リソース名（java:comp/env 形式）による指定

  （例）java:comp/env/ejb/UserInfoService※2
- EJBHome オブジェクトリファレンスの別名付与（ユーザ指定名前空間機能）による指定

  （例）UserInfo※3

注※1

指定例では，J2EE サーバ名称を「MyServer」，J2EE アプリケーション名を「UserInfo」，Enterprise Bean 名称を「UserInfoBean」としています。

注※2

指定例では，Web アプリケーションでの EJB リファレンス名称を「ejb/UserInfoService」としています。

注※3

指定例では，EJBHome オブジェクトリファレンスの別名付与で EJB リファレンス名称を「UserInfo」としています。

JNDI 名前空間については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」を参照してください。

3.12.2　EJB から SOAP サービスを呼び出す場合の注意事項

Stateful Session Bean および Entity Bean を利用する場合は，トランザクション（コンテナ管理トランザクションまたは Bean 管理トランザクション）の指定に関係なく，SOAP サービスへの要求，受信処理はトランザクションの制御外となります。したがって，トランザクションがロールバックされる場合でも，SOAP サービスへのリクエストは必ず実行されます。

3.12.3 EJB を SOAP サービスのエンドポイントとして利用する場合の注意事項

- SOAP サービスのエンドポイントとして利用できるのは，EJB 2.1 までです。EJB 3.0 以降は利用できません。
- ホームインタフェースに引数ありの create メソッドは使用できません。引数ありと引数なしの create メソッドが存在する場合は，引数なしの create メソッドが呼び出されます。引数なしの create メソッドがホームインタフェースにない場合は，次のメッセージが返されます。

  KDCCP0004-E の C4Fault（SOAP Fault）メッセージ
- Stateful Session Bean および Entity Bean を利用する場合，トランザクション（コンテナ管理トランザクションまたは Bean 管理トランザクション）の指定に関係なく，クライアントへの応答処理はトランザクションの制御外となります。したがって，トランザクションがロールバックされる場合でも，クライアントへの応答は必ず実行されます。
- Entity Bean を利用する場合，ファインダメソッド（findByPrimaryKey など）は利用できません。
- SOAP アプリケーションから呼び出す EJB の内容を変更した場合，J2EE アプリケーションの RMI-IIOP インタフェースおよび RMI-IIOP スタブを再取得し，WSDL を再生成する必要があります。ただし，呼び出す EJB 内のメソッドの引数の数，データ型，および戻り値に変更がない場合は，再取得する必要はありません。
- SOAP アプリケーションから呼び出す EJB を実装した EJB-JAR ファイルと，SOAP アプリケーションの WAR ファイルを同じ J2EE アプリケーションに含める場合は，J2EE アプリケーションの RMI-IIOP インタフェースおよび RMI-IIOP スタブを SOAP アプリケーションの WAR ファイルに含めないでください。
- リモートインタフェースを利用する場合はメソッドの throws 節に java.rmi.RemoteException を定義しなければなりません。ユーザ定義例外を定義する場合は，「3.6.2　Java 例外から WSDL へのマッピング」を参照してください。
  - throws 節に java.rmi.RemoteException を定義していない場合

    Java2WSDL コマンドによって KDCCC0228-W のメッセージが出力され，処理は続行されます。
  - throws 節に java.rmi.RemoteException およびユーザ定義例外のどちらでもない例外を定義した場合

    Java2WSDL コマンドによって KDCCC0229-W のメッセージが出力されます。この例外は wsdl:fault 要素にマッピングされません。
- ローカルインタフェースを利用する場合はメソッドの throws 節に java.rmi.RemoteException を定義しないでください。ユーザ定義例外を定義する場合は，「3.6.2　Java 例外から WSDL へのマッピング」を参照してください。
  - throws 節に java.rmi.RemoteException を定義した場合

    Java2WSDL コマンドによって KDCCC0230-W のメッセージが出力され，処理は続行されます。
  - throws 節にユーザ定義例外以外の例外（java.rmi.RemoteException を除く）を定義した場合

Java2WSDL コマンドによって KDCCC0229-W のメッセージが出力されます。この例外は wsdl:fault 要素にマッピングされません。

3.12.4 ラウンドロビンポリシーによる CORBA ネーミングサービスの検索機能を使用する場合の注意事項

ラウンドロビンポリシーによる CORBA ネーミングサービスの検索機能を使用する場合，サービスデプロイ定義を再作成してください。ラウンドロビンポリシーによる CORBA ネーミングサービスの検索については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」を参照してください。

3.12.5 EJBHome オブジェクトリファレンスの別名付与（ユーザ指定名前空間機能）を使用する場合の注意事項

ユーザ指定名前空間機能を使用する場合，CORBA ネーミングサービスの監視機能が持つ，無効なキャッシュ領域を定期的にクリアする機能を利用することを推奨します。ユーザ指定名前空間機能，および無効なキャッシュ領域のクリアについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」を参照してください。

3.12.6 DeployScope に関する注意事項

SOAP サービスの DeployScope を「Session」にする場合，クライアント側の処理ではセッションを終了する際に remove メソッドを呼び出してください。「Session」以外の DeployScope の場合，クライアント側の処理では remove メソッドを呼び出さないでください。

4 RPC 形態の SOAP アプリケーションの開発例

この章では，RPC 形態の SOAP アプリケーションの開発例を手順に沿って説明します。開発した SOAP アプリケーションの実行方法については，「7.　SOAP アプリケーションの実行と運用」を参照してください。

4.1　新規に開発する場合

RPC 形態の SOAP アプリケーションを新規に開発します。開発例の概要と開発の流れを次に示します。

開発例の概要

社員番号，名前，所属，電話番号というユーザ情報を管理し，クライアントからの入力に対して，処理結果を返す SOAP アプリケーションを新規に開発します。

開発例では，スタブを生成して，クライアント側の処理を実装します。

開発の流れ

1. SOAP アプリケーションの設計
2. Java インタフェースの作成
3. WSDL の生成
4. スケルトンおよびサービスデプロイ定義の生成
5. サーバ側の処理の実装
6. アーカイブ（WAR ファイル）の作成
7. スタブの生成
8. クライアント側の処理の実装

開発の流れに沿って，各ステップの作業について説明します。

4.1.1　SOAP アプリケーションを設計する

新規に開発する SOAP アプリケーションの仕様を設計します。開発例では，次のサービス名およびメソッド名を使用します。

サービス名

UserInfo

メソッド名

getUserData

クライアントからの入力時およびクライアントへの出力時の属性について次に示します。出力時の属性は，UserData クラスが保持します。

表 4-1　入力時の属性（RPC・スタブで新規開発する場合）

属性名	変数名	データ型
社員番号	in0	String

表 4-2　出力時の属性（RPC・スタブで新規開発する場合）

属性名	変数名	データ型
名前	name	String
所属	section	String
電話番号	telephone	String

4.1.2　Java インタフェースを作成する

設計した SOAP アプリケーションの WSDL を作成するために，Java インタフェース，およびユーザ定義のデータ型クラスを作成します。

1. Java インタフェースを作成します。

   SOAP アプリケーションで公開するメソッドを Java インタフェースに定義します。

   メソッドを定義した Java インタフェースの例を次に示します。

```
//UserInfo.java

package localhost;

public interface UserInfo extends java.rmi.Remote {
    localhost.UserData getUserData(java.lang.String in0) throws java.rmi.RemoteException;
}
```

2. ユーザ定義のデータ型クラスを作成します。

   Java インタフェースで使用するユーザ定義のデータ型クラスを作成します。

   ユーザ定義のデータ型クラスの例を次に示します。

```
//UserData.java

package localhost;

public class UserData implements java.io.Serializable {
    private java.lang.String name;
    private java.lang.String section;
    private java.lang.String telephone;

    public UserData() {
    }

    public java.lang.String getName() {
        return name;
    }

    public void setName(java.lang.String name) {
        this.name = name;
    }

    public java.lang.String getSection() {
```

```
        return section;
    }

    public void setSection(java.lang.String section) {
        this.section = section;
    }

    public java.lang.String getTelephone() {
        return telephone;
    }

    public void setTelephone(java.lang.String telephone) {
        this.telephone = telephone;
    }
}
```

3. 作成した Java インタフェースおよびユーザ定義のデータ型クラスをコンパイルします。

注意事項

ここで作成した Java インタフェースおよびユーザ定義のデータ型クラスは，WSDL を生成するためだけのもので，WSDL 生成後は使用しません。SOAP アプリケーションとして使用するのは，「4.1.3 WSDL を生成する」で作成する Java インタフェースとデータ型クラスです。

4.1.3 WSDL を生成する

Java2WSDL コマンドを使用して，「4.1.2 Java インタフェースを作成する」で作成したインタフェースおよびユーザ定義のデータ型クラスから，SOAP アプリケーションの WSDL を生成します。

Java2WSDL コマンドの指定例を示します。

```
Java2WSDL.bat -l http://localhost:8080/WebApp1/services/UserInfo localhost.UserInfo
```

このコマンドを実行すると，カレントディレクトリに UserInfo.wsdl という名前の WSDL が生成されます。

この例では，-l オプションで SOAP サービスのエンドポイント URL を指定していますが，必要に応じてほかのオプションも使用してください。オプションについては，「9.1 Java2WSDL コマンド（WSDL の生成）」を参照してください。

この操作によって，生成された WSDL の例を次に示します。背景色付きの太字部分はこの操作によって設定された情報です。

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions name="UserInfo" targetNamespace="http://localhost"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:impl="http://localhost-impl"
    xmlns:intf="http://localhost"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
```

```
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <types>
    <schema targetNamespace="http://localhost"
        xmlns="http://www.w3.org/2001/XMLSchema">
      <complexType name="UserData">
        <sequence>
          <element name="name" nillable="true" type="xsd:string" />
          <element name="section" nillable="true" type="xsd:string" />
          <element name="telephone" nillable="true" type="xsd:string" />
        </sequence>
      </complexType>
      <element name="UserData" nillable="true" type="intf:UserData" />
    </schema>
  </types>
  <wsdl:message name="getUserDataResponse">
     <wsdl:part name="return" type="intf:UserData" />
  </wsdl:message>
  <wsdl:message name="getUserDataRequest">
     <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
  <wsdl:portType name="UserInfo">
    <wsdl:operation name="getUserData" parameterOrder="in0">
        <wsdl:input message="intf:getUserDataRequest" />
        <wsdl:output message="intf:getUserDataResponse" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="UserInfoSoapBinding" type="intf:UserInfo">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="getUserData">
       <soap:operation soapAction="" />
       <wsdl:input>
           <soap:body namespace="http://localhost" use="literal" />
       </wsdl:input>
       <wsdl:output>
           <soap:body namespace="http://localhost" use="literal" />
       </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="UserInfoService">
    <wsdl:port binding="intf:UserInfoSoapBinding" name="UserInfo">
       <soap:address location="http://localhost:8080/WebApp1/services/UserInfo" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

注意事項

- use 属性に"encoded"を指定した場合，SOAP 符号化規則は SOAP エンコーディングとなります。
- use 属性に"literal"を指定した場合，SOAP 符号化規則はリテラルエンコーディングとなります。
- SOAP 符号化規則については「1.4.1　SOAP の概要」を参照してください。

4.1.4　スケルトンおよびサービスデプロイ定義を生成する

WSDL2Java コマンドを使用して，生成された WSDL からサーバ側に必要なソースコードであるスケルトンを生成します。同時に，生成したソースコードを SOAP アプリケーションとして利用するためのサービスデプロイ定義を生成します。

WSDL2Java コマンドの指定例を示します。

```
WSDL2Java.bat -C validS -s UserInfo.wsdl
```

このコマンドを実行すると，カレントディレクトリに次に示すディレクトリおよびファイルが生成されます。

```
/                                カレントディレクトリ
 localhost/
   server-config.xml              サービスデプロイ定義
   UserData.java※                 ユーザ定義のデータ型クラス
   UserInfo.java                  リモートインタフェース
   UserInfoSoapBindingImpl.java   サービス実装クラス
```

注※

このクラスは，JavaBean 形式のクラスとして生成されます。

この例では，-s オプションでサーバ側のソースコードを生成しています。-C オプションは WSDL 検証機能を有効にするため，必ず指定してください。また，必要に応じてほかのオプションも指定してください。オプションについては，「9.2　WSDL2Java コマンド（ソースコードの生成）」を参照してください。

注意事項

生成されたソースコードのうち，処理を加えるのは UserInfoSoapBindingImpl.java だけです。UserInfoSoapBindingImpl.java 以外のクラスを変更しないでください。変更した場合，誤動作するおそれがあります。

4.1.5　サーバ側の処理を実装する

スケルトン生成と同時に生成された実装クラス（UserInfoSoapBindingImpl.java）に，サービスの実装を記述します。サービスの実装を記述したあと，スケルトンをコンパイルします。

生成された実装クラスのソースコードの例を次に示します。

```
package localhost;

public class UserInfoSoapBindingImpl implements localhost.UserInfo {
    public localhost.UserData getUserData(java.lang.String in0) throws java.rmi.RemoteExcept
ion {
        //
        // 実装を記述
        //
```

```
        return null;
    }
}
```

なお，これ以外のソースコードの例については，次に示すディレクトリに格納しているサンプルプログラムを参照してください。ただし，サンプルプログラムとして提供するのは，SOAP アプリケーション開発支援機能だけです。

```
<Application Serverのインストールディレクトリ>/c4web/samples
```

注意事項

WSDL2Java コマンドで生成されたスタブ・スケルトンファイルをコンパイルすると，次に示す警告メッセージが出力されることがあります。

```
注：入力ファイルの操作のうち，未チェックまたは安全ではないものがあります。
注：詳細については，-Xlint:unchecked オプションを指定して再コンパイルしてください。
```

この警告メッセージが出力された場合の対処は不要です。

なお，「入力ファイルの操作のうち」の部分は，コンパイル時の入力ファイル名が出力されることがあります。

4.1.6　アーカイブ（WAR ファイル）を作成する

J2EE サーバにデプロイするためのアーカイブを作成します。

1. WAR ファイルを作成する前に，ディレクトリ構成を整えます。

   ディレクトリ構成を次に示すように整えます。

   ```
   /
     WEB-INF/
       server-config.xml
       web.xml
       classes/
         localhost/
           UserData.class
           UserInfo.class
           UserInfoSoapBindingImpl.class
   ```

   注意事項

   UserData.class および UserInfo.class は，「4.1.4　スケルトンおよびサービスデプロイ定義を生成する」で作成したものを使用してください。「4.1.2　Java インタフェースを作成する」で作成したものは使用しないでください。

2. DD（web.xml）に情報を追加します。

   追加する情報については，「3.9.2　DD（web.xml）の記述」を参照してください。

3. WAR ファイルを作成します。

WAR ファイルの作成方法については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」を参照してください。

4.1.7　スタブを生成する

WSDL2Java コマンドを使用して，WSDL からクライアント側に必要なソースコードであるスタブを生成します。

WSDL2Java コマンドの指定例を示します。

```
WSDL2Java.bat -C validS UserInfo.wsdl
```

このコマンドを実行すると，カレントディレクトリに次に示すディレクトリおよびファイルが生成されます。

```
/                                カレントディレクトリ
 localhost/
  UserData.java※                  ユーザ定義のデータ型クラス
  UserInfo.java                   リモートインタフェース
  UserInfoService.java            サービスインタフェース
  UserInfoServiceLocator.java     サービスクラス
                                  （サービスへの接続情報を保持）
  UserInfoSoapBindingStub.java    スタブクラス
```

注※

このクラスは，JavaBean 形式のクラスとして生成されます。

-C オプションは WSDL 検証機能を有効にするため，必ず指定してください。また，必要に応じてほかのオプションも指定してください。オプションについては，「9.2　WSDL2Java コマンド（ソースコードの生成）」を参照してください。

注意事項

- 生成されたソースコードの内容は変更しないでください。変更した場合，誤動作するおそれがあります。
- 不正な WSDL 定義を指定してエラーになった場合，途中までソースファイルが出力されていることがあります。この場合，ソースファイルをカレントディレクトリから削除し，WSDL 定義を正しく修正したあとに再実行してください。
- WSDL2Java コマンドで生成されたスタブ・スケルトンファイルをコンパイルすると，次に示す警告メッセージが出力されることがあります。

  ```
  注：入力ファイルの操作のうち，未チェックまたは安全ではないものがあります。
  注：詳細については，-Xlint:unchecked オプションを指定して再コンパイルしてください。
  ```

  この警告メッセージが出力された場合の対処は不要です。

  なお，「入力ファイルの操作のうち」の部分は，コンパイル時の入力ファイル名が出力されることがあります。

4.1.8　クライアント側の処理を実装する

SOAP サービスを利用するクライアント側の処理を実装します。クライアント側の処理では，生成されたスタブのうち，UserInfoServiceLocator.java および UserInfo.java を利用します。スタブの内容および実装方法については，「3.7.1　スタブの使用」を参照してください。

ここでは，クライアント側の処理の実装例として，社員番号を入力とし，結果を標準出力に出す場合の例を示します。

```
package localhost;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import com.cosminexus.cws.management.*;

public class RPCSampleClient extends HttpServlet
{
    private static final String CONTENT_TYPE = "text/html; charset=Shift_JIS";

    // クライアント識別子
    private ClientID cltID = null;
    public void init() throws ServletException
    {
        // SOAPクライアントの開始
        cltID = Management.initializeClient();
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response) throws Serv
letException, IOException
    {
        // クライアント識別子と実行スレッドを関連づける
        Management.connectClientIDtoCurrentThread(cltID);

       // ブラウザからパラメタの取得
         String strUserNo = request.getParameter("UserNo");

        if (strUserNo == null) {
            strUserNo = "";
        }

        UserInfo ui = null;
        UserData ud = null;
        UserInfoServiceLocator uis = new UserInfoServiceLocator();
        try
        {
            ui = uis.getUserInfo();
        }
        catch (javax.xml.rpc.ServiceException e)
        {
            // エラー処理を記述
        }

        try
```

```
        {
            ud = ui.getUserData(strUserNo);
        }
        catch (java.rmi.RemoteException e)
        {
            // エラー処理を記述
        }

        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        // ブラウザに表示する内容を設定

        // クライアント識別子と実行スレッドの関連づけを解除する
        Management.disconnectClientIDtoCurrentThread(cltID);
    }

    public void destroy()
    {
        // SOAPクライアントの終了
        Management.finalizeClient(cltID);
    }
}
```

4.2　既存の Java クラスを利用して開発する場合

既存の Java クラスを利用して，RPC 形態の SOAP アプリケーションを開発します。開発例の概要と開発の流れを次に示します。

開発例の概要

社員番号，名前，所属，電話番号というユーザ情報を管理し，クライアントからの入力に対して，処理結果を返す SOAP アプリケーションを開発します。

開発例では，サービスクラスとユーザ定義のデータ型クラスは，作成済みの Java クラスを利用します。また，スタブを生成して，クライアント側の処理を実装します。

開発の流れ

1. SOAP アプリケーションの設計
2. サービスデプロイ定義の生成
3. アーカイブ（WAR ファイル）の作成
4. WSDL の生成
5. スタブの生成
6. クライアント側の処理の実装

開発の流れに沿って，各ステップの作業について説明します。

4.2.1　SOAP アプリケーションを設計する

RPC 形態の SOAP アプリケーションの仕様を設計します。開発例では，次に示すサービスを，既存の Java クラスを利用して開発します。

サービス名

UserInfo

メソッド名

getUserData

クライアントからの入力時およびクライアントへの出力時の属性について次に示します。出力時の属性は，UserData クラスが保持します。

表 4-3　入力時の属性（RPC で既存の Java クラスを利用する場合）

属性名	変数名	データ型
社員番号	in0	String

表 4-4　出力時の属性（RPC で既存の Java クラスを利用する場合）

属性名	変数名	データ型
名前	name	String
所属	section	String
電話番号	telephone	String

開発例で使用する，既存の Java クラス（サービスクラスとユーザ定義のデータ型クラス）を次に示します。

- サービスクラス

```
package localhost;

public class UserInfo {
    public UserData getUserData(String in0)
    {
        UserData ud = new UserData();

        //
        // 業務ロジックの呼び出しや，EJBの呼び出しなどの実装を記述
        //

        return ud;
    }
}
```

- ユーザ定義のデータ型クラス

```
package localhost;

public class UserData {
  private String name;
  private String section;
  private String telephone;

  public UserData(){}

  public void setName(String name)
  {
    this.name = name;
  }

  public String getName()
  {
    return this.name;
  }

  public void setSection(String section)
  {
    this.section = section;
  }

  public String getSection()
  {
```

```
    return this.section;
  }

  public void setTelephone(String telephone)
  {
    this.telephone = telephone;
  }

  public String getTelephone()
  {
    return this.telephone;
  }
}
```

4.2.2　サービスデプロイ定義を生成する

Java2WSDD コマンドを使用して，既存の Java クラスを SOAP アプリケーションとして利用するためのサービスデプロイ定義を生成します。

Java2WSDD コマンドの指定例を示します。

```
Java2WSDD.bat localhost.UserInfo
```

このコマンドを実行すると，カレントディレクトリにサービスデプロイ定義（server-config.xml）が生成されます。

この例では，オプションを指定していませんが，必要に応じてオプションを指定してください。オプションについては，「9.3　Java2WSDD コマンド（サービスデプロイ定義の生成）」を参照してください。

4.2.3　アーカイブ（WAR ファイル）を作成する

J2EE サーバにデプロイするためのアーカイブを作成します。

操作手順

1. WAR ファイルを作成する前に，ディレクトリ構成を整えます。

   ディレクトリ構成を次に示すように整えます。

```
/
  WEB-INF/
    server-config.xml
    web.xml
    classes/
      localhost/
        UserData.class
        UserInfo.class
```

2. DD（web.xml）に情報を追加します。

追加する情報については，「3.9.2　DD（web.xml）の記述」を参照してください。

3. WAR ファイルを作成します。

WAR ファイルの作成方法については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」を参照してください。

4.2.4　WSDL を生成する

Java2WSDL コマンドを使用して，既存の Java クラスを SOAP サービスとして公開するための WSDL を生成します。WSDL は，クライアント側の実装に必要なファイルを生成するために必要です。

Java2WSDL コマンドの指定例を示します。

```
Java2WSDL.bat -l http://localhost:8080/WebApp1/services/UserInfo localhost.UserInfo
```

このコマンドを実行すると，カレントディレクトリに UserInfo.wsdl という名前の WSDL が生成されます。

この例では，-l オプションで SOAP サービスのエンドポイント URL を指定していますが，必要に応じてほかのオプションも指定してください。オプションについては，「9.1　Java2WSDL コマンド（WSDL の生成）」を参照してください。

4.2.5　スタブを生成する

WSDL2Java コマンドを使用して，生成した WSDL からクライアント側に必要なソースコードであるスタブを生成します。

WSDL2Java コマンドの指定例を示します。

```
WSDL2Java.bat -C validS UserInfo.wsdl
```

このコマンドを実行すると，カレントディレクトリに次に示すディレクトリおよびファイルが生成されます。

```
/                                カレントディレクトリ
 localhost/
  UserData.java※                  ユーザ定義のデータ型クラス
  UserInfo.java                  リモートインタフェース
  UserInfoService.java           サービスインタフェース
  UserInfoServiceLocator.java    サービスクラス
                                 （サービスへの接続情報を保持）
  UserInfoSoapBindingStub.java   スタブクラス
```

注※

このクラスは，JavaBean 形式のクラスとして生成されます。

-C オプションは WSDL 検証機能を有効にするため，必ず指定してください。また，必要に応じてほかのオプションも指定してください。オプションについては，「9.2　WSDL2Java コマンド（ソースコードの生成）」を参照してください。

注意事項

- 生成されたソースコードの内容は変更しないでください。変更した場合，誤動作するおそれがあります。
- 不正な WSDL 定義を指定してエラーになった場合，途中までソースファイルが出力されていることがあります。この場合，ソースファイルをカレントディレクトリから削除し，WSDL 定義を正しく修正したあとに再実行してください。
- WSDL2Java コマンドで生成されたスタブおよびスケルトンファイルをコンパイルすると，次に示す警告メッセージが出力されることがあります。

```
注：入力ファイルの操作のうち，未チェックまたは安全ではないものがあります。
注：詳細については，-Xlint:unchecked オプションを指定して再コンパイルしてください。
```

  この警告メッセージが出力された場合の対処は不要です。

  なお，「入力ファイルの操作のうち」の部分は，コンパイル時の入力ファイル名が出力されることがあります。

4.2.6　クライアント側の処理を実装する

SOAP サービスを利用するクライアント側の処理を実装します。クライアント側の処理では，生成されたスタブのうち，UserInfoServiceLocator.java および UserInfo.java を利用します。スタブの内容および実装方法については，「3.7.1　スタブの使用」を参照してください。

クライアント側の処理の実装例については，「4.1.8　クライアント側の処理を実装する」を参照してください。

4.3　既存の EJB を利用して開発する場合

EJB を利用して，RPC 形態の SOAP アプリケーションを開発します。開発例の概要と開発の流れを次に示します。

開発例の概要

社員番号，名前，所属，電話番号というユーザ情報を管理し，クライアントからの入力に対して，処理結果を返す SOAP アプリケーションを開発します。

開発例では，SOAP アプリケーションのサービスに EJB（Stateless Session Bean）を利用します。また，スタブを生成して，クライアント側の処理を実装します。

開発の流れ

1. SOAP アプリケーションの設計
2. EJB 呼び出し環境の設定
3. サービスデプロイ定義の生成
4. アーカイブ（WAR ファイル）の作成
5. WSDL の生成
6. スタブの生成
7. クライアント側の処理の実装

開発の流れに沿って，各ステップの作業について説明します。

4.3.1　SOAP アプリケーションを設計する

EJB を利用した SOAP アプリケーションの仕様を設計します。開発例では，次に示すサービスの仕様を持つ Stateless Session Bean を，SOAP アプリケーションとして利用する場合について示します。なお，localhost.UserData という名前のデータ型クラスは，Stateless Session Bean 側で定義されているものとします。

サービス名

UserInfo

メソッド名

getUserData

クライアントからの入力時およびクライアントへの出力時の属性について次に示します。出力時の属性は，UserData クラスが保持します。

表 4-5　入力時の属性（既存の EJB を利用する場合）

属性名	変数名	データ型
社員番号	user_no	String

表 4-6　出力時の属性（既存の EJB を利用する場合）

属性名	変数名	データ型
名前	name	String
所属	section	String
電話番号	telephone	String

ホームインタフェース

userinfo.SLBUserInfoHome

リモートインタフェース

userinfo.SLBUserInfo

Enterprise Bean

userinfo.SLBUserInfoBean

データ型クラス

localhost.UserData

JNDI 名前空間

HITACHI_EJB/SERVERS/MyServer/EJB/UserInfoBean/SLBUserInfo

4.3.2　EJB 呼び出し環境を設定する

J2EE アプリケーションの RMI-IIOP インタフェースおよび RMI-IIOP スタブを取得します。RMI-IIOP インタフェースおよび RMI-IIOP スタブの取得方法については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(EJB コンテナ)」を参照してください。

4.3.3　サービスデプロイ定義を生成する

Java2WSDD コマンドを使用して，読み込んだ RMI-IIOP インタフェースを SOAP アプリケーションとして利用するためのサービスデプロイ定義を生成します。

Java2WSDD コマンドの指定例を示します。

```
Java2WSDD.bat -p EJB -B userinfo.SLBUserInfoBean -H userinfo.SLBUserInfoHome  -J HITACHI_EJB
/SERVERS/MyServer/EJB/UserInfoBean/SLBUserInfo userinfo.SLBUserInfo
```

このコマンドを実行すると，カレントディレクトリにサービスデプロイ定義（server-config.xml）が生成されます。

この例で指定しているオプションの意味を示します。

- -p オプションでプロバイダ種別を指定します。EJB を指定することで，EJB のサービスデプロイ定義が生成されます。
- -B オプションで Bean クラスを指定します。
- -H オプションでホームインタフェースを指定します。
- -J オプションで JNDI 名前空間を指定します。

必要に応じてほかのオプションも指定してください。オプションについては，「9.3　Java2WSDD コマンド（サービスデプロイ定義の生成）」を参照してください。

注意事項

EJB を SOAP サービスのエンドポイントとして利用する場合で，Stateful Session Bean および Entity Bean を利用する場合は，DeployScope を「Session」にして，クライアント定義ファイルの「app_maintainsession」オプションを true にしてください。なお，次の場合は，SOAP サービスへのリクエストごとに EJB オブジェクトが生成されるためメモリ不足またはリソース不足が発生するおそれがあります。

- DeployScope を「Request」にしたとき
- DeployScope を「Session」にして，かつクライアント定義ファイルの「app_maintainsession」オプションが false のとき

このセッションは，マニュアル「アプリケーションサーバ 機能解説 拡張編」で記述している「J2EE サーバ間のセッション情報の引き継ぎ」には対応していません。

4.3.4　アーカイブ（WAR ファイル）を作成する

J2EE サーバにデプロイするためのアーカイブを作成します。

1. WAR ファイルを作成する前に，ディレクトリ構成を整えます。

   ディレクトリ構成を次に示すように整えます。

```
/
  WEB-INF/
    server-config.xml
    web.xml
    classes/
    lib/
      *.jar※1
      stubs.jar※2
```

注※1

「4.3.2　EJB 呼び出し環境を設定する」で取得した RMI-IIOP インタフェースです。

注※2

「4.3.2　EJB 呼び出し環境を設定する」で取得した RMI-IIOP スタブです。

2. DD（web.xml）に情報を追加します。

追加する情報については，「3.9.2　DD（web.xml）の記述」を参照してください。

3. WAR ファイルを作成します。

WAR ファイルの作成方法については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」を参照してください。

4.3.5　WSDL を生成する

Java2WSDL コマンドを使用して，既存の EJB サービスを SOAP サービスとして公開するための WSDL を生成します。WSDL は，クライアント側の実装に必要なファイルを生成するために必要です。

Java2WSDL コマンドの指定例を次に示します。

```
Java2WSDL.bat -l http://localhost:8080/WebApp1/services/SLBUserInfo userinfo.SLBUserInfo
```

このコマンドを実行すると，カレントディレクトリに SLBUserInfo.wsdl という名前の WSDL が生成されます。

この例では，-l オプションで SOAP サービスのエンドポイント URL を指定していますが，必要に応じてほかのオプションも指定してください。オプションについては，「9.1　Java2WSDL コマンド（WSDL の生成）」を参照してください。

4.3.6　スタブを生成する

WSDL2Java コマンドを使用して，WSDL からクライアント側に必要なソースコードであるスタブを生成します。

WSDL2Java コマンドの指定例を示します。

```
WSDL2Java.bat -C validS SLBUserInfo.wsdl
```

このコマンドを実行すると，カレントディレクトリに次に示すディレクトリおよびファイルが生成されます。

```
/                                カレントディレクトリ
 userinfo/
  UserData.java※                 ユーザ定義のデータ型クラス
```

```
SLBUserInfo.java                  リモートインタフェース
SLBUserInfoService.java           サービスインタフェース
SLBUserInfoServiceLocator.java    サービスクラス
                                  （サービスへの接続情報を保持）
SLBUserInfoSoapBindingStub.java   スタブクラス
```

注※

このクラスは，JavaBean 形式のクラスとして生成されます。

-C オプションは WSDL 検証機能を有効にするため，必ず指定してください。また，必要に応じてほかのオプションも指定してください。オプションについては，「9.2　WSDL2Java コマンド（ソースコードの生成）」を参照してください。

注意事項

- 生成されたソースコードの内容は変更しないでください。変更した場合，誤動作するおそれがあります。
- 不正な WSDL 定義を指定してエラーになった場合，途中までソースファイルが出力されていることがあります。この場合，ソースファイルをカレントディレクトリから削除し，WSDL 定義を正しく修正したあとに再実行してください。
- WSDL2Java コマンドで生成されたスタブ・スケルトンファイルをコンパイルすると，次に示す警告メッセージが出力されることがあります。

  ```
  注：入力ファイルの操作のうち，未チェックまたは安全ではないものがあります。
  注：詳細については，-Xlint:unchecked オプションを指定して再コンパイルしてください。
  ```

  この警告メッセージが出力された場合の対処は不要です。

  なお，「入力ファイルの操作のうち」の部分は，コンパイル時の入力ファイル名が出力されることがあります。

4.3.7　クライアント側の処理を実装する

実装用の新規クラスを作成し，クライアント側の処理を実装します。次に，クライアント側の処理の実装例を示します。

```
package userinfo;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import com.cosminexus.cws.management.*;

public class RPCSampleClient extends HttpServlet
{
    private static final String CONTENT_TYPE = ”text/html; charset=Shift_JIS”;
    private ClientID cltID = null;
    public void init() throws ServletException
```

```
    {
        cltID = Management.initializeClient();
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response) throws Serv
letException, IOException
    {
        // クライアントIDとスレッドの関連づけを行なう
        Management.connectClientIDtoCurrentThread(cltID);

        // ブラウザからパラメタの取得
        String strUserNo = request.getParameter("UserNo");

        if (strUserNo == null) {
            strUserNo = "";
        }

        SLBUserInfo ui = null;
        UserData ud = null;

        SLBUserInfoServiceLocator uis = new SLBUserInfoServiceLocator();

        try
        {
            ui = uis.getUserInfo();
        }
        catch (javax.xml.rpc.ServiceException e)
        {
            // エラー処理を記述
        }

        try
        {
            ud = ui.getUserData(strUserNo);
        }
        catch (java.rmi.RemoteException e)
        {
            // エラー処理を記述
        }

        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        // ブラウザに表示する内容を設定

        try
        {
            //EJBオブジェクトの削除※
            ui.remove();
        }
        catch (java.rmi.RemoteException e)
        {
            // エラー処理を記述
        }
        // クライアントとスレッドの関連づけを解除する
        Management.disconnectClientIDtoCurrentThread(cltID);
    }

    public void destroy()
```

```
    {
        Management.finalizeClient(cltID);
    }
}
```

注※

ui.remove（EJB オブジェクトの削除）の呼び出し処理は，DeployScope が「Session」の SOAP サービスを呼び出すクライアントの，セッションを終了したい個所に実装してください。

4.4　添付ファイルを使用して開発する場合

RPC 形態の添付ファイルを使用した SOAP アプリケーションを開発します。なお，添付ファイル付き SOAP メッセージを受信すると，メモリ使用量を抑制するために退避ファイルが作成されることがあります。退避ファイルの詳細については，「付録 B　添付ファイル使用時に作成される退避ファイル」を参照してください。

開発例の概要と開発の流れを次に示します。

開発例の概要

社員番号，名前，所属というユーザ情報を管理し，クライアントからの入力に対して，処理結果を返す SOAP アプリケーションを新規に開発します。

開発例では，RPC 形態の添付ファイルを使用します。また，スタブを生成して，クライアント側の処理を実装します。

開発の流れ

1. SOAP アプリケーションの設計
2. Java インタフェースの作成
3. WSDL の生成
4. スケルトンおよびサービスデプロイ定義の生成
5. サーバ側の処理の実装
6. アーカイブ（WAR ファイル）の作成
7. スタブの生成
8. クライアント側の処理の実装

開発の流れに沿って，各ステップの作業について説明します。

4.4.1　SOAP アプリケーションを設計する

RPC 形態の添付ファイルを使用した SOAP アプリケーションの仕様を設計します。開発例では，次のサービス名およびメソッド名を使用します。

サービス名

UserInfo

メソッド名

getUserData

クライアントからの入力時およびクライアントへの出力時の属性について次に示します。出力時の属性は，UserData クラスが保持します。

表 4-7　入力時の属性（RPC 形態の添付ファイルを使用した場合）

属性名	変数名	データ型
社員番号	in0	java.lang.String
顔写真	in1	javax.activation.DataHandler

表 4-8　出力時の属性（RPC 形態の添付ファイルを使用した場合）

属性名	変数名	データ型
登録確認メッセージ	message	java.lang.String
名前	name	java.lang.String
所属	section	java.lang.String

4.4.2　Java インタフェースを作成する

設計した SOAP アプリケーションの WSDL を作成するために，Java インタフェース，およびユーザ定義のデータ型クラスを作成します。

1. Java インタフェースを作成します。

   SOAP アプリケーションで公開するメソッドを Java インタフェースに定義します。

   メソッドを定義した Java インタフェースの例を次に示します。

```
//UserInfo.java

package localhost;

public interface UserInfo extends java.rmi.Remote{
    public localhost.UserData getUserData(java.lang.String in0,
        javax.activation.DataHandler in1)throws java.rmi.RemoteException;
}
```

2. ユーザ定義のデータ型クラスを作成します。

   Java インタフェースで使用するユーザ定義のデータ型クラスを作成します。

   ユーザ定義のデータ型クラスの例を次に示します。

```
//UserData.java

package localhost;

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() throws java.rmi.RemoteException{
        return this.message;
    }

    public void setMessage(java.lang.String message) throws java.rmi.RemoteException{
        this.message = message;
    }

    public java.lang.String getName() throws java.rmi.RemoteException{
        return this.name;
    }

    public void setName(java.lang.String name) throws java.rmi.RemoteException{
        this.name = name;
    }

    public java.lang.String getSection() throws java.rmi.RemoteException{
        return this.section;
    }

    public void setSection(java.lang.String section) throws java.rmi.RemoteException{
        this.section = section;
    }
}
```

3. 作成した Java インタフェースおよびユーザ定義のデータ型クラスをコンパイルします。

注意事項

ここで作成した Java インタフェースおよびユーザ定義のデータ型クラスは，WSDL を生成するためだけのもので，WSDL 生成後は使用しません。SOAP アプリケーションとして使用するのは，「4.1.3 WSDL を生成する」で作成する Java インタフェースとデータ型クラスです。

4.4.3 WSDL を生成する

Java2WSDL コマンドを使用して，「4.4.2 Java インタフェースを作成する」で作成したインタフェースおよびユーザ定義のデータ型クラスから，SOAP アプリケーションの WSDL を生成します。

RPC 形態の添付ファイルは，document/literal に対応した SOAP アプリケーションだけで使用できます。そのため，Java2WSDL コマンドの実行時には，-z オプションに「DOCUMENT」，-u オプションに「LITERAL」を指定してください。

Java2WSDL コマンドの指定例を示します。

```
Java2WSDL.bat -l http://localhost:8080/WebApp1/services/UserInfo -z DOCUMENT -u LITERAL loca
lhost.UserInfo
```

このコマンドを実行すると，カレントディレクトリに UserInfo.wsdl という名前の WSDL が生成されます。

この例では，-l オプションで SOAP サービスのエンドポイント URL を指定していますが，必要に応じてほかのオプションも使用してください。オプションについては，「9.1　Java2WSDL コマンド（WSDL の生成）」を参照してください。

この操作によって，生成された WSDL の例を次に示します。背景色付きの太字部分はこの操作によって設定された情報です。

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://localhost"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://localhost-impl"
    xmlns:intf="http://localhost"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsi="http://ws-i.org/profiles/basic/1.1/xsd"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <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>
      <element name="getUserDataResponse">
        <complexType>
          <sequence>
            <element name="getUserDataReturn" type="intf:UserData"/>
          </sequence>
        </complexType>
      </element>
      <complexType name="UserData">
        <sequence>
          <element name="message" nillable="true" type="xsd:string"/>
          <element name="name" nillable="true" type="xsd:string"/>
          <element name="section" nillable="true" type="xsd:string"/>
        </sequence>
      </complexType>
    </schema>
  </wsdl:types>
  <wsdl:message name="getUserDataRequest">
    <wsdl:part element="intf:getUserData" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="getUserDataResponse">
    <wsdl:part element="intf:getUserDataResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:portType name="UserInfo">
    <wsdl:operation name="getUserData">
      <wsdl:input message="intf:getUserDataRequest" name="getUserDataRequest"/>
      <wsdl:output message="intf:getUserDataResponse" name="getUserDataResponse"/>
    </wsdl:operation>
  </wsdl:portType>
```

```
  <wsdl:binding name="UserInfoSoapBinding" type="intf:UserInfo">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="getUserData">
      <soap:operation soapAction=""/>
      <wsdl:input name="getUserDataRequest">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="getUserDataResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="UserInfoService">
    <wsdl:port binding="intf:UserInfoSoapBinding" name="UserInfo">
      <soap:address location="http://localhost:8080/WebApp1/services/UserInfo"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

注意事項

- use 属性に"literal"を指定した場合，SOAP 符号化規則はリテラルエンコーディングとなります。
- SOAP 符号化規則については「1.4.1　SOAP の概要」を参照してください。

4.4.4　スケルトンおよびサービスデプロイ定義を生成する

WSDL2Java コマンドを使用して，生成された WSDL からサーバ側に必要なソースコードであるスケルトンを生成します。同時に，生成したソースコードを SOAP アプリケーションとして利用するためのサービスデプロイ定義を生成します。

WSDL2Java コマンドの指定例を示します。

```
WSDL2Java.bat -C validS -s UserInfo.wsdl
```

このコマンドを実行すると，カレントディレクトリに次に示すディレクトリおよびファイルが生成されます。

```
/                                カレントディレクトリ
 localhost/
   server-config.xml              サービスデプロイ定義
   UserData.java※                 ユーザ定義のデータ型クラス
   UserInfo.java                  リモートインタフェース
   UserInfoSoapBindingImpl.java   サービス実装クラス
```

注※

このクラスは，JavaBean 形式のクラスとして生成されます。

この例では，-s オプションでサーバ側のソースコードを生成しています。-C オプションは WSDL 検証機能を有効にするため，必ず指定してください。また，必要に応じてほかのオプションも指定してください。オプションについては，「9.2　WSDL2Java コマンド（ソースコードの生成）」を参照してください。

注意事項

生成されたソースコードのうち，処理を加えるのは UserInfoSoapBindingImpl.java だけです。UserInfoSoapBindingImpl.java 以外のクラスを変更しないでください。変更した場合，誤動作するおそれがあります。

4.4.5　サーバ側の処理を実装する

スケルトン生成と同時に生成された実装クラス（UserInfoSoapBindingImpl.java）に，サービスの実装を記述します。サービスの実装を記述したあと，スケルトンをコンパイルします。

生成された実装クラスのソースコードの例を次に示します。

```
package localhost;

public class UserInfoSoapBindingImpl implements localhost.UserInfo {
    public localhost.UserData getUserData( java.lang.String id , javax.activation.DataHandle
r photo) throws java.rmi.RemoteException {
        //
        // 実装を記述
        //

        return null;
    }
}
```

なお，これ以外のソースコードの例については，次に示すディレクトリに格納しているサンプルプログラムを参照してください。ただし，サンプルプログラムとして提供するのは，SOAP アプリケーション開発支援機能だけです。

```
<Application Serverのインストールディレクトリ>/c4web/samples
```

注意事項

WSDL2Java コマンドで生成されたスタブ・スケルトンファイルをコンパイルすると，次に示す警告メッセージが出力されることがあります。

```
注：入力ファイルの操作のうち，未チェックまたは安全ではないものがあります。
注：詳細については，-Xlint:unchecked オプションを指定して再コンパイルしてください。
```

この警告メッセージが出力された場合の対処は不要です。

なお，「入力ファイルの操作のうち」の部分は，コンパイル時の入力ファイル名が出力されることがあります。

4.4.6　アーカイブ（WAR ファイル）を作成する

J2EE サーバにデプロイするためのアーカイブを作成します。

1. WAR ファイルを作成する前に，ディレクトリ構成を整えます。

ディレクトリ構成を次に示すように整えます。

```
/
  WEB-INF/
    server-config.xml
    web.xml
    classes/
      localhost/
        UserData.class
        UserInfo.class
        UserInfoSoapBindingImpl.class
```

注意事項

UserData.class および UserInfo.class は，「4.4.4　スケルトンおよびサービスデプロイ定義を生成する」で作成したものを使用してください。「4.4.2　Java インタフェースを作成する」で作成したものは使用しないでください。

2. DD（web.xml）に情報を追加します。

追加する情報については，「3.9.2　DD（web.xml）の記述」を参照してください。

3. WAR ファイルを作成します。

WAR ファイルの作成方法については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」を参照してください。

4.4.7　スタブを生成する

WSDL2Java コマンドを使用して，WSDL からクライアント側に必要なソースコードであるスタブを生成します。

WSDL2Java コマンドの指定例を示します。

```
WSDL2Java.bat -C validS UserInfo.wsdl
```

このコマンドを実行すると，カレントディレクトリに次に示すディレクトリおよびファイルが生成されます。

```
/                                カレントディレクトリ
 localhost/
  UserData.java※                  ユーザ定義のデータ型クラス
  UserInfo.java                  リモートインタフェース
  UserInfoService.java           サービスインタフェース
  UserInfoServiceLocator.java    サービスクラス
                                 （サービスへの接続情報を保持）
  UserInfoSoapBindingStub.java   スタブクラス
```

注※

このクラスは，JavaBean 形式のクラスとして生成されます。

-C オプションは WSDL 検証機能を有効にするため，必ず指定してください。また，必要に応じてほかのオプションも指定してください。オプションについては，「9.2　WSDL2Java コマンド（ソースコードの生成）」を参照してください。

注意事項

- 生成されたソースコードの内容は変更しないでください。変更した場合，誤動作するおそれがあります。
- 不正な WSDL 定義を指定してエラーになった場合，途中までソースファイルが出力されていることがあります。この場合，ソースファイルをカレントディレクトリから削除し，WSDL 定義を正しく修正したあとに再実行してください。
- WSDL2Java コマンドで生成されたスタブ・スケルトンファイルをコンパイルすると，次に示す警告メッセージが出力されることがあります。

  ```
  注：入力ファイルの操作のうち，未チェックまたは安全ではないものがあります。
  注：詳細については，-Xlint:unchecked オプションを指定して再コンパイルしてください。
  ```

  この警告メッセージが出力された場合の対処は不要です。

  なお，「入力ファイルの操作のうち」の部分は，コンパイル時の入力ファイル名が出力されることがあります。

4.4.8　クライアント側の処理を実装する

SOAP サービスを利用するクライアント側の処理を実装します。

ここでは，クライアント側の処理の実装例として，社員番号と顔写真を入力とし，結果を標準出力に出す場合の例を示します。

```
package localhost;

import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import com.cosminexus.cws.management.*;

public class  RPCAttachmentsSampleClient extends HttpServlet
{
    private static final String CONTENT_TYPE = ”text/html; charset=Shift_JIS”;

    // クライアント識別子
    private ClientID cltID = null;
    public void init() throws ServletException
    {
        cltID = Management.initializeClient();
    }
```

```
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws Serv
letException, IOException
    {
        Management.connectClientIDtoCurrentThread(cltID);

       // ブラウザからパラメタの取得

        // 社員番号の取得
        String strUserNo = request.getParameter("UserNo");
        if (strUserNo == null) {
            strUserNo = "";
        }


        String strContextPath =
           (String)getServletConfig().getServletContext().getRealPath("/");
        String filepath = strContextPath + "attachment.txt";
        File imagefile = new File(filepath);
        FileDataSource fdSource = new FileDataSource(imagefile);
        DataHandler dhandler = new DataHandler(fdSource);


        UserInfo ui = null;
        UserData ud = null;
        UserInfoServiceLocator uis = new UserInfoServiceLocator();
        try
        {
            ui = uis.getUserInfo();
        }
        catch (javax.xml.rpc.ServiceException e)
        {
            // エラー処理を記述
        }

        try
        {
        ud = ui.getUserData(strUserNo, dhandler);
        }
        catch (java.rmi.RemoteException e)
        {
            // エラー処理を記述
        }

        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        // ブラウザに表示する内容を設定

        Management.disconnectClientIDtoCurrentThread(cltID);
    }

    public void destroy()
    {
        Management.finalizeClient(cltID);
    }
}
```

4.5　DII を使用して開発する場合

次に示す場合の WSDL の例およびクライアント側の処理の実装例を示します。

- WSDL をインポートしない場合
- WSDL をインポートする場合
- INOUT パラメタを使用する場合
- ユーザ定義のデータ型クラスを使用する場合
- 配列を使用する場合

4.5.1　WSDL をインポートしない場合

WSDL をインポートしない場合の WSDL の例および DII のコーディング例を示します。なお，WSDL の例では次の情報が定義されています。

- WSDL のサービス名
  名前空間 URI…http://example.com
  ローカル名…StockQuoteService
- WSDL のポート名
  名前空間 URI…http://example.com
  ローカル名…StockQuote
- WSDL のオペレーション名
  名前空間 URI…http://example.com
  ローカル名…getLastTradePrice
- オペレーションのパラメタ名…in0

(1)　WSDL の例（StockQuote.wsdl）

```
<wsdl:definitions
targetNamespace="http://example.com"
xmlns:intf="http://example.com"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<wsdl:message name="getLastTradePriceResponse">
  <wsdl:part name="getLastTradePriceReturn" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="getLastTradePriceRequest">
<wsdl:part name="in0" type="xsd:string"/>
  </wsdl:message>
```

```
<wsdl:portType name="StockQuote">
<wsdl:operation name="getLastTradePrice" parameterOrder="in0">
<wsdl:input message="intf:getLastTradePriceRequest" name="getLastTradePriceRequest"/>
<wsdl:output message="intf:getLastTradePriceResponse" name="getLastTradePriceResponse"/>
</wsdl:operation>
 </wsdl:portType>

<wsdl:binding name="StockQuoteSoapBinding" type="intf:StockQuote">
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="getLastTradePrice">
<soap:operation soapAction=""/>
<wsdl:input name="getLastTradePriceRequest">
<soap:body namespace="http://example.com" use="literal"/>
</wsdl:input>
<wsdl:output name="getLastTradePriceResponse">
<soap:body namespace="http://example.com" use="literal"/>
</wsdl:output>
</wsdl:operation>
 </wsdl:binding>
<wsdl:service name="StockQuoteService">
<wsdl:port binding="intf:StockQuoteSoapBinding" name="StockQuote">
<soap:address location="http://localhost:8080/StockQuote/services/StockQuote"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

(2) DII のコーディング例

```
//SOAPクライアントの開始
ClientID cltID = Management.initializeClient();

// クライアント識別子と実行スレッドを関連づける
Management.connectClientIDtoCurrentThread(cltID);

try {
// Serviceオブジェクトの生成
QName serviceName = new QName(
        "http://example.com",
        "StockQuoteService");
URL wsdlDocumentLocation = null;
try {
        wsdlDocumentLocation = new URL("file:///C:/foo/StockQuote.wsdl");
} catch(MalFormedURLException e) {
// 異常時の処理
}

ServiceFactory factory = null;
Service service = null;
try {
factory = ServiceFactory.newInstance();
        service = factory.createService(wsdlDocumentLocation, serviceName)
} catch(ServiceException e) {
// 異常時の処理
}

// Callオブジェクトの生成
```

```
QName portName = new QName(
        "http://example.com",
        "StockQuote");
QName operationName = new QName {
        "http://example.com",
        "getLastTradePrice");

javax.xml.rpc.Call call = null;
try {
call = service.createCall(portName, operationName);
} catch(ServiceException e) {
        // 異常時の処理
}

// サービス呼び出し
// 入力パラメタを生成
Class inputClass = ((com.cosminexus.cws.xml.rpc.Call) call ).getParameterClassByName("in0");
Object in0 = null;
in(inputClass == String.class) {
        in0 = "foo";
}
Object inParams = new Object[] {in0};

Object ret = null;
try {
        Object ret = call.invoke(inParams);
} catch(RemoteException e) {
// 異常時の処理
Throwable cause = e.getCause();
if(cause != null) {
// 原因例外が存在する場合の処理
             System.out.println(cause.getMessage());
}
}

//戻り値の処理
Class retClass = ret.getClass();
int price = 0;
if(retClass == Integer.class) {
        price = ((Integer)ret).intValue();
}
System.out.println("price : " + price);

} finally {
   // クライアント識別子と実行スレッドの関連づけを解除する
   Management.disconnectClientIDtoCurrentThread(cltID);

   // SOAPクライアントの終了
   Management.finalizeClient(cltID);
}
```

例では，javax.xml.rpc.Call クラスと com.cosminexus.cws.xml.rpc.Call クラスという二つの Call クラスが使われています。Java では，別パッケージに存在する，同じ名前のクラスを import 宣言に書けないため，同じクラス内で javax.xml.rpc.Call クラスと com.cosminexus.cws.xml.rpc.Call クラスを使用する場合には，例のように完全修飾名で指定する必要があります。

4.5.2　WSDL をインポートする場合

WSDL のインタフェース定義（portType 要素）をインポートする場合の WSDL の例および DII のコーディング例を示します。なお，WSDL の例では次の情報が定義されています。

- WSDL のサービス名
  名前空間 URI…http://example.com
  ローカル名…StockQuoteService
- WSDL のポート名
  名前空間 URI…http://example.com
  ローカル名…StockQuote
- WSDL のオペレーション名
  名前空間 URI…http://portType.example.com
  ローカル名…getLastTradePrice
- オペレーションのパラメタ名…in0

(1)　WSDL の例（StockQuote.wsdl）

```
<wsdl:definitions
targetNamespace="http://example.com"
xmlns:intf="http://example.com"
xmlns:portType="http://portType.example.com"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <wsdl:import
      namespace="http://portType.example.com"
      location="StockQuotePortType.wsdl"/>

<wsdl:binding name="StockQuoteSoapBinding" type="portType:StockQuote">
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="getLastTradePrice">
<soap:operation soapAction=""/>
<wsdl:input name="getLastTradePriceRequest">
<soap:body namespace="http://example.com" use="literal"/>
</wsdl:input>
<wsdl:output name="getLastTradePriceResponse">
<soap:body namespace="http://example.com" use="literal"/>
</wsdl:output>
</wsdl:operation>
 </wsdl:binding>
<wsdl:service name="StockQuoteService">
<wsdl:port binding="intf:StockQuoteSoapBinding" name="StockQuote">
<soap:address location="http://localhost:8080/StockQuote/services/StockQuote"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

wsdl:import 要素の location 属性には，インポートする WSDL への相対パスを記述してください。

(2) インポートされる WSDL の例（StockQuotePortType.wsdl）

```
<wsdl:definitions
targetNamespace="http://portType.example.com"
xmlns:intf="http://portType.example.com"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<wsdl:message name="getLastTradePriceResponse">
  <wsdl:part name="getLastTradePriceReturn" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="getLastTradePriceRequest">
<wsdl:part name="in0" type="xsd:string"/>
  </wsdl:message>
<wsdl:portType name="StockQuote">
<wsdl:operation name="getLastTradePrice" parameterOrder="in0">
<wsdl:input message="intf:getLastTradePriceRequest" name="getLastTradePriceRequest"/>
<wsdl:output message="intf:getLastTradePriceResponse" name="getLastTradePriceResponse"/>
</wsdl:operation>
 </wsdl:portType>
</wsdl:definitions>
```

(3) Dll のコーディング例

```
//SOAPクライアントの開始
ClientID cltID = Management.initializeClient();

// クライアント識別子と実行スレッドを関連づける
Management.connectClientIDtoCurrentThread(cltID);

try {
// Serviceオブジェクトの生成
QName serviceName = new QName(
        "http://example.com",
        "StockQuoteService");
URL wsdlDocumentLocation = null;
try {
        wsdlDocumentLocation = new URL("file:///C:/foo/StockQuote.wsdl");
} catch(MalFormedURLException e) {
// 異常時の処理
}

ServiceFactory factory = null;
Service service = null;
try {
factory = ServiceFactory.newInstance();
        service = factory.createService(wsdlDocumentLocation, serviceName)
} catch(ServiceException e) {
// 異常時の処理
}

// Callオブジェクトの生成
```

```
QName portName = new QName(
        "http://example.com",
        "StockQuote");
QName operationName = new QName {
        "http://portType.example.com",
        "getLastTradePrice");

javax.xml.rpc.Call call = null;
try {
call = service.createCall(portName, operationName);
} catch(ServiceException e) {
        // 異常時の処理
}

// サービス呼び出し
// 入力パラメタを生成
Class inputClass = ((com.cosminexus.cws.xml.rpc.Call) call ).getParameterClassByName("in0");
Object in0 = null;
in(inputClass == String.class) {
        in0 = "foo";
}
Object inParams = new Object[] {in0};

Object ret = null;
try {
        Object ret = call.invoke(inParams);
} catch(RemoteException e) {
// 異常時の処理
Throwable cause = e.getCause();
if(cause != null) {
// 原因例外が存在する場合の処理
             System.out.println(cause.getMessage());
}
}

//戻り値の処理
Class retClass = ret.getClass();
int price = 0;
if(retClass == Integer.class) {
        price = ((Integer)ret).intValue();
}
System.out.println("price : " + price);

} finally {
   // クライアント識別子と実行スレッドの関連づけを解除する
   Management.disconnectClientIDtoCurrentThread(cltID);

   // SOAPクライアントの終了
   Management.finalizeClient(cltID);
}
```

例では，javax.xml.rpc.Call クラスと com.cosminexus.cws.xml.rpc.Call クラスという二つの Call クラスが使われています。Java では，別パッケージに存在する，同じ名前のクラスを import 宣言に書けないため，同じクラス内で javax.xml.rpc.Call クラスと com.cosminexus.cws.xml.rpc.Call クラスを使用する場合には，例のように完全修飾名で指定する必要があります。

4.5.3　INOUT パラメタを使用する場合

INOUT パラメタを使用する場合の WSDL の例および DII のコーディング例を示します。なお，WSDL の例では次の情報が定義されています。

- WSDL のサービス名
  名前空間 URI…http://example.com
  ローカル名…StockQuoteService
- WSDL のポート名
  名前空間 URI…http://example.com
  ローカル名…StockQuote
- WSDL のオペレーション名
  名前空間 URI…http://example.com
  ローカル名…getLastTradePrice
- オペレーションのパラメタ名…in0

(1)　WSDL の例（StockQuote.wsdl）

```
<wsdl:definitions
targetNamespace="http://example.com"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://example.com-impl"
xmlns:intf="http://example.com"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:message name="getLastTradePriceResponse">
  <wsdl:part name="getLastTradePriceReturn" type="xsd:int"/>
  <wsdl:part name="in0" type="xsd:string"/>
 </wsdl:message>
 <wsdl:message name="getLastTradePriceRequest">
  <wsdl:part name="in0" type="xsd:string"/>
 </wsdl:message>
 <wsdl:portType name="StockQuote">
  <wsdl:operation name="getLastTradePrice" parameterOrder="in0">
   <wsdl:input message="intf:getLastTradePriceRequest" name="getLastTradePriceRequest"/>
   <wsdl:output message="intf:getLastTradePriceResponse" name="getLastTradePriceResponse"/>
  </wsdl:operation>
 </wsdl:portType>
 <wsdl:binding name="StockQuoteSoapBinding" type="intf:StockQuote">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="getLastTradePrice">
   <soap:operation soapAction=""/>
   <wsdl:input name="getLastTradePriceRequest">
    <soap:body namespace="http://example.com" use="literal"/>
   </wsdl:input>
   <wsdl:output name="getLastTradePriceResponse">
    <soap:body namespace="http://example.com" use="literal"/>
```

```
   </wsdl:output>
  </wsdl:operation>
 </wsdl:binding>
 <wsdl:service name="StockQuoteService">
  <wsdl:port binding="intf:StockQuoteSoapBinding" name="StockQuote">
   <soap:address location="http://localhost:8081/StockQuote/services/StockQuote"/>
  </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

(2) DII のコーディング例

```
ClientID cltID = null;

try {
// SOAPクライアントの開始
cltID = Management.initializeClient();
// クライアント識別子と実行スレッドを関連づける
Management.connectClientIDtoCurrentThread(cltID);

// Serviceオブジェクトの生成
QName serviceName = new QName(
        "http://example.com",
        "StockQuoteService");
URL wsdlDocumentLocation = null;
try {
        wsdlDocumentLocation = new URL("file:///F:/soap/V71/test/dii/module/fsSample/out/Sto
ckQuote.wsdl");
} catch (MalformedURLException e) {
System.out.println(e.getMessage());
}

ServiceFactory factory = null;
Service service = null;
try {
factory = ServiceFactory.newInstance();
          service = factory.createService(wsdlDocumentLocation,
serviceName);
} catch (ServiceException e) {
System.out.println(e.getMessage());
}

// Callオブジェクトの生成
QName portName = new QName(
        "http://example.com",
        "StockQuote");
QName operationName = new QName(
        "http://example.com",
        "getLastTradePrice");
javax.xml.rpc.Call call = null;
try {
call = service.createCall(portName, operationName);
} catch (ServiceException e) {
System.out.println(e.getMessage());
}
```

```
// サービスメソッドの呼出し
// 入力パラメタを生成
Class inputClass = ((com.cosminexus.cws.xml.rpc.Call) call ).getParameterClassByName("in0");
Object in0 = null;
if(inputClass == String.class) {
        in0 = "foo";
}
Object[] inParams = new Object[] {in0};

Object ret = null;
try {
        Object ret = call.invoke(inParams);
} catch(RemoteException e) {
// 異常時の処理
Throwable cause = e.getCause();
if(cause != null) {
// 原因例外が存在する場合の処理
            System.out.println(cause.getMessage());
}
}

// 出力パラメタの処理
Map out = call.getOutputParams();
Iterator itr = out.entrySet().iterator();
while(itr.hasNext()) {
Map.Entry entry = (Map.Entry) itr.next();
String key = (String) entry.getKey();
if(key.equals("in0")) {
    System.out.println(entry.getValue());
}
}

// 戻り値の処理
Class retClass = ret.getClass();
int price = 0;
if(retClass == Integer.class) {
price = ((Integer)ret).intValue();
}
System.out.println("price: " + price);

} finally {
// クライアント識別子と実行スレッドの関連づけを解除する
Management.disconnectClientIDtoCurrentThread(cltID);

// SOAPクライアントの終了
Management.finalizeClient(cltID);
}
```

例では，javax.xml.rpc.Call クラスと com.cosminexus.cws.xml.rpc.Call クラスという二つの Call クラスが使われています。Java では，別パッケージに存在する，同じ名前のクラスを import 宣言に書けないため，同じクラス内で javax.xml.rpc.Call クラスと com.cosminexus.cws.xml.rpc.Call クラスを使用する場合には，例のように完全修飾名で指定する必要があります。

DII では，引数に Holder クラスを指定しません。Holder クラスに対応する Java の基本型クラスを指定してください。例では，javax.xml.rpc.holders.StringHolder に対応する java.lang.String クラスを指定しています。

4.5.4　ユーザ定義のデータ型クラスを使用する場合

ユーザ定義のデータ型クラスを使用する場合の WSDL の例および DII のコーディング例を示します。なお，WSDL の例では次の情報が定義されています。

- WSDL のサービス名
  名前空間 URI…http://example.com
  ローカル名…StockQuoteService
- WSDL のポート名
  名前空間 URI…http://example.com
  ローカル名…StockQuote
- WSDL のオペレーション名
  名前空間 URI…http://example.com
  ローカル名…getLastTradePrice
- オペレーションのパラメタ名…in0
- XML スキーマの複合型（パラメタに定義）
  名前空間 URI…http://example.com
  ローカル名…TradeDate
- XML スキーマの複合型（戻り値に定義）
  名前空間 URI…http://example.com
  ローカル名…TradePrice
- XML スキーマの複合型（ユーザ定義例外に定義）
  名前空間 URI…http://example.com
  ローカル名…TradePriceException

(1)　WSDL の例（StockQuote.wsdl）

```
<wsdl:definitions
targetNamespace="http://example.com"
xmlns:intf="http://example.com"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<wsdl:types>
  <schema elementFormDefault="qualified"
```

```
    targetNamespace="http://example.com"
 xmlns="http://www.w3.org/2001/XMLSchema">

    <element name="getLastTradePrice">
      <complexType>
        <sequence>
          <element name="in0" type="intf:TradeDate"/>
        </sequence>
      </complexType>
    </element>

    <complexType name="TradeDate">
      <sequence>
        <element name="date" nillable="true" type="xsd:string"/>
      </sequence>
    </complexType>

    <element name="getLastTradePriceResponse">
      <complexType>
        <sequence>
          <element name="getLastTradePriceReturn" type="intf:TradePrice"/>
        </sequence>
      </complexType>
    </element>

    <complexType name="TradePrice">
      <sequence>
        <element name="price" type="xsd:int"/>
      </sequence>
    </complexType>

    <element name="TradePriceException" type="intf:TradePriceException"/>

    <complexType name="TradePriceException">
      <sequence>
        <element name="errcode" type="xsd:int"/>
      </sequence>
    </complexType>

 </schema>
</wsdl:types>

<wsdl:message name="getLastTradePriceResponse">
  <wsdl:part name="parameters" element="intf:getLastTradePrice"/>
</wsdl:message>

<wsdl:message name="getLastTradePriceRequest">
  <wsdl:part name="parameters" element="intf:getLastTradePriceResponse"/>
</wsdl:message>

<wsdl:message name="TradePriceException">
  <wsdl:part name="fault" type="intf:TradePriceException"/>
</wsdl:message>

<wsdl:portType name="StockQuote">
  <wsdl:operation name="getLastTradePrice">
    <wsdl:input message="intf:getLastTradePriceRequest" name="getLastTradePriceRequest"/>
    <wsdl:output message="intf:getLastTradePriceResponse" name="getLastTradePriceResponse"/>
```

```
    <wsdl:fault message="intf:TradePriceException" name="TradePriceException"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="StockQuoteSoapBinding" type="intf:StockQuote">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>

  <wsdl:operation name="getLastTradePrice">
    <soap:operation soapAction=""/>

    <wsdl:input name="getLastTradePriceRequest">
      <soap:body use="literal"/>
    </wsdl:input>

    <wsdl:output name="getLastTradePriceResponse">
      <soap:body use="literal"/>
    </wsdl:output>

    <wsdl:fault name="TradePriceException">
      <soap:fault name="TradePriceException" use="literal"/>
    </wsdl:fault>
  </wsdl:operation>
</wsdl:binding>

<wsdl:service name="StockQuoteService">
  <wsdl:port binding="intf:StockQuoteSoapBinding" name="StockQuote">
   <soap:address location="http://localhost:8080/StockQuote/services/StockQuote"/>
  </wsdl:port>
</wsdl:service>

</wsdl:definitions>
```

(2) DII のコーディング例

```
// SOAPクライアントの開始
ClientID cltID = Management.initializeClient();

// クライアント識別子と実行スレッドを関連づける
Management.connectClientIDtoCurrentThread(cltID);

try {
// Serviceオブジェクトの生成
QName serviceName = new QName(
    "http://example.com",
    "StockQuoteService");
URL wsdlDocumentLocation = null;
try {
wsdlDocumentLocation = new URL("file:///C:/foo/StockQuote.wsdl");
} catch (MalformedURLException e) {
// 異常時の処理
}

ServiceFactory factory = null;
Service service = null;
try {
    factory = ServiceFactory.newInstance();
```

```
    service = factory.createService(wsdlDocumentLocation, serviceName);
} catch (ServiceException e) {
// 異常時の処理
}

// Callオブジェクトの生成
QName portName = new QName(
        "http://example.com",
        "StockQuote");
QName operationName = new QName(
        "http://example.com",
        "getLastTradePrice");
javax.xml.rpc.Call call = null;
try {
call = service.createCall(portName, operationName);
} catch (ServiceException e) {
// 異常時の処理
}

// サービス呼び出し
// 入力パラメータを生成
Class inputClass =((com.cosminexus.cws.xml.rpc.Call)
call).getParameterClassByName("in0");
Object in0 = null;

try {
    if (inputClass != null) {
        if (inputClass.getName().equals("com.example.TradeDate")) {
            in0 = inputClass.newInstance();

// メンバメソッドをリフレクトするMethodオブジェクトを取得
            Class[] parameterTypes = {String.class};
            Method method = inputClass.getMethod("setDate", parameterTypes);

// メンバメソッドを実行
            Object[] initargs = {"20070101"};
            method.invoke(in0, initargs);
        }
    }
} catch (Exception e) {
// 異常時の処理
}
Object[] inParams = new Object[]{in0};

Object ret = null;
try {
    ret = call.invoke(inParams);
} catch (RemoteException e) {
// 異常時の処理
    Throwable cause = e.getCause();
    if (cause instanceof C4Fault) {
        C4Fault fault = (C4Fault)cause;

// 例外メッセージの取得
        String msg = fault.getMessage();
        System.out.println(msg);

        Throwable uerr = fault.getCause();
```

```
        if (uerr != null) {
// ユーザ定義例外が含まれる場合
            Class errClass = uerr.getClass();

            int errcode = 0;
            try {
                if (errClass != null) {
                    if (errClass.getName().equals(
                        "com.example.TradePriceException")) {
                        Class[] clParam = {};
                        Object[] objParam = {};

 // メンバメソッドをリフレクトするMethodオブジェクトを取得
                        Method method = errClass.getMethod(
                                    "getErrcode", clParam);

// メンバメソッドを実行し，戻り値を取得
                        Object retErrcode = method.invoke(uerr, objParam);
                        if (retErrcode.getClass() == Integer.class) {
                            errcode = ((Integer)retErrcode).intValue();
                        }
                    }
                }
            } catch (Exception ex) {
// 異常時の処理
            }
            System.out.println("errcode : " + errcode);
        } else {
// ユーザ定義例外が含まれない場合
// C4Fault例外からSOAP Faultの情報を取得する
        }
    }
}

// 戻り値の処理
Class retClass = ret.getClass();
int price = 0;
try {
if (retClass != null) {
    if (retClass.getName().equals("com.example.TradePrice")) {
        Class[] clParam = {};
        Object[] objParam = {};

// メンバメソッドをリフレクトするMethodオブジェクトを取得
            Method method = retClass.getMethod("getPrice", clParam);

// メンバメソッドを実行し，戻り値を取得
            Object retPrice = method.invoke(ret, objParam);
            if (retPrice.getClass() == Integer.class) {
                price = ((Integer)retPrice).intValue();
            }
        }
    }
} catch (Exception e) {
// 異常時の処理
}
System.out.println("price : " + price);
```

```
} finally {
// クライアント識別子と実行スレッドの関連づけを解除する
Management.disconnectClientIDtoCurrentThread(cltID);

// SOAPクライアントの終了
Management.finalizeClient(cltID);
}
```

例では，javax.xml.rpc.Call クラスと com.cosminexus.cws.xml.rpc.Call クラスという二つの Call クラスが使われています。Java では，別パッケージに存在する，同じ名前のクラスを import 宣言に書けないため，同じクラス内で javax.xml.rpc.Call クラスと com.cosminexus.cws.xml.rpc.Call クラスを使用する場合には，例のように完全修飾名で指定する必要があります。

4.5.5　配列を使用する場合

配列を使用する場合の WSDL の例および DII のコーディング例を示します。なお，WSDL の例では次の情報が定義されています。

- WSDL のサービス名
  名前空間 URI…http://example.com
  ローカル名…StockQuoteService
- WSDL のポート名
  名前空間 URI…http://example.com
  ローカル名…StockQuote
- WSDL のオペレーション名
  名前空間 URI…http://example.com
  ローカル名…getLastTradePrice
- オペレーションのパラメタ名…in0
- 配列（パラメタに定義）
  ローカル名…getLastTradePrice
- 配列（戻り値に定義）
  ローカル名…getLastTradePriceResponse

(1)　WSDL の例（StockQuote.wsdl）

```
<wsdl:definitions
targetNamespace="http://example.com"
xmlns:intf="http://example.com"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<wsdl:types>
```

```
  <schema elementFormDefault="qualified"
   targetNamespace="http://example.com"
   xmlns="http://www.w3.org/2001/XMLSchema">

    <element name="getLastTradePrice">
      <complexType>
        <sequence>
          <element name="in0" maxOccurs="unbounded" minOccurs="0" type="xsd:string"/>
        </sequence>
      </complexType>
    </element>

    <element name="getLastTradePriceResponse">
      <complexType>
        <sequence>
          <element name="getLastTradePriceReturn" maxOccurs="unbounded" minOccurs="0" type="
xsd:in/>
        </sequence>
      </complexType>
    </element>

  </schema>
</wsdl:types>

<wsdl:message name="getLastTradePriceRequest">
  <wsdl:part element="intf:getLastTradePrice" name="parameters"/>
</wsdl:message>

<wsdl:message name="getLastTradePriceResponse">
  <wsdl:part element="intf:getLastTradePriceResponse" name="parameters"/>
</wsdl:message>

<wsdl:portType name="StockQuote">
  <wsdl:operation name="getLastTradePrice">
    <wsdl:input message="intf:getLastTradePriceRequest" name="getLastTradePriceRequest"/>
    <wsdl:output message="intf:getLastTradePriceResponse" name="getLastTradePriceResponse"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="StockQuoteSoapBinding" type="intf:StockQuote">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>

  <wsdl:operation name="getLastTradePrice">
    <soap:operation soapAction=""/>

    <wsdl:input name="getLastTradePriceRequest">
      <soap:body use="literal"/>
    </wsdl:input>

    <wsdl:output name="getLastTradePriceResponse">
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>

<wsdl:service name="StockQuoteService">
  <wsdl:port binding="intf:StockQuoteSoapBinding" name="StockQuote">
    <soap:address location="http://localhost:8080/StockQuote/services/StockQuote"/>
```

```
  </wsdl:port>
</wsdl:service>

</wsdl:definitions>
```

(2) DII のコーディング例

```
// SOAPクライアントの開始
ClientID cltID = Management.initializeClient();
// クライアント識別子と実行スレッドを関連づける
Management.connectClientIDtoCurrentThread(cltID);

try {
// Serviceオブジェクトの生成
QName serviceName = new QName(
    "http://example.com",
    "StockQuoteService");
    URL wsdlDocumentLocation = null;
try {
    wsdlDocumentLocation = new URL("file:///C:/foo/StockQuote.wsdl");
} catch (MalformedURLException e) {
// 異常時の処理
}

ServiceFactory factory = null;
Service service = null;
try {
    factory = ServiceFactory.newInstance();
    service = factory.createService(wsdlDocumentLocation, serviceName);
} catch (ServiceException e) {
// 異常時の処理
}

// Callオブジェクトの生成
QName portName = new QName(
        "http://example.com",
        "StockQuote");
QName operationName = new QName(
        "http://example.com",
        "getLastTradePrice");
javax.xml.rpc.Call call = null;
try {
call = service.createCall(portName, operationName);
} catch (ServiceException e) {
// 異常時の処理
}

// サービス呼び出し
// 入力パラメータを生成
Class inputClass = ((com.cosminexus.cws.xml.rpc.Call)call).getParameterClassByName("in0");
Object in0 = null;
if (inputClass == String[].class) {
    in0 = new String[]{"20070101", "20070201"};
}

Object[] inParams = new Object[]{in0};
```

```
Object ret = null;
try {
    ret = call.invoke(inParams);
} catch (RemoteException e) {
// 異常時の処理
    Throwable cause = e.getCause();
    if (cause != null) {
// 原因例外が存在する場合の処理
        System.out.println(cause.getMessage());
    }
}

// 戻り値の処理
Class retClass = ret.getClass();
int price = 0;
if (retClass == Integer[].class) {
    Integer[] prices = (Integer[])ret;
    if (prices.length > 0) {
        price = prices[0].intValue();
    }
}
System.out.println("price : " + price);
} finally {
// クライアント識別子と実行スレッドの関連づけを解除する
Management.disconnectClientIDtoCurrentThread(cltID);
// SOAPクライアントの終了
Management.finalizeClient(cltID);
}
```

5 メッセージング形態の SOAP アプリケーションの開発

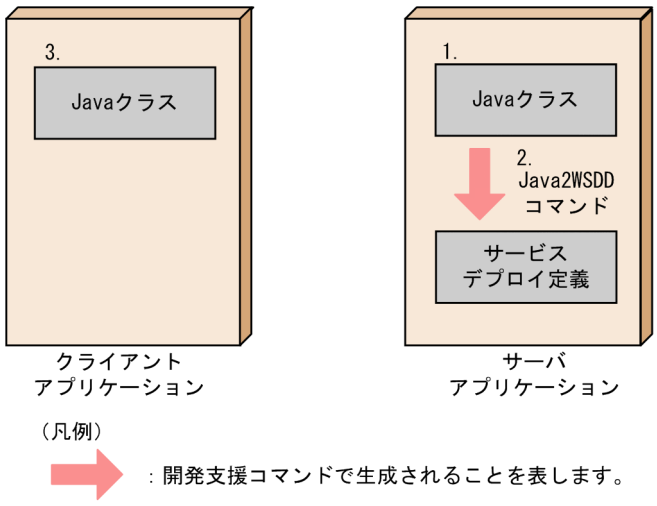
メッセージング形態の SOAP アプリケーションは，SAAJ や開発支援コマンドを使用して開発できます。

この章では，メッセージング形態の SOAP アプリケーションの開発の流れについて説明します。また，添付ファイル付き SOAP メッセージを実装する方法，および XML Processor を使用して開発する方法についても説明します。

5.1　メッセージング形態の SOAP アプリケーション開発の流れ

メッセージング形態の SOAP アプリケーションを新規に開発する場合の流れを示します。

図 5-1　メッセージング形態の SOAP アプリケーションの新規開発



メッセージング形態の SOAP アプリケーションを新規に開発する場合，サーバ側の処理を実装し，実装した Java クラスからサービスデプロイ定義を生成します。また，SAAJ を使用して，クライアント側の処理を実装します。

SOAP アプリケーションを新規に開発する場合の手順を示します。括弧内の数字は，開発例の参照先を示します。

1. **サーバ側の処理を実装する（6.2）**
   ReqResListener インタフェースを実装します。

2. **サービスデプロイ定義を生成する（6.3）**
   実装した Java クラスを SOAP アプリケーションとして利用するための定義ファイルを生成します。

3. **クライアント側の処理を実装する（6.5）**
   SAAJ を使用して，SOAP サービスを利用するクライアント側の処理を実装します。

5.2　メッセージング形態での添付ファイルの使用

メッセージング形態の SOAP アプリケーションでは，SAAJ を利用することで，添付ファイル付きの SOAP メッセージを送受信できます。ここでは，添付ファイル付き SOAP メッセージを実装するために必要な内容について説明します。

なお，添付ファイル付き SOAP メッセージを受信すると，メモリ使用量を抑制するために退避ファイルが作成されることがあります。退避ファイルの詳細については，「付録 B　添付ファイル使用時に作成される退避ファイル」を参照してください。

5.2.1　添付できるファイル

メッセージング形態では，SOAP メッセージで添付できるファイルの種類に制限はありません。ここでは，添付ファイルのサイズおよび添付ファイルの個数の上限値を示します。また，添付ファイルに対応する MIME タイプ，および添付ファイル使用時の注意事項について説明します。

(1)　添付ファイルのサイズの上限値

添付できるファイルサイズは，デフォルトで 100MB（104,857,600 バイト）までです。この値は次に示す動作定義ファイルで変更できます。

サーバ定義ファイル

- c4web.attachment.<識別子>.send_max_attachment_size
- c4web.attachment.<識別子>.receive_max_attachment_size

クライアント定義ファイル

- c4web.attachment.send_max_attachment_size
- c4web.attachment.receive_max_attachment_size

なお，上限値を超える添付ファイルを送信しようとした場合，javax.xml.soap.SOAPConnection クラスの call メソッドの呼び出しで，javax.xml.soap.SOAPException 例外が発生します。

(2)　添付ファイルの個数の上限値

添付できるファイル個数は，デフォルトで 100 個までです。この値は次に示す動作定義ファイルで変更できます。

サーバ定義ファイル

- c4web.attachment.<識別子>.send_max_attachment_count
- c4web.attachment.<識別子>.receive_max_attachment_count

クライアント定義ファイル

- c4web.attachment.send_max_attachment_count
- c4web.attachment.receive_max_attachment_count

なお，上限値を超える添付ファイルを送信しようとした場合，javax.xml.soap.SOAPConnection クラスの call メソッドの呼び出しで，javax.xml.soap.SOAPException 例外が発生します。

(3) 添付ファイルに対応する MIME タイプ

添付したファイルに対応する MIME タイプは，添付したファイルの拡張子によって決まります。次の表に，SOAP 通信基盤が提供するインタフェース（AttachmentPart クラスの setContent メソッド）で添付ファイルを設定した場合に，デフォルトで設定される MIME タイプを示します。なお，SOAP 通信基盤が提供するインタフェース（AttachmentPart クラスの setContentType メソッド）を利用することで，任意の MIME タイプを設定できますが，その場合でも添付したファイルに対応したタイプを設定してください。

表 5-1 添付ファイルの拡張子と MIME タイプの対応（メッセージング）

添付ファイルとして設定したファイルの拡張子	設定される MIME タイプ
html，htm	text/html
txt，text	text/plain
gif，GIF	image/gif
ief	image/ief
jpeg，jpg，jpe，JPG	image/jpeg
tiff，tif	image/tiff
xwd	image/x-xwindowdump
ai，eps，ps	application/postscript
rtf	application/rtf
tex	application/x-tex
texinfo，texi	application/x-texinfo
t，tr，roff	application/x-troff
au	audio/basic
midi，mid	audio/midi
aifc	audio/x-aifc
aif，aiff	audio/x-aiff
wav	audio/x-wav
mpeg，mpg，mpe	video/mpeg

添付ファイルとして設定したファイルの拡張子	設定される MIME タイプ
qt，mov	video/quicktime
avi	video/x-msvideo

この表にない拡張子のファイルを設定した場合，MIME タイプは「application/octet-stream」となります。

(4) 添付ファイルの文字コードと改行コード

SOAP 通信基盤では，テキストベースの添付ファイルの文字コードと改行コードを意識しないで，そのまま SOAP メッセージを転送します。SOAP アプリケーション間で文字コードや改行コードの情報を受け渡したい場合は，MIME ヘッダのユーザ定義フィールド（「X-」で始まる定義フィールド）に，受け渡すコードの情報を設定してください。

(5) MIME ヘッダの扱い

SOAP 通信基盤で設定される MIME ヘッダ値は，RFC822 に従って，すべて US-ASCII として扱います。US-ASCII 以外の文字列を MIME ヘッダ内に指定する場合，RFC2047 に準拠した MIME ヘッダのエンコーディングやデコーディングを SOAP アプリケーション側で実装する必要があります。

5.2.2 添付ファイル付き SOAP メッセージの実装

メッセージング形態の SOAP アプリケーションでは，SAAJ を利用して必要な処理を実装します。SAAJ が提供する API の詳細については，SAAJ に関するドキュメントを参照してください。

5.3 XML Processor を使用した SOAP メッセージの送受信

送受信する XML データが複雑な構造をしている場合や，あらかじめ別のプログラムで DOM オブジェクトを生成してある場合など，SAAJ を使わないで DOM のデータを直接利用したい場合があります。その場合，XML Processor を利用して，XML データを生成／解析できます。

XML データを操作するという点では，XML Processor の提供する DOM インタフェースは，SAAJ インタフェースに比べ自由度が高く，機能も充実しています。

ここでは，XML Processor を使用した場合の送信メッセージの生成方法，および受信メッセージの解析方法について説明します。

5.3.1 XML Processor を使用した送信メッセージの生成

複雑な XML データを生成する必要がある場合や，あらかじめ用意された XML データを利用する場合は，XML Processor で必要な XML データを生成したあとに，SAAJ に取り込む方法を推奨します。

XML Processor で作成した DOM オブジェクト（XML データ）を埋め込んだ SOAP メッセージを作成する場合，次の手順で処理してください。

1. XML Processor を使用して，DOM ドキュメントのデータを作成します。
2. 送信するメッセージの SOAPPart オブジェクトにある importNode メソッドで，データを変換します。
3. SAAJ を使用して，変換したデータを追加します。
   追加先は，SOAP ボディ要素の下，SOAP ヘッダ要素の下，または SOAP Fault の Detail 要素の下にしてください。SOAP ボディ要素や SOAP ヘッダ要素そのものを変換することはできません。
4. 作成したメッセージを送信します。
   SAAJ に変換したデータは，SAAJ のインタフェースを使って変更しないで，そのまま送信してください。

インポートしたデータに，SOAP ヘッダのヘッダ属性が含まれる場合は，その値がそのまま送信されます。ただし，SOAP ヘッダ要素の直下の要素で，mustUnderstand 属性を省略した場合には，次のデフォルト定義が自動的に付加されます。

```
mustUnderstand="0"
```

ヘッダ属性を利用する場合，デフォルト値を意識して，定義がある場合でもない場合でも，正しく処理できるようにユーザプログラムを作成してください。

送信メッセージの SOAP ボディ要素の下に XML を追加する場合のコーディング例を次に示します。

```
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPMessage;
```

```
import javax.xml.soap.SOAPPart;
import org.w3c.dom.Document;
import org.w3c.dom.Node;

public class Sample1 {
    /**
     * Documentオブジェクトのルート要素をSOAPボディ要素の下に格納する。
     *
     * @param msg 格納先の SOAPMessage (MessageFactoryで作成しただけのもの)
     * @param doc 格納元の Document (XML Processorで作成したもの)
     * @return    格納先の SOAPMessage
     */
    public SOAPMessage sample(SOAPMessage msg, Document doc) {
        try {
            SOAPPart part = msg.getSOAPPart();

            // DOMのNodeオブジェクトを取り出しSAAJオブジェクトに変換する
            Node source = doc.getDocumentElement();
            SOAPElement e = (SOAPElement)part.importNode(source, true);

            // メッセージのSOAPBodyの下に追加する
            SOAPBody body = msg.getSOAPBody();
            e = body.addChildElement(e);
        }
        catch (Exception e) {
            // 省略
        }
        return msg;
    }
}
```

送信メッセージの SOAP Fault の Detail 要素下に，XML を追加する場合のコーディング例を次に示します。

```
import javax.xml.soap.Detail;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPFault;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import org.w3c.dom.Document;
import org.w3c.dom.Node;

public class Sample2 {
    /**
     * Documentオブジェクトのルート要素をDetail要素の下に格納する。
     *
     * @param msg 格納先の SOAPMessage (MessageFactoryで作成しただけのもの)
     * @param doc 格納元の Document (XML Processorで作成したもの)
     * @return    格納先の SOAPMessage
     */
    public SOAPMessage sample(SOAPMessage msg, Document doc) {
        try {
            SOAPPart part = msg.getSOAPPart();

            // DOMのNodeオブジェクトを取り出しSAAJオブジェクトに変換する
```

```
            Node source = doc.getDocumentElement();
            SOAPElement e = (SOAPElement)part.importNode(source, true);

            // メッセージのSOAPFault中のDetail要素の下に追加する
            SOAPBody body = msg.getSOAPBody();
            SOAPFault fault = body.addFault();
            Detail detail = fault.addDetail();
            e = detail.addChildElement(e);

            // faultString, faultActor, faultCodeなどを設定する
            fault.setFaultString("sample error");
            // 省略
        }
        catch (Exception e) {
            // 省略
        }
        return msg;
    }
}
```

5.3.2 XML Processor を使用した受信メッセージの解析

受信したメッセージを XML データとして処理する場合は，XML Processor のオブジェクトに変換して解析する方法を推奨します。

SOAP 通信基盤で XML データを埋め込んだ SOAP メッセージを受信し，データを取り出す場合，次の手順で処理してください。

1. XML Processor を利用し，Document オブジェクトを作成します。
2. 作成した Document オブジェクトの importNode メソッドで，受信した SAAJ のデータを変換します。
3. 取得した Node オブジェクトを操作して，必要な要素を取得します。

受信メッセージを解析する場合のコーディング例を次に示します。

```
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPMessage;
import org.w3c.dom.Document;
import org.w3c.dom.Node;

public class Sample3 {
    /**
     * 受信したSOAPMessageオブジェクト全体を，DOMのオブジェクトに変換する
     * @param doc 変換先のDocument（XML Processorで作成したもの）
     * @param msg 変換元のSOAPMessage(受信したもの)
     * @return    変換した要素
     */
    public Node sample(Document doc, SOAPMessage msg) {
        Node n = null;
        try {
            // SAAJのデータを，XML Processorにインポートする
```

```
            SOAPEnvelope env = msg.getSOAPPart().getEnvelope();
            n = doc.importNode(env, true);
        }
        catch (Exception e) {
            // 省略
        }
        return n;
    }
}
```

受信メッセージが SOAP Fault メッセージである場合も，DOM として解析できます。また，SOAPFault インタフェースを利用して，faultString などにアクセスすることもできます。

SOAPFault インタフェースを利用する場合でも，Detail 要素以下に XML データを格納している場合は，XML Processor のオブジェクトに変換して解析してください。

SOAP Fault メッセージである受信メッセージを解析する場合のコーディング例を次に示します。

```
import javax.xml.soap.Detail;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPFault;
import javax.xml.soap.SOAPMessage;
import org.w3c.dom.Document;
import org.w3c.dom.Node;

public class Sample4 {
    /**
     * 受信したSOAPFaultのdetail要素をDOMのNodeオブジェクトに変換する。
     *
     * @param doc 変換先の Document（XML Processorで作成したもの）
     * @param msg 変換元の SOAPMessage（受信したもの）
     * @return    変換した要素
     */
    public Node sample(Document doc, SOAPMessage msg) {
        Node n = null;
        try {
            SOAPBody body = msg.getSOAPBody();
            if (body.hasFault()) {
                SOAPFault fault = body.getFault();
                // SOAPFaultのほかの要素(faultStringなど)にアクセスする
                String faultString = fault.getFaultString();
                // 省略

                // DetailだけはDOMとして取得する
                Detail detail = fault.getDetail();
                n = doc.importNode(detail, true);
            }
            else {
                // 省略
            }
        }
        catch (Exception e) {
            // 省略
        }
        return n;
```

```
    }
}
```

5.4 SAAJ を使用した SOAP アプリケーション開発時の注意事項

SAAJ を使用して，SOAP アプリケーションを開発する場合の注意事項について説明します。

5.4.1 SOAP アプリケーションのサービス名に関する注意

RPC 形態，メッセージング形態，および EJB 形態の SOAP サービスで，同一のサービス名は指定しないでください。同一のサービス名の SOAP サービスがデプロイされていると，予期しない動作をするおそれがあります。

5.4.2 複数個の AttachmentPart オブジェクトを追加する場合の注意

複数個の AttachmentPart オブジェクトを追加する場合，AttachmentPart オブジェクトを追加する個数分，生成するようにしてください。同一の AttachmentPart オブジェクトを SOAPMessage#addAttachmentPart（AttachmentPart AttachmentPart）メソッドの引数として指定し，処理を複数回実行しても，SOAP メッセージ上には一つしか存在しない場合があります。

5.4.3 setProperty メソッドで指定するプロパティのデフォルト値

SOAPMessage#setProperty（java.lang.String property,java.lang.Object value）メソッドで指定できるプロパティは次の二つだけです。

- CHARACTER_SET_ENCODING
- WRITE_XML_DECLARATION

CHARACTER_SET_ENCODING のデフォルト値は"utf-8"，WRITE_XML_DECLARATION のデフォルト値は"false"です。CHARACTER_SET_ENCODING，WRITE_XML_DECLARATION 以外をプロパティとして指定しても，javax.xml.soap.SOAPException 例外は発生しません。

5.4.4 setProperty メソッドで指定するプロパティ値に関する注意

SOAPMessage#setProperty（java.lang.String property,java.lang.Object value）メソッドで指定できる CHARACTER_SET_ENCODING，および WRITE_XML_DECLARATION の value 値には，String オブジェクトを指定してください。String オブジェクト以外を設定した場合，java.lang.ClassCastException 例外が発生することがあります。

5.4.5 MimeHeader オブジェクトで指定できる文字コードに関する注意

MimeHeader オブジェクト生成時に，引数で，name，および value に非 US-ASCII 文字列を指定しないでください。MimeHeader オブジェクト生成時の引数で，name，および value に非 US-ASCII 文字列を指定しても，java.lang.IllegalArgumentException 例外が発生しない場合があります。

5.4.6 各メソッドで発生する例外

各 API に関して不正な引数を与えた場合にスローされる例外を次に示します。

表 5-2 各 API に関して不正な引数を与えた場合にスローされる例外

SAAJ 仕様書に記載されている各メソッドの throws：の内容	各 API でスローされる例外
javax.xml.soap.SOAPException	javax.xml.soap.SOAPException
java.lang.IllegalArgumentException	java.lang.IllegalArgumentException
javax.xml.soap.SOAPException java.lang.IllegalArgumentException	java.lang.IllegalArgumentException
javax.xml.soap.SOAPException java.lang.IllegalArgumentException 以外	javax.xml.soap.SOAPException
記述なし	java.lang.IllegalArgumentException

5.4.7 SOAPPart クラスの setContent メソッドの引数に関する注意

SOAPPart#setContent（javax.xml.transform.Source source）メソッドの引数には，正しい SOAP メッセージを表すソースオブジェクトを指定してください。正しい SOAP メッセージを表すソースオブジェクトを指定しなかった場合，不正な SOAP メッセージを送信してしまう場合があります。

5.4.8 AttachmentPart オブジェクトの Content-Type ヘッダが text/plain の場合の上限サイズ

AttachmentPart オブジェクトの Content-Type ヘッダが text/plain の場合，送受信できる添付ファイルのサイズの上限は 1MB です。1MB より大きいサイズの添付ファイルを受信して AttachmentPart の Content を取得すると，1MB を超えた部分が切り捨てられた String オブジェクトが返されます。

5.4.9　クッキーについて

Web サーバから受け取ったクッキーは，SOAPConnection オブジェクトが保持します。クッキーを利用する場合，メッセージングサービスを呼び出すたびに，SOAPConnection オブジェクトを破棄することなく，継続して使用してください。

5.4.10　クッキーの有効期限について

expires 属性で設定したクッキーの有効期限に達しても，SOAP クライアントライブラリは保持しているクッキーを削除しません。

期限切れのクッキーを受け取っても処理できるようにサーバで対処してください。

5.4.11　マルチスレッドでの動作について

SAAJ の各メソッドは，スレッドセーフではありません。SAAJ のオブジェクトをマルチスレッドで共有しないでください。共有した場合には，動作を保証できません。

5.4.12　受信電文の SOAP ヘッダおよび SOAP ボディの子孫ノードにテキストノードが存在する場合の注意

受信電文の SOAP ヘッダおよび SOAP ボディの子孫ノードにテキストノードが存在する場合，兄弟ノードのすべてのテキストノードを連結し，そのテキストノードを同じ階層の最後に追加します。

受信電文の Body 要素に対する javax.xml.soap.SOAPElement#getChildElements および org.w3c.dom.Node#getChildNodes の結果を示します。

- 受信電文の Body 要素

```
<soapenv:Body><a/>△△foo△△<b/>△△bar△△</soapenv:Body>
```

- javax.xml.soap.SOAPElement#getChildElements の戻り値

```
Iteratorの最初の要素 : Elementノード <a/>
Iteratorの二番目の要素 : Elementノード <b/>
Iteratorの三番目の要素 : Textノード △△foo△△△△bar△△
```

- org.w3c.dom.Node#getChildNodes の戻り値

```
NodeListの最初のノード : Elementノード <a/>
NodeListの二番目のノード : Elementノード <b/>
NodeListの三番目のノード : Textノード △△foo△△△△bar△△
```

注　△は半角スペースを表します。

5.4.13　名前空間に属さない要素を追加する場合の注意

SOAP ボディの子要素以下の要素（SOAP ボディの子要素を含む）に，その子要素として名前空間に属さない要素を追加する場合，javax.xml.soap.SOAPFactory クラスの createElement メソッドで javax.xml.soap.SOAPElement オブジェクトを生成し，生成したオブジェクトを指定して，javax.xml.soap.SOAPElement クラスの addChildElement メソッドを呼び出してください。例を次に示します。

```
SOAPFactory soapFactory = SOAPFactory.newInstance();

// 名前空間に属さない”entry”という要素名の要素をsoapElementに追加する例
// soapElementは，子要素を追加するSOAPボディの子要素以下の要素（SOAPボディの子要素を含む）
soapElement.addChildElement( soapFactory.createElement( ”entry” ) );
```

5.4.14　SOAP メッセージの要素の名前空間プレフィクスに関する注意

同じ名前空間 URI で異なる名前空間プレフィクスは使用しないでください。

要素の名前空間プレフィクスに，同じ名前空間 URI で異なる名前空間プレフィクスを付けたメッセージを作成した場合，送信される SOAP メッセージ内の要素の名前空間プレフィクスが作成時と異なることがあります（いちばん近い名前空間宣言の名前空間プレフィクスが付けられます）。作成したメッセージと送信される SOAP メッセージの例を次に示します。

```
＜作成したメッセージ＞
<prefix01:elm1 xmlns:prefix01=”http://localhost”>
<prefix02:elm2 xmlns:prefix02=”http://localhost”>
<prefix01:elm3>
<prefix02:elm4/>
</prefix01:elm3>
</prefix02:elm2>
</prefix01:elm1>

＜送信されるSOAPメッセージ＞
<prefix01:elm1 xmlns:prefix01=”http://localhost”>
<prefix02:elm2 xmlns:prefix02=”http://localhost”>
<prefix02:elm3>
<prefix02:elm4/>
</prefix02:elm3>
</prefix02:elm2>
</prefix01:elm1>
```

5.4.15　setProperty メソッドで指定したエンコード方式に関する注意

SOAPMessage#setProperty()メソッドの引数で，CHARACTER_SET_ENCODING プロパティに指定できる値は"utf-8"だけです。

SOAPMessage#setProperty()メソッドの引数で，CHARACTER_SET_ENCODING プロパティに"utf-16"を指定して SOAP メッセージを作成し，送信した場合，実際の送信メッセージにある SOAP メッセージのエンコード方式が"utf-16"と異なることがあります。

5.4.16　SOAPPart クラスから取得した MimeHeader に関する注意

SOAPPart クラスから取得した MimeHeader は有効にならないため，利用しないことを推奨します。

SOAPPart#getAllMimeHeaders()メソッドを発行して値を取得した場合，実際の送信メッセージにある SOAP メッセージに含まれる MIME ヘッダと異なることがあります。

5.4.17　MIME ヘッダに関する注意

SOAPPart クラスに対し指定した MimeHeader は有効にならないため，利用しないでください。

SOAPPart#addMimeHeader()メソッド，SOAPPart#setMimeHeader()メソッドで MIME ヘッダを追加した場合，または SOAPPart#setContentId()メソッド，SOAPPart#setContentLocation()メソッドで値を指定した場合，実際の送信メッセージにある SOAP メッセージに含まれる MIME ヘッダに反映されないことがあります。

5.4.18　SOAPFault クラスの setFaultCode メソッドの引数に関する注意

SOAPFault クラスの setFaultCode メソッドの引数に，"prefix:localName"形式の String オブジェクトを指定する場合は，SOAPFault#setFaultCode(String faultCode)メソッドではなく，SOAPFault#setFaultCode（Name faultCodeQName）メソッドを使用してください。

SOAPFault#setFaultCode(String faultCode)メソッドの引数に"prefix:localName"形式の String オブジェクトを指定した場合，javax.xml.soap.SOAPException 例外が発生することがあります。

5.4.19　AttachmentPart クラスの setContent メソッドの引数に関する注意

AttachmentPart#setContent（Object object, String contentType）メソッドの引数 object には，JAF 仕様に準拠する型のオブジェクトを指定し，引数 contentType には，RFC2045 に準拠する Content-Type ヘッダの値を指定してください。

AttachmentPart#setContent（Object object, String contentType）メソッドの引数 object に JAF 仕様に準拠する型のオブジェクト以外を指定した場合，または引数 contentType に RFC2045 に準拠する Content-Type ヘッダの値以外を指定した場合，タイムアウト待ちが発生し，不正な SOAP メッセージが送信されることがあります。

5.4.20 名前空間プレフィクスを引数として渡すメソッドを発行した場合の注意

名前空間プレフィクスを引数として渡すメソッドを発行した場合，次の現象が発生することがあります。

- 名前空間プレフィクスが付与されない。
- 指定した名前空間プレフィクスと異なるプレフィクスが付与される。

5.4.21 引数に null オブジェクトを指定した場合の注意

メソッドを発行する前に，引数が null オブジェクトでないことを確認してください。メソッドの引数に null オブジェクトを指定した場合，次の現象が発生することがあります。

- 不正な SOAP メッセージが生成される。
- java.lang.NullPointerException 例外が発生する。
- 対象オブジェクトに不正な値が設定される。
- 対象オブジェクトに対して何も処理をしない。
- null オブジェクトを指定後，取得系のメソッドを発行しても正しい値が取得できない。

5.4.22 CHARACTER_SET_ENCODING プロパティの値の取得に関する注意

マルチパート形式の SOAP メッセージを受信した場合，SOAPMessage#getProperty()メソッドを使用して，CHARACTER_SET_ENCODING プロパティの値を取得することはできません。null オブジェクトが返されます。

5.4.23 SOAPMessage クラスの saveChanges メソッドおよび saveRequired メソッドの発行に関する注意

SOAPMessage#saveChanges()メソッドおよび SOAPMessage#saveRequired()メソッドを発行しないでください。

SOAPMessage#saveChanges()メソッドおよび SOAPMessage#saveRequired()メソッドを発行した場合，生成される SOAP メッセージが予期しない形式になることがあります。

5.4.24 SOAPMessage クラスの writeTo メソッドの発行に関する注意

SOAPMessage#writeTo()メソッドは発行しないでください。

SOAPMessage#writeTo()メソッドを発行後に，SOAPConnection#call()メソッドを実行した場合，SOAPConnection#call()メソッド発行の延長で javax.xml.soap.SOAPException 例外が発生することがあります。

また，SOAPMessage#writeTo()メソッドの引数に，すでに close された OutputStream オブジェクトを指定した場合，java.io.IOException 例外が発生しないで処理が続行することがあります。

5.4.25　Node クラスの detachNode メソッドの発行に関する注意

Node#detachNode()メソッドの発行によって，対象オブジェクトが存在しない状態でメソッドを発行すると，例外が発生します。Node#detachNode()メソッドの発行対象となったオブジェクトを包含しているオブジェクトに対してメソッドを発行すると，例外が発生することがあります。

Node#detachNode()メソッドを発行した場合は，包含するオブジェクトを再生成してからメソッドを発行してください。

5.4.26　添付ファイルの削除に関する注意

次に示す表（MIME ヘッダのコンテントタイプの分類）の項番 1～3 の場合を除いて，AttachmentPart#getContent()メソッド発行後に，取得した入力ストリームをユーザプログラムで閉じていない場合，添付ファイルを削除できなくなることがあります。

なお，AttachmentPart#getDataHandler()メソッドの延長で，java.io.InputStream オブジェクトを取得した場合も同様です。

表 5-3　MIME ヘッダのコンテントタイプの分類

項番	MIME ヘッダの コンテントタイプ	AttachmentPart#getContent() メソッドの戻り値	入力ストリームを明示的にユーザプログラムで閉じる必要の有無
1	"text/plain"	java.lang.String	×
2	"image/gif"	java.awt.Image	×
3	"image/jpeg"	java.awt.Image	×
4	"text/xml"	javax.xml.transform.stream.StreamSource	○
5	上記以外	java.io.InputStream	○

（凡例）

×：入力ストリームを明示的にユーザプログラムで閉じる必要はありません。

○：入力ストリームを明示的にユーザプログラムで閉じる必要があります。

添付ファイルを削除できない場合，J2EE サーバを停止後，手動でファイルを削除してください。なお，入力ストリームを閉じないで連続稼働した場合，ファイルディスクリプタなどの OS 資源が枯渇するおそれがあります。

5.4.27 引数の名前空間 URI または名前空間プレフィクスの指定に関する注意

名前空間 URI または名前空間プレフィクスを引数として指定する場合は，引数が null オブジェクトまたは""（空文字列）でないことを確認してください。

引数の名前空間 URI または名前空間プレフィクスとして null オブジェクトまたは""（空文字列）を指定した場合，次の現象が発生することがあります。

- 予期しない名前空間 URI または名前空間プレフィクスが設定される。
- 名前空間 URI または名前空間プレフィクスが不正な SOAP メッセージを送信する。

5.4.28 SOAPElement クラスの addChildElement メソッドの指定に関する注意

SOAPElement#addChildElement()メソッドの引数に SOAP 名前空間（"http://schemas.xmlsoap.org/soap/envelope/"）に属する SOAP エンベロープ，SOAP ヘッダ，SOAP ボディ，および自分自身を含む要素を指定した場合，java.lang.StackOverflowError 例外が発生することがあります。

5.4.29 SOAPHeader クラスの examineHeaderElements メソッドおよび extractHeaderElements メソッドの実行に関する注意

SOAPHeader#examineHeaderElements()メソッドおよび SOAPHeader#extractHeaderElements()メソッドを実行すると，次の SOAPHeaderElement オブジェクトを含む Iterator が返されます。

- 引数で指定した actor を持つ SOAPHeaderElement オブジェクト
- actor として"http://schemas.xmlsoap.org/soap/actor/next"を持つ SOAPHeaderElement オブジェクト
- actor 属性が存在しない SOAPHeaderElement オブジェクト

SOAPHeader#examineHeaderElements()メソッドの場合，取得した Iterator の各要素から要求する actor を持つ SOAPHeaderElement を抽出してください。

SOAPHeader#extractHeaderElements()メソッドの場合，引数で指定した actor を持つ SOAPHeaderElement オブジェクト以外に，actor として"http://schemas.xmlsoap.org/soap/actor/next"を持つ SOAPHeaderElement オブジェクト，および actor 属性が存在しない SOAPHeaderElement

オブジェクトが，SOAPHeader オブジェクトから削除されます。指定した actor を持つ SOAPHeaderElement オブジェクトだけを削除するには，いったん SOAPHeader#examineHeaderElements()メソッドで Iterator を取得後，要求する actor を持つ SOAPHeaderElement を抽出して削除してください。

5.4.30 MessageFactory クラスの createMessage メソッドの引数に関する注意

SOAP アプリケーションでは，引数に close した InputStream オブジェクトを指定しないでください。

MessageFactory#createMessage（MimeHeaders, InputStream）で，close した InputStream オブジェクトを指定すると，java.lang.RuntimeException 例外が発生します。

5.4.31 SOAPElement クラスの getVisibleNamespacePrefixes メソッドの指定に関する注意

SOAPEnvelope の子孫ノードに対して，SOAPElement#getVisibleNamespacePrefixes()を発行すると，戻り値の Iterator に名前空間プレフィクス soapenv が二つ含まれます。二つの名前空間プレフィクスは同じ名前空間 URI を指します。

5.4.32 添付オブジェクトに対応する Content-Type ヘッダの設定に関する注意

メソッドを使用して Content-Type ヘッダを設定する場合は，添付オブジェクトに対応する Content-Type を指定してください。

次のメソッドを使用して添付オブジェクトに対応しない Content-Type を指定した場合，タイムアウト待ちが発生する，または不正な SOAP メッセージが送信されることがあります。

- AttachmentPart#setContentType（String）
- AttachmentPart#addMimeHeader（String, String）
- AttachmentPart#setMimeHeader（String, String）

5.4.33 MIME ヘッダまたは SOAP ヘッダを取得するメソッドに関する注意

次のメソッドを使用して MIME ヘッダまたは SOAP ヘッダを取得する場合，該当する MIME ヘッダまたは SOAP ヘッダが存在しないと，null オブジェクトが返されます。

- MimeHeaders#getHeader（String）
- SOAPPart#getMimeHeader（String）
- AttachmentPart#getMimeHeader（String）
- SOAPHeader#examineMustUnderstandHeaderElements（String）

5.4.34　送信メッセージの HTTP ヘッダについて

Content-Length ヘッダ，および Content-Type ヘッダの値は自動で付与されます。Content-Type ヘッダの charset に文字コードを設定する場合は，javax.xml.soap.SOAPMessage#setProperty メソッドを使用して，CHARACTER_SET_ENCODING を設定してください。

javax.xml.soap.SOAPMessage オブジェクトの MimeHeaders に Content-Length ヘッダ，および Content-Type ヘッダを設定していても，送信されるメッセージの HTTP ヘッダには MimeHeaders で設定した値は付与されません。

5.4.35　Management クラスの使用に関する注意事項

クライアントプログラムでは必ず Management クラスを使用してください。SAAJ を利用して添付ファイルを送受信するときに生成される退避ファイルが，削除されないで残ることがあります。

5.4.36　toString メソッドの使用に関する注意事項

detail entry 以外の内容を含む detail 要素を利用している SOAP Fault の受信メッセージから得られた javax.xml.soap.SOAPBody オブジェクト，および javax.xml.soap.SOAPFault オブジェクトに対して，toString メソッドを呼び出さないでください。toString メソッドを呼び出した場合，オブジェクトの内容を表現する XML 文字列ではなく，オブジェクトのクラス名とハッシュコードから構成される文字列を返すことがあります。

6 メッセージング形態の SOAP アプリケーションの開発例

この章では，メッセージング形態の SOAP アプリケーションの開発例を手順に沿って説明します。開発した SOAP アプリケーションの実行方法については，「7.　SOAP アプリケーションの実行と運用」を参照してください。

6.1 SOAP アプリケーションを設計する

メッセージング形態の SOAP アプリケーションを新規に開発します。開発例の概要と開発の流れを次に示します。

開発例の概要

SAAJ を使用して，任意の XML データを送受信する SOAP アプリケーションを新規に開発します。

開発の流れ

1. SOAP アプリケーションの設計
2. サーバ側の処理の実装
3. サービスデプロイ定義の生成
4. アーカイブ（WAR ファイル）の作成
5. クライアント側の処理の実装

最初に，SAAJ を使用した SOAP アプリケーションの仕様を設計します。開発例では，次に示す SOAP アプリケーションを新規に開発します。

サービス名

MsgService

入出力

任意の XML データ

6.2　サーバ側の処理を実装する

新しくクラスを作成して，処理を実装します。新しく作成したクラスは，SOAP 通信基盤が提供する ReqResListener インタフェースを実装します。実装後に，新しく作成したクラスをコンパイルします。

ReqResListener インタフェースの実装例を次に示します。

```
package msgservice;

import com.cosminexus.cws.xml.soap.ReqResListener;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPException;

public class MessageServiceImpl implements ReqResListener
{
     public SOAPMessage onMessage(SOAPMessage soapmessage) throws SOAPException
     {
          //
          // メッセージングサービスの実装を行う
          //
     }
}
```

なお，これ以外の実装例については，次に示すディレクトリに格納しているサンプルプログラムを参照してください。ただし，サンプルプログラムとして提供するのは，SOAP アプリケーション開発支援機能だけです。

```
<Application Serverのインストールディレクトリ>/c4web/samples
```

6.3　サービスデプロイ定義を生成する

Java2WSDD コマンドを使用して，作成した Java プログラムを SOAP アプリケーションとして利用するための定義ファイルを生成します。

Java2WSDD コマンドの指定例を示します。

```
Java2WSDD.bat -p C4MSG -s MsgService msgservice.MessageServiceImpl
```

このコマンドを実行すると，カレントディレクトリにサービスデプロイ定義ファイル（server-config.xml）が生成されます。

この例では，-p オプションで C4MSG を指定することでメッセージングのサービスデプロイ定義ファイルを生成し，-s オプションでサービス名を指定していますが，必要に応じてほかのオプションも指定してください。オプションについては，「9.3　Java2WSDD コマンド（サービスデプロイ定義の生成）」を参照してください。

SOAP ヘッダ項目の解釈処理を SOAP サービスに実装している場合は，-S オプションで"true"を指定してください。また，SOAP メッセージをほかの SOAP サービスに中継し，そこで SOAP ヘッダを処理する場合も-S オプションで"true"を指定してください。

注意事項

-S オプションで"true"を指定した場合，SOAP 通信基盤は mustUnderstand 属性をチェックしません。このため，次の処理をサーバ側で実装してください。

- すべての SOAP ヘッダ項目のうち，処理が必須であると設定されている SOAP ヘッダ項目※に対して，処理を実装してください。
- 処理が必須であると設定されている SOAP ヘッダ項目※の処理に失敗した場合は，SOAP Fault を送信する処理を実装してください。
- SOAP メッセージを中継する場合は，中継先で処理されるように，未処理の SOAP ヘッダ項目は削除しないでください（処理済の SOAP ヘッダ項目は削除してください）。
- 中継先から SOAP Fault を受けた場合は，クライアントに SOAP Fault を送信する処理を実装してください。

注※

「処理が必須であると設定されている SOAP ヘッダ項目」とは，actor 属性値が SOAP サービス自身を示す URI であり，かつ mustUnderstand 属性値での処理が必須であると設定されている SOAP ヘッダ項目のことを示します。

次に，送信する SOAP Fault の詳細について示します。

表 6-1　送信する SOAP Fault の詳細（-S オプションで"true"を指定した場合）

SOAP Fault の要素	指定値
faultcode	名前空間識別子：MustUnderstand

SOAP Fault の要素	指定値
	（名前空間識別子は，「http://schemas.xmlsoap.org/soap/envelope/」を推奨します）
faultstring	Did not understand "MustUnderstand" header(s):{名前空間識別子：ヘッダ項目のローカル名}
faultactor	SOAP Fault 発生元を識別する URI
detail	null

-S オプションで"false"を指定した場合，SOAP 通信基盤はサーバ側の実装を呼び出す前に mustUnderstand 属性をチェックします。mustUnderstand 属性が「1」となっている SOAP ヘッダが存在した場合，サーバ側の実装を呼び出さないでクライアントに SOAP Fault を送信します。

次にクライアントに送信する SOAP Fault の詳細について示します。

表 6–2　送信する SOAP Fault の詳細（-S オプションで"false"を指定した場合）

SOAP Fault の要素	指定値
faultcode	名前空間識別子：MustUnderstand （名前空間識別子は，「http://schemas.xmlsoap.org/soap/envelope/」）
faultstring	Did not understand "MustUnderstand" header(s):{名前空間識別子：ヘッダ項目のローカル名}
faultactor	null
detail	null

6.4　アーカイブ（WAR ファイル）を作成する

J2EE サーバにデプロイするためのアーカイブを作成します。

1. WAR ファイルを作成する前に，ディレクトリ構成を整えます。

   ディレクトリ構成を次に示すように整えます。

```
/
  WEB-INF/
    server-config.xml
    web.xml
    classes/
      messagesampleservice/
        MessageSampleService.class
```

2. DD（web.xml）に情報を追加します。

   追加する情報については，「3.9.2　DD（web.xml）の記述」を参照してください。

3. WAR ファイルを作成します。

   WAR ファイルの作成方法については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」を参照してください。

6.5　クライアント側の処理を実装する

実装用の新規クラスを作成し，クライアント側の処理を実装します。クライアント側の処理の実装は，SAAJ を使用してください。実装後に新しく作成したクラスをコンパイルします。

クライアント側の処理の実装例を次に示します。

```
package messagesampleclient;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

import javax.xml.soap.AttachmentPart;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConnection;
import javax.xml.soap.SOAPConnectionFactory;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPFactory;
import javax.xml.soap.SOAPMessage;

import com.cosminexus.cws.management.Management;
import com.cosminexus.cws.management.ClientID;

public class MessageSampleClient extends HttpServlet
{
    private static final String CONTENT_TYPE = "text/html; charset=Shift_JIS";

    //Client Identifier
    private ClientID cltID = null;

    public void init() throws ServletException
    {
        //Initializing the client
        cltID = Management.initializeClient();
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response) throws Serv
letException, IOException
    {
        //Service endpoint url
        String SERVICE_URL = "http://localhost:8080/MessageSampleServer/services/MsgService"
;

        //Response message output
        String strResultMessage = null;
        String strResultMessageAttach = null;

        //Connecting client to the current thread
        Management.connectClientIDtoCurrentThread(cltID);

        //Getting the parameters set by user
        String strName = request.getParameter("Name");
```

```
        if (strName == null)
        {
            strName = "";
        }

        String strSection = request.getParameter("Section");
        if (strSection == null)
        {
            strSection = "";
        }

        String strTelephone = request.getParameter("Telephone");
        if (strTelephone == null)
        {
            strTelephone = "";
        }

        //Request and Response message
        SOAPMessage msgRequest, msgResponse;

        try
        {
            //Creating the soapmessage using MessageFactory
            MessageFactory msgfactory = MessageFactory.newInstance();
            msgRequest = msgfactory.createMessage();

            //Creating the factory instance
            SOAPFactory factory = SOAPFactory.newInstance();

            //Adding retrieved parameters to the body
            SOAPBody bdy = msgRequest.getSOAPBody();
            SOAPElement bdyElmName = bdy.addChildElement("Name", "", "http://localhost");
            bdyElmName.addTextNode(strName);
            SOAPElement bdyElmSection = bdy.addChildElement("Section", "", "http://localhost
");
            bdyElmSection.addTextNode(strSection);
            SOAPElement bdyElmTelephone = bdy.addChildElement("Telephone", "", "http://local
host");
            bdyElmTelephone.addTextNode(strTelephone);
            SOAPElement bdyElmAttachment = bdy.addChildElement("Attachment");
            bdyElmAttachment.setAttribute("href","attachment.txt");

            //Getting the path of the file to be attached
            String UserDir = System.getProperty("user.dir");
            String strFileSeperator=File.separator;
            String strContextPath =  (String)getServletConfig().getServletContext().getRealP
ath("/");
            String strAttachment_c4webcl_Dir = strContextPath + "attachment.txt";
            //Adding an attachment to the request message
            AttachmentPart apPart = msgRequest.createAttachmentPart(new java.io.FileInputStr
eam(new File(strAttachment_c4webcl_Dir)),"text/plain");
            apPart.setContentLocation("attachment.txt");
            msgRequest.addAttachmentPart(apPart);

            //Create Connection
            SOAPConnectionFactory connFactory = SOAPConnectionFactory.newInstance();
            SOAPConnection conn = connFactory.createConnection();
            //Invoke the Service
```

```
            msgResponse = conn.call(msgRequest, SERVICE_URL);

            //Retrieving the body elements
            SOAPBody body = msgResponse.getSOAPBody();
            Iterator itr = body.getChildElements();

            if (itr.hasNext())
            {
                SOAPElement elmResult = (SOAPElement)itr.next();
                strResultMessage = elmResult.getFirstChild().getNodeValue();
                if ( itr.hasNext() )
                {
                    SOAPElement elmResultAttach = (SOAPElement)itr.next();
                    strResultMessageAttach = elmResultAttach.getFirstChild().getNodeValue();
                }
            }
        }
        catch( Exception exp )
        {
            //Displaying the exception
            response.setContentType(CONTENT_TYPE);
            PrintWriter out = response.getWriter();
            out.println("<html>");
            out.println("<head><title>MessageSampleClient</title></head>");
            out.println("<body>");
            out.println("Exception Occurred");
            out.println("<BR>Message  :  "+exp.getMessage());
            out.println("</body></html>");
            return;
        }
        finally
        {
            //Disconnecting the client thread
            Management.disconnectClientIDtoCurrentThread(cltID);
        }

        //Displaying the required output message
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>MessageSampleClient</title></head>");
        out.println("<body>");
        out.println("¥"" + strResultMessage + "¥" was sent by Client.");
        out.println("<BR><BR>Service is invoked.<BR>");
        out.println("<BR>Attachment's contents are : ¥n" + strResultMessageAttach + "<BR>");
        out.println("</body></html>");

    }

    public void destroy()
    {
        //Finalizing the client
        Management.finalizeClient(cltID);
    }
}
```

なお，これ以外の実装例については，次に示すディレクトリに格納しているサンプルプログラムを参照してください。ただし，サンプルプログラムとして提供するのは，SOAP アプリケーション開発支援機能だけです。

```
<Application Serverのインストールディレクトリ>/c4web/samples
```

7

SOAP アプリケーションの実行と運用

開発した SOAP アプリケーションは，実行環境にデプロイして，SOAP アプリケーションを開始することで利用できます。開始した SOAP アプリケーションは，Web ブラウザまたはコマンドラインから利用します。

この章では，SOAP アプリケーションの実行方法と，コマンドラインからの利用方法について説明します。また，SOAP アプリケーション運用時の注意事項についても説明します。

7.1 SOAP アプリケーションの開始と停止

SOAP アプリケーションの開始または停止は，J2EE サーバ上で実行します。

サーバ管理コマンドで，SOAP アプリケーションを J2EE サーバにデプロイしてから開始してください。サーバ管理コマンドの使用方法については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」を参照してください。

7.2 SOAP サービスの表示

SOAP サービスを提供しているアプリケーションは，サーバ管理コマンドを使用して一覧表示できます。

サーバ管理コマンドの使用方法については，マニュアル「アプリケーションサーバ アプリケーション設定操作ガイド」を参照してください。

7.3　コマンドラインを利用した SOAP アプリケーションの実行

RPC 形態の SOAP アプリケーションでは，コマンドラインで動作する Java アプリケーションを，SOAP アプリケーションのクライアント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 SOAP アプリケーションの利用時に必要な設定，コマンドラインの指定例，および注意事項について示します。また，参考として，コマンドラインを利用する場合のクライアントの実装例を示します。

7.3.1　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 SOAP アプリケーションを利用するために必要な，次の設定について説明します。

- Java アプリケーション用オプション定義ファイルの設定
- 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
- パスの追加
- クライアント定義ファイルの設定

(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

Java アプリケーション用オプション定義ファイルを，クライアント SOAP アプリケーションを実行するカレントディレクトリに作成し，次のキーと値を追加します。

Java アプリケーション用オプション定義ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の 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。

- add.class.path=<クライアント SOAP アプリケーション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=<クライアント定義ファイルが格納された JAR またはディレクトリのパス>
- add.class.path=<Application Server のインストールディレクトリ>/c4web/lib/hitc4web.jar
- add.class.path=<Application Server のインストールディレクトリ>/c4web/lib/hitsaaj.jar
- add.class.path=<Application Server のインストールディレクトリ>/c4web/lib/hitjaxrpc.jar
- add.class.path=<Application Server のインストールディレクトリ>/c4web/lib/wsdl4j.jar
- add.class.path=hntrlib2j64.jar のパス※1
- add.jvm.arg=-Dcosminexus.home=<Application Server のインストールディレクトリ>※2
- add.jvm.arg=-Dejbserver.server.prf.PRFID=<PRF 識別子>※3

注※1

クラスパスの指定内容を示します。

<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> の部分は，次に示すコマンドの実行結果を指定します。

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

Windows 以外の場合：

/opt/hitachi/HNTRLib2/classes/hntrlib2j64.jar

注※2

プロパティ cosminexus.home を指定しなかった場合は，KDCCE0608-W のメッセージが出力され，処理が続行されます。

注※3

cprfstart で指定した PRF 識別子と同じものを指定します。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

Java アプリケーション用ユーザプロパティファイルを，クライアント SOAP アプリケーションを実行するカレントディレクトリに作成し，必要に応じて設定します。

Java アプリケーション用ユーザプロパティファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の 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。

(3) パスの追加

<Application Server のインストールディレクトリ>/PRF/bin を環境変数 PATH に追加します。

(4) クライアント定義ファイルの設定

クライアント定義ファイルでクライアント SOAP アプリケーションの動作を変更する場合，クライアント定義ファイルを格納したディレクトリ名を，クラスパスに追加します。詳細については，「7.3.1(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」を参照してください。

注意事項

- クライアント定義ファイルがない場合，またはクライアント定義ファイルを格納したディレクトリ名をクラスパスに追加していない場合は，クライアント定義ファイルは読み込まれません。この場合，実行時オプションの値はすべてデフォルト値で動作します。
- クライアント定義ファイルを格納したディレクトリ名を，クラスパスに複数追加した場合は，先に指定したディレクトリ下にあるクライアント定義ファイルが利用されます。

7.3.2　コマンドラインの実行

SOAP アプリケーションのクライアントを，コマンドラインから実行する場合の指定例を次に示します。次の例では，<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 識別子と同じものを指定してください。異なる場合は，KDCCT0001-E のメッセージが出力されますが，処理は続行されます。

(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：

```
<Application Serverのインストールディレクトリ>/PRF/bin/cprfstart
```

PRF 識別子としてデフォルト値以外を使用する場合：

```
<Application Serverのインストールディレクトリ>/PRF/bin/cprfstart <prfid>
```

cprfstart コマンドの詳細については，マニュアル「アプリケーションサーバ リファレンス コマンド編」を参照してください。

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

実行例を次に示します。

```
cjclstartap localhost.testMain
```

7.3.3　コマンドライン利用時の注意事項

(1)　トレースファイルおよびアプリケーションログの出力について

- トレースファイルおよびアプリケーションログは，デフォルトでは「<カレントディレクトリ>/logs/system/ejbcl/WS」以下に出力されます。
- 次の条件にあてはまる場合は，トレースファイルおよびアプリケーションログが出力されません。
  - クライアント動作定義ファイル中の「c4web.logger.log_file_prefix」の指定によって出力パスが OS の最大長を超える
  - アクセス権のないパスを含む
  - デバイスが用意されていないパスを含む（A:¥など）
  - パスとして許可されない文字を含む

この場合，デフォルトトレースファイルおよびデフォルトアプリケーションログに出力されます。

- デフォルトトレースファイル，およびデフォルトアプリケーションログに出力できない場合は，アプリケーションの起動に失敗します。
- Windows 系以外の OS 上で実行する場合，umask の設定値を"000"にしてください。"000"以外の値を設定すると，トレースファイルおよびアプリケーションログが出力されないことがあります。
- コマンドライン使用時の PRF トレースを取得する場合は，SOAP アプリケーション実行前に PRF デーモンを起動しておく必要があります。PRF デーモンが起動されていない場合はエラーメッセージおよび警告メッセージが出力され，PRF トレースが取得されないまま処理が続行されます。

7.4　コネクションプーリング

コネクションプーリングとは，SOAP クライアントが SOAP サーバと通信するときに，接続された一つのコネクションを複数の通信で再利用する機能です。一度接続された SOAP クライアントと SOAP サーバ間のコネクションをプールしておき，再度通信するときに，そのコネクションをプールから取り出して利用します。コネクション利用後は，コネクションを解放しないでプールに戻し，再度通信できる状態にします。コネクションプーリングによって，データの通信が発生するたびにコネクションの確立処理および解放処理を行わなくなるため，レスポンスタイムを短縮できます。

注意事項

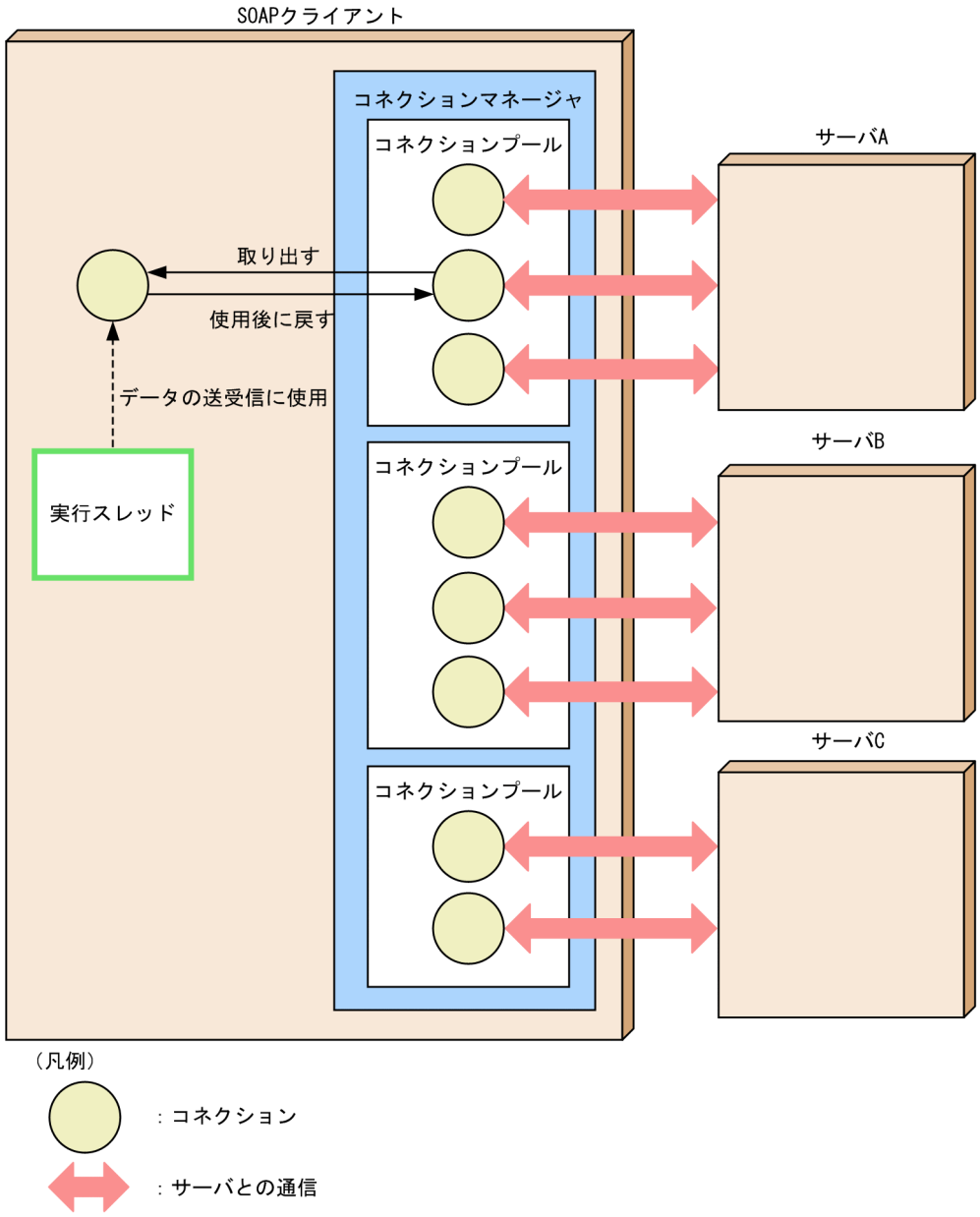
Application Server 08-00 および Developer 08-00 から，コネクションプーリングのデフォルト設定を無効（使用しない）に変更しています。コネクションプーリング機能を使用する場合，次に示す内容を十分に理解した上で使用してください。

- コネクションプーリングを使用する場合，SOAP クライアント側だけでなく，接続先 SOAP サービスが稼働する Web サーバや中継するネットワーク機器などの設定も見直す必要があります。また，Web サーバが TCP 接続を保持したままの状態となり，Web サーバを正常停止できなくなることがあります。
- Web サーバや中継するネットワーク機器などの設定を適切に変更しなかった場合，クライアント側で<状態コード>が 0，<詳細>が null，および<コンテントタイプ>が null であるメッセージを保持する KDCCP0005-E のエラーが発生することがあります。また，SOAP クライアントと接続先 SOAP サービスの間にあるネットワーク経路などでコネクションが切断されることでも同様のエラーが発生することがあります。

7.4.1　コネクションプーリングの概要

コネクションプーリングの概要を次の図に示します。

図 7-1　コネクションプーリングの概要



コネクションプールは，接続先サーバごとに作成されます。すべてのコネクションプールは，コネクションマネージャが管理しています。

サーバ A に対するコネクションプールは，サーバ A と接続されたコネクションを幾つか含んでいます。同様に，サーバ B およびサーバ C に対するコネクションプールも，サーバ B およびサーバ C と接続されたコネクションを幾つか含んでいます。

SOAP クライアントの実行スレッドの処理で，サーバ A との通信が必要になった場合，サーバ A に対するコネクションプールの中から，接続されたコネクションを取り出して通信します。

なお，コネクションプールには，最大コネクション数までコネクションを生成できます。最大コネクション数を超えないかぎり，一つのサーバに対するコネクションプールに生成できるコネクションの数は制限

されません。最大コネクション数の設定値および変更については，「7.4.2　コネクションプーリングに関する設定」を参照してください。

7.4.2　コネクションプーリングに関する設定

コネクションプーリングに関する設定は，共通定義ファイルで変更できます。共通定義ファイルで変更できる項目を次に示します。なお，共通定義ファイルの詳細については，「10.4　共通定義ファイルの設定」を参照してください。

なお，コネクションプールを使用する場合，「7.4　コネクションプーリング」に記載された注意事項を十分に理解した上で使用してください。

(1)　コネクションプーリングの有効化

コネクションプーリングはデフォルトで無効に設定されています。コネクションプーリングを有効にするには，共通定義ファイルに「c4web.common.connection_pool.enable=true」を追加し，J2EE サーバを停止・再起動してください。

なお，SSL 通信の場合，「c4web.common.connection_pool.enable=true」を追加しても，コネクションプーリングは使用できません。

(2)　コネクションの利用回数

コネクションを再利用できる最大数を変更できます。コネクションを再利用した回数がこの値を超えると，コネクションは切断および破棄されます。共通定義ファイルの「c4web.common.connection_pool.max_use」で変更します。デフォルトは，10,000（回）です。0 を指定すると，利用回数によるコネクションの切断および破棄は行われません。

(3)　コネクションのタイムアウト

コネクションを最後に利用してから，コネクションのタイムアウトが発生するまでの時間を変更できます。コネクションのタイムアウトが発生すると，そのコネクションは切断および破棄されます。共通定義ファイルの「c4web.common.connection_pool.timeout」で変更します。デフォルトは，1,800（秒）です。0 を指定すると，タイムアウトによるコネクションの切断および破棄は行われません。

コネクションのタイムアウトは，SOAP クライアント側では，共通定義ファイルの「c4web.common.connection_pool.timeout」で設定でき，Web サーバ側では，Web サーバ固有の設定方法で設定できます。SOAP クライアントと Web サーバの両方でタイムアウトが発生した場合は，コネクションは切断および破棄されます。そのため，Web サーバ側のコネクションタイムアウトの時間は，共通定義ファイルの「c4web.common.connection_pool.timeout」で設定した時間の倍以上を設定し，Web サーバ側のタイムアウトによってコネクションが切断されることのないようにしてください。例えば，共通定義ファイルの「c4web.common.connection_pool.timeout」の設定が 1,800（秒）の場合は，Web サーバ側のコネクションタイムアウトの設定を 3,600（秒）以上に設定してください。

(4) 最大コネクション数

各コネクションプールに生成できるコネクションの合計値である最大コネクション数を変更できます。例えば，コネクションプール A，B，C があり，最大コネクション数に 10（本）を設定した場合，コネクションプール A，B，C の三つに生成できるコネクション数の合計が 10（本）になります。共通定義ファイルの「c4web.common.connection_pool.max_connection」で変更します。デフォルトは，150（本）です。

7.4.3 コネクションが維持された状態での HTTP Server の終了方法

コネクションプーリングによってコネクションが維持された状態で，HTTP Server の終了処理を実行すると，HTTP Server は，維持されたコネクションが解放されるまで最大 30 秒待機します。そのため，Web サーバに HTTP Server を利用している場合は，HTTP Server の終了処理を実行しても，すぐに終了しないことがあります。このような場合に，HTTP Server をすぐに終了させる方法を次に示します。

(1) コマンドで終了させる方法

コマンドで終了させる方法を次に示します。

1. HTTP Server の HWSGracefulStopTimeout ディレクティブに，HTTP Server が終了するまでの最大待ち時間を設定します。
2. gracefulstop オプションを指定して，次のコマンドを実行します。

```
httpsd.exe -k gracefulstop
```

gracefulstop オプションを指定してコマンドを実行すると，最大で HWSGracefulStop ディレクティブに設定した時間だけ待機して，HTTP Server が終了します。HWSGracefulStop ディレクティブに小さい値を設定することで，コネクションプーリングによってコネクションが維持された状態でも，すぐに HTTP Server が終了します。

(2) Management Server から終了させる方法

Management Server から終了させる方法を次に示します。

1. 運用管理ポータルで［論理サーバの起動/停止］アンカーをクリックします。
2. ［サーバビュー］タブの［論理 Web サーバ］－［Web サーバ］－［<Web サーバ名>］をクリックします。
3. ［起動/停止］タブをクリックします。
4. ［強制停止］ボタンをクリックします。

7.4.4　コネクションの再利用条件

コネクションの再利用の可否は，SOAP クライアントと SOAP サーバが通信したときの状態によって異なります。

コネクションを再利用できる条件（コネクションを解放しない）

クライアントからのリクエストに含まれる HTTP バージョン，およびサーバからのレスポンスに含まれる HTTP バージョンが 1.1 の場合に，次のどちらかの状態のとき

- Connection ヘッダが Keep-Alive である
- Connection ヘッダが存在しない

コネクションを再利用できない条件（コネクションを解放する）

次のどれかの状態のとき

- サーバからのレスポンスに含まれる HTTP バージョンが 1.1 で，Connection ヘッダが close となっている場合
- サーバからのレスポンスに含まれる HTTP バージョンが 1.0 の場合
- SSL 通信（リクエスト URL に"https"を使用）の場合
- サーバからコネクションを切断された場合※1
- ソケット通信で例外が発生した場合※2
- HTTP 1.1 仕様に従っていない Chunk フォーマットがレスポンスデータに含まれている場合※2
- レスポンスデータが"HTTP"で開始されていない場合※2

注※1

切断されたコネクションは破棄されたあと再作成されます。再作成されたコネクションを利用して，サーバに再接続します。

注※2

ユーザプログラムには C4Fault が返されます。

7.5 SOAP アプリケーション運用時の注意事項

SOAP アプリケーション運用時の注意事項について説明します。

7.5.1 他社製品と接続する場合の多重配列送受信形式の確認

SOAP 通信基盤では，多重配列のデータ型を通信する場合，次に示すような SOAP メッセージが送受信されます。他社製品と接続する場合は，この形式が扱えるかどうかを確認して，多重配列のデータ型を送受信してください。他社製品と通信できない場合があります。

```
<in xsi:type="soapenc:Array" soapenc:arrayType="ns1:UserData[][2]"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
 <item xsi:type="soapenc:Array"
       soapenc:arrayType="ns1:UserData[3]">
  <item xsi:type="ns1:UserData">
   <Key xsi:type="xsd:string">key00</Key>
   <Value xsi:type="xsd:string">value00</Value>
  </item>
  <item xsi:type="ns1:UserData">
   <Key xsi:type="xsd:string">key01</Key>
   <Value xsi:type="xsd:string">value01</Value>
  </item>
  <item xsi:type="ns1:UserData">
   <Key xsi:type="xsd:string">key02</Key>
   <Value xsi:type="xsd:string">value02</Value>
  </item>
 </item>
 <item xsi:type="soapenc:Array"
       soapenc:arrayType="ns1:UserData[3]">
  <item xsi:type="ns1:UserData">
   <Key xsi:type="xsd:string">key10</Key>
   <Value xsi:type="xsd:string">value10</Value>
  </item>
  <item xsi:type="ns1:UserData">
   <Key xsi:type="xsd:string">key11</Key>
   <Value xsi:type="xsd:string">value11</Value>
  </item>
  <item xsi:type="ns1:UserData">
   <Key xsi:type="xsd:string">key12</Key>
   <Value xsi:type="xsd:string">value12</Value>
  </item>
 </item>
</in>
```

7.5.2　SOAP Fault に関する注意事項

(1)　faultactor 要素について

SOAP サービスから返される Fault メッセージには，faultactor 要素は含まれません。ただし，ユーザ実装で fault 生成者を指定した C4Fault 例外をスローした場合は除きます。

(2)　detail 要素について

SOAP ヘッダ要素に関係するエラーによって，SOAP サービスから Fault メッセージが返される場合，返される Fault メッセージは<detail/>を含む場合があります。また，SOAP ボディ要素に関係するエラーによって，SOAP サービスから Fault メッセージが返される場合，ユーザ実装で Fault 詳細を指定した C4Fault 例外をスローした場合を除いて，返される Fault メッセージに detail 要素は含まれません。

(3)　faultcode 要素について

SOAP では，Client，Server，VersionMismatch，MustUnderstand の四つの faultcode を定めていますが，SOAP 通信基盤では VersionMismatch，MustUnderstand だけ使用します。Client，Server に代わる faultcode として，SOAP 通信基盤独自の名前空間（http://c4web.cosminexus.com）を持つ Client，Server を使用します。また，これら SOAP 通信基盤独自の faultcode は，ドット表記による詳細コードを持つものがあります。

7.5.3　不正リクエストメッセージ受信時の動作について

リクエストのメッセージが HTTP リクエストとして正しい形式ではない，または XML が整形式（well-formed）ではないメッセージを受信した場合，HTTP 状態コードとして"400 Bad Request"を返します。その時，fault メッセージも送信しますので，他社製品と接続するときは注意してください。他社製品と通信できない場合があります。

7.5.4　エラーページ委任機能を使用する場合の注意事項

SOAP サービスが動作する環境（URL）で Application Server が提供するエラーページの委任機能を使用する場合，SOAP サービスが動作する URL に対しては，エラーコード 500 のエラーページ委任機能を設定しないようにしてください。

SOAP では，SOAP サービスでエラーが発生した場合に，SOAP Fault 電文を作成し，エラーコード 500 で SOAP クライアントに返信します。そのため，エラーページ委任機能で，エラーコード 500 を委任すると，プロトコル違反となり，SOAP クライアント側に不正な電文が送信されることになります。

なお，プロトコル違反の SOAP Fault 電文を SOAP クライアントが受け付けた場合の動作については，SOAP クライアント開発元に確認してください。エラーページ委任機能については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」を参照してください。

7.5.5　サーブレットのデフォルトマッピングについて

SOAP クライアント，および SOAP サービスを，Application Server が提供するデフォルトマッピングで呼び出さないでください。SOAP エンジン，または SOAP クライアントライブラリが予期しない動作をするおそれがあります。サーブレットのデフォルトマッピングについては，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」を参照してください。

7.5.6　リロード機能について

SOAP アプリケーションを Application Server が提供するリロード機能で入れ替えないでください。正しくアプリケーションが入れ替わらない場合があります。リロード機能については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」を参照してください。

7.5.7　異常発生時のアプリケーションログ出力機能について

SOAP 通信基盤では，SOAP Fault や C4Fault 例外が発生した場合など，異常を検出した場合にアプリケーションログを出力する機能を動作定義オプションで指定できます。障害発生時のトラブルシュート機能を強化するため，SOAP 通信基盤の動作定義オプションを使用することを強く推奨します。なお，動作定義ファイルの設定方法については，「10.　動作定義ファイルおよび実行時オプションの設定項目」を参照してください。

7.5.8　DD（web.xml）ファイルの記述内容について

SOAP アプリケーションの開発時に作成する DD（web.xml）ファイルの記述については，「3.9.2　DD（web.xml）の記述」を参照してください。

7.5.9　旧バージョンの環境の移行について

旧バージョンの環境で開発した SOAP アプリケーションを利用する場合，移行手順に従い，SOAP アプリケーションを移行する必要があります。移行手順については，「付録 A　SOAP アプリケーションの移行」を参照してください。

7.5.10　受信時サイズチェック機能について

受信したマルチパートメッセージの最初に SOAPEnvelope が含まれない場合，SOAPEnvelope より前に位置する添付データに対して，受信時サイズチェック機能が機能しません。

7.5.11　SOAP メッセージのエンコードに関する注意

UTF-16 でエンコードした SOAP メッセージを受信した場合，予期しないエラーまたは例外が発生することがあります。UTF-16 ではなく，UTF-8 を使用するようにしてください。

7.5.12　通信失敗時の Java 例外

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

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

このメッセージは，OS や環境によって出力内容が異なる場合があります。

8 UDDI クライアントの開発と実行

UDDI クライアントは，UDDI クライアントライブラリを使用して開発します。

この章では，UDDI クライアントライブラリを使用した UDDI クライアントの開発手順について説明します。また，実行時の環境設定，および実行手順についても説明します。

8.1　UDDI クライアントライブラリとは

UDDI クライアントライブラリとは，UDDI レジストリにアクセスするための機能を提供するためのライブラリです。UDDI クライアントライブラリが提供する API を使用して，UDDI レジストリにアクセスする UDDI クライアントを開発できます。

UDDI クライアントライブラリは，JAXR 1.0（Capability Profile Level 0）に準拠した Java の標準 API をサポートしています。JAXR 1.0 のサポート範囲については，「12.4　JAXR 1.0 との対応」を参照してください。

8.2 UDDI クライアントの開発手順

8.2.1 開発手順の詳細

ソースファイルを作成してから，UDDI クライアントとして実行するまでの流れを説明します。

(1) ソースファイルの作成

UDDI クライアントライブラリを使用して，ソースファイルを作成する一般的な手順を説明します（[ ]内は使用するメソッドを示します）。

注意事項

UDDI クライアントは SOAP クライアントとして動作するため，SOAP クライアントの開始，終了時に SOAP 通信基盤が提供する Management クラスの各メソッドを呼び出す必要があります。Management クラスに関しては「13.10 Management クラス（クライアントの開始，終了）」を参照してください。

1. ConnectionFactory オブジェクトを作成します。[ConnectionFactory#newInstance( )]
2. ConnectionFactory に接続する次のプロパティを設定します。
   [ConnectionFactory#setProperties( )]
   - javax.xml.registry.queryManagerURL
   - javax.xml.registry.lifeCycleManagerURL
3. ConnectionFactory オブジェクトから Connection オブジェクトを入手します。
   [ConnectionFactory#createConnection( )]
4. 接続モードを同期または非同期に設定します。[Connection#setSynchronous( )]
5. 認証が必要な操作をする場合は認証情報を設定します。[Connection#setCredentials( )]
6. Connection オブジェクトから RegistryService オブジェクトを入手します。
   [Connection#getRegistryService( )]
7. RegistryService オブジェクトから BusinessQueryManager または BusinessLifeCycleManager オブジェクトを入手します。[RegistryService#getBusinessQueryManager( )]または
   [RegistryService#getBusinessLifeCycleManager( )]
8. 操作に関する処理を実装します。
   - 検索（find）および取得（get）をする場合
     BusinessQueryManager オブジェクトの find 系および get 系メソッドを呼び出します。
   - 登録（save）および削除（delete）をする場合

BusinessLifeCycleManager オブジェクトの save 系および delete 系メソッドを呼び出します。登録の場合には，save 系メソッドのパラメタに渡すオブジェクトを BusinessLifeCycleManager オブジェクトの create 系メソッドで作成します。

9. 各操作を行う API から BulkResponse オブジェクトを取得します。API が正常に実行された場合は，結果の Collection オブジェクトを取得します。[BulkResponse#getCollection( )]

   API が正常に実行できたかどうかは，getStatus メソッドで確認します。

(2) ソースファイルのコンパイル

クラスパスに次の JAR ファイルを追加し，作成したソースファイルをコンパイルします。

- <Application Server のインストールフォルダ>/c4web/lib/hitjaxr.jar
- <Application Server のインストールフォルダ>/c4web/lib/hitc4web.jar

(3) UDDI クライアントの実行

環境設定を行い，UDDI クライアントを実行します。環境設定とプログラムの実行については以降で説明します。

8.3 UDDI クライアント実行時の環境設定

8.3.1 環境設定の詳細

UDDI クライアントを実行するには，事前に環境設定が必要になります。ここでは，設定が必要な JAR ファイルや接続プロパティについて説明します。また，UDDI クライアント実行時のトレースプロパティの設定や，動作定義ファイルの設定についても説明します。

(1) クラスパスに追加する JAR ファイル

UDDI クライアントを実行する場合に，クラスパスに追加する JAR ファイルを示します。追加する方法については，「8.4 UDDI クライアントの実行」を参照してください。

- <Application Server のインストールフォルダ>/c4web/lib/hitjaxr.jar
- <Application Server のインストールフォルダ>/c4web/lib/hitc4web.jar
- <Application Server のインストールフォルダ>/c4web/lib/hitsaaj.jar
- <Application Server のインストールフォルダ>/c4web/lib/hitjaxrpc.jar
- <Application Server のインストールフォルダ>/c4web/lib/wsdl4j.jar
- hntrlib2j.jar のパス※

注※

クラスパスの指定内容を示します。

<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> の部分は，次に示すコマンドの実行結果を指定します。

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

Windows 以外の場合：

/opt/hitachi/HNTRLib2/classes/hntrlib2j64.jar

また，必要に応じて次のシステムプロパティを追加する必要があります。

1. 接続プロパティ
2. トレースプロパティ

具体的な追加方法については，「8.4 UDDI クライアントの実行」を参照してください。

(2) 接続プロパティの設定

レジストリへの接続に関するプロパティをシステムプロパティで指定できます。

接続プロパティはJAXR APIのConnectionFactory#setProperties(Properties)メソッドを使用して，UDDIクライアントのプログラム内から動的に変更することもできます。この場合，setPropertiesメソッドで設定される値がシステムプロパティで指定した値よりも優先されます。

次に接続プロパティに関するシステムプロパティの一覧を示します。

表 8-1　接続プロパティ一覧

プロパティ名称	指定値	デフォルト値
javax.xml.registry.queryManagerURL	アクセスする先のUDDIレジストリサーバの問い合わせURLを指定します。URL形式として正しく指定する必要があります。 このプロパティは必ず指定してください。	−
javax.xml.registry.lifeCycleManagerURL	アクセスする先のUDDIレジストリサーバの発行URLを指定します。URL形式として正しく指定する必要があります。 このプロパティを指定しない場合，queryManagerURLプロパティの値が使用されます。	−
javax.xml.registry.semanticEquivalences	同じ意味を持つConceptのIDの組を次の形式で指定します。 <id1>,<id2>|<id3>,<id4>|… ここで，<id1>と<id2>および<id3>と<id4>が同じ意味を持つIDの組です。主にUDDIにユーザ定義の郵便住所体系を持つ場合に使用します。	−
javax.xml.registry.postalAddressScheme	郵便住所体系を示すClassificationSchemeオブジェクトのIDを指定します。	−
javax.xml.registry.security.authenticationMethod	レジストリサーバとの認証で使用する認証方法を指定します。 "UDDI_GET_AUTHTOKEN"は固定です。これ以外の認証方法はサポートされません。	"UDDI_GET_AUTHTOKEN"
javax.xml.registry.uddi.maxRows	UDDI APIのfind操作で返却される最大エントリ数を0以上の整数値で指定します。 指定しない場合，最大エントリ数はレジストリサーバのデフォルト値に従います。※	−

（凡例）

−：デフォルト値がないことを示します。

注※

このプロパティはUDDI APIのmaxRows属性を指定するプロパティです。したがって，JAXRのfind系APIの最大取得件数を指定する値ではありません。特にfindConceptsやfindClassificationSchemesメソッドでは指定したエントリ数より少なくなる場合があります。

(3)　トレースプロパティの設定

トレース出力に関するプロパティをシステムプロパティに設定できます。

注意事項

トレースプロパティが設定されない場合や，指定した値が有効でない場合には，トレースファイルに警告メッセージが記録され，そのプロパティのデフォルト値で処理が続行されます。

次にトレースプロパティに関するシステムプロパティの一覧を示します。

表 8-2　トレースプロパティ一覧

プロパティ名称	指定値	デフォルト値
com.cosminexus.xml.registry.trace.trace_level	トレースレベルを ERROR，WARN，INFO，DEBUG のどれかを指定します。トレースレベルについては「14.4.3　トレースファイル出力の重要度」を参照してください。	WARN
com.cosminexus.xml.registry.trace.file_path	トレースメッセージを記録するため作成されるログファイルのパスと名前を文字列で指定します。	<Application Server のインストールフォルダ>/c4web/logs/JAXRAPITrace
com.cosminexus.xml.registry.trace.file_num	トレースファイルの面数を 1～16 の数値で指定します。	2
com.cosminexus.xml.registry.trace.file_size	一つのトレースファイルの最大サイズをバイト単位で指定します。4,096～2,147,483,647 の数値を設定します。	2,097,152

(4)　SOAP アプリケーションの動作定義ファイルの設定

UDDI クライアントは SOAP クライアントとして動作するため，必要に応じて SOAP アプリケーションのクライアント定義ファイル，共通定義ファイルを設定します。

特に，HTTP プロキシサーバ経由でレジストリサーバにアクセスする場合，クライアント定義ファイルに次のキーを設定する必要があります。

- c4web.application.proxy_host
- c4web.application.non_proxy_hosts
- c4web.application.proxy_port
- c4web.application.proxy_user
- c4web.application.proxy_password

SOAP アプリケーションの動作定義ファイルの設定については，「10.　動作定義ファイルおよび実行時オプションの設定項目」を参照してください。

8.4 UDDI クライアントの実行

8.4.1 実行手順の詳細

UDDI クライアントライブラリを使用するには，J2EE サーバ上で実行する場合と，コマンドラインから実行する場合があります。次に，各場合でのプログラムの実行手順を説明します。

ここでは UDDI クライアントライブラリに関係のある手順だけを説明します。そのほかの詳細については，次のマニュアルを参照してください。

- アプリケーションサーバ システム構築・運用ガイド
- アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)
- アプリケーションサーバ 機能解説 基本・開発編(EJB コンテナ)

(1) J2EE サーバ上で実行する場合

J2EE サーバ上で動作するサーブレット，JSP，EJB などのコンポーネントから UDDI クライアントライブラリを使用する場合は，次の手順で実行します。

1. システムプロパティの設定

   usrconf.properties ファイルにシステムプロパティを設定します。

   ```
   （例）com.cosminexus.xml.registry.trace.trace_level=INFO
   ```

2. 実行

   J2EE サーバに，サーブレット，JSP，EJB などを配置し，サーバを実行します。

(2) コマンドラインから実行する

コマンドラインから実行するプログラムから UDDI クライアントライブラリを使用する場合，次の手順で実行します。

1. クラスパスを設定します。

   「8.3 UDDI クライアント実行時の環境設定」でクラスパスに追加するすべての JAR ファイルを，コンテナ拡張ライブラリ用の JAR として，Java アプリケーション用オプション定義ファイルの add.class.path キーに設定し，追加します。

   Java アプリケーション用オプション定義ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の 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。

2. 環境変数 PATH を設定します。

   <Application Server のインストールフォルダ>/PRF/bin を環境変数 PATH に設定します。

3. PRF トレースに関するプロパティを設定します。

PRF トレース使用時は，PRF トレースに関するプロパティの指定をする必要があります。次のプロパティを，JavaVM を起動するオプションとして，Java アプリケーション用オプション定義ファイルの add.jvm.arg キーに設定し，追加します。

プロパティ名称：ejbserver.server.prf.PRFID

値：cprfstart コマンドで指定した PRF 識別子

デフォルト値：PRF_ID

なお，PRF デーモンを起動しないでプログラムを実行した場合，KDCCT0001-E および KDCCT0004-W のメッセージが出力されますが，プログラムの動作に影響はありません。

Java アプリケーション用オプション定義ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の 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. 接続プロパティおよびトレースプロパティを設定します。

「8.3　UDDI クライアント実行時の環境設定」に示したシステムプロパティを，JavaVM を起動するオプションとして，必要に応じて Java アプリケーション用オプション定義ファイルの add.jvm.arg キーに設定し，追加します。

Java アプリケーション用オプション定義ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の 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. Application Server のインストールディレクトリをプロパティに設定します。

次のプロパティを，JavaVM を起動するオプションとして，Java アプリケーション用オプション定義ファイルの add.jvm.arg キーに設定し，追加します。

プロパティ名称：cosminexus.home

値：Application Server のインストールディレクトリ

デフォルト値：なし

Java アプリケーション用オプション定義ファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の 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. Java アプリケーション用ユーザプロパティファイルを設定します。

Java アプリケーション用ユーザプロパティファイルを，クライアント SOAP アプリケーションを実行するカレントディレクトリに作成し，必要に応じて設定します。

Java アプリケーション用ユーザプロパティファイルの詳細については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」の 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. Java アプリケーションの開始コマンド（cjclstartap）を実行します。

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

実行例を次に示します。

```
cjclstartap SampleClass
```

8.5 UDDI インタフェースおよびクラス

UDDI クライアントライブラリが提供する JAXR API のパッケージ，および JAXR API の一覧を示します。

8.5.1 JAXR のパッケージ

JAXR API は，主に次の二つのパッケージから成り立ちます。

- レジストリ

  このパッケージは，レジストリの情報内容について接続，問い合わせ，保存，更新，削除を行うためのクラスとインタフェースを含みます。パッケージ名称は，javax.xml.registry です。
- インフォモデル

  このパッケージは，リポジトリ内容のためのメタデータを記述して表現する，JAXR API のためのデータモデルを含みます。パッケージ名称は，javax.xml.registry.infomodel です。

8.5.2 JAXR API の一覧

JAXR API が提供するインタフェースとクラスの一覧を次に示します。

表 8-3 インタフェース／クラス一覧

種類	インタフェース／クラス名	説明
レジストリ	BulkResponse	API の実行結果（複数）を格納します。
	BusinessLifeCycleManager	ライフサイクル管理の上位 API を提供します。
	BusinessQueryManager	問い合わせ管理の上位 API を提供します。
	CapabilityProfile	JAXR 実装の機能範囲に関する情報を提供します。
	Connection	JAXR 実装と接続します。
	ConnectionFactory	JAXR の接続に関するファクトリを提供します。
	DeclarativeQueryManager	SQL などの明示的な形式で問い合わせます。
	FederatedConnection	複数のレジストリを単一の接続情報で扱います。
	FindQualifier	検索条件を示す定数を提供します。
	JAXRResponse	API の実行結果を格納するベースインタフェースです。
	LifeCycleManager	ライフサイクル管理のベース API を提供します。
	Query	明示的な形式の問い合わせ（SQL など）を管理します。
	QueryManager	問い合わせ管理のベース API を提供します。
	RegistryService	レジストリアクセスの基本となるインタフェースです。

種類	インタフェース／クラス名	説明
インフォモデル	Association	UDDI の publisherAssertion に相当します。
	AuditableEvent	履歴を表します。UDDI では使用しません。
	Classification	UDDI の categoryBag/keyedReference に相当します。
	ClassificationScheme	UDDI の tModel のうち分類を表現するものに相当します。
	Concept	UDDI の tModel のうち識別を表現するものに相当します。
	EmailAddress	UDDI の contact/email に相当します。
	ExtensibleObject	属性の拡張ができるオブジェクトのベースインタフェースです。
	ExternalIdentifier	UDDI の identifierBag/keyedReference に相当します。
	ExternalLink	UDDI の discoveryURL, overviewDoc に相当します。
	ExtrinsicObject	レジストリ外で管理するデータです。UDDI では使用しません。
	InternationalString	複数のロケール情報付き文字列（LocalizedString）の集合です。
	Key	RegistryObject を一意に識別するキーです。
	LocalizedString	ロケール情報付きの文字列です。
	Organization	UDDI の businessEntity に相当します。
	PersonName	UDDI の contact/personName に相当します。
	PostalAddress	UDDI の contact/address に相当します。
	RegistryEntry	オブジェクトの状態や期限を表します。UDDI では使用しません。
	RegistryObject	レジストリのオブジェクトを表すベースインタフェースです。
	RegistryPackage	RegistryEntry の集合です。UDDI では使用しません。
	Service	UDDI の businessService に相当します。
	ServiceBinding	UDDI の bindingTemplate に相当します。
	Slot	任意の属性を表現します。
	SpecificationLink	技術仕様に対するリンクです。UDDI の tModelInstanceInfo や instanceDetail に相当します。
	TelephoneNumber	UDDI の contact/phone に相当します。
	URIValidator	URI の検証を行うベースインタフェースです。
	User	UDDI の contact に相当します。
	Versionable	バージョンを表します。UDDI では使用しません。
例外	DeleteException	削除操作に関する例外です。
	FindException	検索操作に関する例外です。
	InvalidRequestException	不正な API メソッドを実行した場合の例外です。

種類	インタフェース／クラス名	説明
	JAXRException	すべての例外の基本クラスです。
	RegistryException	レジストリ側で検出される例外です。
	SaveException	保存操作に関する例外です。
	UnexpectedObjectException	不正な型のオブジェクトが指定された場合の例外です。
	UnsupportedCapabilityException	JAXR 実装がサポートしていないレベルのメソッドが発行された場合の例外です。

8.6 UDDI クライアント開発，実行時の注意事項

8.6.1 API リファレンスについて

SOAP 通信基盤は，レジストリアクセスの API である JAXR1.0 に準拠しています。API の使用方法は，http://java.sun.com/j2ee/1.4/docs/api/index.html を参照してください。JAXR1.0 に対応したパッケージ名を次に示します。

- javax.xml.registry
- javax.xml.registry.infomodel

8.6.2 Association で boolean を返すメソッドについて

Association インタフェースで，次に示す boolean を返すメソッドに対する正しい値を必要とする場合は，BusinessQueryManager#findCallerAssociations メソッド，または BusinessQueryManager#findAssociations メソッドを発行してください。

LifeCycleManager#createAssociation メソッドから生成された Association のインスタンスに対して次のメソッドを呼び出すと，一部の UDDI レジストリ環境では正しい結果が取得できない場合があります。

- boolean isConfirmed()
- boolean isConfirmedBySourceOwner()
- boolean isConfirmedByTargetOwner()
- boolean isExtramural()

8.6.3 findCallerAssociations メソッドの findQualifiers パラメタについて

BusinessQueryManager#findCallerAssociations メソッドの第一パラメタである findQualifiers の値は使用されません。値を指定した場合の動作は，値を指定していない場合の動作と同様になります。

8.6.4 findServiceBindings メソッドの classifications パラメタについて

BusinessQueryManager#findServiceBindings メソッドの第三パラメタである classifications の値は使用されません。値を指定した場合の動作は，値を指定していない場合の動作と同様になります。

8.6.5　Collection を返すメソッドについて

次に示すインタフェース，またはインタフェースを継承するインタフェースでの Collection のインスタンスを返すメソッドについては，Collection の要素はコピーされません。そのため，要素自体の内容を変更しないようにしてください。

- javax.xml.registry.BulkResponse
- javax.xml.registry.infomodel.ClassificationScheme
- javax.xml.registry.infomodel.Concept
- javax.xml.registry.infomodel.ExtensibleObject
- javax.xml.registry.infomodel.ExternalLink
- javax.xml.registry.infomodel.InternationalString
- javax.xml.registry.infomodel.Organization
- javax.xml.registry.infomodel.RegistryObject
- javax.xml.registry.infomodel.Service
- javax.xml.registry.infomodel.ServiceBinding
- javax.xml.registry.infomodel.Slot
- javax.xml.registry.infomodel.SpecificationLink
- javax.xml.registry.infomodel.User

Collection の要素自体を変更した場合，Collection を取得したインスタンスの状態も変更されるため，そのインスタンスの，ほかのメソッドの動作が不正になるおそれがあります。

8.6.6　SOAP アプリケーションを運用する場合の注意

UDDI クライアントライブラリを利用して運用する場合，固有の注意事項はありません。共通の注意事項を参照してください。共通の注意事項については，「7.5　SOAP アプリケーション運用時の注意事項」を参照してください。

9 開発支援コマンド

SOAP アプリケーション開発支援機能では，Java2WSDL コマンド（WSDL の生成），WSDL2Java コマンド（ソースコードの生成），および Java2WSDD コマンド（サービスデプロイ定義の生成）という 3 種類の開発支援コマンドを提供しています。

この章では，開発支援コマンドの使用方法および注意事項について説明します。

9.1　Java2WSDL コマンド（WSDL の生成）

WSDL の生成は次のコマンドを使用します。

形式

```
Java2WSDL.bat -l 〈サービスロケーション〉 ［その他のオプション群］〈対象となるクラス名〉
```

必須の-l オプションおよび任意に指定するオプションの一覧を示します。

表 9–1　Java2WSDL コマンドの-l オプション（必須）

オプション	意味
-l	サービスロケーションを指定します（service 要素の address location 属性）。サービスロケーションの指定方法については，「3.3.1　サービスロケーションの指定」を参照してください。

表 9–2　Java2WSDL コマンドのオプション一覧（任意）

オプション	意味
-h	ヘルプが表示されます。
-o	WSDL の出力先ファイル名を指定します。 省略時："servicePort 名.wsdl"の名称でカレントディレクトリに生成されます。
-P	portType 名を指定します（portType 要素の name 属性）。 省略時：指定クラス名が設定されます。
-b	binding 名を指定します（binding 要素の name 属性）。 省略時："servicePort 名"＋"SOAPBinding"が設定されます。
-S	serviceElement 名を指定します（service 要素の name 属性）。 省略時："portType 名"＋"Service"が設定されます。
-s	servicePort 名を指定します（service 要素内の port 要素の name 属性）。 省略時：サービス名（サービスロケーションの最終ノード名）が設定されます。
-n	targetNamespace を指定します。 省略時：クラスのパッケージ名から生成されます。 クラスにパッケージ名の指定がない場合は，"http://DefaultNamespace"が設定されます。
-p	パッケージ名と名前空間の関係を package=namespace の形式で指定します（名前空間の変更）。
-m	対象となるメソッドを空白またはコンマ（,）区切りで指定します。 省略時：すべてのメソッドが対象になります。
-w	WSDL の生成モードを All，Interface，Implementation のどれかで指定します。 省略時：All が設定されます。
-x	非対象メソッドを空白またはコンマ（,）区切りで指定します。 省略時：非対象メソッドは設定されません。
-T	TypeMappingVersion を 1.1 または 1.2 で指定します。

オプション	意味
	typeMappingVersion 値によって，Java クラスから生成される WSDL 内のデータ型が異なります。詳細は次の表を参照してください。 省略時：1.1 が設定されます。
-A	soapAction の値を指定します。DEFAULT を指定した場合は""を設定します。NONE を指定した場合，soapAction は生成されません。 省略時：DEFAULT が設定されます。
-z	WSDL のバインディングスタイルを RPC または DOCUMENT で指定します。大文字／小文字は区別されません。 省略時：RPC を設定します。 DOCUMENT を指定した場合は，wrapped 形式※の WSDL が生成されます。
-u	use 属性を LITERAL または ENCODED で指定します。大文字／小文字は区別されません。 省略時：LITERAL が設定されます。

注※

wrapped 形式については，「3.3.3(3)　DOCUMENT／LITERAL を指定した場合」を参照してください。

表 9-3　typeMappingVersion 値とデータ型の関係

クラス名	WSDL 内のデータ型			
	1.1 指定時		1.2 指定時	
	データ型	名前空間名	データ型	名前空間名
java.lang.String	string	xsd	string	soapenc
java.lang.Boolean	boolean	xsd	boolean	soapenc
java.lang.Double	double	xsd	double	soapenc
java.lang.Float	float	xsd	float	soapenc
java.lang.Integer	int	xsd	int	soapenc
java.math.BigInteger	integer	xsd	integer	soapenc
java.math.BigDecimal	decimal	xsd	decimal	soapenc
java.lang.Long	long	xsd	long	soapenc
java.lang.Short	short	xsd	short	soapenc
java.lang.Byte	byte	xsd	byte	soapenc
byte[]	base64Binary	xsd	base64	soapenc

（凡例）

xsd は名前空間 URI が"http://www.w3.org/2001/XMLSchema"であることを示します。

soapenc は名前空間 URI が"http://schemas.xmlsoap.org/soap/encoding/"であることを示します。

終了コード

0：正常終了

1：異常または警告終了

メッセージ

メッセージの詳細については，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」の開発支援コマンドによって出力されるメッセージの記述を参照してください。

注意事項

- 対象となるメソッドの引数または戻り値に，ユーザ定義のデータ型クラスを含む場合，WSDL 内でユーザ定義のデータ型クラスを表す型が定義されますが，このときの要素の並び順は，ユーザ定義のデータ型クラス内でのフィールドの順序に関係なく，アルファベット順となります。
- -z オプションで DOCUMENT，-u オプションで ENCODED の組み合わせは指定できません。
- Java2WSDL コマンドを実行して Java インタフェースから WSDL を生成する場合，Java インタフェースのメソッドのパラメタ名は，WSDL に「inXX」（XX は整数）の名称でマッピングされます。生成例については，「3.3.9(2)　Java インタフェースのメソッドのパラメタ名と生成される WSDL の対応」を参照してください。
- Java2WSDL コマンドを実行する場合，指定する Java インタフェースへのクラスパスを環境変数 CLASSPATH に設定してください。CLASSPATH の設定がない場合，Java2WSDL コマンドは，カレントディレクトリをクラスパスに設定し，指定された Java インタフェースを検索します。

9.2 WSDL2Java コマンド（ソースコードの生成）

WSDL からのソースコード生成は次のコマンドを使用します。

形式

```
WSDL2Java.bat -C validS ［その他のオプション群］〈対象となるWSDLファイル名〉
```

必須の-C オプションおよび任意に指定するオプションの一覧を示します。

表 9-4 WSDL2Java コマンドの-C オプション（必須）

オプション	意味
-C	WSDL を検証するかどうかを指定します。検証モードに validS，validOnlyS のどちらかを指定します。validS を指定した場合，指定した WSDL の検証が行われ，Java ソースコードが生成されます。validOnlyS を指定した場合，WSDL の検証だけが行われます。WSDL を検証した結果，WSDL に不正がある場合はエラーとなります。 省略時：WSDL は検証されません。

-C オプションの指定について

-C オプション（WSDL 検証機能）の指定は必須です。このオプションは入力された WSDL 内で，サポートされていない要素を使用していないかどうかを検証する機能です。

このオプションを指定しなかった場合，入力された WSDL 内でサポートされていない要素を使用していても，WSDL2Java コマンドが正常に終了することがあります。

表 9-5 WSDL2Java コマンドのオプション一覧（任意）

オプション	意味
-h	ヘルプが表示されます。
-v	生成過程が画面に表示されます。
-O	タイムアウト値を秒で指定します。 省略時：45 が設定されます。
-s	スケルトン（サーバ側に必要なソースコード）とサービスデプロイ定義が生成されます。 省略時：スタブ（クライアント側に必要なソースコード）が生成されます。
-N	名前空間とパッケージ名の関係を namespace=package の形式で指定します（パッケージ名）。 省略時：パッケージ名は変更されません。
-f	ファイル指定で名前空間とパッケージ名の関係を指定します。 省略時：パッケージ名は変更されません。
-p	すべてのクラスのパッケージ名を変更します。 省略時：パッケージ名は変更されません。
-o	ソース出力先ディレクトリを指定します。 省略時：カレントディレクトリが設定されます。

オプション	意味
-d	deployScope を Application，Request，Session のどれかで指定します。 省略時：Request が設定されます。
-a	メソッドのパラメタや，パラメタのクラスで参照されないユーザ定義のデータ型についても，ソースコードが生成されます。 省略時：参照されないユーザ定義のデータ型については，ソースコードは生成されません。
-L	入力された WSDL の記述内容に対するチェック機能を緩和します。 省略時：チェック機能は緩和されません。

WSDL2Java コマンドで生成されるソースコード

ソースコードは，カレントディレクトリ下にあるパッケージ名のフォルダ下に生成されます。例えば，カレントディレクトリが「C:¥SOAP」でパッケージ名が「jp.co.hitachi.soft」の場合は，「C:¥SOAP¥jp¥co¥hitachi¥soft」に生成されます。WSDL2Java コマンドで生成されるファイルの一覧を次の表に示します。

表 9-6　WSDL2Java コマンドで生成されるファイル一覧

生成ファイル名	名称	説明
<portType 要素の name 属性>.java	リモートインタフェース	スタブ，スケルトン共通インタフェースクラスです。
<service 要素の name 属性>.java	サービスインタフェース	サービスのインタフェースクラスです。
<service 要素の name 属性>Locator.java	サービスクラス	サービスへの接続情報を保持するクラスです。
<binding 要素の name 属性>Impl.java	スケルトンクラス	サーバ側に処理を実装するためのクラスです。
<binding 要素の name 属性>Stub.java	スタブクラス	Call オブジェクト生成などを行うスタブクラスです。
<types 要素に定義したデータ型名>.java	ユーザ定義のデータ型クラス	ユーザ作成のデータ型クラスです。0 または 1 個以上のデータ型クラスを作成します。

生成されるファイル名称に関して，次の注意が必要です。

- 生成されるフォルダ内にすでに同じ名称のソースコードがある場合，<binding 要素の name 属性>Impl.java（スケルトンクラス）以外は上書きされます。したがって，上書きされないようにするには，各要素の name 属性を変更したあとに，再度生成し直してください。
- 生成されるフォルダ内にすでに同じ名称のソースコードがある場合，<binding 要素の name 属性>Impl.java（スケルトンクラス）だけは上書きされません。したがって，インタフェースを変更するときは，<binding 要素の name 属性>Impl.java（スケルトンクラス）を直接修正するか，<binding 要素の name 属性>Impl.java（スケルトンクラス）を削除してから再度生成し直してください。

終了コード

0：正常終了

1：異常または警告終了

メッセージ

メッセージの詳細については，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」の開発支援コマンドによって出力されるメッセージの記述を参照してください。

注意事項

- -s オプションを指定して生成したサービスデプロイ定義（server-config.xml）は編集しないでそのままご使用ください。
- 指定する WSDL ファイルにはローカルファイルを指定してください。
- 実行中にエラーが発生した場合，ソースが途中まで生成されている場合があります。必要に応じて削除してください。
- WSDL ファイルのパスには，次の文字は使用できません。

  ? : * # % | < > "
- WSDL 定義の message 要素に含まれる part 要素に，element 属性と type 属性の両方を指定して，コマンドを実行したときの挙動を次に示します。

  WSDL のスタイルおよび use 属性が「RPC／LITERAL」または「RPC／ENCODED」の場合：

  element 属性を無視し，type 属性の記述に従って Java ソースコードが生成されます。

  WSDL のスタイルおよび use 属性が「DOCUMENT／LITERAL」の場合：

  WSDL2Java コマンドの実行時に，KDCCC0240-E メッセージが出力されて異常終了します。
- XML Schema に同じ名前の要素と型を定義する場合，wsdl:portType，wsdl:binding，および wsdl:service には同じ名前を使用しないでください。同じ名前を使用した場合，コンパイルできないソースコードが生成されます。
- WSDL には，同じ名前を設定できる要素があるため，生成される Java インタフェースおよび Java クラスの名前が衝突することがあります。名前が衝突した場合，WSDL2Java コマンドは，衝突したすべての Java インタフェースおよび Java クラスの名前に次のようなサフィックスを追加します。

表 9–7　名前衝突時に追加されるサフィックス

項番	WSDL または XML Schema の要素および型	Java インタフェースおよびクラスに追加されるサフィックス	例
1	Java クラスにマッピングされる XML Schema の型	_Type	（例 1） XML Schema: <xsd:complexType name="Shared"> <xsd:sequence> <xsd:element name="elem" type="xsd:string"/> </xsd:sequence> </xsd:complexType> Java: Shared_Type.java

項番	WSDL または XML Schema の要素および型	Java インタフェースおよびクラスに追加されるサフィックス	例
			(例 2) XML Schema： <xsd:simpleType name="Shared"> <xsd:restriction base="xsd:string"> <xsd:enumeration value="foo"/> </xsd:restriction> </xsd:simpleType> Java： Shared_Type.java
2	Java クラスにマッピングされる XML Schema の要素	_ElemType	XML Schema: <xsd:element name="Shared"> <xsd:complexType> <xsd:sequence> <xsd:element name="elem" type="xsd:string"/> </xsd:sequence> </xsd:complexType> </xsd:element> Java: Shared_ElemType.java
3	wsdl:portType 要素	_Port	WSDL: <wsdl:portType name="Shared"> ... Java: Shared_Port.java
4	wsdl:service 要素	_Service	WSDL： <wsdl:service name="Shared"> ... Java： Shared_Service.java
5	wsdl:fault 要素が参照している XML Schema の型	_Exception	WSDL： <wsdl:types> <xsd:schema ...> <xsd:complexType name="Shared"> ... </xsd:complexType> </xsd:schema> </wsdl:types> <wsdl:message name="UserException">

項番	WSDL または XML Schema の要素および型	Java インタフェースおよびクラスに追加されるサフィックス	例
			<wsdl:part name="fault" type="intf:Shared"/> </wsdl:message> ... <wsdl:portType name="UserInfo"> <wsdl:operation name="getUserInfo" parameterOrder="in0"> ... <wsdl:fault message="intf:UserException" name="UserException"/> </wsdl:operation> </wsdl:portType> Java： Shared_Exception.java

- wsdl:definitions 要素の targetNamespace 属性値からマッピングされる Java パッケージの名前と，次の要素および型の名前が同じ場合，Java パッケージの名前にサフィックスとして"_pkg"を追加します。
  - ・Java クラスにマッピングされる XML Schema の型
  - ・Java クラスにマッピングされる XML Schema の要素
  - ・wsdl:portType 要素
  - ・wsdl:service 要素
  - ・wsdl:fault 要素が参照している XML Schema の型
- -p オプションで IPv6 アドレスを指定した場合，指定した文字列がそのままパッケージ名になります。このとき，生成したクラスのパッケージ名が不正になり，コンパイルできないことがあります。コンパイルできない場合は，生成したクラスのパッケージ名を修正してください。
- スタブおよびスケルトンファイルをコンパイルすると，次に示す警告メッセージが出力されることがあります。

  ```
  注：入力ファイルの操作のうち，未チェックまたは安全ではないものがあります。
  注：詳細については，-Xlint:unchecked オプションを指定して再コンパイルしてください。
  ```

  この警告メッセージが出力された場合の対処は不要です。

  なお，「入力ファイルの操作のうち」の部分は，コンパイル時の入力ファイル名が出力されることがあります。
- 生成されるソースコードは，SOAP 通信基盤のバージョン 08-50（Component Container 08-50）対応のソースコードとなります。08-50 以外のバージョンの SOAP 通信基盤には，対応する SOAP アプリケーション開発支援機能でソースコードの再生成が必要になります。

-L オプションについて

-L オプションを指定した場合，WSDL の記述内容に関する次のチェックを緩和します。

- 別ファイルの XML Schema をインポートまたはインクルードし，対象の XML Schema がさらに別ファイルの XML Schema をインポートまたはインクルードする場合，間接的に XML Schema の要素および属性を参照することはできません。間接的に要素および属性を参照しようとした場合，WSDL2Java コマンドは KDCCC0137-E メッセージを出力し異常終了します。
- -s オプションを指定した場合，soap:address 要素の location 属性に次のどれかを指定したとき，WSDL2Java コマンドは KDCCC0182-E メッセージを出力し異常終了します。

  ・java.net.URL(String)コンストラクタで java.net.MalformedURLException が発生する文字列

  ・authority 部がない URL 文字列

  ・path 部がない URL 文字列

  ・プロトコルが「http」および「https」以外の URL 文字列
- 複数の wsdl:types 要素を記述した場合，WSDL2Java コマンドは KDCCC0247-E メッセージを出力し異常終了します。
- -s オプションを指定した場合，soap:address 要素の location 属性に，RFC3986 で規定された予約文字および非予約文字以外の文字列を指定したとき，WSDL2Java コマンドは KDCCC0273-E メッセージを出力し異常終了します。
- -s オプションを指定した場合，soap:address 要素の location 属性に，文字列長が 8,190 バイトを超える文字列を指定したときに，WSDL2Java コマンドは KDCCC0274-E メッセージを出力し異常終了します。
- 次の条件のどちらかの場合，WSDL2Java コマンドは KDCCC0275-E メッセージを出力し異常終了します。

  ・「http://www.w3.org/1999/XMLSchema」の XML Schema を使用した場合。

  ・「http://www.w3.org/2000/10/XMLSchema」の XML Schema を使用した場合。
- 次の条件のどれかの場合，WSDL2Java コマンドは KDCCC0277-E メッセージを出力し異常終了します。

  ・WSDL のインポートで，wsdl:import 要素の location 属性で指定したファイルパスを絶対パスに換算した文字数が 256 文字を超える場合。

  ・XML Schema のインポートで，xsd:import 要素の schemaLocation 属性で指定したファイルパスを絶対パスに換算した文字数が 256 文字を超える場合。

  ・XML Schema のインクルードで，xsd:include 要素の schemaLocation 属性で指定したファイルパスを絶対パスに換算した文字数が 256 文字を超える場合。
- xsd:schema 要素の子要素に複数の xsd:import 要素を記述した場合，schemaLocation 属性の値が同じとき，WSDL2Java コマンドは KDCCC0278-E メッセージを出力し異常終了します。
- 次の条件のどれかの場合，WSDL2Java コマンドは KDCCC0279-E メッセージを出力し異常終了します。

  ・WSDL のインポートで，インポート対象の wsdl:definitions 要素の targetNamespace 属性と，インポート元の wsdl:import 要素の namespace 属性とが異なる場合。

  ・XML Schema のインポートで，インポート対象の xsd:schema 要素の targetNamespace 属性と，インポート元の xsd:import 要素の namespace 属性とが異なる場合。

・XML Schema のインクルードで，インクルード対象の xsd:schema 要素の targetNamespace 属性と，インクルード元の xsd:schema 要素の targetNamespace 属性とが異なる場合。

- 同じ WSDL 内の XML Schema をインポートまたはインクルードし，対象の XML Schema がさらに同じ WSDL 内の XML Schema をインポートまたはインクルードする場合，間接的に XML Schema の要素および属性を参照しようとしたとき，WSDL2Java コマンドは KDCCC0280-E メッセージを出力し異常終了します。
- wsdl:types 要素を記述する場合，wsdl:documentation 要素および wsdl:import 要素を除く，ほかのすべての要素よりも前に記述していないとき，WSDL2Java コマンドは KDCCC0283-E メッセージを出力し異常終了します。
- WSDL をインポートする場合，次の条件のどちらも満たさないとき，WSDL2Java コマンドは KDCCC0284-E メッセージを出力し異常終了します。

  ・次の要素の組み合わせ

  　・インポート元：wsdl:binding，wsdl:service

  　・インポート対象：wsdl:types※，wsdl:message，wsdl:portType

  ・次の要素の組み合わせ

  　・インポート元：wsdl:service

  　・インポート対象：wsdl:types※，wsdl:message，wsdl:portType，wsdl:binding

  注※

  　wsdl:types 要素を省略する場合は不要です。
- wsdl:import 要素を記述する場合，wsdl:documentation 要素を除く，ほかのすべての要素よりも前に記述していないとき，WSDL2Java コマンドは KDCCC0285-E メッセージを出力し異常終了します。
- xsd:schema 要素の子要素に複数の xsd:include 要素を記述した場合，schemaLocation 属性の値が同じとき，WSDL2Java コマンドは KDCCC0287-E メッセージを出力し異常終了します。
- wsdl:definitions 要素の子要素に複数の wsdl:import 要素を記述した場合，location 属性の値が同じとき，WSDL2Java コマンドは KDCCC0288-E メッセージを出力し異常終了します。
- 次の条件のどちらかの場合，WSDL2Java コマンドは KDCCC0290-E メッセージを出力し異常終了します。

  ・XML Schema のインクルードで，インクルード対象の xsd:schema 要素の targetNamespace 属性，およびインクルード元の xsd:schema 要素の targetNamespace 属性が省略されている場合。

  ・XML Schema のインクルードで，インクルード対象の xsd:schema 要素の targetNamespace 属性，およびインクルード元の xsd:schema 要素の targetNamespace 属性の値が空文字の場合。
- 次のどれかの属性に対して，RFC3986 で規定された予約文字および非予約文字以外の文字列を使用する場合，UTF-8 でパーセントエンコードしていないとき，WSDL2Java コマンドは KDCCC0291-E メッセージを出力し異常終了します。

  ・wsdl:definitions 要素の targetNamespace 属性

  ・wsdl:definitions 要素の名前空間宣言

・wsdl:import 要素の namespace 属性
・wsdl:import 要素の location 属性
・xsd:schema 要素の targetNamespace 属性
・xsd:schema 要素の名前空間宣言
・xsd:import 要素の namespace 属性
・xsd:import 要素の schemaLocation 属性
・xsd:include 要素の namespace 属性
・xsd:include 要素の schemaLocation 属性

- -s オプションを指定した場合，wsdl:port 要素の name 属性を日本語で定義し，wsdl:port 要素の name 属性をパーセントエンコードした値と soap:address 要素の location 属性の URL パスの末尾が一致しないとき，WSDL2Java コマンドは KDCCC0292-E メッセージを出力し異常終了します。
- -s オプションを指定した場合，wsdl:port 要素の name 属性と soap:address 要素の location 属性の URL パスの末尾が一致しないとき，WSDL2Java コマンドは KDCCC0293-E メッセージを出力し異常終了します。
- 次の条件のどちらかの場合，WSDL2Java コマンドは KDCCC0300-E メッセージを出力し異常終了します。

  ・WSDL のインポートで，wsdl:definitions 要素の targetNamespace 属性と wsdl:import 要素の namespace 属性が同じ名前空間の場合。

  ・XML Schema のインポートで，xsd:schema 要素の targetNamespace 属性と xsd:import 要素の namespace 属性が同じ名前空間の場合。
- 次の条件のどれかの場合，WSDL2Java コマンドは wsdl:import 要素，xsd:import 要素または xsd:include 要素を無視します。

  ・WSDL のインポートで，wsdl:import 要素の location 属性に相対パス以外（絶対パスおよび http://や ftp://などのプロトコルで始まる URI など）を指定した場合。

  ・XML Schema のインポートで，xsd:import 要素の schemaLocation 属性に相対パス以外（絶対パスおよび http://や ftp://などのプロトコルで始まる URI など）を指定した場合。

  ・XML Schema のインクルードで，xsd:include 要素の schemaLocation 属性に相対パス以外（絶対パスおよび http://や ftp://などのプロトコルで始まる URI など）を指定した場合。

9.3　Java2WSDD コマンド（サービスデプロイ定義の生成）

サービスデプロイ定義（server-config.xml）の生成は，次のコマンドを使用します。

形式

```
Java2WSDD.bat ［オプション群］〈対象となるクラス名〉
```

表 9–8　Java2WSDD（server-config.xml）コマンドのオプション一覧

オプション	意味	備考
-h	ヘルプが表示されます。	−
-o	server-config.xml の出力先ディレクトリを指定します。 省略時：カレントディレクトリが設定されます。	−
-s	serviceElement 名を指定します（service 要素の name 属性）。 省略時：クラス名が設定されます。	−
-p	provider 種別名を指定します。RPC，EJB または C4MSG（メッセージング）で指定します。 省略時：RPC が設定されます。	−
-d	deployScope を Application，Request，Session のどれかで指定します。 省略時：Request が設定されます。	−
-u	use 属性を LITERAL または ENCODED で指定します。大文字／小文字は区別されません。 省略時：LITERAL が設定されます。	−
-T	typeMappingVersion を 1.1 または 1.2 で指定します。 typeMappingVersion 値で，Java クラスから生成される WSDL 内のデータ型が異なります。詳細は「9.1　Java2WSDL コマンド（WSDL の生成）」を参照してください。 省略時：1.1 が設定されます。	−
-z	WSDL のバインディングスタイルを RPC または DOCUMENT で指定します。大文字／小文字は区別されません。 省略時：RPC が設定されます。	−
-m	対象となるメソッドを空白またはコンマ（,）区切りで指定します。 省略時：すべてのメソッドが対象になります。	RPC 用
-x	非対象メソッドを空白またはコンマ（,）区切りで指定します。 省略時：非対象メソッドは設定されません。	RPC 用
-B	EJB の Bean クラス名を指定します。	EJB 用 省略不可
-H	EJB のホームインタフェースクラス名を指定します。	EJB 用 省略不可
-J	JNDI 名前空間を指定します。 EJBHome オブジェクトリファレンスの JNDI 名前空間を次のどちらかの方式で指定します。 ・JNDI 名前空間の直接指定します。	EJB 用 省略不可

オプション	意味	備考
	（例）HITACHI_EJB/SERVERS/MyServer/EJB/UserInfo/UserInfoBean ・ネーミング機能で指定します。 （例）java:comp/env/ejb/UserInfoBean 指定値の詳細は，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(コンテナ共通機能)」を参照してください。	
-C	CORBA Naming Service へアクセスするための URL を指定します。 省略時：corbaname::localhost:900 が設定されます。	EJB 用
-S	SOAP サービスに SOAP ヘッダ解釈処理を実装している場合は true，実装していない場合は false を指定します。 省略時：false が設定されます。	C4MSG（メッセージング）用

終了コード

0：正常終了

1：異常または警告終了

メッセージ

メッセージの詳細については，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」の開発支援コマンドによって出力されるメッセージの記述を参照してください。

注意事項

- 生成された server-config.xml は編集しないでそのままご使用ください。
- EJB プロバイダ用の server-config.xml を生成する場合，class-of-portType にはインタフェースクラスを指定してください。
- -S オプションの指定値による動作について

  SOAP ヘッダ項目の解釈処理の実装とは次の処理のことです。

  呼び出される SOAP サービスが最終送付先の場合：

  actor 属性値が SOAP サービス自身を示す URI であり，かつ mustUnderstand 属性値が"1"である全ヘッダ項目に関する処理を実装しています。

  呼び出される SOAP サービスが仲介者の場合：

  actor 属性値が SOAP サービス自身を示す URI であり，かつ mustUnderstand 属性値が"1"である全ヘッダ項目に関して処理し，そのヘッダ項目を削除する処理を実装しています。

  処理できない場合は，C4Fault をスローする処理を実装しています。

  false：

  SOAP ヘッダ内の mustUnderstand 属性が"1"の場合，サービスを呼び出さないで C4Fault を返します。mustUnderstand 属性が"0"または指定がない場合はサービスを呼び出します。

  true：

  呼び出されるサービスは上記の SOAP ヘッダの解釈処理を実装しなければなりません。

- EJB プロバイダ用の server-config.xml を生成する場合，<Application Server のインストールフォルダ>/CC/client/lib/j2ee-javax.jar をクラスパスに追加してください。クラスパスが通っていない場合，KDCCC0011-E エラーが発生する場合があります。
- -z オプションで DOCUMENT，-u オプションで ENCODED の組み合わせは指定できません。
- -p オプションに EJB を指定しても，EJB インタフェースを指定しなかった場合に出力される KDCCC0226-E メッセージが，KDCCC0011-E メッセージの details に出力されます。
- Java2WSDD コマンドを実行する場合は，指定するクラスへのクラスパスを環境変数 CLASSPATH に設定してください。CLASSPATH の設定がない場合，Java2WSDD コマンドは，カレントディレクトリをクラスパスに設定し，指定されたクラスを検索します。

9.4　開発支援コマンドに関する注意事項

開発支援コマンドに関する注意事項について説明します。

9.4.1　バッチファイル実行時の注意

各バッチファイルは，インストールされたディレクトリから，ほかのディレクトリへはコピーしないで，セットアップ後の状態で使用してください。KDCCC0012-E のメッセージが出力され，コマンドが正常に実行されない場合があります。

9.4.2　環境変数 CLASSPATH について

Java2WSDL コマンドおよび Java2WSDD コマンドを実行する場合，環境変数 CLASSPATH を設定しているかどうかでカレントディレクトリの扱いが異なります。

- 環境変数 CLASSPATH を設定している場合
  コマンドの対象となるクラスの検索対象に，カレントディレクトリは含まれません。含める場合は，環境変数 CLASSPATH にカレントディレクトリを表す"."を含めてください。例えば，すでに環境変数 CLASSPATH として"C:¥works¥build¥classes;C:¥apps¥someapp¥lib¥"を設定している場合，"C:¥works¥build¥classes;C:¥apps¥someapp¥lib¥;."としてください。
- 環境変数 CLASSPATH を設定していない場合
  コマンドの対象となるクラスの検索対象に，カレントディレクトリが含まれます。

9.4.3　開発支援コマンドの最大入力文字数について

開発支援コマンドで入力できる文字数は，8,190 バイトまでです。8,190 バイトを超えて入力した場合，コマンドが正常に実行されません。

9.4.4　Windows 使用時の注意

開発支援コマンドは，管理者特権で実行する必要があります。開発支援コマンドは，「管理者：コマンドプロンプト」で実行してください。

9.4.5　トレースとコンソール画面の出力内容に関する注意

開発コマンドを実行した場合に，コンソール画面に出力されている内容がトレースに出力されない，またはトレースに出力されている内容がコンソール画面に出力されないことがあります。

障害発生時には，コンソール画面とトレースに出力された内容を参照してください。

9.4.6 INFORMATION メッセージおよび WARN メッセージの出力について

トレースファイル出力の重要度を ERROR または WARN に設定しても，KDCCC0222-I の INFORMATION メッセージがトレースファイルに出力されることがあります。

また，トレースファイル出力の重要度を ERROR に設定しても，KDCCC0202-W，KDCCC0211-W，および KDCCC0221-W の INFORMATION メッセージがトレースファイルに出力されることがあります。次の場合に出力されます。

- Java2WSDL コマンドで KDCCC0202-W をトレースファイルに出力した場合
- WSDL2Java コマンドで KDCCC0211-W をトレースファイルに出力した場合
- Java2WSDD コマンドで KDCCC0211-W，KDCCC0221-W，および KDCCC0222-I をトレースファイルに出力した場合

10 動作定義ファイルおよび実行時オプションの設定項目

SOAP アプリケーションの実行時の動作や運用環境を詳細に定義したい場合，実行時オプションとして動作定義ファイルに定義できます。

この章では，動作定義ファイルおよび実行時オプションの設定項目について説明します。

10.1　動作定義ファイルの記述規則

SOAP アプリケーションの動作定義ファイルには，次の種類があります。

サーバ定義ファイル

サーバ側の動作を制御します。

クライアント定義ファイル

クライアント側の動作を制御します。

共通定義ファイル

サーバ側およびクライアント側の共通の動作を制御します。

これらの動作定義ファイルの記述規則を示します。

- J2SE のプロパティ形式に従い，「キー名称=値」の形式で記述します。
- 1 行の終わりは必ず改行されていなければなりません。
- 「#」で始まる行はコメントとみなされます。
- 値が存在しない行を設定した場合，空文字として処理されます。
- キー名称と=の間，および=と値の間にスペースを入れてはいけません。
- 値に 2 バイト文字を使用する場合は，文字コードを UTF-8 にしなければなりません。
- 設定できるキー名称以外のキー名称を設定した場合，その行は無視されます。
- 識別子以外のキー名称はすべて小文字で設定しなければなりません。大文字が混在した場合，その行は無視されます。
- 値には半角スペースも設定できます。

  （例）

  c4web.logger.log_file_prefix=File Prefix

10.2　サーバ定義ファイルの設定

サーバ定義ファイルは，「c4websv.cfg」というファイル名称で，インストール時に次の場所に格納されます。

Windows の場合

```
<Application Serverのインストールフォルダ>¥c4web¥conf
```

Windows 以外の場合

```
/opt/Cosminexus/c4web/conf
```

サーバ定義ファイルには，設定できるパラメタをサーバごとに複数ブロック設定できます。

サーバ定義ファイルに設定した内容を反映するには，SOAP アプリケーションを再起動する必要があります。設定した内容は，SOAP アプリケーション再起動後の初回リクエストを受信したときに反映されます。なお，サーバ定義ファイルは，SOAP アプリケーションを停止してから変更してください。SOAP アプリケーションの再起動および停止については，「7.1　SOAP アプリケーションの開始と停止」を参照してください。

次の表に設定するキー名称と値の一覧について示します。

表 10-1　サーバ定義ファイルの設定項目

設定項目	キー名称	値	デフォルト値 （値省略時の仮定値）
トレースファイル，アプリケーションログのプレフィクス	c4web.logger.<識別子>.log_file_prefix	任意の文字列を設定します。 「<log_file_prefix>-j2ee-<J2EE サーバのサーバ名>-<log_file_num>.log」がトレースファイルの名称となります。 「<log_file_prefix>-j2ee-<J2EE サーバのサーバ名>-aplog-<log_file_num>.log」がアプリケーションのログ名称となります。	識別子値
多重参照（実行時オプション）	c4web.common.<識別子>.do_multirefs	多重参照を有効にするかどうかを設定します。true または false を設定します。※1	false
データ型定義（実行時オプション）	c4web.common.<識別子>.send_xsi_types	データ型情報を送信するかどうかを設定します。true または false を設定します。※1	true
未宣言要素の無視※3	c4web.common.<識別子>.ignore_undeclared_elements	XML スキーマ（DOCUMENT スタイルの wrapped 形式で引数および戻り値をラップした要素を除く）で要素宣言していない XML インスタンスを含む SOAP メッセージを受信した	false

設定項目	キー名称	値	デフォルト値 （値省略時の仮定値）
		場合，この XML インスタンスを無視するかどうかを設定します。true または false を設定します。※1	
未宣言パラメタの無視※4	c4web.common.<識別子>.ignore_undeclared_parts	WSDL（RPC スタイルの wsdl:part 要素，および DOCUMENT スタイルの wrapped 形式で引数および戻り値をラップした要素）で宣言していないパラメタを含む SOAP メッセージを受信した場合，このパラメタを無視するかどうかを設定します。true または false を設定します。※1	false
SOAP ヘッダの名前修飾チェックオプション（実行時オプション）	c4web.common.<識別子>.enable_soapheader_check	SOAP ヘッダの子エレメントの名前修飾をチェックするかどうかを指定します。true または false を設定します。※1	true
性能解析トレース出力オプション	c4web.common.<識別子>.prf_trace_level	ALL，NONHANDLER のどちらかを指定します。 ALL： 次のトレース取得ポイントでトレースファイルが出力されます。 • 送信側ハンドラの呼び出し • ハンドラの呼び出し • 受信側ハンドラの呼び出し • 送信側ハンドラからの戻り • ハンドラからの戻り • 受信側ハンドラからの戻り NONHANDLER： 次のトレース取得ポイントでトレースファイルが出力されません。 • 送信側ハンドラの呼び出し • ハンドラの呼び出し • 受信側ハンドラの呼び出し • 送信側ハンドラからの戻り • ハンドラからの戻り • 受信側ハンドラからの戻り 性能解析トレースについては，「14.7　性能解析トレース」を参照してください。	ALL
文字参照形式オプション	c4web.common.<識別子>.character_reference	送信メッセージ中の文字を文字参照形式にエンコードするかどうかを指定します。true または false を設定します。※1	false

設定項目	キー名称	値	デフォルト値 （値省略時の仮定値）
送信できる SOAPEnvelope の最大バイト数	c4web.common.<識別子>.send_max_soap_envelope_size	送信できる SOAPEnvelope の最大バイト数を設定します。設定値は 0～2,147,483,647 の数値を設定します。0 を指定した場合は無制限となります。	0
受信できる SOAPEnvelope の最大バイト数	c4web.common.<識別子>.receive_max_soap_envelope_size	受信できる SOAPEnvelope の最大バイト数を設定します。設定値は 0～2,147,483,647 の数値を設定します。0 を設定した場合は無制限となります。	0
HTTP セッションの維持（実行時オプション）	c4web.application.<識別子>.app_maintainsession	ほかの SOAP サーバのサービスメソッドを呼び出す場合に，HTTP セッションを維持するかどうかを設定します。true または false を設定します。※1	false
プロキシサーバのホスト名または IP アドレス（実行時オプション）	c4web.application.<識別子>.proxy_host	ほかの SOAP サーバのサービスメソッドを呼び出す場合に，プロキシサーバのホスト名または IP アドレスを設定します。ホスト名または IP アドレスを示す文字列を設定します。	－
プロキシサーバを使用しないホスト名または IP アドレス群（実行時オプション）	c4web.application.<識別子>.non_proxy_hosts	ほかの SOAP サーバのサービスメソッドを呼び出す場合に，プロキシサーバを使用しないホスト名または IP アドレス群を設定します。ホスト名または IP アドレスを示す文字列を設定します。複数のホスト名または IP アドレスを指定する場合「|」で区切って設定します。※2	－
プロキシサーバのポート番号（実行時オプション）	c4web.application.<識別子>.proxy_port	ほかの SOAP サーバのサービスメソッドを呼び出す場合に，ポート番号を示す文字列を設定します。	－
プロキシサーバの認証ユーザ ID（実行時オプション）	c4web.application.<識別子>.proxy_user	ほかの SOAP サーバのサービスメソッドを呼び出す場合に，プロキシサーバの認証ユーザ ID を設定します。任意の文字列を設定します。※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（実行時オプション）	c4web.application.<識別子>.proxy_password	ほかの SOAP サーバのサービスメソッドを呼び出す場合に，認証ユーザ ID に対応するパスワードを設定します。任意の文字列を設定します。※2	－
サーバ兼クライアントのソケットの書き込みタイムアウト値	c4web.application.<識別子>.socket_write_timeout	ほかの SOAP サーバのサービスメソッドを呼び出す場合に，ソケットの書き込みタイムアウト値を秒単位	60

設定項目	キー名称	値	デフォルト値 (値省略時の仮定値)
		で設定します。0～86,400 の数値を設定します。0 を指定した場合はタイムアウト監視をしません。	
サーバ兼クライアントのソケットの読み込みタイムアウト値	c4web.application.<識別子>.socket_read_timeout	ほかの SOAP サーバのサービスメソッドを呼び出す場合に，ソケットの読み込みタイムアウト値を秒単位で設定します。0～86,400 の数値を設定します。0 を指定した場合はタイムアウト監視をしません。	300
サーバ兼クライアントのソケットの接続タイムアウト値	c4web.application.<識別子>.socket_connect_timeout	ほかの SOAP サーバのサービスメソッドを呼び出す場合に，ソケットの接続タイムアウト値を秒単位で設定します。0～86,400 の範囲で数値を設定します。0 を指定した場合はタイムアウト監視をしません。	60
Java ソースファイルおよびクラスファイルの出力ディレクトリ	c4web.application.<識別子>.dii_temp_directory	DII でユーザ定義のデータ型クラスを使用する場合に，WSDL 定義から動的に出力される Java ソースファイルおよびクラスファイルの出力先ディレクトリを，ローカルディレクトリ上の絶対パスで設定します。設定値は 1～128 文字の範囲で設定します。なお，ディレクトリ名に指定できる文字は，使用している OS に依存します。また，ディレクトリ名のパスの区切りには，次の文字を使用してください。 Windows の場合： 「/」または「¥¥」を使用してください。 Windows 以外の場合： 「/」を使用してください。	<Application Server のインストールディレクトリ>/c4web/dii/temp
WSDL 解析および Java ソース生成のタイムアウト値	c4web.application.<識別子>.dii_wsdl_parse_timeout	DII を使用する場合に，WSDL 解析および Java ソース生成のタイムアウト値を秒単位で設定します。0～3,600 の範囲で数値を設定します。0 を指定した場合はタイムアウト監視をしません。	45
ホスト名の検証	c4web.application.<識別子>.hostname_verification.enable	ホスト名の検証を有効にするかどうかを設定します。true または false を設定します。※1	true

設定項目	キー名称	値	デフォルト値 （値省略時の仮定値）
SOAPAction 値の扱い（実行時オプション）	c4web.server.<識別子>.fault_omit_soapaction	SOAPAction 値がない場合の実行時オプションを設定します。true または false を設定します。※1	true
送信できる添付データの最大個数	c4web.attachment.<識別子>.send_max_attachment_count	送信できる添付データの最大個数を設定します。設定値は 0～2,147,483,647 の数値を設定します。0 を設定した場合は無制限となります。	100
受信できる添付データの最大個数	c4web.attachment.<識別子>.receive_max_attachment_count	受信できる添付データの最大個数を設定します。設定値は 0～2,147,483,647 の数値を設定します。0 を設定した場合は無制限となります。	0
送信できる各添付データの最大バイト数	c4web.attachment.<識別子>.send_max_attachment_size	送信できる各添付データの最大バイト数を設定します。設定値は 0～2,147,483,647 の数値を設定します。0 を設定した場合は無制限となります。	104,857,600
受信できる各添付データの最大バイト数	c4web.attachment.<識別子>.receive_max_attachment_size	受信できる各添付データの最大バイト数を設定します。設定値は 0～2,147,483,647 の数値を設定します。0 を設定した場合は無制限となります。	0
退避ファイルを格納するディレクトリ名	c4web.attachment.<識別子>.attachment_temp_directory	添付ファイル付き SOAP メッセージを受信したときに生成される退避ファイルの出力先ディレクトリを，ローカルディレクトリ上の絶対パスで設定します。設定値は 1～128 文字の範囲で設定します。なお，ディレクトリ名に指定できる文字は，OS に依存します。また，ディレクトリ名のパスの区切りには，次の文字を使用してください。 Windows の場合： 「/」または「¥¥」を使用してください。 Windows 以外の場合： 「/」を使用してください。	• Windows の場合 <Application Server のインストールフォルダ>/c4web/attachments • Windows 以外の場合 /opt/Cosminexus/c4web/attachments

（凡例）

－：デフォルト値がないことを示します。

注※1

true：設定項目を有効にします。

false：設定項目を無効にします。

また，true および false は小文字で記述してください。

注※2

プロキシサーバのホスト名または IP アドレス（c4web.application.<識別子>.proxy_host）とプロキシサーバのポート番号（c4web.application.<識別子>.proxy_port）のどちらかが null または空文字の場合，設定した値は無視されます。

注※3

false の場合，XML スキーマ（DOCUMENT スタイルの wrapped 形式で引数および戻り値をラップした要素を除く）で要素宣言していない XML インスタンスを含む SOAP メッセージを受信した場合，次のどちらかが発生します。

(a) SOAP サービスで受信した場合，KDCCP0006-E メッセージの SOAP フォルトを返信します。

(b) SOAP クライアントで受信した場合，KDCCP0006-E メッセージの C4Fault が発生します。

true の場合，上記の受信チェックを行いません。該当する XML インスタンスを無視します。

注※4

false の場合，WSDL（RPC スタイルの wsdl:part 要素，および DOCUMENT スタイルの wrapped 形式で引数および戻り値をラップした要素）で宣言していないパラメタを含む SOAP メッセージを受信した場合，次のどちらかが発生することがあります。

(a) SOAP サービスで受信した場合，次のどちらかが発生することがあります。

・不正な値の引数を受け取ります。

・KDCCP0006-E メッセージの SOAP フォルトを返信します。

(b) SOAP クライアントで受信した場合，KDCCP0006-E メッセージの C4Fault が発生することがあります。

true の場合，上記の受信チェックを行いません。該当するパラメタを無視します。

識別子について

識別子は，サーバ定義ファイル中から動作中のサーバの情報を取得するためのキーとして使用します。識別子として，サーバにデプロイした SOAP アプリケーションのコンテキストルートから先頭の「/」を抜いた名称を使用します。コンテキストルートとは，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。例えば，接続先 URL が「http://localhost:8080/WebApp1/services/UserInfo」の場合，識別子は「WebApp1」となります。接続先 URL が「http://localhost:8080/WebApp1/Service1/services/UserInfo」の場合，識別子は「WebApp1/Service1」となります。なお，サーバ定義ファイル中の識別子の大文字と小文字は区別されます。

注意事項

ルートコンテキストを使用する場合，サーバ定義ファイルに，次のように識別子を省略した形式で記述する必要があります。

通常指定時

```
c4web.logger.<識別子>.log_file_num=2
```

ルートコンテキスト指定機能使用時

```
c4web.logger.log_file_num=2
```

なお，トレースファイル，アプリケーションログのプレフィクス（log_file_prefix）を省略した場合，デフォルト値は"ROOT"になります。

サーバ定義ファイルの例

```
#  TO BE REPLACED: ”<cosminexus.home>”,
#                  ”<proxy-host>”,”<nonproxy-host1>”,” <nonproxy-host2>”,
```

```
#                   "<proxy-port>","<proxy-user>","<proxy-password>"

#  define WebApp settings.
#c4web.logger.WebApp1.log_file_prefix=WebApp1
#c4web.common.WebApp1.do_multirefs=false
#c4web.common.WebApp1.send_xsi_types=true
#c4web.common.WebApp1.ignore_undeclared_elements=false
#c4web.common.WebApp1.ignore_undeclared_parts=false
#c4web.common.WebApp1.enable_soapheader_check=true
#c4web.common.WebApp1.prf_trace_level=ALL
#c4web.common.WebApp1.character_reference=false
#c4web.common.WebApp1.send_max_soap_envelope_size=0
#c4web.common.WebApp1.receive_max_soap_envelope_size=0
#c4web.application.WebApp1.app_maintainsession=false
#c4web.application.WebApp1.proxy_host=<proxy-host>
#c4web.application.WebApp1.non_proxy_hosts=<nonproxy-host1>|<nonproxy-host2>
#c4web.application.WebApp1.proxy_port=<proxy-port>
#c4web.application.WebApp1.proxy_user=<proxy-user>
#c4web.application.WebApp1.proxy_password=<proxy-password>
#c4web.application.WebApp1.socket_write_timeout=60
#c4web.application.WebApp1.socket_read_timeout=300
#c4web.application.WebApp1.socket_connect_timeout=60
#c4web.application.WebApp1.dii_temp_directory=<cosminexus.home>/c4web/dii/temp
#c4web.application.WebApp1.dii_wsdl_parse_timeout=45
#c4web.application.WebApp1.hostname_verification.enable=true
#c4web.server.WebApp1.fault_omit_soapaction=true
#c4web.attachment.WebApp1.attachment_temp_directory=<cosminexus.home>/c4web/attachments
#c4web.attachment.WebApp1.send_max_attachment_count=100
#c4web.attachment.WebApp1.receive_max_attachment_count=0
#c4web.attachment.WebApp1.send_max_attachment_size=104857600
#c4web.attachment.WebApp1.receive_max_attachment_size=0

#  define other WebApp settings.
#c4web.logger.WebApp2.log_file_prefix=WebApp2
#c4web.common.WebApp2.do_multirefs=false
#c4web.common.WebApp2.send_xsi_types=true
#c4web.common.WebApp2.ignore_undeclared_elements=false
#c4web.common.WebApp2.ignore_undeclared_parts=false
#c4web.common.WebApp2.enable_soapheader_check=true
#c4web.common.WebApp2.prf_trace_level=ALL
#c4web.common.WebApp2.character_reference=false
#c4web.common.WebApp2.send_max_soap_envelope_size=0
#c4web.common.WebApp2.receive_max_soap_envelope_size=0
#c4web.application.WebApp2.app_maintainsession=false
#c4web.application.WebApp2.proxy_host=<proxy-host>
#c4web.application.WebApp2.non_proxy_hosts=<nonproxy-host1>|<nonproxy-host2>
#c4web.application.WebApp2.proxy_port=<proxy-port>
#c4web.application.WebApp2.proxy_user=<proxy-user>
#c4web.application.WebApp2.proxy_password=<proxy-password>
#c4web.application.WebApp2.socket_write_timeout=60
#c4web.application.WebApp2.socket_read_timeout=300
#c4web.application.WebApp2.socket_connect_timeout=60
#c4web.application.WebApp2.dii_temp_directory=<cosminexus.home>/c4web/dii/temp
#c4web.application.WebApp2.dii_wsdl_parse_timeout=45
#c4web.application.WebApp2.hostname_verification.enable=true
#c4web.server.WebApp2.fault_omit_soapaction=true
#c4web.attachment.WebApp2.attachment_temp_directory=<cosminexus.home>/c4web/attachments
#c4web.attachment.WebApp2.send_max_attachment_count=100
```

```
#c4web.attachment.WebApp2.receive_max_attachment_count=0
#c4web.attachment.WebApp2.send_max_attachment_size=104857600
#c4web.attachment.WebApp2.receive_max_attachment_size=0
```

定義ファイル中の<>で囲まれた部分については，利用者の環境に合わせた値に修正する必要があります。なお，<cosminexus.home>は，Windows の場合，Application Server インストールフォルダを表しています。Windows 以外の場合，/opt/Cosminexus を表しています。

10.3　クライアント定義ファイルの設定

クライアント定義ファイルは，「c4webcl.properties」というファイル名称で利用者が作成します。作成したクライアント定義ファイルは，クラスパス上に配置してください。

配置例を次に示します。

コマンドラインを利用する場合

コマンドラインから SOAP アプリケーションを利用する場合，Java コマンド実行時に指定するクラスパス上に，クライアント定義ファイルを配置してください。例えば，次のようにクライアントを実行する場合は，C:¥temp¥に配置します。

```
cjclstartap -classpath C:¥temp¥ <クライアントのJavaクラス名>
```

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

クラスパスは J2EE サーバ用オプション定義ファイルで指定することもできます。J2EE サーバ用オプション定義ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」を参照してください。

サーブレットを利用する場合

サーブレットから SOAP アプリケーションを利用する場合，WAR ファイルの WEB-INF/classes に，クライアント定義ファイルを配置してください。WAR ファイルの構成例を次に示します。

SOAP アプリケーションを利用するサーブレットクラス RPCSampleClient を含む RPCSampleClient.war の例

```
RPCSampleClient.war/
  getUserData.html
  META-INF/
    MANIFEST.MF
  WEB-INF/
    web.xml
    classes/
      c4webcl.properties
      localhost/
        RPCSampleClient.class
          :
```

WAR ファイルの作成方法については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」を参照してください。

サーブレットから SOAP アプリケーションを利用する場合，J2EE サーバ用オプション定義ファイルの add.class.path キーに指定したパスに配置することもできます。J2EE サーバ用オプション定義ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」を参照してください。ただし，この場合，その J2EE サーバにデプロイされた，SOAP アプリケーションを利用するすべてのアプリケーションが同じ設定で動作するので注意してください。なお，「EJB を利用する場合」の次の注意事項も参照してください。

EJB を利用する場合

EJB から SOAP アプリケーションを利用する場合，EJB-JAR ファイルのルートディレクトリに，クライアント定義ファイルを配置してください。EJB-JAR ファイルの構成例を次に示します。

SOAP アプリケーションを利用する EJB"RPCSampleClient"を含む RPCSampleClient-ejb.jar の例

```
RPCSampleClient-ejb.jar/
  c4webcl.properties
  META-INF/
    ejb-jar.xml
    MANIFEST.MF
  localhost/
    RPCSampleClient.class
    RPCSampleClientBean.class
    RPCSampleClientHome.class
      :
```

EJB-JAR ファイルの作成方法については，マニュアル「アプリケーションサーバ アプリケーション開発ガイド」を参照してください。

EJB から SOAP アプリケーションを利用する場合，J2EE サーバ用オプション定義ファイルの add.class.path キーに指定したパスに配置することもできます。J2EE サーバ用オプション定義ファイルについては，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」を参照してください。ただし，この場合，その J2EE サーバにデプロイされた，SOAP アプリケーションを利用するすべてのアプリケーションが同じ設定で動作するので注意してください。なお，次の注意事項も参照してください。

注意事項

- J2EE サーバ上でクライアントを実行する場合，Management クラスのメソッドを必ず使用してください。
- SOAP サービスを利用するクライアントを同じマシン上で複数動作させる場合，クライアント定義ファイルのトレースファイル，アプリケーションログのプレフィクス（log_file_prefix）は，クライアントごとに異なる値を設定してください。同じ値を設定すると，実行時オプションの指定による制御が正しく行われません。その他の設定（重要度，面数，ファイルサイズ）については，すべてのクライアントが同じ設定で動作します。

次の表に設定するキー名称と値の一覧について示します。

表 10-2　クライアント定義ファイルの設定項目

設定項目	キー名称	値	デフォルト値 (値省略時の仮定値)
トレースファイル，アプリケーションログのプレフィクス	c4web.logger.log_file_prefix	任意の文字列を設定します。 「<log_file_prefix>-j2ee-<J2EE サーバのサーバ名>※1-<log_file_num>.log」がトレースファイルの名称となります。 「<log_file_prefix>-j2ee-<J2EE サーバのサーバ名>※1-aplog-	c4webcl

設定項目	キー名称	値	デフォルト値 （値省略時の仮定値）
		<log_file_num>.log」がアプリケーションログの名称となります。	
多重参照（実行時オプション）	c4web.common.do_multirefs	多重参照を有効にするかどうかを設定します。true または false を設定します。※2	false
データ型定義（実行時オプション）	c4web.common.send_xsi_types	データ型情報を送信するかどうかを設定します。true または false を設定します。※2	true
未宣言要素の無視※4	c4web.common.ignore_undeclared_elements	XML スキーマ（DOCUMENT スタイルの wrapped 形式で引数および戻り値をラップした要素を除く）で要素宣言していない XML インスタンスを含む SOAP メッセージを受信した場合，この XML インスタンスを無視するかどうかを設定します。true または false を設定します。※2	false
未宣言パラメタの無視※5	c4web.common.ignore_undeclared_parts	WSDL（RPC スタイルの wsdl:part 要素，および DOCUMENT スタイルの wrapped 形式で引数および戻り値をラップした要素）で宣言していないパラメタを含む SOAP メッセージを受信した場合，このパラメタを無視するかどうかを設定します。true または false を設定します。※2	false
SOAP ヘッダの名前修飾チェックオプション（実行時オプション）	c4web.common.enable_soapheader_check	SOAP ヘッダの子エレメントの名前修飾をチェックするかどうかを指定します。true または false を設定します。※2	true
性能解析トレース出力オプション	c4web.common.prf_trace_level	ALL，NONHANDLER のどちらかを指定します。 ALL： 次のトレース取得ポイントでトレースファイルが出力されます。 • 送信側ハンドラの呼び出し • ハンドラの呼び出し • 受信側ハンドラの呼び出し • 送信側ハンドラからの戻り • ハンドラからの戻り • 受信側ハンドラからの戻り NONHANDLER： 次のトレース取得ポイントでトレースファイルが出力されません。	ALL

設定項目	キー名称	値	デフォルト値 (値省略時の仮定値)
		• 送信側ハンドラの呼び出し • ハンドラの呼び出し • 受信側ハンドラの呼び出し • 送信側ハンドラからの戻り • ハンドラからの戻り • 受信側ハンドラからの戻り 性能解析トレースについては,「14.7　性能解析トレース」を参照してください。	
文字参照形式オプション	c4web.common.character_reference	送信メッセージ中の文字を文字参照形式にエンコードするかどうかを指定します。true または false を設定します。※2	false
送信できる SOAPEnvelope の最大バイト数	c4web.common.send_max_soap_envelope_size	送信できる SOAPEnvelope の最大バイト数を設定します。設定値は 0～2,147,483,647 の数値を設定します。0 を指定した場合は無制限となります。	0
受信できる SOAPEnvelope の最大バイト数	c4web.common.receive_max_soap_envelope_size	受信できる SOAPEnvelope の最大バイト数を設定します。設定値は 0～2,147,483,647 の数値を設定します。0 を設定した場合は無制限となります。	0
HTTP セッションの維持（実行時オプション）	c4web.application.app_maintainsession	HTTP セッションを維持するかどうかを設定します。true または false を設定します。※2	false
プロキシサーバのホスト名または IP アドレス(実行時オプション)	c4web.application.proxy_host	プロキシサーバのホスト名または IP アドレスを設定します。ホスト名または IP アドレスを示す文字列を設定します。	－
プロキシサーバを使用しないホスト名または IP アドレス群（実行時オプション）	c4web.application.non_proxy_hosts	プロキシサーバを使用しないホスト名または IP アドレス群を設定します。ホスト名または IP アドレスを示す文字列を設定します。複数のホスト名または IP アドレスを指定する場合「|」で区切って設定します。※3	－
プロキシサーバのポート番号（実行時オプション）	c4web.application.proxy_port	ポート番号を示す文字列を設定します。	－
プロキシサーバの認証ユーザ ID（実行時オプション）	c4web.application.proxy_user	プロキシサーバの認証ユーザ ID を設定します。任意の文字列を設定します。※3	－

設定項目	キー名称	値	デフォルト値 （値省略時の仮定値）
プロキシサーバの認証ユーザ ID に対応するパスワード（実行時オプション）	c4web.application.proxy_password	認証ユーザ ID に対応するパスワードを設定します。任意の文字列を設定します。※3	−
クライアントのソケットの書き込みタイムアウト値	c4web.application.socket_write_timeout	クライアントのソケットの書き込みタイムアウト値を秒単位で設定します。0～86,400 の数値を設定します。0 を指定した場合はタイムアウト監視をしません。	60
クライアントのソケットの読み込みタイムアウト値	c4web.application.socket_read_timeout	クライアントのソケットの読み込みタイムアウト値を秒単位で設定します。0～86,400 の数値を設定します。0 を指定した場合はタイムアウト監視をしません。	300
クライアントのソケットの接続タイムアウト値	c4web.application.socket_connect_timeout	クライアントのソケットの接続タイムアウト値を秒単位で設定します。0～86,400 の数値を設定します。0 を指定した場合はタイムアウト監視をしません。	60
Java ソースファイルおよびクラスファイルの出力ディレクトリ	c4web.application.dii_temp_directory	DII でユーザ定義のデータ型クラスを使用する場合に，WSDL 定義から動的に出力される Java ソースファイルおよびクラスファイルの出力先ディレクトリを，ローカルディレクトリ上の絶対パスで設定します。設定値は 1～128 文字の範囲で設定します。なお，ディレクトリ名に指定できる文字は，使用している OS に依存します。また，ディレクトリ名のパスの区切りには，次の文字を使用してください。 Windows の場合： 「/」または「¥¥」を使用してください。 Windows 以外の場合： 「/」を使用してください。	<Application Server のインストールディレクトリ>/c4web/dii/temp
WSDL 解析および Java ソース生成のタイムアウト値	c4web.application.dii_wsdl_parse_timeout	DII を使用する場合に，WSDL 解析および Java ソース生成のタイムアウト値を秒単位で設定します。0～3,600 の範囲で数値を設定します。0 を指定した場合はタイムアウト監視をしません。	45

設定項目	キー名称	値	デフォルト値 (値省略時の仮定値)
ホスト名の検証	c4web.application.hostname_verification.enable	ホスト名の検証を有効にするかどうかを設定します。true または false を設定します。※2	true
送信できる添付データの最大個数	c4web.attachment.send_max_attachment_count	送信できる添付データの最大個数を設定します。設定値は 0～2,147,483,647 の数値を設定します。0 を設定した場合は無制限となります。	100
受信できる添付データの最大個数	c4web.attachment.receive_max_attachment_count	受信できる添付データの最大個数を設定します。設定値は 0～2,147,483,647 の数値を設定します。0 を設定した場合は無制限となります。	0
送信できる各添付データの最大バイト数	c4web.attachment.send_max_attachment_size	送信できる各添付データの最大バイト数を設定します。設定値は 0～2,147,483,647 の数値を設定します。0 を設定した場合は無制限となります。	104,857,600
受信できる各添付データの最大バイト数	c4web.attachment.receive_max_attachment_size	受信できる各添付データの最大バイト数を設定します。設定値は 0～2,147,483,647 の数値を設定します。0 を設定した場合は無制限となります。	0
退避ファイルの格納ディレクトリ名	c4web.attachment.attachment_temp_directory	添付ファイル付き SOAP メッセージを受信したときに生成される退避ファイルの出力先ディレクトリを，ローカルディレクトリ上の絶対パスで設定します。設定値は 1～128 文字の範囲で指定してください。なお，ディレクトリ名に指定できる文字は，OS に依存します。また，ディレクトリ名のパスの区切りには，次の文字を使用してください。 Windows の場合： 「/」または「¥¥」を使用してください。 Windows 以外の場合： 「/」を使用してください。	• Windows の場合 <Application Server のインストールフォルダ>/c4web/attachments • Windows 以外の場合 /opt/Cosminexus/c4web/attachments

(凡例)

－：デフォルト値がないことを示します。

注※1

単体 Java アプリケーションとして動作させる場合，"-j2ee-<J2EE サーバのサーバ名>"は付与されません。

注※2

true：設定項目を有効にします。

false：設定項目を無効にします。

また，true および false は小文字で記述してください。

注※3

プロキシサーバのホスト名または IP アドレス（c4web.application.proxy_host）とプロキシサーバのポート番号（c4web.application.proxy_port）のどちらかが null または空文字の場合，設定した値は無視されます。

注※4

false の場合，XML スキーマ（DOCUMENT スタイルの wrapped 形式で引数および戻り値をラップした要素を除く）で要素宣言していない XML インスタンスを含む SOAP メッセージを受信した場合，KDCCP0006-E メッセージの C4Fault が発生します。

true の場合，上記の受信チェックを行いません。該当する XML インスタンスを無視します。

注※5

false の場合，WSDL（RPC スタイルの wsdl:part 要素，および DOCUMENT スタイルの wrapped 形式で引数および戻り値をラップした要素）で宣言していないパラメタを含む SOAP メッセージを受信した場合，KDCCP0006-E メッセージの C4Fault が発生することがあります。

true の場合，上記の受信チェックを行いません。該当するパラメタを無視します。

クライアント定義ファイルの例

```
#  TO BE REPLACED: "<cosminexus.home>",
#                  "<proxy-host>","<nonproxy-host1>"," <nonproxy-host2>",
#                  "<proxy-port>","<proxy-user>","<proxy-password>"

#c4web.logger.log_file_prefix=c4webcl
#c4web.common.do_multirefs=false
#c4web.common.send_xsi_types=true
#c4web.common.ignore_undeclared_elements=false
#c4web.common.ignore_undeclared_parts=false
#c4web.common.enable_soapheader_check=true
#c4web.common.prf_trace_level=ALL
#c4web.common.character_reference=false
#c4web.common.send_max_soap_envelope_size=0
#c4web.common.receive_max_soap_envelope_size=0
#c4web.application.app_maintainsession=false
#c4web.application.proxy_host=<proxy-host>
#c4web.application.non_proxy_hosts=<nonproxy-host1>|<nonproxy-host2>
#c4web.application.proxy_port=<proxy-port>
#c4web.application.proxy_user=<proxy-user>
#c4web.application.proxy_password=<proxy-password>
#c4web.application.socket_write_timeout=60
#c4web.application.socket_read_timeout=300
#c4web.application.socket_connect_timeout=60
#c4web.application.dii_temp_directory=<cosminexus.home>/c4web/dii/temp
#c4web.application.dii_wsdl_parse_timeout=45
#c4web.application.hostname_verification.enable=true
#c4web.attachment.attachment_temp_directory=<cosminexus.home>/c4web/attachments
#c4web.attachment.send_max_attachment_count=100
#c4web.attachment.receive_max_attachment_count=0
#c4web.attachment.send_max_attachment_size=104857600
#c4web.attachment.receive_max_attachment_size=0
```

定義ファイル中の<>で囲まれた部分については，利用者の環境に合わせた値に修正する必要があります。なお，<cosminexus.home>は，Windows の場合，Application Server のインストールフォルダを表しています。Windows 以外の場合，/opt/Cosminexus を表しています。

10.4　共通定義ファイルの設定

サーバおよびクライアント共通の動作を制御するファイルです。ファイル名は「c4webcom.cfg」固定とし，SOAP アプリケーション開発支援機能および SOAP 通信基盤のインストール時に次の場所に格納されます。

Windows の場合

```
<Application Serverのインストールフォルダ>¥c4web¥conf
```

Windows 以外の場合

```
/opt/Cosminexus/c4web/conf
```

次の表に設定するキー名称と値の一覧について示します。

表 10-3　共通定義ファイルの設定項目

設定項目	キー名称	値	デフォルト値 (値省略時の仮定値)
トレースファイル出力の重要度	c4web.logger.log_level	ERROR，WARN，INFO，DEBUG のどれかを設定します。トレースファイル出力の重要度については，「14.4.3　トレースファイル出力の重要度」を参照してください。	WARN
アプリケーションログ出力の重要度	c4web.logger.aplog_level	WARN，INFO，DEBUG のどれかを設定します。アプリケーションログ出力の重要度については，「14.5.2　アプリケーションログ出力の重要度」を参照してください。	WARN
異常発生時のアプリケーションログ出力	c4web.logger.aplog_error_record	ALL，SEND，RECV，NONE のどれかを設定します。異常発生時のアプリケーションログについては，「14.6　異常発生時のアプリケーションログの出力」を参照してください。	NONE
トレースファイルの面数	c4web.logger.log_file_num	1～16 の数値を設定します。	2
トレースファイルのサイズ	c4web.logger.log_file_size	4,096～16,777,216（単位：バイト）の数値を設定します。	2,097,152
アプリケーションログの面数	c4web.logger.aplog_file_num	1～16 の数値を設定します。	2
アプリケーションログのサイズ	c4web.logger.aplog_file_size	4,096～2,147,483,647（単位：バイト）の数値を設定します。	2,097,152
コネクションプーリングの設定	c4web.common.connection_pool.enable	コネクションプーリングの有効／無効を設定します。 true または false を設定します。	false

設定項目	キー名称	値	デフォルト値 （値省略時の仮定値）
		true： コネクションプーリングを有効にします。 false： コネクションプーリングを無効にします。 コネクションプーリングを有効にする場合，「7.4　コネクションプーリング」に記載されている注意事項を十分に理解した上で，設定してください。	
コネクションの利用回数	c4web.common.connection_pool.max_use	0～2,147,483,647（単位：回）の数値を設定します。	10,000
コネクションのタイムアウト	c4web.common.connection_pool.timeout	0～86,400（単位：秒）の数値を設定します。	1,800
最大コネクション数	c4web.common.connection_pool.max_connection	1～2,147,483,647（単位：本）の数値を設定します。	150

10.5　動作定義ファイル設定時の注意事項

動作定義ファイルは，次に示す内容に注意して設定してください。

- サーバ定義ファイルで，サーバにデプロイした SOAP アプリケーションのコンテキストルートから先頭の「/」を抜いた名称と，識別子が一致しない場合は，すべてデフォルト値で動作します。
- トレースファイル出力の重要度の値に ERROR，INFO，WARN，DEBUG 以外の文字列を設定した場合はデフォルト値（WARN）が仮定されます。
- トレースファイル出力の重要度の値はすべて大文字で設定してください。小文字が混在した場合は，その文字列は無効となり，デフォルト値（WARN）が仮定されます。
- トレースファイルの面数，トレースファイルのサイズに範囲外の値を設定した場合は，それぞれデフォルト値が仮定されます。
- トレース出力先ディレクトリ名およびトレースファイルのプレフィクスの最大長は，トレースファイル名を生成したときの，実行するプラットフォーム上のファイル名の最大長に依存します。実行するプラットフォーム上のファイル名の最大長を超えた場合はトレースが出力されません。
- トレースファイル名およびアプリケーションログファイル名の形式は次のようになります。

  トレースファイル名の形式

  <トレースファイルのプレフィクス>-j2ee-<J2EE サーバのサーバ名>-<トレースファイル面数>.log

  アプリケーションログファイル名の形式

  <トレースファイルのプレフィクス>-j2ee-<J2EE サーバのサーバ名>-aplog-<トレースファイル面数>.log

  単体 Java アプリケーションとして動作させる場合，"-j2ee-<J2EE サーバのサーバ名>"は付与されません。
- トレースファイルとアプリケーションログでは，ファイル名に付けられる面数の順序が異なるので注意してください。

  トレースファイルの面数

  トレースファイルの面数は，トレースの出力時刻に従って「1，2，3・・・」という順で生成されます。動作定義ファイル中のトレースファイル面数を超えた場合は，再度 1 から生成されます。

  アプリケーションログの面数

  アプリケーションログは，面数が 0 のものが，いちばん新しいファイルになります（面数が最も大きいものが，いちばん古いファイルとなります）。
- サーバの動作開始時にサーバ定義ファイル（c4websv.cfg）とクライアント定義ファイル（c4webcl.properties）の両方が存在する場合は，サーバ定義ファイル（c4websv.cfg）の設定が有効となります。サーバ定義ファイル（c4websv.cfg）が存在しない場合は，デフォルト値が仮定されます。
- クライアントの動作開始時にサーバ定義ファイル（c4websv.cfg）とクライアント定義ファイル（c4webcl.properties）の両方が存在する場合は，クライアント定義ファイル（c4webcl.properties）の設定が有効となります。クライアント定義ファイル（c4webcl.properties）が存在しない場合は，デフォルト値が仮定されます。

- true および false を設定する場合はすべて小文字で設定してください。大文字または異なる文字列で設定した場合はデフォルト値が仮定されます。
- クライアントを同じマシン上で複数動作させる場合で，同じ内容のクライアント定義ファイル（c4webcl.properties）がそれぞれのクライアント実行時クラスパス上に存在するときは，トレースが同じファイルに書き込まれてしまうためトレースが正常に出力されません。トレースファイル出力の排他制御待ちや，トレースファイル中のデータが解析できなくなる現象が発生するおそれがあります。
- クライアントを同じマシン上で複数動作させる場合でトレースファイルのプレフィクスを省略したとき，すべてのクライアントのトレースが同じファイルに出力され，トレースの内容が不正確になります。トレースファイル出力の排他制御待ちや，トレースファイル中のデータが解析できなくなる現象が発生するおそれがあります。
- 動作定義ファイル中では「¥」は特殊文字として扱われるため，Windows の場合にディレクトリ名のパスの区切り文字に「¥」を使用するときは，「¥¥」のように連続して設定してください。
- 複数の SOAP サービスを利用する場合，Web アプリケーションのコンテキストパス名をそれぞれ異なる名称にしてください。異なる名称にしない場合，トレース出力先が重複します。そのため，複数の SOAP サービスのトレースが同じファイルに出力されます。

  コンテキストパスとは

  コンテキストパスとは，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 上に指定するパスのことです。例えば，接続先 URL が「http://localhost:8080/WebApp1/services/UserInfo」の場合，コンテキストパスは，「/WebApp1/services/UserInfo」となります。

  同一のコンテキストパスを使用する場合

  同一のコンテキストパスを使用する場合は，トレースファイル面数，トレースファイルサイズを必要に応じて大きい値にして，障害発生時に必要な情報が削除されてしまわないよう注意してください。
- サーバ定義ファイルのトレースファイル，アプリケーションログのプレフィクスを省略した場合，識別子をデフォルト値として使用しますが，SOAP アプリケーションのコンテキストルートの文字列に「$」，「%」，「+」，「/」が含まれる場合，トレースファイル，アプリケーションログのファイル名プレフィクスは次のように置換されます。

コンテキストルートに含まれる文字	ファイル名中の文字
$	$24
%	$25
+	$2b
/	$2f

例えば，接続先 URL が「http://localhost:8080/WebApp1/Service1/services/UserInfo」の場合，トレースファイル，アプリケーションログのファイル名プレフィクスは「WebApp1$2fService1-j2ee-<J2EE サーバのサーバ名>-<トレースファイル面数>.log」および「WebApp1$2fService1-j2ee-<J2EE サーバのサーバ名>-aplog-<トレースファイル面数>.log」となります。

なお，上記の文字に置換されたファイルを Windows 以外の OS 上で操作する場合，ファイル名を指定するときに，シングルクォーテーション「'」で囲む必要があります。

ファイルを削除する場合の例

```
rm 'WebApp1$2fService1-server1-1.log'
```

- 複数のクライアントでクライアント定義ファイルを設定する場合，「log_file_prefix」には，同じ値を設定しないでください。
- サーバから別の SOAP アプリケーションを呼び出すような場合（サーバが別のサーバに対するクライアントとなる場合)，サーバトレース中にクライアントのトレースが出力されます。この場合，トレースの内容がラップアラウンドによって上書きされる頻度が高いため，サーバ定義ファイルのトレースファイル面数，およびトレースファイルサイズに十分大きい値を指定してください。
- SOAP サービスを利用するクライアントを同じマシン上で複数動作させる場合，クライアント定義ファイルのトレースファイル，アプリケーションログのプレフィクス（log_file_prefix）は，クライアントごとに異なる値を設定してください。同じ値を設定すると，実行時オプションの指定による制御が正しく行われません。その他の設定（重要度，面数，ファイルサイズ）については，すべてのクライアントが同じ設定で動作します。
- パスワードを設定する場合，動作定義ファイルの読み取り権限を限定するなど，第三者にパスワードを読み取られないように注意してください。

10.6　実行時オプションの設定項目

開発した SOAP アプリケーションを実行するには，SOAP アプリケーションの動作に関する実行時オプションを設定する必要があります。実行時オプションの設定項目を次に示します。

- 多重参照オプション
- データ型定義オプション
- HTTP セッションに関するオプション
- SOAP ヘッダの名前修飾に関するオプション
- SOAPAction 値の扱いに関するオプション
- プロキシオプション
- ソケットタイムアウト値オプション
- 送受信データのサイズチェックオプション
- SOAP メッセージのコードに関するオプション

実行時オプション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については，10.2～10.4 の説明を参照してください。また，SOAP クライアントライブラリの実行時に，実行時オプションを動的に設定するための C4Property クラスを提供します。C4Property クラスについては，「13.3　C4Property クラス（実行時オプションの設定）」を参照してください。

10.6.1　多重参照オプション

サーバおよびクライアントが SOAP メッセージを送信する場合に，扱う複合型データ（構造体）を埋め込み形式または参照形式（ID-REF 型）のどちらの形式にするかを設定できます。

埋め込み形式と参照形式は，動作定義ファイルに「do_multirefs」キーを追加して設定します。デフォルトの動作は埋め込み形式です。次に，埋め込み形式および参照形式（ID-REF 型）の SOAP メッセージの例を示します。

- **埋め込み形式の SOAP メッセージの例**

```
<soapenv:Envelope soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns
:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XM
LSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:soapenc="http://sche
mas.xmlsoap.org/soap/encoding/">
 <soapenv:Body>
  <ns1:test1 xmlns:ns1="http://XMLschemaType2.test">
   <in0>
    <str1>string</str1>
    <b1>true</b1>
    <f1>10.0</f1>
    <d1>40.0</d1>
    <BD1>123</BD1>
    <TI1>88</TI1>
```

```
    <by2>QEFC</by2>
    <QN1>
     <namespaceURI></namespaceURI>
     <localPart>QName</localPart>
    </QN1>
    <BI1>456</BI1>
    <l1>30</l1>
    <i1>43</i1>
    <s1>20</s1>
    <by1>48</by1>
    <da1>99</da1>
   </in0>
  </ns1:test1>
 </soapenv:Body>
</soapenv:Envelope>
```

- 参照形式（ID-REF 型）の SOAP メッセージの例

```
<soapenv:Envelope soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/20
01/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:soapenc="http:/
/schemas.xmlsoap.org/soap/encoding/">
 <soapenv:Body>
  <ns1:test1 xmlns:ns1="http://XMLschemaType2.test">
   <in0 href="#id0"/>
  </ns1:test1>
  <multiRef id="id0"
soapenc:root="0" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/:encodingStyle">
   <str1>string</str1>
   <b1>true</b1>
   <f1>10.0</f1>
   <d1>40.0</d1>
   <BD1>123</BD1>
   <TI1>88</TI1>
   <by2>QEFC</by2>
   <QN1 href="#id1"/>
   <BI1>456</BI1>
   <l1>30</l1>
   <i1>43</i1>
   <s1>20</s1>
   <by1>48</by1>
   <da1>99</da1>
  </multiRef>
  <multiRef id="id1"
soapenc:root="0" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <namespaceURI></namespaceURI>
   <localPart>QName</localPart>
  </multiRef>
 </soapenv:Body>
</soapenv:Envelope>
```

注意事項

リテラルエンコーディング使用時は多重参照オプションの指定は有効になりません。多重参照オプションを使用する場合は，SOAP エンコーディングを使用してください。

10.6.2　データ型定義オプション

送信する SOAP メッセージ中にデータ型を含めるかどうかを設定します。データ型は XML Schema で規定されているデータ型で，接頭辞に「xsi」が付くものを指します。データ型を含めるかどうかは，動作定義ファイルに「send_xsi_types」キーを追加して設定します。

次に，データ型を含んだ場合およびデータ型を含まない場合の SOAP メッセージの例を示します。

- データ型を含んだ場合の SOAP メッセージの例

```
<str1 xsi:type="xsd:string">string</str1>
```

- データ型を含まない場合の SOAP メッセージの例

```
<str1>string</str1>
```

注意事項

リテラルエンコーディング使用時は，「send_xsi_types」キーに true を設定してもデータ型を含まない場合の SOAP メッセージになります。

10.6.3　HTTP セッションに関するオプション

クライアントからサーバ側の SOAP サービスを呼び出したときに，HTTP セッションを継続するかどうかを設定できます。HTTP セッションを継続するかどうかは，動作定義ファイルに「app_maintainsession」キーを追加して設定します。

デフォルトの動作では HTTP セッションを継続しません。WSDL からソースコードを生成する場合，またはサービスデプロイ定義を生成する場合で，Deploy Scope に「Session」を選択した SOAP サービスと接続するときには，true を設定してください。ただし，このセッションは，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」で記述している「J2EE サーバ間のセッション情報の引き継ぎ」には対応していません。

10.6.4　SOAP ヘッダの名前修飾に関するオプション

送受信する SOAP メッセージに SOAP ヘッダが含まれていた場合，SOAP1.1 仕様に基づき，「SOAP ヘッダの子エレメントが名前修飾されているか」のチェックをするかどうかを設定できます。デフォルトの動作では，名前修飾されているかをチェックします。このオプションは，動作定義ファイルに「enable_soapheader_check」キーを追加して設定します。SOAP ヘッダをチェックしない場合は，false を設定します。

10.6.5　SOAPAction 値の扱いに関するオプション

SOAP エンジンでは，HTTP ヘッダ中に SOAPAction 値自体がない場合に SOAP Fault メッセージを返しています。このオプションの設定によって，SOAPAction 値がない場合の動作を，SOAP Fault メッセージを返すか，空文字（""）として振る舞うかを設定できます。SOAPAction 値の扱いは，動作定義ファイルに「fault_omit_soapaction」キーを追加して設定します。デフォルトの動作では，SOAP Fault メッセージを返します。なお，このオプションはサーバ定義ファイルで有効です。次に，SOAPAction 値がある場合，および SOAPAction 値がない場合の HTTP ヘッダの例を示します。

- SOAPAction 値がある場合の HTTP ヘッダの例

```
POST /WebX/services/XMLschemaType2 HTTP/1.0
Content-Length: 774
Host: localhost
Content-Type: text/xml; charset=utf-8
SOAPAction: ””
```

- SOAPAction 値がない場合の HTTP ヘッダの例

```
POST /WebX/services/XMLschemaType2 HTTP/1.0
Content-Length: 774
Host: localhost
Content-Type: text/xml; charset=utf-8
```

10.6.6　プロキシオプション

クライアントが外部の SOAP サービスを利用するための，プロキシサーバ認証に関するオプションです。SOAP アプリケーション開発支援機能ではプロキシオプションの形式はチェックしません。このオプションはクライアント定義ファイルで有効です。次に，プロキシオプションとして設定する項目について示します。なお，次のオプションの説明にあるホスト名は，IP アドレスに置き換えることもできます。

proxy_host

プロキシサーバのホスト名を設定します。このオプションの内容が null または空文字の場合は，プロキシサーバに接続しません。

non_proxy_hosts

プロキシサーバを利用しないホスト名群を設定します。複数のホスト名を設定する場合は，「|」で区切って設定します。プロキシサーバを利用しないホスト名を設定する場合，ホスト名とホスト名の間には，「|」以外の文字（空白など）を設定しないようにしてください。

proxy_port

プロキシサーバのポート番号を設定します。このオプションの内容が null または空文字の場合は，プロキシサーバに接続しません。

proxy_user

プロキシサーバの認証ユーザ ID を設定します。

proxy_password

認証ユーザ ID に対応するパスワードを設定します。

注意事項

パスワードを設定する場合，動作定義ファイルの読み取り権限を限定するなど，第三者にパスワードを読み取られないように注意してください。

10.6.7　ソケットタイムアウト値オプション

クライアントからサーバ側の SOAP サービスを呼び出す場合に利用するソケットの接続・書き込み・読み込みタイムアウト値を設定できます。タイムアウト値は，動作定義ファイルに「socket_write_timeout」，「socket_read_timeout」，「socket_connect_timeout」キーを追加して設定します。このキーには 0～86,400 の数値を指定できます。デフォルトの動作では「socket_write_timeout」と「socket_connect_timeout」が 60 秒（1 分），「socket_read_timeout」が 300 秒（5 分）を仮定します。

0 を指定すると，タイムアウト値は無制限となります。J2EE サーバの同時実行スレッド数を超えて無制限の待ちが発生する場合などには，デッドロックになることがありますので，0 を指定する場合は注意してください。また，このオプションに極端に小さい値を設定すると，タイムアウトが頻繁に発生することがあります。この場合は，タイムアウトが発生しない十分な値を設定してください。

注意事項

ソケットの接続タイムアウト値および書き込みタイムアウト値に次のどちらかの値を設定した場合，TCP の持つタイムアウト時間が適用されます。

- 0
- TCP が持つコネクション確立およびデータ送信の再送タイマより長い時間

10.6.8　送受信データのサイズチェックオプション

クライアントとサーバとの間で送受信される SOAPEnvelope のサイズ，添付データのサイズ，および添付データの個数をチェックするためのオプションです。

(1)　チェックする項目

チェックする項目を次に示します。これらの項目は送信時，および受信時の場合も同様にチェックしてください。

- 送受信時の SOAPEnvelope のバイト数（自動的に付加される XML 宣言を含む）
- 送受信時の添付データの個数
- 送受信時の添付データのバイト数

(2) チェック動作について

チェックする動作について次に示します。

- クライアント側
  実行時のサイズチェックが不正になった場合，送信時，および受信時には下記の例外を通知します。

  表 10-4 実行時のサイズチェックが不正になった場合に通知する例外

モード	通知メソッド	通知する例外
SAAJ	javax.xml.soap.SOAPConnection#call	javax.xml.soap.SOAPException
RPC	サービス呼び出しメソッド	java.rmi.RemoteException （実際は上記を継承する C4Fault）

- サーバ側
  実行時のサイズチェックが不正になった場合，送信時，および受信時には SOAP エンジンによって Fault メッセージを作成し，クライアント側へ送信します。サイズチェックは，SOAP エンジンの内部処理として実施されるため，サーバ側 UP には例外などで通知されないことに注意してください。

(3) Fault メッセージ

サイズチェックに関係する Faultcode を次に示します。SAAJ では Fault メッセージに含まれ，RPC では C4Fault から取得できます。

```
{http://c4web.cosminexus.com}[Client | Server].SOAPMessageNotPredicted
```

10.6.9 SOAP メッセージのコードに関するオプション

送信する SOAP メッセージに 0x7F より大きいコードの文字が含まれていた場合，「常に文字参照形式にエンコードする処理」のチェックをするかどうかを設定できます。このオプションは，デフォルトとして文字参照形式でエンコードしないようにし，送信する SOAP メッセージのサイズを縮小します。

11 SOAP アプリケーションで扱うデータ型

この章では，SOAP アプリケーションで扱うデータ型の対応，および注意事項について説明します。

11.1　WSDL 定義とソースコードのデータ型の関係

WSDL からソースコードを生成する場合，WSDL 定義のデータ型および名前空間の URL によって，生成されるソースコードでのデータ型が異なります。また，Java クラスから WSDL を生成する場合，Java クラスのデータ型によって，WSDL 定義での名前空間の URL が異なります。

次に示す WSDL 定義を例に，生成されるソースコードのデータ型の違いについて説明します。

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions targetNamespace="http://DefaultNamespace"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:impl="http://DefaultNamespace-impl"
  xmlns:intf="http://DefaultNamespace"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <wsdl:message name="soaplongRequest">
    <wsdl:part name="in0" type="soapenc:long" />   (1)
  </wsdl:message>

  <wsdl:message name="xsdlongRequest">
    <wsdl:part name="in0" type="xsd:long" />        (2)
  </wsdl:message>

                    :

</wsdl:definitions>
```

WSDL 定義の（1）の部分と（2）の部分は同じ long 型を使用していますが，使用している名前空間が異なるため，生成されるソースコードのデータ型は異なります。生成されたソースコードでは，（1）の部分は java.lang.Long 型に，（2）の部分は long 型になります。

11.2 WSDL からソースコードを生成する場合のデータ型の関係

WSDL2Java コマンドで WSDL からソースコードを生成した場合の，WSDL 定義のデータ型と名前空間の URL，およびソースコードのデータ型の関係を示します。データ型の対応関係は，次に示す場合によって異なります。

- メソッド引数の入出力種別が「IN」，およびメソッド戻り値の場合
- メソッド引数の入出力種別が「OUT」および「INOUT」の場合

それぞれの場合に分けて，データ型の対応関係を示します。

11.2.1 メソッド引数の入出力種別が「IN」，およびメソッド戻り値の場合

表 11–1 WSDL からソースコードを生成した場合のデータ型の対応（入出力種別が「IN」，およびメソッド戻り値の場合）

WSDL でのデータ型		Java でのデータ型
データ型	名前空間 URL	
anyType※	http://www.w3.org/2001/XMLSchema	java.lang.Object
Array	http://schemas.xmlsoap.org/soap/encoding/	java.lang.Object[]
base64	http://schemas.xmlsoap.org/soap/encoding/	byte[]
base64Binary	http://www.w3.org/2001/XMLSchema	byte[]
boolean	http://www.w3.org/2001/XMLSchema	boolean
	http://schemas.xmlsoap.org/soap/encoding/	java.lang.Boolean
byte	http://www.w3.org/2001/XMLSchema	byte
	http://schemas.xmlsoap.org/soap/encoding/	java.lang.Byte
date	http://www.w3.org/2001/XMLSchema	java.util.Date
dateTime	http://www.w3.org/2001/XMLSchema	java.util.Calendar
decimal	http://www.w3.org/2001/XMLSchema	java.math.BigDecimal
	http://schemas.xmlsoap.org/soap/encoding/	java.math.BigDecimal
double	http://www.w3.org/2001/XMLSchema	double
	http://schemas.xmlsoap.org/soap/encoding/	java.lang.Double
float	http://www.w3.org/2001/XMLSchema	float
	http://schemas.xmlsoap.org/soap/encoding/	java.lang.Float
hexBinary	http://www.w3.org/2001/XMLSchema	byte[]

WSDL でのデータ型		Java でのデータ型
データ型	名前空間 URL	
int	http://www.w3.org/2001/XMLSchema	int
	http://schemas.xmlsoap.org/soap/encoding/	java.lang.Integer
integer	http://www.w3.org/2001/XMLSchema	java.math.BigInteger
	http://schemas.xmlsoap.org/soap/encoding/	java.math.BigInteger
long	http://www.w3.org/2001/XMLSchema	long
	http://schemas.xmlsoap.org/soap/encoding/	java.lang.Long
QName	http://www.w3.org/2001/XMLSchema	javax.xml.namespace.QName
short	http://www.w3.org/2001/XMLSchema	short
	http://schemas.xmlsoap.org/soap/encoding/	java.lang.Short
string	http://www.w3.org/2001/XMLSchema	java.lang.String
	http://schemas.xmlsoap.org/soap/encoding/	java.lang.String
duration	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.Duration
time	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.Time
gYearMonth	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.YearMonth
gYear	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.Year
gMonthDay	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.MonthDay
gDay	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.Day
gMonth	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.Month
anyURI	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.URI
normalizedString	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.NormalizedString
token	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.Token
Name	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.Name
NCName	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.NCName
NMTOKEN	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.NMToken
nonPositiveInteger	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.NonPositiveInteger
negativeInteger	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.NegativeInteger
nonNegativeInteger	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.NonNegativeInteger
unsignedInt	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.UnsignedInt

WSDL でのデータ型		Java でのデータ型
データ型	名前空間 URL	
unsignedShort	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.UnsignedShort
unsignedLong	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.UnsignedLong
unsignedByte	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.UnsignedByte
positiveInteger	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.PositiveInteger
language	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.Language
ID	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.Id
IDREF	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.IDRef
ENTITY	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.Entity
IDREFS	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.IDRefs
ENTITIES	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.Entities
NMTOKENS	http://www.w3.org/2001/XMLSchema	org.apache.axis.types.NMTokens
wsi:swaRef	http://ws-i.org/profiles/basic/1.1/xsd	javax.activation.DataHandler

注※

データ型の変数には指定できません。

11.2.2 メソッド引数の入出力種別が「OUT」および「INOUT」の場合

表 11-2 WSDL からソースコードを生成した場合のデータ型の対応（入出力種別が「OUT」，および「INOUT」の場合）

WSDL でのデータ型		Java でのデータ型
データ型	名前空間 URL	
anyType※	http://www.w3.org/2001/XMLSchema	javax.xml.rpc.holders.ObjectHolder
base64	http://schemas.xmlsoap.org/soap/encoding/	javax.xml.rpc.holders.ByteArrayHolder
base64Binary	http://www.w3.org/2001/XMLSchema	javax.xml.rpc.holders.ByteArrayHolder
boolean	http://www.w3.org/2001/XMLSchema	javax.xml.rpc.holders.BooleanHolder
	http://schemas.xmlsoap.org/soap/encoding/	javax.xml.rpc.holders.BooleanWrapperHolder
byte	http://www.w3.org/2001/XMLSchema	javax.xml.rpc.holders.ByteHolder
	http://schemas.xmlsoap.org/soap/encoding/	javax.xml.rpc.holders.ByteWrapperHolder
date	http://www.w3.org/2001/XMLSchema	org.apache.axis.holders.DateHolder
dateTime	http://www.w3.org/2001/XMLSchema	javax.xml.rpc.holders.CalendarHolder

WSDL でのデータ型		Java でのデータ型
データ型	名前空間 URL	
decimal	http://www.w3.org/2001/XMLSchema	javax.xml.rpc.holders.BigDecimalHolder
	http://schemas.xmlsoap.org/soap/encoding/	javax.xml.rpc.holders.BigDecimalHolder
double	http://www.w3.org/2001/XMLSchema	javax.xml.rpc.holders.DoubleHolder
	http://schemas.xmlsoap.org/soap/encoding/	javax.xml.rpc.holders.DoubleWrapperHolder
float	http://www.w3.org/2001/XMLSchema	javax.xml.rpc.holders.FloatHolder
	http://schemas.xmlsoap.org/soap/encoding/	javax.xml.rpc.holders.FloatWrapperHolder
hexBinary	http://www.w3.org/2001/XMLSchema	javax.xml.rpc.holders.ByteArrayHolder
int	http://www.w3.org/2001/XMLSchema	javax.xml.rpc.holders.IntHolder
	http://schemas.xmlsoap.org/soap/encoding/	javax.xml.rpc.holders.IntegerWrapperHolder
integer	http://www.w3.org/2001/XMLSchema	javax.xml.rpc.holders.BigIntegerHolder
	http://schemas.xmlsoap.org/soap/encoding/	javax.xml.rpc.holders.BigIntegerHolder
long	http://www.w3.org/2001/XMLSchema	javax.xml.rpc.holders.LongHolder
	http://schemas.xmlsoap.org/soap/encoding/	javax.xml.rpc.holders.LongWrapperHolder
QName	http://www.w3.org/2001/XMLSchema	javax.xml.rpc.holders.QNameHolder
short	http://www.w3.org/2001/XMLSchema	javax.xml.rpc.holders.ShortHolder
	http://schemas.xmlsoap.org/soap/encoding/	javax.xml.rpc.holders.ShortWrapperHolder
string	http://www.w3.org/2001/XMLSchema	javax.xml.rpc.holders.StringHolder
	http://schemas.xmlsoap.org/soap/encoding/	javax.xml.rpc.holders.StringHolder
duration	http://www.w3.org/2001/XMLSchema	org.apache.axis.holders.DurationHolder
time	http://www.w3.org/2001/XMLSchema	org.apache.axis.holders.TimeHolder
gYearMonth	http://www.w3.org/2001/XMLSchema	org.apache.axis.holders.YearMonthHolder
gYear	http://www.w3.org/2001/XMLSchema	org.apache.axis.holders.YearHolder
gMonthDay	http://www.w3.org/2001/XMLSchema	org.apache.axis.holders.MonthDayHolder
gDay	http://www.w3.org/2001/XMLSchema	org.apache.axis.holders.DayHolder
gMonth	http://www.w3.org/2001/XMLSchema	org.apache.axis.holders.MonthHolder
anyURI	http://www.w3.org/2001/XMLSchema	org.apache.axis.holders.URIHolder
normalizedString	http://www.w3.org/2001/XMLSchema	org.apache.axis.holders.NormalizedStringHolder
token	http://www.w3.org/2001/XMLSchema	org.apache.axis.holders.TokenHolder

WSDL でのデータ型		Java でのデータ型
データ型	名前空間 URL	
nonPositiveInteger	http://www.w3.org/2001/XMLSchema	org.apache.axis.holders.NonPositiveIntegerHolder
negativeInteger	http://www.w3.org/2001/XMLSchema	org.apache.axis.holders.NegativeIntegerHolder
nonNegativeInteger	http://www.w3.org/2001/XMLSchema	org.apache.axis.holders.NonNegativeIntegerHolder
unsignedInt	http://www.w3.org/2001/XMLSchema	org.apache.axis.holders.UnsignedIntHolder
unsignedShort	http://www.w3.org/2001/XMLSchema	org.apache.axis.holders.UnsignedShortHolder
unsignedLong	http://www.w3.org/2001/XMLSchema	org.apache.axis.holders.UnsignedLongHolder
unsignedByte	http://www.w3.org/2001/XMLSchema	org.apache.axis.holders.UnsignedByteHolder
positiveInteger	http://www.w3.org/2001/XMLSchema	org.apache.axis.holders.PositiveIntegerHolder
wsi:swaRef	http://ws-i.org/profiles/basic/1.1/xsd	org.apache.axis.holders.DataHandlerHolder

注※

データ型の変数には指定できません。

11.3　Java クラスから WSDL を生成する場合のデータ型の関係

Java2WSDL コマンドで Java クラスから WSDL を生成した場合の，WSDL 定義のデータ型と名前空間の URL，およびソースコードのデータ型の関係を示します。

表 11-3　Java クラスから WSDL を生成した場合のデータ型の対応

Java でのデータ型	WSDL でのデータ型	
	データ型	名前空間の URL
boolean	boolean	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.BooleanHolder		
byte	byte	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.ByteHolder		
byte[]	base64※3	http://schemas.xmlsoap.org/soap/encoding/※3
javax.xml.rpc.holders.ByteArrayHolder		
double	double	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.DoubleHolder		
float	float	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.FloatHolder		
int	int	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.IntHolder		
long	long	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.LongHolder		
short	short	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.ShortHolder		
java.lang.Byte	byte	http://schemas.xmlsoap.org/soap/encoding/※3
javax.xml.rpc.holders.ByteWrapperHolder		
java.lang.Byte[]	SequenceOf_soapenc_byte※3	definitions 要素の targetNamespace※1
javax.xml.rpc.holders.ByteWrapperArrayHolder		
java.lang.Double	double	http://schemas.xmlsoap.org/soap/encoding/※3
javax.xml.rpc.holders.DoubleWrapperHolder		
java.lang.Float	float	http://schemas.xmlsoap.org/soap/encoding/※3
javax.xml.rpc.holders.FloatWrapperHolder		

Java でのデータ型	WSDL でのデータ型	
	データ型	名前空間の URL
java.lang.Integer	int	http://schemas.xmlsoap.org/soap/encoding/※3
javax.xml.rpc.holders.IntegerWrapperHolder		
java.lang.Long	long	http://schemas.xmlsoap.org/soap/encoding/※3
javax.xml.rpc.holders.LongWrapperHolder		
java.lang.Object	anyType	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.ObjectHolder		
java.lang.Object[]	SequenceOf_xsd_anyType	definitions 要素の targetNamespace※2
java.lang.Short	short	http://schemas.xmlsoap.org/soap/encoding/※3
javax.xml.rpc.holders.ShortWrapperHolder		
java.lang.String	string	http://schemas.xmlsoap.org/soap/encoding/※3
javax.xml.rpc.holders.StringHolder		
java.math.BigDecimal	decimal	http://schemas.xmlsoap.org/soap/encoding/※3
javax.xml.rpc.holders.BigDecimalHolder		
java.math.BigInteger	integer	http://schemas.xmlsoap.org/soap/encoding/※3
javax.xml.rpc.holders.BigIntegerHolder		
java.util.Date	date	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.DateHolder		
javax.xml.namespace.QName	QName	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.QNameHolder		
java.lang.Boolean	boolean	http://schemas.xmlsoap.org/soap/encoding/※3
javax.xml.rpc.holders.BooleanWrapperHolder		
java.util.Calendar	dateTime	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.CalendarHolder		
org.apache.axis.types.Duration	duration	http://www.w3.org/2001/XMLSchema
org.apache.axis.holders.DurationHolder		
org.apache.axis.types.Time	time	http://www.w3.org/2001/XMLSchema
org.apache.axis.holders.TimeHolder		
org.apache.axis.types.YearMonth	gYearMonth	http://www.w3.org/2001/XMLSchema

Java でのデータ型	WSDL でのデータ型	
	データ型	名前空間の URL
org.apache.axis.holders.YearMonthHolder		
org.apache.axis.types.Year	gYear	http://www.w3.org/2001/XMLSchema
org.apache.axis.holders.YearHolder		
org.apache.axis.types.MonthDay	gMonthDay	http://www.w3.org/2001/XMLSchema
org.apache.axis.holders.MonthDayHolder		
org.apache.axis.types.Day	gDay	http://www.w3.org/2001/XMLSchema
org.apache.axis.holders.DayHolder		
org.apache.axis.types.Month	gMonth	http://www.w3.org/2001/XMLSchema
org.apache.axis.holders.MonthHolder		
org.apache.axis.types.URI	anyURI	http://www.w3.org/2001/XMLSchema
org.apache.axis.holders.URIHolder		
org.apache.axis.types.NormalizedString	normalizedString	http://www.w3.org/2001/XMLSchema
org.apache.axis.holders.NormalizedStringHolder		
org.apache.axis.types.Token	token	http://www.w3.org/2001/XMLSchema
org.apache.axis.holders.TokenHolder		
org.apache.axis.types.Name	Name	http://www.w3.org/2001/XMLSchema
org.apache.axis.types.NCName	NCName	http://www.w3.org/2001/XMLSchema
org.apache.axis.types.NMToken	NMTOKEN	http://www.w3.org/2001/XMLSchema
org.apache.axis.types.NonPositiveInteger	nonPositiveInteger	http://www.w3.org/2001/XMLSchema
org.apache.axis.holders.NonPositiveIntegerHolder		
org.apache.axis.types.NegativeInteger	negativeInteger	http://www.w3.org/2001/XMLSchema
org.apache.axis.holders.NegativeIntegerHolder		
org.apache.axis.types.NonNegativeInteger	nonNegativeInteger	http://www.w3.org/2001/XMLSchema
org.apache.axis.holders.NonNegativeIntegerHolder		
org.apache.axis.types.UnsignedInt	unsignedInt	http://www.w3.org/2001/XMLSchema
org.apache.axis.holders.UnsignedIntHolder		
org.apache.axis.types.UnsignedLong	unsignedLong	http://www.w3.org/2001/XMLSchema
org.apache.axis.holders.UnsignedLongHolder		

Java でのデータ型	WSDL でのデータ型	
	データ型	名前空間の URL
org.apache.axis.types.UnsignedShort	unsignedShort	http://www.w3.org/2001/XMLSchema
org.apache.axis.holders.UnsignedShortHolder		
org.apache.axis.types.UnsignedByte	unsignedByte	http://www.w3.org/2001/XMLSchema
org.apache.axis.holders.UnsignedByteHolder		
org.apache.axis.types.PositiveInteger	positiveInteger	http://www.w3.org/2001/XMLSchema
org.apache.axis.holders.PositiveIntegerHolder		
org.apache.axis.types.Language	language	http://www.w3.org/2001/XMLSchema
org.apache.axis.types.Id	ID	http://www.w3.org/2001/XMLSchema
org.apache.axis.types.IDRef	IDREF	http://www.w3.org/2001/XMLSchema
org.apache.axis.types.Entity	ENTITY	http://www.w3.org/2001/XMLSchema
org.apache.axis.types.IDRefs	IDREFS	http://www.w3.org/2001/XMLSchema
org.apache.axis.types.Entities	ENTITIES	http://www.w3.org/2001/XMLSchema
org.apache.axis.types.NMTokens	NMTOKENS	http://www.w3.org/2001/XMLSchema
javax.activation.DataHandler	wsi:swaRef	http://ws-i.org/profiles/basic/1.1/xsd
org.apache.axis.holders.DataHandlerHolder		

注※1

WSDL 内に次に示す types 要素が生成されます。出力形式は Java2WSDL コマンドでの引数のオプションの設定値で異なります。

(1) Java2WSDL コマンドの引数で"-T 1.1"を指定した場合

```
<schema targetNamespace="http://localhost" (definitions要素のtargetNamespace)
  xmlns="http://www.w3.org/2001/XMLSchema">
  <complexType name="SequenceOf_xsd_byte">
    <sequence>
      <element name="item" minOccurs="0" maxOccurs="unbounded" type="xsd:byte" />
    </sequence>
  </complexType>
</schema>
```

(2) Java2WSDL コマンドの引数で"-T 1.2"を指定した場合

```
<schema targetNamespace="http://localhost" (definitions要素のtargetNamespace)
  xmlns="http://www.w3.org/2001/XMLSchema">
  <complexType name="SequenceOf_soapenc_byte">
    <complexContent>
      <restriction base="soapenc:Array">
        <attribute ref="soapenc:arrayType" wsdl:arrayType="soapenc:byte[]"/>
      </restriction>
    </complexContent>
```

```
  </complexType>
</schema>
```

注※2

WSDL 内に次に示す types 要素が生成されます。出力形式は Java2WSDL コマンドでの引数の設定値で異なります。

(1)Java2WSDL コマンドの引数で"-T 1.1"を指定した場合

```
<schema targetNamespace="http://localhost" (definitions要素のtargetNamespace)
  xmlns="http://www.w3.org/2001/XMLSchema">
  <complexType name="SequenceOf_xsd_anyType">
    <sequence>
      <element name="item" minOccurs="0" maxOccurs="unbounded" type="xsd:anyType" />
    </sequence>
  </complexType>
</schema>
```

(2)Java2WSDL コマンドの引数で"-T 1.2"を指定した場合

```
<schema targetNamespace="http://localhost" (definitions要素のtargetNamespace)
  xmlns="http://www.w3.org/2001/XMLSchema">
  <complexType name="SequenceOf_xsd_anyType">
    <complexContent>
      <restriction base="soapenc:Array">
        <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:anyType[]"/>
      </restriction>
    </complexContent>
  </complexType>
</schema>
```

注※3

Java2WSDL コマンドの引数で"-T 1.1"を指定した場合の，データ型の対応を次の表に示します。

表 11-4　Java クラスから WSDL を生成した場合のデータ型の対応

Java でのデータ型	WSDL でのデータ型	
	データ型	名前空間の URL
byte[]	base64Binary	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.ByteArrayHolder		
java.lang.Byte	byte	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.ByteWrapperHolder		
java.lang.Byte[]	SequenceOf_xsd_byte	definitions 要素の targetNamespace
javax.xml.rpc.holders.ByteWrapperArrayHolder		
java.lang.Double	double	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.DoubleWrapperHolder		
java.lang.Float	float	http://www.w3.org/2001/XMLSchema

Java でのデータ型	WSDL でのデータ型	
	データ型	名前空間の URL
javax.xml.rpc.holders.FloatWrapperHolder		
java.lang.Integer	int	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.IntegerWrapperHolder		
java.lang.Long	long	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.LongWrapperHolder		
java.lang.Short	short	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.ShortWrapperHolder		
java.lang.String	string	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.StringHolder		
java.math.BigDecimal	decimal	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.BigDecimalHolder		
java.math.BigInteger	integer	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.BigIntegerHolder		
java.lang.Boolean	boolean	http://www.w3.org/2001/XMLSchema
javax.xml.rpc.holders.BooleanWrapperHolder		

Java2WSDL コマンドの引数については「9.1　Java2WSDL コマンド（WSDL の生成）」を参照してください。

11.4　データ型についての注意事項

SOAP アプリケーション開発支援機能および SOAP 通信基盤で扱うデータ型についての注意事項を示します。

java.util.Calendar クラスのタイムゾーンについて

SOAP クライアントと SOAP サービス間のインタフェースとして，java.util.Calendar クラスを利用した場合，この製品から渡ってくる java.util.Calendar クラスのオブジェクトにはタイムゾーンとして"GMT"が設定されています。したがって，java.util.Calendar クラスのオブジェクトに設定されているタイムゾーンが異なる場合，オブジェクトの比較ができないことがありますので注意してください。

org.apache.axis.types.Language クラスのコンストラクタに xsd:language 型で許容しない文字を指定した場合の注意

org.apache.axis.types.Language クラスのコンストラクタに，xsd:language 型で許容しない文字を指定しても，例外が発生しないでオブジェクトの生成が成功します。

12 標準仕様との対応

この章では，SOAP，WSDL，SAAJ，および JAXR の仕様と，SOAP アプリケーション開発支援機能および SOAP 通信基盤で利用できる機能の関係について説明します。

12.1 SOAP 1.1 との対応

SOAP 1.1 の仕様と，SOAP アプリケーション開発支援機能および SOAP 通信基盤で利用できる機能の関係について次の表に示します。大分類に示した章番号は，W3C の SOAP 1.1 の仕様に記載されている章番号を表します。

表 12–1 SOAP 1.1 仕様のサポート範囲の一覧

分類		サポート	備考
大分類	小分類		
SOAP メッセージ交換モデル (2 章)	ワンウェイ	×	メッセージの一方送信について規定しています。
	リクエスト・レスポンス	○	要求メッセージの送信，および応答メッセージの受信について規定しています。
	マルチキャスト	×	特定多数のメッセージ送信について規定しています。
XML との関係 (3 章)	SOAP エンベロープ名前空間	○	SOAP エンベロープで使用する名前空間について規定しています。SOAP エンベロープは，http://schemas.xmlsoap.org/soap/envelope/で識別します。
	SOAP 直列化名前空間	○	SOAP 直列化の名前空間について規定しています。SOAP の直列化規則は，http://schemas.xmlsoap.org/soap/encoding/で識別します。
	uri-reference 型の参照	○	XML1.0，XML Schema1.0，および XML Linking Language1.0 に従った仕様です。
	一般の属性，値の SOAP エンベロープへの出現	○	SOAP エンベロープでの，一般の属性と値の出現について規定しています。
SOAP エンベロープ (4 章)	構文	○	SOAP エンベロープの構文について規定しています。
	SOAP encodingStyle 属性	○	エンコーディングスタイルについて規定しています。エンコーディングスタイルで，任意の要素に対して直列化規則を指定できます。
SOAP ヘッダ (4 章)	構文	○	SOAP ヘッダの構文について規定しています。
	要素の無視	○	SOAP Header 要素の子要素以外の SOAP ヘッダ属性を無視します。
	actor	○	指定した SOAP ヘッダ項目の受信者について規定しています。
	mustUnderstand	○	指定した SOAP ヘッダ項目が必須の処理であることを示します。
SOAP ボディ (4 章)	構文	○	SOAP ボディの構文について規定しています。
	名前空間修飾	○	SOAP Body 要素の子要素は，名前空間で修飾されていてもかまいません。
SOAP Fault (4 章)	コード	○	SOAP Fault のコードを規定します。

分類		サポート	備考
大分類	小分類		
SOAP 符号化 (5 章)	単一参照	○	一つのアクセサだけが参照している値の直列化について規定しています。
	多重参照	○	二つ以上のアクセサによって参照される可能性がある値の直列化について規定しています。
	単純データ型	△	WSDL で simpleType で表されるデータ型の直列化について規定しています。
	バイト配列・文字列	○	バイト配列および文字列の直列化について規定しています。
	列挙	△	列挙型の直列化について規定しています。
	構造体（連結）	○	構造体（連結構造体を含む）の直列化について規定しています。
	複合データ型	△	WSDL で complexType で表されるデータ型の直列化について規定しています。
	配列（疎，部分，Jagged，多次元）	△	疎である配列，部分配列，Jagged 配列，および多次元配列の直列化について規定しています。Jagged 配列とは，要素そのものが配列となっているものをいいます。RPC 形態のアプリケーションでは，疎である配列，部分配列を生成できません。
	デフォルト値	△	アクセサ要素の省略について規定しています。 RPC 形態のアプリケーションで扱える NULL 値は，xsi:nil="true"の形式だけです。
	多態的アクセサ	×	多態的アクセサとは，共用体など実行時にデータ型が決まるものを示します。
HTTP (6 章)	SOAPAction	○	SOAP のリクエストメッセージの意図を示す URI を示します。
	SOAPHTTP レスポンス	○	HTTP1.0 の Status コードに従います。
	HTTP 拡張フレームワーク	×	SOAP の HTTP 拡張フレームワークでの使用について規定しています。

(凡例)

○：サポートされます。

△：制限付きでサポートされます。

×：サポートされません。

12.2　WSDL 1.1 との対応

SOAP アプリケーションを開発するときに留意が必要な，WSDL のサポート範囲を示します。

12.2.1　WSDL 1.1 仕様のサポート範囲

WSDL 1.1 仕様と，SOAP アプリケーション開発支援機能および SOAP 通信基盤で利用できる機能の関係について説明します。また，日本語で定義する場合のサポート範囲についても示します。

(1)　WSDL 1.1 仕様のサポート範囲

表 12–2　WSDL 1.1 仕様のサポート範囲

分類		サポート	備考
大分類	小分類		
サービス定義：WSDL ドキュメント構造	ドキュメントの名前付けとリンク	○	名前空間のスコープについて規定しています。
	作成スタイル（部品の import 要素による取り込み）	△	ほかのファイルの import 要素による取り込みについて規定しています。外部のインポート情報（URL）は取り込めません。
	言語の拡張性とバインディング	×	wsdl:required 属性のデプロイについて規定しています。
	ドキュメンテーション	○	要素内のコメントについて規定しています。
サービス定義：タイプ		△	扱うデータ型について規定しています。扱えるデータ型については，「11.　SOAP アプリケーションで扱うデータ型」を参照してください。
サービス定義：メッセージ		△	メッセージの論理定義を規定しています。
サービス定義：ポートタイプ	一方向操作	×	メッセージの一方向操作について規定しています。
	要求/応答操作	○	メッセージの要求/応答操作について規定しています。
	送信請求/応答操作	×	メッセージの送信要求/応答操作について規定しています。
	通知操作	×	メッセージの通知操作について規定しています。
	操作内の要素の名前	○	input および output 要素の name 属性を示します。オーバーロードはできません。WSDL 内では名称を一意にする必要があります。
	操作内のパラメタの順序	○	操作内のパラメタの順序について規定しています。parameterOrder 属性によって，パラメタのリストを指定できます。
サービス定義：バインディング		○	メッセージ形式とプロトコルの詳細の定義を規定しています。
サービス定義：ポート		○	サービスの物理的定義について規定しています。

分類		サポート	備考
大分類	小分類		
サービス定義：サービス		△	サービスの位置について規定しています。一つの WSDL に複数の SOAP サービスを対応させることはできません。
SOAP バインディング	soap:binding	○	SOAP 形式のバインディングを規定しています。
	soap:operation	△	SOAP オペレーションへの情報について規定しています。RPC 利用時の指定要素で使用できます。
	soap:body	△	SOAP Body 要素内でのメッセージ部分の表示方法について規定しています。RPC 利用時の指定要素で使用できます。
	soap:fault	○	SOAP Fault Details 要素の内容について規定しています。
	soap:header と soap:headerfault	×	SOAP Header 要素内の内容について規定しています。
	soap:address	○	port 要素のアドレスについて規定しています。
MIME バインディング	mime:content	×	MIME タイプを規定しています。
	mime:multipartRelated	×	MIME パートの任意のセットを集約しています。
	mime:part	×	個々の MIME パートを規定しています。
	mime:mimeXml	×	特定のスキーマを持っている XML ペイロードを規定しています。SOAP には準拠していません。

（凡例）

○：サポートされます。

△：制限付きでサポートされます。

×：サポートされません。

(2) WSDL 1.1 仕様での日本語のサポート範囲

document/literal 使用時の WSDL のサポート範囲を示します。

表 12–3 WSDL の要素と日本語のサポート範囲

要素			属性	データ型	日本語サポート
definitions			name	NMTOKEN	○
┃			targetNamespace	anyURI	△
┃			xmlns:xxx	anyURI	△
┃			xmlns	anyURI	△
┃					
┣	documentation		−	−	−
┃	┗	text	−	−	○

要素			属性	データ型	日本語サポート
├	import		namespace	anyURI	△
│			location	anyURI	△
│	└	documentation	−	−	○
├	types		−	−	−
│	├	documentation	−	−	○
│	└	any	−	−	「12.2.4　XML Schema のサポート範囲」を参照してください。
├	message		name	NCName	○
│	├	documentation	−	−	○
│	└	part	name	NMTOKEN	○
│			element	QName	○
│			type	QName	○
│		└ documentation	−	−	×
├	portType		name	NMTOKEN	○
│	├	documentation	−	−	○
│	└	operation	−	−	−
│		├ documentation	−	−	○
│		│	name	NMTOKEN	○
│		│	parameterOrder	NMTOKENS	○
│		├ input	name	NMTOKEN	○
│		│	message	QName	○
│		│ └ documentation	−	−	○
│		├ output	name	NMTOKEN	○
│		│	message	QName	○
│		│ └ documentation	−	−	○
│		└ fault	name	NMTOKEN	○

要素					属性	データ型	日本語サポート
│					message	QName	○
│			└	documentation	−	−	○
├	binding				name	NMTOKEN	○
					type	QName	○
	├	documentation			−	−	○
	├	soap:binding			style	string	×
	│				transport	anyURI	×
	└	operation			name	NMTOKEN	○
		├	documentation		−	−	○
		├	soap:operation		soapAction	anyURI	×
		│			style	string	×
		├	input		name	NMTOKEN	○
			├	documentation	−	−	○
			├	soap:header※1	−	−	×
			└	soap:body	parts	NMTOKEN	○
					use	string	×
					encodingStyle	anyURI-list	×
					namespace	anyURI	△
		├	output		name	NMTOKEN	○
			├	documentation	−	−	○
			├	soap:header※1	−	−	×
			└	soap:body	parts	NMTOKEN	○
					use	string	×
					encodingStyle	anyURI-list	×
					namespace	anyURI	△
		└	fault		name	NMTOKEN	○
			├	documentation	−	−	○
			└	soap:fault	name	NMTOKEN	○

要素					属性	データ型	日本語サポート
┃					use	string	×
┃					encodingStyle	anyURI-list	×
┃					namespace	anyURI	△
┗	service				name	NMTOKEN	○
	┣	documentation			－	－	○
	┗	port			name	NCName	○
					binding	QName	○
		┣	documentation		－	－	○
		┗	soap:address		location	anyURI	△※2

（凡例）

○：サポートされます。

△：サポートされます。ただし，パーセントエンコードした値を定義する必要があります。パーセントエンコードするときに使用できる符号化形式は UTF-8 です。

×：サポートされません。

－：該当しないことを表します。

注※1

soap:header 要素，および soap:header 要素に含まれる子要素（soap:headerfault 要素）は，属性を含めてすべてサポートされません。

注※2

URL の末尾（例の下線部分）にだけ日本語を指定できます。

（例）http://localhost:8080/RPCSampleService/service/UserInfo

12.2.2　Java の基本データ型のサポート範囲

SOAP アプリケーション開発支援機能および SOAP 通信基盤を使用する場合の，Java の基本データ型のサポート範囲について次の表に示します。

表 12–4　Java 基本データ型のサポート範囲の一覧

Java の基本データ型	サポート	Java の基本データ型	サポート
boolean	○	short	○
byte	○	int	○
byte[]	○	long	○

Java の基本データ型	サポート	Java の基本データ型	サポート
char	×	float	○
char[]	×	double	○

（凡例）

○：サポートされます。

×：サポートされません。

12.2.3　XML Schema のデータ型のサポート範囲

SOAP アプリケーション開発支援機能および SOAP 通信基盤を使用する場合の，XML Schema のデータ型，およびデータ型の列挙のサポート範囲について次の表に示します。

表 12–5　XML Schema のデータ型，およびデータ型列挙のサポート範囲の一覧

XML Schema のデータ型	データ型	データ型の列挙	attribute 要素	list 要素※1
xsd:duration	○	○	○	○
xsd:dateTime※2	○	×	○	○
xsd:time	○	×	○	○
xsd:date	○	×	○	○
xsd:gYearMonth	○	○	○	○
xsd:gYear	○	○	○	○
xsd:gMonthDay	○	○	○	○
xsd:gDay	○	○	○	○
xsd:gMonth※3	○	○	○	○
xsd:boolean	○	×	○	○
xsd:base64Binary※4	○	×	○	×
xsd:hexBinary※4	○	×	○	×
xsd:float	○	○	○	○
xsd:double	○	○	○	○
xsd:anyURI※5	○	×	○	○
xsd:QName	○	○	○	○
xsd:string	○	○	○	○
xsd:normalizedString※5	○	×	○	○
xsd:token※5	○	○	○	○

XML Schema のデータ型	データ型	データ型の列挙	attribute 要素	list 要素※1
xsd:language※5	△	○	○	○
xsd:Name※5	△	○	○	○
xsd:NCName※5	△	○	○	○
xsd:ID※5	△	○	○	○
xsd:IDREF※5	△	○	○	○
xsd:ENTITY※5	△	○	○	○
xsd:IDREFS※5	△	×	○	×
xsd:ENTITIES※5	△	×	○	×
xsd:NMTOKEN※5	△	○	○	○
xsd:NMTOKENS※5	△	×	○	×
xsd:decimal	○	○	○	○
xsd:integer	○	○	○	○
xsd:nonPositiveInteger	○	○	○	○
xsd:negativeInteger	○	○	○	○
xsd:long	○	○	○	○
xsd:int	○	○	○	○
xsd:short	○	○	○	○
xsd:byte	○	○	○	○
xsd:nonNegativeInteger	○	○	○	○
xsd:unsignedInt	○	○	○	○
xsd:unsignedShort	○	○	○	○
xsd:unsignedByte	○	○	○	○
xsd:unsignedLong※3	○	○	○	○
xsd:positiveInteger	○	○	○	○
xsd:anyType※4※6※7	○	×	×	×
xsd2000:timeInstant※8	×	×	×	×

（凡例）

○：サポートされます。

△：IN パラメタおよび戻り値だけサポートされます。

×：サポートされません。

注※1

xsd:list 要素の itemType 属性に指定するデータ型を指します。

注※2

xsd:dateTime 型で紀元前の日付を送信すると，SOAP メッセージ中の日付が正しい値にならないため，受信側で正しい値を受信できません。

注※3

gMonth および UnsignedLong には次に示す制限があります。サポート範囲外の値を使用した場合，他社製品と通信できないおそれがあります。

表 12-6　gMonth および UnsignedLong の制限

WSDL でのデータ型	注意事項
gMonth	gMonth の送受信パターンは--MM--です。--MM のパターンは扱えません。
UnsignedLong	UnsignedLong の範囲は 0～9,007,199,254,740,992 の整数です。最大値を超えたデータを送受信した場合，値が異なる可能性があるため，範囲内で使用してください。

注※4

WSDL のスタイルが RPC／ENCODED の場合，wsdl:arrayType 属性を使用して soapenc:Array 型を制限した複合型を使用できません。wsdl:arrayType 属性に指定しないでください。

注※5

空文字列を送信すると受信時に null になります。

注※6

複合型の中で maxOccurs 属性を指定した要素は，配列として使用できません。maxOccurs 属性を指定した要素の type 属性には指定しないでください。

注※7

xsd:anyType 型で，配列型の値を送信しないでください。xsd:anyType 型で，配列型の値を送信すると，SOAP Fault または例外が発生する場合があります。

注※8

xsd2000:timeInstant は使用できません。代わりに，xsd:dateTime を使用してください。

12.2.4　XML Schema のサポート範囲

document/literal 使用時の XML Schema のサポート範囲を示します。

表 12-7　XML Schema のサポート範囲

要素名	サポート	属性名	サポート	日本語サポート
schema	○	id	○	○
		attributeFormDefault	○	×
		blockDefault	△	×
		elementFormDefault	○	×
		finalDefault	△	×

要素名	サポート	属性名	サポート	日本語サポート
		targetNamespace	○	△
		version	△	○
		xml:lang	△	×
		xmlns:xxx	○	△
		xmlns	○	△
include	○	id	○	○
		schemaLocation	○	△
import	○	id	○	○
		namespace	○	△
		schemaLocation	○	△
redefine	×	id	×	×
		schemaLocation	×	×
element （グローバル）	○	id	○	○
		abstract	○	×
		final	×	×
		substitutionGroup	×	×
		block	△	×
		default	×	×
		fixed	×	×
		name	○	○
		nillable	○	×
		type	○	○
element （ローカル）	○	id	○	○
		minOccurs	○	×
		maxOccurs	○	×
		form	○	×
		block	△	×
		default	×	×
		fixed	×	×
		name	○	○
		nillable	○	×

要素名	サポート	属性名	サポート	日本語サポート
		type	○	○
		ref	○	○
simpleType （グローバル）※1※2	○	id	○	○
		final	△	×
		name	○	○
simpleType （ローカル）※1※2	○	id	○	○
complexType （グローバル）	○	id	○	○
		abstract	○	×
		block	△	×
		final	△	×
		mixed	×	×
		name	○	○
complexType （ローカル）	○	id	○	○
		mixed	×	×
all	○※3	id	○	○
		minOccurs	○	×
		maxOccurs	○	×
sequence	○※3	id	○	○
		minOccurs	×	×
		maxOccurs	×	×
choice	×	id	×	×
		minOccurs	×	×
		maxOccurs	×	×
any	×	id	×	×
		minOccurs	×	×
		maxOccurs	×	×
		namespace	×	×
		processContents	×	×
group	×	id	×	×
		minOccurs	×	×

要素名	サポート	属性名	サポート	日本語サポート
		maxOccurs	×	×
		ref	×	×
		name	×	×
attribute (グローバル)	○	id	○	○
		default	×	×
		fixed	×	×
		name	○	○
		type	○	○
		wsdl:arrayType	○	○
attribute (ローカル)	○※3	id	○	○
		default	×	×
		fixed	×	×
		name	○	○
		type	○	○
		wsdl:arrayType	○	○
		ref	○	○
		form	○	×
		use	×	×
anyAttribute	×	id	×	×
		namespace	×	×
attributeGroup	×	id	×	×
		name	×	×
		ref	×	×
list※4	○	id	○	○
		itemType	○	○
union	×	id	×	×
		memberTypes	×	×
enumeration	○	value	○	×
minExclusive	△	value	△	×
		fixed	×※5	×
minInclusive	△	value	△	×

要素名	サポート	属性名	サポート	日本語サポート
		fixed	×※5	×
maxExclusive	△	value	△	×
		fixed	×※5	×
maxInclusive	△	value	△	×
		fixed	×※5	×
totalDigits	△	value	△	×
fractionDigits	△	value	△	×
length	△	value	△	×
minLength	△	value	△	×
maxLength	△	value	△	×
whiteSpace	△	value	△	○
		fixed	×※5	×
pattern	△	value	△	○
complexContent※6	○	id	○	○
		mixed	×	×
simpleContent	×	id	×	×
restriction※2※6	○	id	○	○
		base	○	○
extension※7	○	id	○	○
		base	○	○
unique	×	id	×	×
		name	×	×
key	×	id	×	×
		name	×	×
keyref	×	id	×	×
		name	×	×
		refer	×	×
selector	×	id	×	×
		xpath	×	×
field	×	id	×	×

要素名	サポート	属性名	サポート	日本語サポート
		xpath	×	×
notation	×	id	×	×
		name	×	×
		public	×	×
		system	×	×
annotation	○	id	○	○
appinfo※8	○	source	○	×
documentation	○	source	○	×
		xml:lang	○	×

（凡例）

○：サポートされます。

△：指定しても機能は有効になりません。アプリケーションで対応してください。

×：サポートされません。

注※1

simpleType 要素および union 要素は子要素として指定できません。また，restriction 要素および list 要素の記述には制限があります。注意事項およびサポートされない要素の代替記述については，「12.2.8　XML Schema 記述時の注意事項」の該当する注意事項を参照してください。

注※2

restriction 要素の base 属性は必ず指定してください。また，simpleType 要素の子要素に restriction 要素を記述する場合には，base 属性を指定し，その子要素には enumeration 要素だけを記述してください。この組み合わせ以外では，不正なソースが生成されます。

注※3

次の要素を合計した数は 254 個までです。255 個以上は記述できません。

・all 要素または sequence 要素の子要素

・all 要素または sequence 要素の兄弟要素である attribute 要素

合計で 255 個以上記述した場合，不正なソースが生成されます。

注※4

list 要素を記述する場合，itemType 属性を必ず指定してください。また，子要素に simpleType 要素は記述できません。記述できない属性，要素を記述した場合，不正なソースが生成されます。

注※5

facet の fixed 属性とは，facet を設定している型を派生（extension または restriction）させるときに派生する型のことです。fixed 属性は，facet の値の変更を禁止します（デフォルトは false）。

注※6

complexContent 要素の子要素に restriction 要素を記述する場合には，base 属性を指定し，その属性値には"soapenc:Array"だけを指定できます。この組み合わせ以外では，不正なソースが生成されます。

注※7

complexContent の子要素だけをサポートしています。

注※8

WSDL2Java コマンドは，記述した内容を無視します。

12.2.5　soapencoding 型のサポート範囲

SOAP アプリケーション開発支援機能および SOAP 通信基盤を使用する場合の，soapencoding 型のサポート範囲について次の表に示します。

表 12–8　soapencoding 型のサポート範囲の一覧

データ型	サポート	attribute 要素，データ型を参照する修飾名※1
string	○	×
boolean	○	×
double	○	×
float	○	×
int	○	×
long	○	×
short	○	×
byte	○	×
integer	○	×
decimal	○	×
Array	△	×
base64※2	○	×

（凡例）

○：サポートされます。

△：IN パラメタで使用する場合だけサポートされます。

×：サポートされません。

注※1

データ型を参照する修飾名とは，base 属性および itemType 属性に指定するデータ型を指します。

注※2

WSDL のスタイルが RPC／ENCODED の場合，wsdl:arrayType 属性を使用して soapenc:Array 型を制限した複合型を使用できません。wsdl:arrayType 属性に指定しないでください。

12.2.6　DII 使用時のサポート範囲

DII 使用時に留意が必要な XML データ型，XML Schema のデータ型，および soapencoding 型のサポート範囲を示します。

(1) XML データ型のサポート範囲（DII 使用時）

表 12-9 XML データ型のサポート範囲（DII 使用時）

XML データ型	サポート	備考
XML Schema のデータ型	○	パラメタの種別によって未サポートの場合があります。詳細は，表 12-10 を参照してください。
XML Schema のデータ型の列挙	×	−
XMLSchema のデータ型の配列	○	−
soapencoding 型	○	パラメタの種別によって未サポートの場合があります。詳細は，表 12-11 を参照してください。
soapencoding 型の列挙	×	−
soapencoding 型の配列	○	−
ユーザ定義のデータ型クラス	○	−
ユーザ定義例外	○	−

（凡例）

○：サポートされます。

×：サポートされません。

(2) XML Schema のデータ型のサポート範囲（DII 使用時）

表 12-10 XML Schema のデータ型のサポート範囲（DII 使用時）

XML Schema のデータ型	Java データ型	サポート			
		IN	RET	OUT	INOUT
xsd:anyType	java.lang.Object	○	○	○	○
xsd:base64Binary	byte[]	○	○	×	×
xsd:boolean	java.lang.Boolean	○	○	○	○
xsd:byte	java.lang.Byte	○	○	○	○
xsd:date	java.util.Date	○	○	○	○
xsd:dateTime※1	java.util.GregorianCalendar	○	○	○	○
xsd:decimal	java.math.BigDecimal	○	○	○	○
xsd:double	java.lang.Double	○	○	○	○
xsd:float	java.lang.Float	○	○	○	○
xsd:hexBinary	byte[]	○	○	○	○

XML Schema のデータ型	Java データ型	サポート			
		IN	RET	OUT	INOUT
xsd:int	java.lang.Integer	○	○	○	○
xsd:integer	java.math.BigInteger	○	○	○	○
xsd:long	java.lang.Long	○	○	○	○
xsd:QName	javax.xml.namespace.QName	○	○	○	○
xsd:short	java.lang.Short	○	○	○	○
xsd:string	java.lang.String	○	○	○	○
xsd:duration	org.apache.axis.types.Duration	○	○	○	○
xsd:time	org.apache.axis.types.Time	○	○	○	○
xsd:gYearMonth	org.apache.axis.types.YearMonth	○	○	○	○
xsd:gYear	org.apache.axis.types.Year	○	○	○	○
xsd:gMonthDay	org.apache.axis.types.MonthDay	○	○	○	○
xsd:gDay	org.apache.axis.types.Day	○	○	○	○
xsd:gMonth	org.apache.axis.types.Month	○	○	○	○
xsd:anyURI※2	org.apache.axis.types.URI	○	○	○	○
xsd:normalizedString※2	org.apache.axis.types.NormalizedString	○	○	○	○
xsd:token※2	org.apache.axis.types.Token	○	○	○	○
xsd:Name※2	org.apache.axis.types.Name	○	○	×	×
xsd:NCName※2	org.apache.axis.types.NCName	○	○	×	×
xsd:NMTOKEN※2	org.apache.axis.types.NMTOKEN	○	○	×	×
xsd:nonPositiveInteger	org.apache.axis.types.NonPositiveInteger	○	○	○	○
xsd:negativeInteger	org.apache.axis.types.NegativeInteger	○	○	○	○
xsd:nonNegativeInteger	org.apache.axis.types.NonNegativeInteger	○	○	○	○
xsd:unsignedInt	org.apache.axis.types.UnsignedInt	○	○	○	○
xsd:unsignedShort	org.apache.axis.types.UnsignedShort	○	○	○	○
xsd:unsignedLong	org.apache.axis.types.UnsignedLong	○	○	○	○
xsd:unsignedByte	org.apache.axis.types.UnsignedByte	○	○	○	○
xsd:positiveInteger	org.apache.axis.types.PositiveInteger	○	○	○	○
xsd:language※2	org.apache.axis.types.Language	○	○	×	×

XML Schema のデータ型	Java データ型	サポート			
		IN	RET	OUT	INOUT
xsd:ID※2	org.apache.axis.types.Id	○	○	×	×
xsd:IDREF※2	org.apache.axis.types.IDRef	○	○	×	×
xsd:ENTITY※2	org.apache.axis.types.Entity	○	○	×	×
xsd:IDREFS※2	org.apache.axis.types.IDRefs	○	○	×	×
xsd:ENTITIES※2	org.apache.axis.types.Entities	○	○	×	×
xsd:NMTOKENS※2	org.apache.axis.types.NMTokens	○	○	×	×
xsd2000:timeInstant※3	java.util.GregorianCalendar	×	×	×	×

（凡例）

IN：IN パラメタとして使用します。

RET：戻り値として使用します。

OUT：OUT パラメタとして使用します。

INOUT：INOUT パラメタとして使用します。

○：サポートされます。

×：サポートされません。

注※1

xsd:dateTime 型で紀元前の日付を送信すると，SOAP メッセージ中の日付が正しい値にならないため，受信側で正しい値を受信できません。

注※2

空文字列を送信すると受信時に null になります。

注※3

xsd2000:timeInstant は使用できません。代わりに，xsd:dateTime を使用してください。

(3) soapencoding 型のサポート範囲（DII 使用時）

表 12–11 soapencoding 型のサポート範囲（DII 使用時）

XML Schema のデータ型	Java データ型	サポート			
		IN	RET	OUT	INOUT
soapenc:Array	java.lang.Object[]	○	○	×	×
soapenc:base64	byte[]	○	○	×	×
soapenc:boolean	java.lang.Boolean	○	○	○	○
soapenc:byte	java.lang.Byte	○	○	○	○

XML Schema のデータ型	Java データ型	サポート			
		IN	RET	OUT	INOUT
soapenc:decimal	java.math.BigDecimal	○	○	○	○
soapenc:double	java.lang.Double	○	○	○	○
soapenc:float	java.lang.Float	○	○	○	○
soapenc:int	java.lang.Integer	○	○	○	○
soapenc:integer	java.math.BigInteger	○	○	○	○
soapenc:long	java.lang.Long	○	○	○	○
soapenc:short	java.lang.Short	○	○	○	○
soapenc:string	java.lang.String	○	○	○	○

（凡例）

IN：IN パラメタとして使用します。

RET：戻り値として使用します。

OUT：OUT パラメタとして使用します。

INOUT：INOUT パラメタとして使用します。

○：サポートされます。

×：サポートされません。

12.2.7　.NET Framework 使用時のサポート範囲

.NET Framework 使用時の XML Schema のデータ型，および XML Schema のサポート範囲を示します。

(1)　XML Schema のデータ型のサポート範囲（.NET Framework 使用時）

表 12–12　XML Schema のデータ型のサポート範囲（.NET Framework 使用時）

XML Schema のデータ型	データ型	データ型の列挙	attribute 要素	list 要素※1
xsd:duration	○	○	○	○
xsd:dateTime※2	○	×	○	○
xsd:time	×※3	×	×	×
xsd:date	×※3	×	×	×
xsd:gYearMonth	○	○	○	○
xsd:gYear	○	○	○	○
xsd:gMonthDay	○	○	○	○

XML Schema のデータ型	データ型	データ型の列挙	attribute 要素	list 要素※1
xsd:gDay	○	○	○	○
xsd:gMonth	○	○	○	○
xsd:boolean	○	×	○	○
xsd:base64Binary	○※4	×	○	×
xsd:hexBinary	○※4	×	○	×
xsd:float	○	○	○	○
xsd:double	○	○	○	○
xsd:anyURI※5	○	×	○	○
xsd:QName	×	×	×	×
xsd:NOTATION	×	×	×	×
xsd:string	○	○	○	○
xsd:normalizedString※5	○	×	○	○
xsd:token※5	○	○	○	○
xsd:language※5	△	○	○	○
xsd:Name※5※6	△	○	○	○
xsd:NCName※5※6	△	○	○	○
xsd:ID※5	△	○	○	○
xsd:IDREF※5	△	○	○	○
xsd:ENTITY※5	△	○	○	○
xsd:IDREFS※5	△	×	○	×
xsd:ENTITIES※5	△	×	○	×
xsd:NMTOKEN※5※6	△	○	○	○
xsd:NMTOKENS※5※6	△	×	○	×
xsd:decimal	○	○	○	○
xsd:integer	○	○	○	○
xsd:nonPositiveInteger	○	○	○	○
xsd:negativeInteger	○	○	○	○
xsd:long	○	○	○	○
xsd:int	○	○	○	○
xsd:short	○	○	○	○

XML Schema のデータ型	データ型	データ型の列挙	attribute 要素	list 要素※1
xsd:byte	○	○	○	○
xsd:nonNegativeInteger	○	○	○	○
xsd:unsignedInt	○	○	○	○
xsd:unsignedShort	○	○	○	○
xsd:unsignedByte	○	○	○	×
xsd:unsignedLong	○	○	○	○
xsd:positiveInteger	○	○	○	○
xsd:anyType※7	○	×	×	×
xsd2000:timeInstant※8	×	×	×	×

（凡例）

○：サポートされます。

△：IN パラメタおよび戻り値だけサポートされます。

×：サポートされません。

注※1

xsd:list 要素の itemType 属性に指定するデータ型を指します。

注※2

xsd:dateTime 型で紀元前の日付を送信すると，SOAP メッセージ中の日付が正しい値にならないため，受信側で正しい値を受信できません。

注※3

xsd:time および xsd:date のデータ型はサポートされません。代わりに xsd:dateTime を使用してください。

注※4

xsd:base64Binary および xsd:hexBinary のデータ型を使用する場合，.NET Framework 上のクライアントから送受信した値と，SOAP 通信基盤で送受信した値が異なる場合があります。バイナリ値として解釈すると同じ値として解釈されますが，数値として解釈すると異なる値で解釈されることがあります。

例えば，0xFF のバイナリ値は，.NET クライアントでは 255 と解釈されますが，SOAP 通信基盤では-1 と解釈されます。

注※5

空文字列を送信すると受信時に null になります。

注※6

XML Schema の仕様で許容されない文字を送信すると，SOAP メッセージ中のこれらの文字が.NET Framework 独自の形式にエンコードされるため，受信側で正しい値を受信できません。

注※7

xsd:anyType 型で，配列型の値を送信しないでください。xsd:anyType 型で，配列型の値を送信すると，SOAP Fault または例外が発生する場合があります。

注※8

xsd:timeInstant は使用できません。代わりに，xsd:dateTime を使用してください。

(2) XML Schema のサポート範囲（.NET Framework 使用時）

表 12-13 XML Schema のサポート範囲（.NET Framework 使用時）

要素名	サポート	属性名	サポート
schema	○	id	×
		attributeFormDefault	○
		blockDefault	×
		elementFormDefault	○
		finalDefault	×
		targetNamespace	○
		version	×
		xml:lang	△
include	○	id	○
		schemaLocation	○
import	○	id	○
		namespace	○
		schemaLocation※1	○
redefine	×	id	×
		schemaLocation	×
element（グローバル）	○	id	○
		abstract	○
		final	×
		substitutionGroup	×
		block	△
		default	×
		fixed	×
		name	○
		nillable	○
		type	○
element（ローカル）	○	id	○
		minOccurs	○
		maxOccurs	○

要素名	サポート	属性名	サポート
		form	○
		block	△
		default	×
		fixed	×
		name	○
		nillable	○
		type	○
		ref	○
simpleType （グローバル）※2※3	○	id	○
		final	△
		name	○
simpleType （ローカル）※2※3	○	id	○
complexType （グローバル）	○	id	○
		abstract	○
		block	△
		final	△
		mixed	×
		name	○
complexType （ローカル）	○	id	○
		mixed	×
all	○※4	id	○
		minOccurs	△
		maxOccurs	○
sequence	○※4	id	○
		minOccurs	×
		maxOccurs	×
choice	×	id	×
		minOccurs	×
		maxOccurs	×
any	×	id	×

要素名	サポート	属性名	サポート
		minOccurs	×
		maxOccurs	×
		namespace	×
		processContents	×
group	×	id	×
		minOccurs	×
		maxOccurs	×
		ref	×
		name	×
attribute (グローバル)	○	id	○
		default	×
		fixed	×
		name	○
		type	○
		wsdl:arrayType	×
attribute (ローカル)	○※4	id	○
		default	×
		fixed	×
		form	○
		name	○
		ref	○
		type	○
		use	×
		wsdl:arrayType	×
anyAttribute	×	id	×
		namespace	×
attributeGroup	×	id	×
		name	×
		ref	×
list※5	○	id	○
		itemType	○

要素名	サポート	属性名	サポート
union	×	id	×
		memberTypes	×
enumeration	○	value	○
minExclusive	△	value	△
		fixed	×※6
minInclusive	△	value	△
		fixed	×※6
maxExclusive	△	value	△
		fixed	×※6
maxInclusive	△	value	△
		fixed	×※6
totalDigits	△	value	△
fractionDigits	△	value	△
length	△	value	△
minLength	△	value	△
maxLength	△	value	△
whiteSpace	△	value	△
		fixed	×※6
pattern	△	value	△
complexContent※7	○	id	○
		mixed	×
simpleContent	×	id	×
restriction※3※7	○	id	○
		base	○
extension※8	○	id	○
		base	○
unique	×	id	×
		name	×
key	×	id	×
		name	×

要素名	サポート	属性名	サポート
keyref	×	id	×
		name	×
		refer	×
selector	×	id	×
		xpath	×
field	×	id	×
		xpath	×
notation	×	id	×
		name	×
		public	×
		system	×
annotation	○	id	○
appinfo※9	○	source	○
documentation	○	source	○
		xml:lang	○

（凡例）

○：サポートされます。

△：指定しても機能は有効になりません。アプリケーションで対応してください。

×：サポートされません。

注※1

.NET Framework を使用する場合，schemaLocation 属性は無視されます。インポートするファイルの指定方法は，.NET Framework のドキュメントを参照してください。

注※2

simpleType 要素および union 要素は子要素として指定できません。また，restriction 要素および list 要素の記述には制限があります。注意事項およびサポートされない要素の代替記述については，「12.2.8　XML Schema 記述時の注意事項」の該当する注意事項を参照してください。

注※3

restriction 要素の base 属性は必ず指定してください。また，simpleType 要素の子要素に restriction 要素を記述する場合には，base 属性を指定し，その子要素には enumeration 要素だけを記述してください。この組み合わせ以外では，不正なソースが生成されます。

注※4

次の要素を合計した数は 254 個までです。255 個以上は記述できません。

・all 要素または sequence 要素の子要素

・all 要素または sequence 要素の兄弟要素である attribute 要素

合計で 255 個以上記述した場合，不正なソースが生成されます。

注※5

list 要素を記述する場合，itemType 属性を必ず指定してください。また，子要素に simpleType 要素は記述できません。記述できない属性，要素を記述した場合，不正なソースが生成されます。

注※6

facet の fixed 属性とは，facet を設定している型を派生（extension または restriction）させるときに派生する型のことです。fixed 属性は，facet の値の変更を禁止します（デフォルトは false）。

注※7

complexContent 要素の子要素に，restriction 要素を記述することはサポートしていません。

注※8

complexContent の子要素だけをサポートしています。

注※9

WSDL2Java コマンドは，記述した内容を無視します。

12.2.8　XML Schema 記述時の注意事項

XML Schema の要素を記述するときの注意事項について説明します。また，非サポート要素の代替記述についても説明します。

(1)　maxOccurs 属性を記述する場合の注意事項

- element 要素に maxOccurs 属性を指定する場合，アプリケーションでは XML Schema で指定されている値の範囲で配列を使用してください。

  element 要素に maxOccurs を記述した場合，生成されたソースでは，該当する要素は配列型の変数となりますが，配列の最大値は設定されません。したがって，要素の配列数はアプリケーションに依存することになります。スキーマに指定されている値より大きい配列をアプリケーションで指定すると，不正な SOAP メッセージが送信されることになります。

- maxOccurs を記述した element 要素を OUT 属性および INOUT 属性にして使用することはできません。OUT 属性および INOUT 属性にして使用した場合，不正な Holder クラスのソースが生成されます。

- nonwrapped 形式では，maxOccurs を記述した element 要素は引数として使用できません。引数として使用した場合，配列の最初の要素しか取得できません。

(2)　restriction 要素を記述する場合の注意事項

- simpleType 要素の子要素として記述する場合

  simpleType 要素の子要素に restriction 要素を記述する場合には，base 属性を指定し，その子要素には enumeration 要素だけを記述してください。この組み合わせ以外では，不正なソースが生成されます。

- complexContent 要素の子要素として記述する場合

  complexContent 要素の子要素に restriction 要素を記述する場合には，base 属性を指定し，その属性値には"soapenc:Array"だけを指定できます。この組み合わせ以外では，不正なソースが生成されます。

(3) list 要素を記述する場合の注意事項

list 要素を記述する場合の注意事項を示します。

- itemType 属性を必ず指定してください。
- 子要素に simpleType 要素は記述できません。
- list 要素の itemType 属性には，list データ型は指定できません。例えば，次の例では IntListList 型は使用できません。

```
<simpleType name="IntList">
  <list itemType="xsd:int"/>
</simpleType>
<simpleType name="IntListList">
  <list itemType="intf:IntList"/>
</simpleType>
```

記述できない要素および属性を記述した場合，不正なソースが生成されます。

(4) 単純型を基準型として拡張して，複合型を定義する場合の代替記述

simpleContent 要素は非サポート要素です。単純型を拡張した複合型を定義するには，complexType 要素を使用して記述してください。

単純型を基準型として拡張して，複合型を定義する場合の修正例を示します。

<修正前（非サポート）>

- simpleContent 要素を使用した記述

```
<xsd:element name="elm1">
   <xsd:complexType>
      <xsd:simpleContent>
         <xsd:extension base="xsd:string">
            <xsd:attribute name="att1" type="xsd:int" />
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
</xsd:element>
```

- XML インスタンス

```
<elm1 att1="123">abc</elm1>
```

<修正後（代替記述）>

- complexType 要素を使用した記述

```
<xsd:element name="elm1">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="data" type="xsd:string"/>
```

```
      </xsd:sequence>
      <xsd:attribute name="att1" type="xsd:int"/>
   </xsd:complexType>
</xsd:element>
```

- XML インスタンス

```
<elm1 att1="123">
   <data>abc</data>
</elm1>
```

(5) 単純型を使用した派生によって，制限された複合型を定義する場合の代替記述

simpleContent 要素は非サポートです。complexType 要素を使用して，別の型として定義してください。

単純型を使用した派生によって，制限された複合型を定義する場合の修正例を示します。

<修正前（非サポート）>

- simpleContent 要素を使用した記述

```
<xsd:complexType name="attDataBase" >
   <xsd:simpleContent >
      <xsd:extension base="xsd:int" >
         <xsd:attribute name="currency"
                     type="xsd:string" />
         <xsd:attribute name="country"
                     type="xsd:string" />
      </xsd:extension>
   </xsd:simpleContent>
</xsd:complexType >

<xsd:element name="elm1">
   <xsd:complexType>
      <xsd:simpleContent>
         <xsd:restriction base="intf:attDataBase" >
            <xsd:attribute name="currency"
                     type="xsd:string" />
      </xsd:restriction>
      </xsd:simpleContent>
   </xsd:complexType >
</xsd:element>
```

- XML インスタンス

```
<elm1 currency="yen" >
1000
</elm1>
```

<修正後（代替記述）>

- complexType 要素を使用した記述

```
<xsd:element name="elm1" type="intf:attDataBase2"/>
<xsd:complexType name="attDataBase1" >
   <xsd:sequence>
      <xsd:element name="price" type="xsd:int"/>
   </xsd:sequence>
   <xsd:attribute name="currency"
                  type="xsd:string" />
   <xsd:attribute name="country"
                  type="xsd:string" />
</xsd:complexType >

<xsd:complexType name="attDataBase2" >
   <xsd:sequence>
      <xsd:element name="price" type="xsd:int"/>
   </xsd:sequence>
   <xsd:attribute name="currency"
                  type="xsd:string" />
</xsd:complexType >
```

- XML インスタンス

```
<elm1 currency="yen">
   <price xmlns="">1000</price>
</elm1>
```

(6) use 属性の代替記述

attribute 要素に指定する use 属性は非サポートです。attribute 要素の出現可能性を型定義によって記述する場合には，その用途によって complexType 要素を使用して，別の型として定義してください。

use 属性を使用した記述の修正例を示します。

<修正前（非サポート）>

- use 属性を使用した記述

```
<xsd:complexType name="Use1">
   <xsd:sequence>
      <xsd:element name="data1"
                type="intf:dataType" />
      <xsd:element name="data2">
         <xsd:complexType>
            <xsd:complexContent>
               <xsd:restriction
                     base="intf:dataType">
                  <xsd:sequence>
                     <xsd:element
                     name="data"
                     type="xsd:int"/>
                  </xsd:sequence>
                  <xsd:attribute
                     name="currency"
                     type="xsd:string"
                     use="prohibited" />
               </xsd:restriction>
```

```
            </xsd:complexContent>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType >
<xsd:complexType name="dataType" >
      <xsd:sequence>
            <xsd:element
                  name="data" type="xsd:int"/>
      </xsd:sequence>
      <xsd:attribute name="currency" type="xsd:string" />
</xsd:complexType >
<xsd:element name="use-1" type="intf:Use1"/>
```

- XML インスタンス

```
<use-1 xmlns="http://localhost">
   <data1 currency="yen" xmlns="" >
      <data xmlns="">1000</data>
   </data1>
   <data2 xmlns="" >
      <data xmlns="">1000</data>
   </data2>
</use-1>
```

<修正後（代替記述）>

- complexType 要素を使用した記述

```
<xsd:complexType name="Use1">
   <xsd:sequence>
      <xsd:element name="data1" type="intf:dataType" />
      <xsd:element name="data2" type="intf:dataType2" />
   </xsd:sequence>
</xsd:complexType >
<xsd:complexType name="dataType" >
   <xsd:sequence>
      <xsd:element name="data" type="xsd:int"/>
   </xsd:sequence>
   <xsd:attribute name="currency" type="xsd:string" />
</xsd:complexType >
<xsd:complexType name="dataType2" >
   <xsd:sequence>
      <xsd:element name="data" type="xsd:int"/>
   </xsd:sequence>
</xsd:complexType >
<xsd:element name="use-1" type="intf:Use1"/>
```

- XML インスタンス

  修正前と同じです。

(7) default 属性の代替記述

element 要素または attribute 要素に default 属性を指定する場合で，SOAP メッセージの送信側のアプリケーションで値を設定しないとき，default 属性の属性値は SOAP メッセージに設定されません。

SOAP メッセージの受信側のアプリケーションで，該当する属性値が SOAP メッセージに設定されていない場合は，default 属性の属性値に指定されている値を設定してください。

(8) fixed 属性の代替記述

element 要素または attribute 要素に fixed 属性を指定する場合，SOAP メッセージの送信側のアプリケーションでどのような値を設定しても，fixed 属性の属性値が SOAP メッセージに設定されません。

SOAP メッセージの受信側のアプリケーションで，fixed 属性に指定されている値を設定してください。

(9) substitutionGroup 属性の代替記述

substitutionGroup 属性は非サポートです。グローバルの element 要素に非サポートである substitutionGroup 属性を指定することはできません。

次に示す修正例を基に，代替記述について説明します。

このスキーマのインスタンスでは，トップレベルの要素"SSG"は，子要素として，data 要素を内容とする構造です。この data 要素は，dataPattern1 要素，dataPattern2 要素の代わりに使用できます。したがって，インスタンス 2，インスタンス 3 も正しい XML インスタンスとなります。Application Server では，substitutionGroup 属性をサポートしないため，代替要素となる型（ここでは，dataPattern1 要素，dataPattern2 要素）を使用する場合は，別の WSDL で記述し，別のアプリケーションとする必要があります。

<修正前（非サポート）>

- substitutionGroup 属性を使用した記述

```
<xsd:complexType name="SSG">
   <xsd:sequence>
      <xsd:element ref="intf:data" />
   </xsd:sequence>
</xsd:complexType>
<xsd:element name="data" type="intf:dataType" />
<xsd:element name="dataPattern1"
            type="intf:dataPattern1Type"
            substitutionGroup="intf:data" />
<xsd:element name="dataPattern2"
            type="intf:dataPattern2Type"
            substitutionGroup="intf:data" />
<xsd:complexType name="dataType">
   <xsd:sequence>
      <xsd:element name="pno" type="xsd:NMTOKEN" />
      <xsd:element name="date" type="xsd:string" />
      <xsd:element name="season" type="xsd:string" />
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="dataPattern1Type">
   <xsd:complexContent>
         <xsd:extension base="intf:dataType" />
   </xsd:complexContent>
```

```
</xsd:complexType>
<xsd:complexType name="dataPattern2Type">
   <xsd:complexContent>
      <xsd:extension base="intf:dataType">
         <xsd:sequence>
               <xsd:element name="on"
                     type="xsd:string" />
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
<xsd:element name="SSG" type="intf:SSG"/>
```

- XML インスタンス

```
インスタンス1
<SSG xmlns="http://localhost">
   <data>
      <pno xmlns="">1967</pno>
      <date xmlns="">Date</date>
      <season xmlns="">SPRING</season>
   </data>
</SSG>

インスタンス2
<SSG xmlns="http://localhost">
   <dataPattern1>
      <pno xmlns="">1967</pno>
      <date xmlns="">Date</date>
      <season xmlns="">SPRING</season>
   </dataPattern1>
</SSG>

インスタンス3
<SSG xmlns="http://localhost">
   <dataPattern2>
      <pno xmlns="">1967</pno>
      <date xmlns="">Date</date>
      <season xmlns="">SPRING</season>
      <on xmlns="">OFF</on>
   </dataPattern2>
</SSG>
```

<修正後（代替記述）>

- インスタンスごとに記述

```
インスタンス１のスキーマ
<xsd:complexType name="SSG">
   <xsd:sequence>
      <xsd:element ref="intf:data" />
   </xsd:sequence>
</xsd:complexType>
<xsd:element name="data" type="intf:dataType" />
<xsd:complexType name="dataType">
   <xsd:sequence>
      <xsd:element name="pno" type="xsd:NMTOKEN" />
```

```
      <xsd:element name="date" type="xsd:string" />
      <xsd:element name="season" type="xsd:string" />
   </xsd:sequence>
</xsd:complexType>
<xsd:element name="SSG" type="intf:SSG"/>

インスタンス２のスキーマ
<xsd:complexType name="SSG">
   <xsd:sequence>
      <xsd:element ref="intf:dataPattern1Type" />
   </xsd:sequence>
</xsd:complexType>
<xsd:element name="dataPattern1" type="intf:dataType" />
<xsd:complexType name="dataType">
   <xsd:sequence>
      <xsd:element name="pno" type="xsd:NMTOKEN" />
      <xsd:element name="date" type="xsd:string" />
      <xsd:element name="season" type="xsd:string" />
   </xsd:sequence>
</xsd:complexType>
<xsd:element name="SSG" type="intf:SSG"/>

インスタンス３のスキーマ
<xsd:complexType name="SSG">
   <xsd:sequence>
      <xsd:element ref="intf:dataPattern2" />
   </xsd:sequence>
</xsd:complexType>
<xsd:element name="dataPattern2"
            type="intf: dataPattern2Type" />
<xsd:complexType name="dataPattern2Type">
   <xsd:sequence>
      <xsd:element name="pno" type="xsd:NMTOKEN" />
      <xsd:element name="date" type="xsd:string" />
      <xsd:element name="season" type="xsd:string" />
      <xsd:element name="on" type="xsd:string" />
   </xsd:sequence>
</xsd:complexType>
<xsd:element name="SSG" type="intf:SSG"/>
```

- XML インスタンス
  修正前と同じです。

(10) mixed 属性の代替記述

mixed 属性は非サポートです。complexType 要素，complexContent 要素に mixed 属性で実現する，内容の中に要素が入る構造は実現できません。SOAP メッセージで送信しなければならないデータを，それぞれ独立した要素となる型を作成してください。

mixed 属性を使用した記述の修正例を示します。

＜修正前（非サポート）＞

- mixed 属性を使用した記述

```
<xsd:complexType name="mixed">
   <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="address">
         <xsd:complexType>
            <xsd:complexContent mixed="true" >
               <xsd:extension base="intf:addressType">
                  <xsd:sequence>
                     <xsd:element name="postcode" type="xsd:int"/>
                  </xsd:sequence>
               </xsd:extension>
            </xsd:complexContent>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="addressType" mixed="true" >
   <xsd:sequence>
      <xsd:element name="street" type="xsd:string"/>
      <xsd:element name="city" type="xsd:string"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:element name="mixed" type="intf:mixed"/>
```

- XML インスタンス

```
<mixed xmlns="http://localhost">
   <name xmlns="" >abcdef</name>
   <address xmlns="" >
      abcde<street>XXXX51st.</street>fghijk<city>ZZZZZZ</city>lmnopqrstuvwxyz <postcode>0
45981</postcode>
   </address>
</mixed>
```

＜修正後（代替記述）＞

- 独立した要素となる型を作成

```
<xsd:complexType name="mixed">
   <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="address">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="data1" type="xsd:string"/>
               <xsd:element name="street" type="xsd:string"/>
               <xsd:element name="data2" type="xsd:string"/>
               <xsd:element name="city" type="xsd:string"/>
               <xsd:element name="data3" type="xsd:string"/>
               <xsd:element name="postcode" type="xsd:int"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
   </xsd:sequence>
</xsd:complexType>
<xsd:element name="mixed" type="intf:mixed"/>
```

- XML インスタンス

```
<mixed xmlns="http://localhost">
   <name xmlns="" >abcdef</name>
   <address xmlns="" >
      <data1>abcde</data1>
      <street>XXXX51st.</street>
      <data2>fghijk</data2>
      <city>ZZZZZZ</city>
      <data3>lmnopqrstuvwxyz</data3>
      <postcode>045981</postcode>
   </address>
</mixed>
```

(11) XML Schema のビルトインデータ型使用時の注意事項

ENTITY 型を指定してアプリケーションでデータを送受信できますが，ENTITY 型としての機能は有効になりません。

(12) soapenc:Array 型の制限に関する注意事項

WSDL の use 属性が literal の場合，soapenc:Array 型を制限した複合型を使用することはできません。配列を定義するには，代わりに maxOccurs 属性を指定した要素を子要素に持つ複合型を定義してください。

(13) wsdl:arrayType 属性を記述する場合の注意事項

wsdl:arrayType 属性を記述した場合の配列は，OUT 属性および INOUT 属性で使用できません。

(14) ref 属性，minOccurs 属性，および minOccurs 属性を使用する場合の代替記述

ref 属性，minOccurs 属性，および maxOccurs 属性を同じ element 要素で使用した場合，不正なソースが生成されます。ref 属性，minOccurs 属性，および maxOccurs 属性を同じ element 要素で使用しないでください。

ref 属性，minOccurs 属性，および maxOccurs 属性を使用する場合の修正例を示します。

<修正前（非サポート）>

- ref 属性，minOccurs 属性，および maxOccurs 属性を同じ element 要素で使用した記述

```
<xsd:complexType name="getUserData">
   <xsd:sequence>
      <xsd:element ref="intf:in0"
          minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:element name="in0" type="xsd:string"/>
```

<修正後（代替記述）>

- ref 属性を使用しない記述

```
<xsd:complexType name="getUserData">
   <xsd:sequence>
      <xsd:element name="in0" type="xsd:string"
          minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
```

(15) sequence 要素に指定する属性に関する注意

sequence 要素の属性に，minOccurs 属性および maxOccurs 属性は使用しないでください。

minOccurs 属性，maxOccurs 属性のどちらか，または両方を指定した場合，WSDL2Java の WSDL 検証機能でエラーにならないで，ソースコードが生成されます。

(16) xsd:enumeration 要素を使用した場合の注意

クライアントアプリケーションまたはサーバアプリケーションでは，xsd:enumeration 要素を使用して列挙型として定義した値だけを指定してください。

xsd:enumeration 要素を使用して定義した列挙型の値以外の値をクライアントアプリケーション，またはサーバアプリケーションで指定した場合，スキーマに違反した SOAP メッセージが送信されます。このとき，エラーになりません。

(17) xsd:string 型の xsd:list 型で空文字列の要素を含む配列を送信した場合の注意

RPC 形態のアプリケーションの場合，itemType が xsd:string 型の xsd:list 型で，空文字列の要素を含む配列を送信すると，受信時に配列の要素から空文字が削除されます。

(18) xsd:schema 要素の targetNamespace 属性に関する注意

WSDL2Java コマンド実行時に，targetNamespace 属性を省略した xsd:schema 要素を記述している WSDL ファイルを指定しても，エラーになりません。WSDL2Java コマンドで指定する WSDL ファイル内で xsd:schema 要素を記述する場合は，targetNamespace 属性を持つ xsd:schema 要素を記述してください。

(19) 複数の xsd:schema 要素を記述する場合の注意

WSDL2Java コマンド実行時に，同じ targetNamespace 属性値を持つ xsd:schema 要素を wsdl:types 要素以下に複数記述している WSDL ファイルを指定しても，エラーになりません。WSDL2Java コマンドで指定する WSDL ファイル内で wsdl:types 要素以下に複数の xsd:schema 要素を記述する場合は，それぞれ異なる targetNamespace 属性値を持つ xsd:schema 要素を記述してください。

(20) element 要素に nillable 属性を指定していない場合の注意

nillable 属性を指定していない element 要素に対してアプリケーションで値を設定しないと，要素に xsi：nil="true" 属性が設定された空要素が送信され，不正な SOAP メッセージになります。

次の条件がすべて重なる場合に問題が発生します。なお，element 要素の型は問いません。

- element 要素に nillable 属性が属性値"false"で指定されている，または nillable 属性が指定されていない。
- minOccurs 属性の属性値に"1"以上が設定されている（minOccurs 属性の省略時も含む）。
- アプリケーションで該当する要素のメンバに値を設定していない。

XML Schema の指定で空要素指定がされていない要素には，アプリケーションで必ず値を設定してください。

12.3 SAAJ 1.2 との対応

SAAJ 1.2 の javax.xml.soap パッケージのうち，SOAP アプリケーションの開発に利用できるインタフェースおよびクラスを次の表に示します。

表 12–14 SAAJ1.2 のサポート範囲

分類	インタフェース名／クラス名	メソッド名	サポート
インタフェース	Detail	すべてのメソッド	○
	DetailEntry	すべてのメソッド	○
	Name	すべてのメソッド	○
	Node	すべてのメソッド	○
	SOAPBody	すべてのメソッド	○
	SOAPBodyElement	すべてのメソッド	○
	SOAPConstants	すべてのメソッド	○
	SOAPElement	すべてのメソッド	○
	SOAPEnvelope	すべてのメソッド	○
	SOAPFault	すべてのメソッド	○
	SOAPFaultElement	すべてのメソッド	○
	SOAPHeader	すべてのメソッド	○
	SOAPHeaderElement	すべてのメソッド	○
	Text	すべてのメソッド	○
クラス	AttachmentPart	setContentType(java.lang.String contentType)※1	○
		setDataHandler(javax.activation.DataHandler dataHandler)※1	○
		上記以外のメソッド	○
	MessageFactory	すべてのメソッド	○
	MimeHeader	すべてのメソッド	○
	MimeHeaders	すべてのメソッド	○
	SOAPConnection	すべてのメソッド	○
	SOAPConnectionFactory	すべてのメソッド	○
	SOAPElementFactory	すべてのメソッド	×
	SOAPException	すべてのメソッド	○
	SOAPFactory	すべてのメソッド	○
	SOAPMessage	addAttachmentPart(Attachment AttachmentPart)※2	○

分類	インタフェース名／クラス名	メソッド名	サポート
		createAttachmentPart(javax.activation.DataHandler dataHandler)※1	○
		createAttachmentPart(java.lang.Object content, java.lang.String contentType)※1	○
		上記以外のメソッド	○
	SOAPPart	すべてのメソッド※3	○

（凡例）

○：サポートされます。

×：サポートされません。

注※1

添付ファイルが添付できるファイルサイズの上限を超えた場合，IllegalArgumentException をスローします。

注※2

添付ファイルの数が追加できる上限を超えた場合，IllegalArgumentException をスローします。

注※3

DOM 系の API でサポートしていないメソッド（UnsupportedOperationException 例外が発生するメソッド）があります。該当するメソッドを次に示します。

- getEncoding()
- setEncoding(java.lang.String)
- getStandalone()
- setStandalone(boolean)
- getStrictErrorChecking()
- setStrictErrorChecking(boolean)
- getVersion()
- getVersion(java.lang.String)
- adoptNode(org.w3c.dom.Node)

12.4　JAXR 1.0 との対応

UDDI クライアントライブラリでは JAXR 1.0 Capability Profile Level 0 の API をサポートします。次にサポート範囲を示します。

表 12–15　javax.xml.registry パッケージ

項番	インタフェース/クラス名	メソッド名	サポート	Capability Profile
1	BulkResponse	すべてのメソッド	○	Level 0
2	BusinessLifeCycleManager	すべてのメソッド	○	Level 0
3	BusinessQueryManager	findRegistryPackages	×	Level 1
		上記以外のメソッド	○	Level 0
4	CapabilityProfile	すべてのメソッド	○	Level 0
5	Connection	すべてのメソッド	○	Level 0
6	ConnectionFactory	createFederatedConnection	×	Level 0(Optional)
		上記以外のメソッド	○	Level 0
7	DeclarativeQueryManager	すべてのメソッド	×	Level 1
8	FederatedConnection	すべてのメソッド	×	Level 0(Optional)
9	FindQualifier	メソッドなし	−	−
10	JAXRResponse	すべてのメソッド	○	Level 0
11	LifeCycleManager	createExtrinsicObject createPersonName createRegistryPackage deprecateObjects unDeprecateObjects deleteObjects(keys)	×	Level 1
		上記以外のメソッド	○	Level 0
12	Query	すべてのメソッド	×	Level 1
13	QueryManager	getRegistryObject(id) getRegistryObjects(objectKeys)	×	Level 1
		上記以外のメソッド	○	Level 0
14	RegistryService	getDeclarativeQueryManager	×	Level 1
		上記以外のメソッド	○	Level 0

（凡例）

○：サポートされます。

×：サポートされません。

−：対象外です。

表 12–16　javax.xml.registry.infomodel パッケージ

項番	インタフェース/クラス名	メソッド名	サポート	Capability Profile
1	Association	すべてのメソッド	○	Level 0
2	AuditableEvent	すべてのメソッド	×	Level 1
3	Classification	すべてのメソッド	○	Level 0
4	ClassificationScheme	getValueType setValueType	×	Level 1
		上記以外のメソッド	○	Level 0
5	Concept	すべてのメソッド	○	Level 0
6	EmailAddress	すべてのメソッド	○	Level 0
7	ExtensibleObject	すべてのメソッド	○	Level 0
8	ExternalIdentifier	すべてのメソッド	○	Level 0
9	ExternalLink	すべてのメソッド	○	Level 0
10	ExtrinsicObject	すべてのメソッド	×	Level 1
11	InternationalString	すべてのメソッド	○	Level 0
12	Key	すべてのメソッド	○	Level 0
13	LocalizedString	すべてのメソッド	○	Level 0
14	Organization	getPostalAddress setPostalAddress addChildOrganization removeChildOrganization getChildOrganizationCount getChildOrganizations getDescendantOrganizations getParentOrganization getRootOrganization	×	Level 1
		上記以外のメソッド	○	Level 0
15	PersonName	getLastName setLastName getFirstName setFirstName getMiddleName setMiddleName	×	Level 1
		上記以外のメソッド	○	Level 0
16	PostalAddress	すべてのメソッド	○	Level 0
17	RegistryEntry	すべてのメソッド	×	Level 1

項番	インタフェース/クラス名	メソッド名	サポート	Capability Profile
18	RegistryObject	toXML	×	Level 0
		getAuditTrail getAssociatedObjects getObjectType getRegistryPackages	×	Level 1
		上記以外のメソッド	○	Level 0
19	RegistryPackage	すべてのメソッド	×	Level 1
20	Service	すべてのメソッド	○	Level 0
21	ServiceBinding	すべてのメソッド	○	Level 0
22	Slot	すべてのメソッド	○	Level 0
23	SpecificationLink	すべてのメソッド	○	Level 0
24	TelephoneNumber	getCountryCode getAreaCode getExtension getUrl setCountryCode setAreaCode setExtension setUrl	×	Level 1
		上記以外のメソッド	○	Level 0
25	URIValidator	すべてのメソッド	○	Level 0
26	User	getUrl setUrl	×	Level 1
		上記以外のメソッド	○	Level 0
27	Versionable	すべてのメソッド	×	Level 1

（凡例）

○：サポートされます。

×：サポートされません。

注意事項

例外インタフェースのメソッドについてはすべてサポートされています。

12.5　WS-I Attachments Profile - Version 1.0 との対応

RPC 形態の添付ファイルは，WS-I Attachments Profile - Version 1.0 仕様に準拠します。RPC 形態の添付ファイルでサポートする，WS-I Attachments Profile - Version 1.0 のデータ型の範囲を次に示します。

表 12–17　WS-I Attachments Profile - Version 1.0 のデータ型のサポート範囲

項番	Attachments Profile のデータ型	データ型	データ型の列挙	attribute 要素	list 要素
1	wsi:swaRef	○	×	×	×

（凡例）

○：サポートされます。

×：サポートされません。

注意事項

- wsi:swaRef を利用できるのは，document/literal を使用した場合だけです。
- wsi:swaRef を利用する場合，RPC 形態のクライアント処理の実装方法には，スタブを使用してください。DII は使用できません。
- .NET Framework を使用した場合は，wsi:swaRef を利用できません。

13 SOAP 通信基盤が提供する API

この章では，SOAP 通信基盤が提供する API について説明します。

13.1　インタフェースおよびクラスの一覧

SOAP 通信基盤はインタフェースおよびクラスを提供します。これらのインタフェースおよびクラスは，使用する JAR ファイルによって使用できるものと使用できないものとに分類されます。JAR ファイルごとのインタフェースおよびクラスの対応を次の表に示します。なお，スタブクラスが提供するインタフェースについては，「3.7.1　スタブの使用」を参照してください。

表 13–1　JAR ファイルに対するインタフェースおよびクラスの対応

インタフェース名およびクラス名	説明
C4Fault	SOAP Fault の情報を保持するクラスです。
C4Property	実行時オプションを設定するクラスです。
C4QName	修飾された名前を保持するクラスです。
C4Session	セッション管理機能を提供するクラスです。
Call（com.cosminexus.cws.xml.rpc.Call）	サービスの呼び出しに必要な情報を取得するためのインタフェースです。
Call（javax.xml.rpc.Call）	サービスを呼び出すための機能を提供するインタフェースです。
ClientID	クライアント識別子を表すクラスです。
JAXRPCException	JAX-RPC ランタイムに関連した例外を表す例外クラスです。
Management	RPC 形態，メッセージング形態の各クラスを使用する前，および終了時に呼び出すクラスです。
ReqResListener	リクエスト・レスポンス型の SOAP アプリケーションが実装するメッセージングインタフェースです。
Service	Call インスタンスを生成するためのファクトリインタフェースです。
ServiceException	Service クラスおよび ServiceFactory クラスのメソッドによってスローされる例外クラスです。
ServiceFactory	Service インスタンスを生成するためのファクトリクラスです。

注意事項

- このマニュアルに記載していないクラスおよびメソッドは使用しないでください。使用した場合，動作は保証できません。
- SOAP 通信基盤が提供するクラスのうち，次のクラスのインスタンスはマルチスレッドで共有した場合，動作は保証できません。
  - C4Session クラス
  - C4QName クラス
  - ClientID クラス
  - C4Fault クラス

- Call インタフェース
- Service インタフェース

13.2　C4Fault クラス（SOAP Fault 情報の保持）

SOAP Fault の情報を保持するクラスです。

クラス定義

```
public final class C4Fault
extends java.rmi.RemoteException
```

パッケージ

```
com.cosminexus.cws.service.exception
```

C4Fault クラスのメソッドを次の表に示します。

表 13-2　C4Fault クラスのメソッド一覧

メソッド	機能概要
C4Fault	C4Fault オブジェクトを生成します。
getFaultActor	Fault 生成者（SOAP Fault の faultactor）を取得します。
getFaultCode	Fault のコード（SOAP Fault の faultcode）を取得します。
getFaultDetails	Fault の詳細（SOAP Fault の detail）を取得します。
getFaultString	Fault の文字列（SOAP Fault の faultstring）を取得します。

C4Fault

クラス名：C4Fault

機能

C4Fault オブジェクトを生成するコンストラクタです。

構文

```
public C4Fault(
     com.cosminexus.cws.service.common.C4QName    a_c4QName,
     java.lang.String                             a_strFaultString,
     java.lang.String                             a_strFaultActor,
     org.w3c.dom.Element[]                        a_elmDetail
)
```

引数

表 13-3　C4Fault メソッドの引数

仮引数名	名称	in/out	説明
a_c4QName	Fault コード	in	生成する C4Fault オブジェクトに設定する Fault コード（SOAP Fault の faultcode）を指定します。Fault コードを設定しない場合は null を指定します。 なお，Fault コードに null を指定して C4Fault オブジェクトを生成した場合，getFaultCode メソッドを呼び出すと，ローカル部と接頭辞が空文字の C4QName オブジェクトが返ります。
a_strFaultString	Fault 文字列	in	生成する C4Fault オブジェクトに設定する Fault 文字列（SOAP Fault の faultstring）を指定します。Fault 文字列を設定しない場合は null を指定します。C4Fault オブジェクトを生成したあと，getFaultString メソッドを呼び出した場合，KDCCP9000-E の＜詳細＞に Fault 文字列を設定して返します。また，Fault 文字列に null を指定した場合には，＜詳細＞は空文字になります。
a_strFaultActor	Fault 生成者	in	生成する C4Fault オブジェクトに設定する Fault 生成者（SOAP Fault の faultactor）を指定します。Fault 生成者を設定しない場合は null を指定します。
a_elmDetail	Fault 詳細	in	生成する C4Fault オブジェクトに設定する Fault 詳細（SOAP Fault の detail）を org.w3c.dom.Element の配列で指定します。Fault 詳細を設定しない場合は null を指定します。要素数が 0 の配列を指定した場合は，Fault 詳細に null を指定した場合と同じです。なお，配列の要素内に null の要素が存在する場合，その要素に対応する SOAP Fault メッセージの Fault 詳細は作成されません。

getFaultActor

クラス名：C4Fault

機能

Fault 生成者（SOAP Fault の faultactor）を取得します。

構文

```
public java.lang.String getFaultActor()
```

引数

ありません。

戻り値

Fault 生成者を格納する String オブジェクトを返します。Fault 生成者が設定されていない場合は null を返します。

getFaultCode

クラス名：C4Fault

機能

Fault のコード（SOAP Fault の faultcode）を取得します。

構文

```
public com.cosminexus.cws.service.common.C4QName getFaultCode()
```

引数

ありません。

戻り値

Fault コードを格納した C4QName オブジェクトを返します。

注意事項

Fault コードに null を指定して C4Fault オブジェクトを生成した場合，ローカル部と接頭辞が空文字の C4QName オブジェクトが返されます。

getFaultDetails

クラス名：C4Fault

機能

Fault の詳細（SOAP Fault の detail）を取得します。

構文

```
public org.w3c.dom.Element[] getFaultDetails()
```

引数

ありません。

戻り値

Fault 詳細を org.w3c.dom.Element 型の配列で返します。Fault 詳細がない場合は null を返します。

注意事項

- C4Fault オブジェクト生成した場合に，要素数が 0 の配列の Fault 詳細を指定した場合は，null が返されます。
- null の要素を含む Fault 詳細を指定した場合は，このメソッドを呼び出す個所によって，取得する情報が異なります。

  ・MessageService インタフェースクラスの onMessage メソッドを実装した SOAP サービスから，SOAP エンジンに制御が戻るまでにこのメソッドを呼び出した場合は，C4Fault オブジェクト生成時に指定した，Fault 詳細の null の要素はそのまま取得されます。

・SOAP エンジンから SOAP Fault メッセージがクライアントに返され，クライアントでこのメソッドを呼び出した場合は，C4Fault オブジェクト生成時に指定した，Fault 詳細の null の要素は取得されません。

- detail 要素の子ノードにテキストノードが存在する場合，戻り値の Element 型配列を構成する最後の一つの要素として null が返されます。ただし，テキストノードが空白文字，改行文字，およびタブだけで構成されている場合は，null は返されません。
- detail entry のサブツリーにテキストノードが存在する場合，兄弟ノードのすべてのテキストノードが連結されます。連結される文字列の前後に存在する空白文字，改行文字，およびタブは省略され，そのテキストノードは同じ階層の最後に追加されます。ただし，テキストノードが空白文字，改行文字，およびタブだけで構成されている場合は，テキストノードは追加されません。

  例を次に示します。

  ・受信電文の detail 要素

  <detail><ns1:Test1 xmlns:ns1="http://test.cosminexus.com/">△△foo△△
  <TestData1>TESTDATA1</TestData1>△△bar△△
  </ns1:Test1></detail>

  ・getFaultDetails メソッドの戻り値

  Element[0]：<ns1:Test1 xmlns:ns1="http://
  test.cosminexus.com/"><TestData1>TESTDATA1</TestData1>foo△△△△bar</
  ns1:Test1>

  注　△は半角スペースを表します。

getFaultString

クラス名：C4Fault

機能

Fault の文字列（SOAP Fault の faultstring）を取得します。

構文

```
public java.lang.String getFaultString()
```

引数

ありません。

戻り値

Fault 文字列を格納した String オブジェクトを返します。ユーザ実装の SOAP アプリケーションから C4Fault 例外をスローした場合は，KDCCP9000-E のメッセージテキストを格納した String オブジェクトを返します。このメッセージの＜詳細＞は，C4Fault オブジェクト生成時に指定した Fault 文字列が設定されます。

注意事項

Fault 文字列に null を指定して C4Fault オブジェクトを生成した場合，KDCCP9000-E の＜詳細＞は空文字になります。

13.3　C4Property クラス（実行時オプションの設定）

SOAP エンジンの実行時オプションを，SOAP クライアントライブラリおよび SOAP サーバの処理の実行中に動的に切り替えるためのクラスです。

クラス定義

```
public final class C4Property
extends java.lang.Object
```

パッケージ

```
com.cosminexus.cws.property
```

C4Property クラスのメソッドを次の表に示します。

表 13-4　C4Property クラスのメソッド一覧

メソッド	機能概要
getInstance	プロパティクラスのインスタンスを取得します。
getProperty	実行時オプションを取得します。
setProperty	実行時オプションを設定します。

getInstance

クラス名：C4Property

機能

C4Property クラスのインスタンスを取得します。

構文

```
public static com.cosminexus.cws.property.C4Property getInstance()
```

引数

ありません。

戻り値

C4Property クラスのインスタンスを返します。

getProperty

クラス名：C4Property

機能

実行時オプションを取得します。

構文

```
public java.lang.String getProperty(
     java.lang.String key
)
```

引数

表 13-5　getProperty メソッドの引数

仮引数名	名称	in/out	説明
key	キー名称	in	取得するキー名称を指定します。取得するキー名称は，setProperty メソッドで指定するキー名称と同じものを指定します。

戻り値

オプションの値を示す文字列を返します。

注意事項

- このメソッドで取得するオプション値を setProperty メソッドで指定していない場合は，動作定義ファイル中の値を返します。動作定義ファイル中に設定していない場合は，次のように動作します。

  次のキーを指定した場合は null を返します。

  ・C4Property.CONFIGURATION_KEY_SOCKET_WRITE_TIMEOUT

  ・C4Property.CONFIGURATION_KEY_SOCKET_READ_TIMEOUT

  ・C4Property.CONFIGURATION_KEY_SOCKET_CONNECT_TIMEOUT

  上記以外のキーを指定した場合はデフォルト値を返します。デフォルト値がない場合は null を返します。
- 実行時オプション以外のキーを指定した場合で，setProperty メソッドで設定していないキーを指定した場合は null を返します。
- SOAP クライアントライブラリの実行中にこのメソッドを使用する場合は，このメソッドを使用する前に，Management クラスの connectClientIDtoCurrentThread メソッドでクライアント識別子とスレッドを関連づけてください。connectClientIDtoCurrentThread メソッドが呼び出されていない場合にこのメソッドを呼び出した場合，null を返す場合があります。SOAP サーバの処理の実行中にこのメソッドを使用する場合は，connectClientIDtoCurrentThread メソッドを呼び出す必要はありません。

setProperty

クラス名：C4Property

機能

実行時オプションを設定します。

構文

```
public void setProperty(java.lang.String name,
                        java.lang.Object value)
```

引数

表 13-6　setProperty メソッドの引数

仮引数名	名称	in/out	説明
key	キー名称	in	設定するキー名称を指定します。
value	値	in	設定する値を文字列で指定します。

setProperty メソッドで指定するキー名称および値を次に示します。

表 13-7　setProperty メソッドで指定するキー名称および値

メソッドで指定するキー名称	動作定義ファイルのキー名称	メソッドで指定する値
C4Property.CONFIGURATION_KEY_PROXYHOST	c4web.application.proxy_host（クライアント定義ファイル） c4web.application.<識別子>.proxy_host（サーバ定義ファイル）	ホスト名（または IP アドレス）を示す文字列を指定します。
C4Property.CONFIGURATION_KEY_NONPROXYHOSTS	c4web.application.non_proxy_hosts（クライアント定義ファイル） c4web.application.<識別子>.non_proxy_hosts（サーバ定義ファイル）	ホスト名（または IP アドレス，以降同様）群を示す文字列を指定します。複数のホスト名を指定する場合は「|」（ストローク）で区切って指定します。 プロキシサーバを利用しないホスト名を指定する場合，ホスト名とホスト名の間には，「|」（ストローク）以外の文字（空白など）を指定しないようにしてください。
C4Property.CONFIGURATION_KEY_PROXYPORT	c4web.application.proxy_port（クライアント定義ファイル） c4web.application.<識別子>.proxy_port（サーバ定義ファイル）	ポート番号を示す文字列を指定します。
C4Property.CONFIGURATION_KEY_PROXYUSER	c4web.application.proxy_user（クライアント定義ファイル） c4web.application.<識別子>.proxy_user（サーバ定義ファイル）	認証ユーザ ID を示す文字列を指定します。
C4Property.CONFIGURATION_KEY_PROXYPASSWORD	c4web.application.proxy_password（クライアント定義ファイル） c4web.application.<識別子>.proxy_password（サーバ定義ファイル）	認証ユーザ ID に対応するパスワード文字列を指定します。
C4Property.CONFIGURATION_KEY_MAINTAINSESSION	c4web.application.app_maintainsession（クライアント定義ファイル）	「true」または「false」を指定します。

メソッドで指定するキー名称	動作定義ファイルのキー名称	メソッドで指定する値
	c4web.application.<識別子>.app_maintainsession（サーバ定義ファイル）	
C4Property.CONFIGURATION_KEY_DOMULTIREFS	c4web.common.do_multirefs（クライアント定義ファイル） c4web.common.<識別子>.do_multirefs（サーバ定義ファイル）	「true」または「false」を指定します。
C4Property.CONFIGURATION_KEY_SENDXSITYPES	c4web.common.send_xsi_types（クライアント定義ファイル） c4web.common.<識別子>.send_xsi_types（サーバ定義ファイル）	「true」または「false」を指定します。
C4Property.CONFIGURATION_KEY_ENABLESOAPHEADERCHECK	c4web.common.enable_soapheader_check（クライアント定義ファイル） c4web.common.<識別子>.enable_soapheader_check（サーバ定義ファイル）	「true」または「false」を指定します。
C4Property.CONFIGURATION_KEY_SOCKET_WRITE_TIMEOUT	c4web.application.socket_write_timeout（クライアント定義ファイル） c4web.application.<識別子>.socket_write_timeout（サーバ定義ファイル）	ソケットの書き込みタイムアウト値を指定します。
C4Property.CONFIGURATION_KEY_SOCKET_READ_TIMEOUT	c4web.application.socket_read_timeout（クライアント定義ファイル） c4web.application.<識別子>.socket_read_timeout（サーバ定義ファイル）	ソケットの読み込みタイムアウト値を指定します。
C4Property.CONFIGURATION_KEY_SOCKET_CONNECT_TIMEOUT	c4web.application.socket_connect_timeout（クライアント定義ファイル） c4web.application.<識別子>.socket_connect_timeout（サーバ定義ファイル）	ソケットの接続タイムアウト値を指定します。
C4Property.CONFIGURATION_KEY_CHARACTERREFERENCE	c4web.common.character_reference（クライアント定義ファイル） c4web.common.<識別子>.character_reference（サーバ定義ファイル）	「true」または「false」を指定します。

実行時オプションとして設定する各オプションの意味については，「10.6　実行時オプションの設定項目」を参照してください。

戻り値

ありません。

注意事項

- このメソッドでは，クライアント定義ファイルおよびサーバ定義ファイルの設定項目のうち，トレースファイルに関する項目（キー名称の先頭に c4web.logger が付くもの）の設定はできません。
- 「true」または「false」の値を指定するキーでは，すべて小文字で記述してください。すべて大文字の場合，または大文字と小文字が混在した場合は，指定が無効となり，それぞれのキーのデフォルト値が仮定されます。
- SOAP クライアントライブラリの実行中にこのメソッドを使用する場合は，このメソッドを使用する前に，Management クラスの connectClientIDtoCurrentThread メソッドでクライアント識別子とスレッドを関連づけてください。connectClientIDtoCurrentThread メソッドが呼び出されていない場合，このメソッドを呼び出しても無視されます。SOAP サーバの処理の実行中にこのメソッドを使用する場合は，connectClientIDtoCurrentThread メソッドを呼び出す必要はありません。
- 仮引数 value に null を指定した場合，すでに設定している実行時オプションの値は変更されません。

13.4　C4QName クラス（名前空間の保持）

修飾された名前を保持するクラスです。

クラス定義

```
public final class C4QName
extends javax.xml.namespace.QName
```

パッケージ

```
com.cosminexus.cws.service.common
```

C4QName クラスのメソッドを次の表に示します。

表 13-8　C4QName クラスのメソッド一覧

メソッド	機能概要
C4QName	C4QName オブジェクトを生成します。
equals	引数で指定されたオブジェクトと等しいか比較します。
getLocalPart	修飾された名前のローカル部を取得します。
getNamespaceURI	修飾された名前の接頭辞を取得します。
hashCode	保持している修飾された名前のハッシュコード値を取得します。
toString	保持している修飾された名前の文字列表現を返します。

C4QName

クラス名：C4QName

機能

C4QName オブジェクトを生成するコンストラクタです。

構文

- ローカル部だけを指定する場合

```
public C4QName(
java.lang.String a_strLocalPart
)
```

- 接頭辞およびローカル部を指定する場合

```
public C4QName(
java.lang.String a_strNamespaceURI,
java.lang.String a_strLocalPart
```

)

引数

表 13-9　C4QName メソッドの引数

仮引数名	名称	in/out	説明
a_strNamespaceURI	接頭辞	in	修飾された名前の接頭辞を指定します。
a_strLocalPart	ローカル部	in	修飾された名前のローカル部を指定します。

注意事項

このクラスのインスタンスは，マルチスレッドで共有できません。

equals

クラス名：C4QName

機能

引数で指定されたオブジェクトと等しいか比較します。

構文

```
public boolean equals(
     java.lang.Object a_oObject
)
```

引数

表 13-10　equals メソッドの引数

仮引数名	名称	in/out	説明
a_oObject	比較対象オブジェクト	in	比較するオブジェクトを指定します。

戻り値

引数に指定されたオブジェクトと等しければ true を，異なれば false を返します。

getLocalPart

クラス名：C4QName

機能

修飾された名前のローカル部を取得します。

構文

```
public java.lang.String getLocalPart()
```

引数

ありません。

戻り値

修飾された名前のローカル部を返します。

getNamespaceURI

クラス名：C4QName

機能

修飾された名前の接頭辞を取得します。

構文

```
public java.lang.String getNamespaceURI()
```

引数

ありません。

戻り値

修飾された名前の接頭辞を返します。ローカル部だけを設定している場合は，空文字を返します。

hashCode

クラス名：C4QName

機能

保持している修飾された名前のハッシュコード値を取得します。

構文

```
public int hashCode()
```

引数

ありません。

戻り値

このオブジェクトが保持している修飾された名前のハッシュコード値を返します。

toString

クラス名：C4QName

機能

保持している修飾された名前の文字列表現を返します。

構文

```
public java.lang.String toString()
```

引数

ありません。

戻り値

保持している修飾された名前の文字列表現を返します。接頭辞とローカル部が設定されている場合は「{接頭辞}ローカル部」の形式で返し，ローカル部だけが設定されている場合は「ローカル部」を返します。

13.5　C4Session クラス（セッションの管理）

セッションの管理に関する機能を提供するクラスです。

クラス定義

```
public final class C4Session extends java.lang.Object
```

パッケージ

```
com.cosminexus.cws.service.common
```

C4Session クラスのメソッドを次の表に示します。

表 13-11　C4Session クラスのメソッド一覧

メソッド	機能概要
getInstance	C4Session インスタンスを取得します。
invalidate	現在継続中のセッションの終了を指示します。

getInstance

クラス名：C4Session

機能

C4Session インスタンスを取得します。

構文

```
public static C4Session getInstance()
        throws java.lang.UnsupportedOperationException
```

引数

ありません。

戻り値

RPC 形態の SOAP アプリケーションのサービスで，DeployScope が「Session」の場合に，次の値を返します。

- 現在継続中のセッションがあれば C4Session インスタンス
- 現在継続中のセッションがなければ null

例外

次のどれかの場合に java.lang.UnsupportedOperationException が発生します。

- クライアントから呼び出された場合
- RPC 形態の SOAP アプリケーションのサービスから呼び出された場合

- RPC 形態の SOAP アプリケーションのサービスで，DeployScope が「Session」以外のサービスから呼び出された場合

invalidate

クラス名：C4Session

機能

現在継続中のセッションの終了を指示します。

構文

```
public void invalidate()
        throws java.lang.IllegalStateException
```

引数

ありません。

戻り値

ありません。

例外

すでに無効化されているセッションに対して実行した場合に java.lang.IllegalStateException が発生します。

13.6　Call インタフェース（サービス呼び出しの情報取得）

サービスの呼び出しに必要な情報を取得するためのインタフェースです。

インタフェース定義

```
public interface Call
```

パッケージ

```
com.cosminexus.cws.xml.rpc.Call
```

Call インタフェースのメソッドを次の表に示します。

表 13-12　Call インタフェース（com.cosminexus.cws.xml.rpc.Call）のメソッド一覧

メソッド	機能概要
getParameterClassByName	パラメタの Java 型を取得します。

getParameterClassByName

インタフェース名：Call

機能

パラメタの Java 型を取得します。

構文

```
public java.lang.Class getParameterClassByName(
                                        java.lang.String paramName)
                                  throws IllegalArgumentException
```

引数

表 13-13　getParameterClassByName メソッドの引数

仮引数名	名称	in/out	説明
paramName	パラメタのローカル名	in	パラメタのローカル名を指定します。

戻り値

パラメタの Java 型（java.lang.Class）を返します。存在しない場合は null を返します。

例外

paramName が null の場合，IllegalArgumentException 例外がスローされます。

13.7　Call インタフェース（サービスの呼び出し）

サービスを呼び出すための機能を提供するインタフェースです。

インタフェース定義

```
public interface Call
```

パッケージ

```
javax.xml.rpc.Call
```

Call インタフェースのメソッドを次の表に示します。

表 13-14　Call インタフェース（javax.xml.rpc.Call）のメソッド一覧

メソッド	機能概要
getOutputParams	Call オブジェクトを使用して，最後に呼び出されたオペレーションの出力パラメタをキー:name，値:value の Map で返します。
getOutputValues	最後に呼び出されたオペレーションの出力パラメタの値を List で返します。
getProperty	指定した名前のプロパティ値を取得します。
invoke	Synchronous request-response モードでオペレーションを呼び出します。
removeProperty	指定された名前のプロパティを削除します。
setProperty	プロパティを設定します。

getOutputParams

インタフェース名：Call

機能

Call オブジェクトを使用して，最後に呼び出されたオペレーションの出力パラメタをキー:name，値:value の Map で返します。キー:name の型は java.lang.String です。

構文

```
public java.util.Map getOutputParams()
```

引数

ありません。

戻り値

最後に呼び出されたオペレーションの出力パラメタを Map（java.util.Map）で返します。出力パラメタがない場合は，空の Map を返します。

例外

invoke メソッドが呼び出される前に実行した場合，JAXRPCException 例外がスローされます。

getOutputValues

インタフェース名：Call

機能

最後に呼び出されたオペレーションの出力パラメタの値を List で返します。

構文

```
public java.util.List getOutputValues()
```

引数

ありません。

戻り値

最後に呼び出されたオペレーションの出力パラメタの値をリスト（java.util.List）で返します。出力の値がない場合は，空の List を返します。List に格納される出力パラメタの順番は，WSDL の parameterOrder の順番に一致します。

例外

invoke メソッドが呼び出される前に実行した場合，JAXRPCException 例外がスローされます。

getProperty

インタフェース名：Call

機能

指定した名前のプロパティ値を取得します。指定できるプロパティについては，setProperty の説明を参照してください。

構文

```
public java.lang.Object getProperty(java.lang.String name)
```

引数

表 13-15　getProperty メソッドの引数

仮引数名	名称	in/out	説明
name	プロパティ名	in	プロパティ名を指定します。

戻り値

プロパティ値（Object）を返します。

例外

次の場合に JAXRPCException 例外がスローされます。

- name に null を指定した場合
- 指定できるプロパティ以外のプロパティ名を指定した場合
- 指定した名前のプロパティが存在しない場合

invoke

インタフェース名：Call

機能

Synchronous request-response モードでオペレーションを呼び出します。引数なしのオペレーションを呼び出す場合は，空の Object 配列を引数に指定します。

構文

```
public java.lang.Object invoke(java.lang.Object[] inputParams)
                        throws java.rmi.RemoteException
```

引数

表 13–16　invoke メソッドの引数

仮引数名	名称	in/out	説明
inputParams	オペレーションの引数	in	オペレーションの引数を指定します。

戻り値

オペレーションの戻り値（java.lang.Object）または null を返します。

例外

次の場合に RemoteException 例外がスローされます。

- リモート呼び出しでエラーが発生した場合
- 通信先のサービスが上がっていない場合
- SOAP Fault が発生した場合

次の場合に JAXRPCException 例外がスローされます。

- 引数 inputParams に null を指定した場合
- Call オブジェクトの初期化でエラーが発生した場合
- inputParams の数，順番，型が WSDL の内容と一致しない場合

removeProperty

インタフェース名：Call

機能

指定された名前のプロパティを削除します。指定できるプロパティについては，setProperty の説明を参照してください。

構文

```
public void removeProperty(java.lang.String name)
```

引数

表 13-17　removeProperty メソッドの引数

仮引数名	名称	in/out	説明
name	プロパティ名	in	プロパティ名を指定します。

戻り値

ありません。

例外

次の場合に JAXRPCException 例外がスローされます。

- name に null を指定した場合
- 指定できるプロパティ以外のプロパティ名を指定した場合
- 指定した名前のプロパティが存在しない場合

setProperty

インタフェース名：Call

機能

設定するプロパティを指定します。指定できるプロパティを次の表に示します。

表 13-18　指定できるプロパティ

キー名称	設定内容	型	指定値
javax.xml.rpc.security.auth.username	HTTP ベーシック認証のユーザ名	java.lang.String	「:」（コロン）以外の任意の文字列を指定します。空文字列も指定できます。次の値を指定した場合は，JAXRPCException がスローされます。 • null を指定した場合 • 「:」（コロン）を含む文字列を指定した場合

キー名称	設定内容	型	指定値
			サービスの URL にユーザ名を指定していた場合でも，このプロパティの設定が有効になります。このプロパティを設定することで既存の値が上書きされるので注意してください。
javax.xml.rpc.security.auth.password	HTTP ベーシック認証のパスワード	java.lang.String	任意の文字列を指定します。空文字列も指定できます。null は指定できません。null を指定した場合は，JAXRPCException がスローされます。 サービスの URL にパスワードを指定していた場合でも，このプロパティの設定が有効になります。このプロパティを設定することで既存の値が上書きされるので注意してください。

両プロパティの指定値の組み合わせによって，HTTP Authorization ヘッダの値は次のとおりとなります。

表 13–19　username／password の指定値と HTTP Authorization ヘッダの値の対応

username 指定値	password 指定値	Authorization ヘッダの値
指定なし	指定なし	ヘッダなし
空文字列	指定なし	":"の Base64 エンコード
空文字列	空文字列	":"の Base64 エンコード
空文字列	任意文字列	":任意文字列"の Base64 エンコード
":"以外の任意文字列	指定なし	"任意文字列:"の Base64 エンコード
":"以外の任意文字列	空文字列	":任意文字列:"の Base64 エンコード
":"以外の任意文字列	任意文字列	"任意文字列:任意文字列"の Base64 エンコード

構文

```
setProperty(String name, Object value)
```

引数

表 13–20　setProperty メソッドの引数

仮引数名	名称	in/out	説明
name	プロパティ名	in	プロパティ名を指定します。
value	プロパティ値	in	プロパティ値を指定します。

戻り値

ありません。

例外

次の場合に JAXRPCException 例外がスローされます。

- name に null を設定した場合
- value に不正な値を指定した場合
- value が指定の型と異なる場合
- 指定できるプロパティ以外のプロパティ名を指定した場合

13.8　ClientID クラス（クライアント識別子）

クライアント識別子を表すクラスです。このクラスは，公開メンバおよび公開メソッドを持ちません。

クラス定義

```
public final class ClientID
extends java.lang.Object
```

パッケージ

```
com.cosminexus.cws.management
```

13.9　JAXRPCException クラス（JAX-RPC に関する例外）

JAX-RPC ランタイムに関連した例外を表す例外クラスです。

クラス定義

```
public class JAXRPCException
extends java.lang.RuntimeException
```

パッケージ

```
javax.xml.rpc.JAXRPCException
```

JAXRPCException クラスのメソッドを次の表に示します。

表 13–21　JAXRPCException クラスのメソッド一覧

メソッド	機能概要
getLinkedCause	原因例外を取得します。

getLinkedCause

クラス名：JAXRPCException

機能

原因例外を取得します。

構文

```
public java.lang.Throwable getLinkedCause()
```

引数

ありません。

戻り値

原因例外（java.lang.Throwable）を返します。存在しない場合は null を返します。

13.10 Management クラス（クライアントの開始，終了）

RPC 形態，メッセージング形態の各クラスを使用する前，および終了時に呼び出すクラスです。SOAP 通信基盤でクライアントを開発する場合は必ずこのクラスを使用してください。このクラスでは，次の処理を行います。

- トレースファイルおよびアプリケーションログの初期化
- トレースファイルおよびアプリケーションログの終了処理
- クライアント識別子とスレッドの関連づけ

なお，サーバから別の SOAP アプリケーションを呼び出すような場合（サーバが別のサーバに対するクライアントとなる場合）は，このクラスのメソッドを使用する必要はありません。

クライアント識別子とスレッドの関連づけ

クライアント識別子とは，SOAP サービスを利用するクライアントの実行時オプション，トレースファイル，およびアプリケーションログの出力先を個々のクライアントごとに区別するために使用する情報です。J2EE サーバ上でクライアントを実行する場合，実行時オプション，トレースファイル，およびアプリケーションログの出力先をクライアントごとに分けるために，このクラスのメソッドを使用して，クライアント識別子とクライアントの各スレッドを関連づける必要があります。

注意事項

クライアントをサーブレット，JSP，および EJB として実装する場合は，必ずこのクラスのメソッドを使用してください。

このクラスのメソッドを使用するクライアントと使用しないクライアントが混在すると，J2EE サーバ上でクライアントを実行するときに，トレースファイル，アプリケーションログ，および実行時オプションの出力先が動作定義ファイルの指定どおりにならないことがあります。

クラス定義

```
public final class Management
extends java.lang.Object
```

パッケージ名

```
com.cosminexus.cws.management
```

Management クラスが提供するメソッドを次に示します。

表 13–22 Management クラスのメソッド一覧

メソッド	機能概要
initializeClient	クライアントの開始処理をします。
connectClientIDtoCurrentThread	クライアント識別子をスレッドに関連づけます。
disconnectClientIDtoCurrentThread	クライアント識別子とスレッドの関連づけを解除します。

メソッド	機能概要
finalizeClient	クライアントの終了処理をします。

initializeClient

クラス名：Management

機能

トレースファイル，アプリケーションログの初期化，動作定義ファイルの読み込みなど，クライアントの開始処理をします。このメソッドはマルチスレッドセーフです。

構文

```
public static com.cosminexus.cws.management.ClientID initializeClient()
```

引数

ありません。

戻り値

クライアント識別子を示す ClientID クラスを返します。

注意事項

- このメソッドを呼び出さなかった場合，動作定義ファイルは読み込まれないで，実行時オプションはすべてデフォルト値で動作します。また，トレース出力結果はデフォルトトレースに出力されます。
- クライアントをサーブレット，JSP で実装する場合，このメソッドを HttpServlet.init や jspInit メソッド内で呼び出してください。
- クライアントを EJB で実装する場合，このメソッドを ejbCreate メソッド内で呼び出してください。
- このメソッドを同じスレッド内で複数回コールした場合，同じクライアント識別子を示す ClientID クラスを返します。

connectClientIDtoCurrentThread

クラス名：Management

機能

クライアント識別子を現在のスレッドに関連づけます。

構文

```
public static void connectClientIDtoCurrentThread (
     com.cosminexus.cws.management.ClientID key
)
```

引数

表 13-23　connectClientIDtoCurrentThread メソッドの引数

仮引数名	名称	in/out	説明
key	クライアント識別子	in	initializeClient メソッドの戻り値として取得したクライアント識別子を指定します。null を指定した場合は何もしないで終了します。

戻り値

ありません。

注意事項

- クライアントをサーブレットおよび JSP で実装する場合，サーブレットでは HttpServlet クラスの HTTP リクエストに対応するハンドラメソッド（doGet メソッドや doPost メソッドなど）の最初で，JSP の場合はボディ部分の最初でこのメソッドを呼び出してください。
- クライアントを EJB で実装する場合，このメソッドを EnterpriseBean クラス内のサービスメソッドの最初で呼び出してください。
- このメソッドを呼び出さなかった場合は，同じ J2EE サーバ上で動作するほかのクライアントで disconnectClientIDtoCurrentThread メソッドが呼び出されていないと，実行時オプションやトレース出力先が不定となります。
- このメソッドを呼び出さなかった場合で，SOAP クライアント識別子と現在のスレッドの関連づけがされていないとき（初回実行時など）は，SOAP エンジンの実行時オプションはすべてデフォルト値で動作します。またトレース出力結果はデフォルトトレースに出力されます。

disconnectClientIDtoCurrentThread

クラス名：Management

機能

クライアント識別子と現在のスレッドの関連づけを解除します。

構文

```
public static void disconnectClientIDtoCurrentThread (
     com.cosminexus.cws.management.ClientID key
)
```

引数

表 13-24　disconnectClientIDtoCurrentThread メソッドの引数

仮引数名	名称	in/out	説明
key	クライアント識別子	in	initializeClient メソッドの戻り値として取得したクライアント識別子を指定します。

戻り値

ありません。

注意事項

- connectClientIDtoCurrentThread メソッドを使用した場合，必ずこのメソッドを呼び出してください。このメソッドを呼び出さなかった場合，connectClientIDtoCurrentThread メソッドで関連づけたクライアント識別子の情報が解除されないで，このメソッドを呼び出したほかのクライアントの実行時オプション，トレースファイル，およびアプリケーションログの出力先や重要度が有効になってしまうなどの影響を及ぼす場合があります。
- クライアントをサーブレットおよび JSP で実装する場合，サーブレットでは HttpServlet クラスの HTTP リクエストに対応するハンドラメソッド（doGet メソッドや doPost メソッドなど）の最後で，JSP ではボディ部分の最後でこのメソッドを呼び出してください。
- クライアントを EJB で実装する場合，このメソッドを EnterpriseBean クラス内のサービスメソッドの最後で呼び出してください。
- key が null の場合，または key で指定したクライアント識別子が connectClientIDtoCurrentThread メソッドで指定したものと異なる場合，警告メッセージがトレース上に出力されます。

finalizeClient

クラス名：Management

機能

トレースファイルおよびアプリケーションログのクローズなど，クライアントの終了処理をします。このメソッドはマルチスレッドセーフです。

構文

```
public static void finalizeClient(com.cosminexus.cws.management.clientID key)
```

引数

表 13-25　finalizeClient メソッドの引数

仮引数名	名称	in/out	説明
key	SOAP クライアント識別子	in	initializeClient メソッドの戻り値として取得した SOAP クライアント識別子を指定します。

戻り値

ありません。

注意事項

- initializeClient メソッドを呼び出す前にこのクラスを呼び出した場合は，何もしないで終了します。

- このメソッドを呼び出したあと，initializeClient メソッドを再度呼び出した場合は，クライアント開始処理を再度行います。
- クライアントをサーブレットおよび JSP で実装する場合，このメソッドは HttpServlet.destroy（サーブレットの場合）や jspDestroy（JSP の場合）で呼び出してください。
- クライアントを EJB で実装する場合，このメソッドを ejbRemove メソッド内で呼び出してください。

13.11　ReqResListener インタフェース（SAAJ を利用した SOAP アプリケーションの実装）

リクエスト・レスポンス型の SOAP アプリケーションが実装するメッセージングインタフェースです。

インタフェース定義

```
public interface ReqResListener
```

パッケージ

```
com.cosminexus.cws.xml.soap
```

ReqResListener インタフェースのメソッドを次の表に示します。

表 13-26　ReqResListener インタフェースのメソッド一覧

メソッド	機能概要
onMessage	メッセージングサービスを処理するメソッドです。

onMessage

インタフェース名：ReqResListener

機能

メッセージングサービスを処理します。

構文

```
public javax.xml.soap.SOAPMessage onMessage(javax.xml.soap.SOAPMessage soapmessage)
        throws javax.xml.soap.SOAPException
```

引数

表 13-27　onMessage メソッドの引数（ReqResListener インタフェース）

仮引数名	名称	in/out	説明
soapmessage	リクエストメッセージ	in	リクエストメッセージを格納したクラスです。

戻り値

javax.xml.soap.SOAPMessage オブジェクトを返します。

例外

javax.xml.soap.SOAPException

レスポンスメッセージを返せない場合にスローされます。

13.12　Service インタフェース（サービスインタフェース）

Call インスタンスを生成するためのファクトリインタフェースです。

インタフェース定義

```
public interface Service
```

パッケージ

```
javax.xml.rpc.Service
```

Service インタフェースのメソッドを次の表に示します。

表 13-28　Service インタフェースのメソッド一覧

メソッド	機能概要
createCall	指定のポートとオペレーションに関連づけられた Call オブジェクトを生成します。

createCall

インタフェース名：Service

機能

指定したポートとオペレーションに関連づけられた Call オブジェクトを生成します。このメソッドは，Service オブジェクトが WSDL の情報を保持している状態で，初期化された Call オブジェクトを生成するために使用されます。生成された Call オブジェクトを初期化する必要はありません。

構文

```
public Call createCall(javax.xml.namespace.QName portName
                       javax.xml.namespace.QName operationName)
                throws ServiceException
```

引数

表 13-29　createCall メソッドの引数

仮引数名	名称	in/out	説明
portName	ポートの QName	in	ポートの QName を指定します。
operationName	オペレーションの QName	in	オペレーションの QName を指定します。

戻り値

Call オブジェクトを返します。

例外

Service オブジェクトが WSDL を読み込んでポートの情報を保持している状態では，次の場合に ServiceException 例外がスローされます。

- portName または operationName に null を指定した場合
- 指定した portName を Service オブジェクトが保持していない場合
- 指定した operationName を Service オブジェクトが保持していない場合

また，Call オブジェクトの生成でエラーが発生した場合も ServiceException 例外がスローされます。

13.13　ServiceException クラス（サービスに関する例外）

Service インタフェースおよび ServiceFactory クラスのメソッドによってスローされる例外クラスです。

クラス定義

```
public class ServiceException
extends java.lang.Exception
```

パッケージ

```
javax.xml.rpc.ServiceException
```

ServiceException クラスのメソッドを次の表に示します。

表 13–30　ServiceException クラスのメソッド一覧

メソッド	機能概要
getLinkedCause	原因例外を取得します。

getLinkedCause

クラス名：ServiceException

機能

原因例外を取得します。

構文

```
public java.lang.Throwable getLinkedCause()
```

引数

ありません。

戻り値

原因例外（java.lang.Throwable）を返します。存在しない場合は null を返します。

13.14　ServiceFactory クラス（サービスのファクトリクラス）

Service インスタンスを生成するためのファクトリクラスです。

クラス定義

```
public abstract class ServiceFactory
extends Object
```

パッケージ

```
javax.xml.rpc.ServiceFactory
```

ServiceFactory クラスのメソッドを次の表に示します。

表 13-31　ServiceFactory クラスのメソッド一覧

メソッド	機能概要
createService	Service オブジェクトを生成します。
newInstance	ServiceFactory オブジェクトを生成します。

createService

クラス名：ServiceFactory

機能

Service オブジェクトを生成します。WSDL を読み込んでパースし，WSDL に記述されている情報を保持します。

構文

```
public abstract Service createService(
                              java.net.URL wsdlDocumentation,
                              javax.xml.namespace.QName serviceName)
                           throws ServiceException
```

引数

表 13-32　createService メソッドの引数

仮引数名	名称	in/out	説明
wsdlDocumentation	WSDL の URL	in	WSDL の URL として，ローカルファイルへのパスを指定します。
serviceName	SOAP サービスの QName	in	SOAP サービスの QName を指定します。

戻り値

Service オブジェクトを返します。

例外

次の場合に，ServiceException 例外がスローされます。

- wsdlDocumentation または serviceName に null を指定した場合
- wsdlDocumentation にローカルファイル以外のパスを指定した場合
- wsdlDocumentation に WSDL が存在しないローカルファイルのパスを指定した場合
- serviceName に指定したサービス名が WSDL に存在しない場合
- WSDL の解析中にエラーが発生した場合
- Service オブジェクトの作成でエラーが発生した場合

newInstance

クラス名：ServiceFactory

機能

ServiceFactory オブジェクトを生成します。

構文

```
public static ServiceFactory newInstance()
                                throws ServiceException
```

引数

ありません。

戻り値

ServiceFactory オブジェクトを返します。

14 障害対策

この章では，SOAP アプリケーションの実行，運用中の障害対策について説明します。SOAP を利用した通信ができない場合の，考えられる原因や対処方法を FAQ で説明します。

14.1　障害対策の流れ

SOAP アプリケーションを運用中に障害が発生した場合，次に示す流れで対処します。

1. 現象の確認

   障害が発生したときの現象を確認します。メッセージが出力されている場合は，メッセージの内容を確認してください。

2. 資料の取得

   トラブルの要因を調査するために必要な資料を取得します。取得が必要な資料については，「14.2　障害発生時に取得する資料」を参照してください。

3. 原因の調査と対処

   取得した資料を基にトラブルの原因を調査し，対処します。問題発生個所の切り分けについては，「14.3　問題発生個所の切り分け」を参照してください。

14.2　障害発生時に取得する資料

システム管理者への連絡が必要になる障害が発生した場合，または対処方法が不明な障害の場合は，次に示す資料を取得した上，システム管理者に連絡してください。

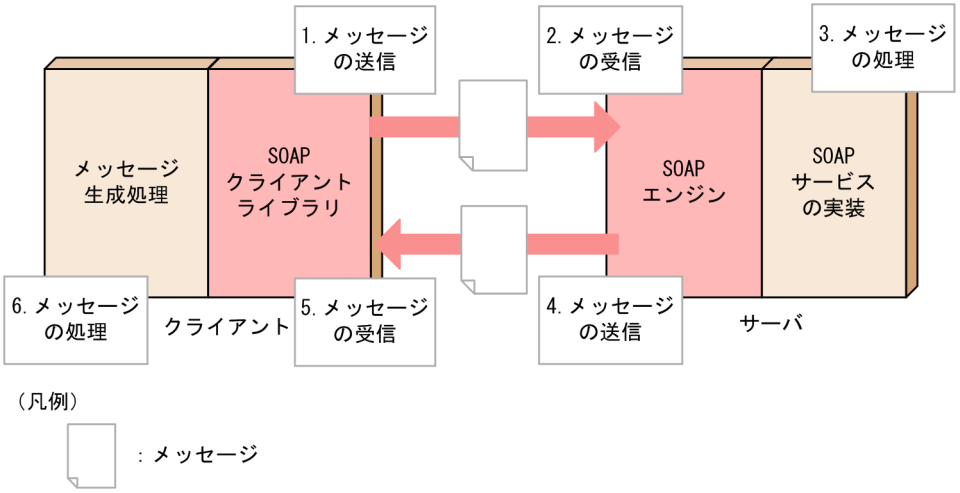
- トレースファイル
- アプリケーションログファイル
- SOAP アプリケーションの環境設定ファイル（web.xml，server-config.xml）
- サーバおよびクライアントに設定したシステムプロパティとシステムクラスパス
- サーバおよびクライアントの標準出力と標準エラー出力
- Component Container および Web サーバのログ
- Component Container で規定する障害時取得情報
  Component Container で規定する障害時取得情報を次に示します。
  - J2EE サーバのユーザ定義ファイル
  - J2EE サーバの保守情報
  - サーバ管理コマンドのユーザ定義ファイル
  - サーバ管理コマンドの保守情報
  - J2EE サーバの標準出力，標準エラー出力
  - J2EE サーバ側の TPBroker のログ
  - EJB クライアントアプリケーションに設定したシステムプロパティ，およびシステムクラスパス
  - EJB クライアントアプリケーションの標準出力，標準エラー出力
- ネットワークモニタツールで採取したパケットの内容

Component Container で規定する障害時取得情報の取得方法については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」を参照してください。

14.3　問題発生個所の切り分け

SOAP による通信ができない場合，処理が行われる個所のうち，どこで問題が発生しているかを把握した上で適切な対処をする必要があります。SOAP アプリケーションのトラブル発生時に，原因となる可能性がある個所を次の図に示します。

図 14–1　SOAP アプリケーションの処理の流れでの問題発生個所



図に示す 1～6 の個所に対し，次に示す処理が正常に行われているかを確認することで，問題発生個所を絞り込めます。

1. クライアントから SOAP メッセージが正しく出力されているか
2. メッセージは SOAP エンジンに届いているか
3. SOAP サービスでメッセージが処理できているか
4. サーバからの返信用 SOAP メッセージは正しく出力されているか
5. 返信用 SOAP メッセージはクライアントに届いているか
6. クライアントで返信用 SOAP メッセージを処理できているか

1.～6.で示した内容が正常に処理されていない場合，次に示すメッセージ ID を持つメッセージが出力されることが考えられます。

表 14–1　問題発生個所と出力されることが考えられるメッセージ ID の対応

問題発生個所	出力されるメッセージのメッセージ ID
1.の処理	KDCCS1149-E
2.の処理	KDCCP0001-E
3.の処理	KDCCP0002-E，KDCCP0003-E，KDCCP0004-E，KDCCP0009-E，KDCCP0013-E，KDCCP0020-E
4.の処理	KDCCP0001-E，KDCCS1149-E

問題発生個所	出力されるメッセージのメッセージ ID
5.の処理	KDCCP0001-E
6.の処理	KDCCP0009-E

メッセージの内容については，マニュアル「アプリケーションサーバ メッセージ(構築／運用／開発用)」を参照してください。メッセージのほかに，処理が正常に行われているかどうかを確認するには，トレースファイルおよびアプリケーションログを使用します。トレースファイルについては，「14.4　トレースファイル」を参照してください。アプリケーションログについては，「14.5　アプリケーションログ」を参照してください。

14.4　トレースファイル

SOAP アプリケーション開発支援機能および SOAP 通信基盤では，障害対策に必要な情報をトレースとしてトレースファイルに出力します。トレースファイルは障害の発生個所の割り出しや原因の調査などに利用します。出力するトレースとトレースファイルの種類を次に示します。

SOAP トレース

SOAP アプリケーション開発支援機能および SOAP 通信基盤で出力するトレースです。SOAP トレースの種類を次に示します。

サーバトレース

SOAP アプリケーション開発支援機能および SOAP 通信基盤のサーバ側で出力するトレースです。サーバトレースファイルに出力します。

クライアントトレース

SOAP アプリケーション開発支援機能および SOAP 通信基盤のクライアント側で出力するトレースです。クライアントトレースファイルに出力します。

デフォルトトレース

警告メッセージなどを出力するトレースです。デフォルトトレースファイルに出力します。

Web サービスセキュリティトレース

Web サービスセキュリティ機能の実行時に出力するトレースです。サーバ側はサーバトレースファイルに，クライアント側はクライアントトレースファイルに出力します。

提供コマンドトレース

提供コマンドの実行時に出力するトレースです。提供コマンドトレースファイルに出力します。

JAXR トレース

UDDI クライアントライブラリが提供する API の実行時に出力するトレースです。JAXR トレースファイルに出力します。

14.4.1　トレースファイルに出力される内容

トレースファイルに出力される内容を次に示します。

表 14–2　トレースファイルの出力内容

項目	内容
日付	出力時の日付（yyyy/mm/dd 形式）が出力されます。
時刻	出力時の時刻（hh:mm:ss.sss 形式）が出力されます。
アプリケーション名	トレースの種類によって，アプリケーション名は異なります。 • サーバトレース：「c4websv」 • クライアントトレース：「c4webcl」

項目	内容
	• デフォルトトレース：「c4webcl」 • Web サービスセキュリティトレース：「wss」 • JAXR トレース：「hitjaxr」 • 提供コマンドトレース：「c4webcl」
プロセス識別子	プロセス識別子（16 進数）が出力されます。
スレッド識別子	スレッド識別子（16 進数）が出力されます。
メッセージ ID	メッセージ ID が出力されます。メッセージ ID を持たないものは出力されません。
メッセージ種別	メッセージテキストの種別が出力されます。 1. SOAP トレースの場合 EC：例外をキャッチしたことを表します。 FB：SOAP 通信基盤が提供する API の開始を表します。 FE：SOAP 通信基盤が提供する API の終了を表します。 PB：ほかのプログラムの処理の呼び出しを表します。 PE：呼び出したほかのプログラムの処理が終了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。 2. JAXR トレースの場合 FB：メソッドの入口を表します。 FE：メソッドの出口を表します。 EC：例外をキャッチしたことを表します。 ER：エラーメッセージを表示したことを表します。 PB：ほかのプログラムの処理の呼び出しを表します。 PE：呼び出したほかのプログラムの処理の終了を表します。
メッセージテキスト	実行時の情報を示すメッセージが出力されます。

14.4.2　トレースファイルの出力先

出力される SOAP トレースおよび JAXR トレースの出力先と，トレースファイル名を示します。

(1)　SOAP トレース

出力される SOAP トレースファイルの種類やトレースファイル名は，次に示す場合で異なります。

- **J2EE サーバ上で動作する場合**

  J2EE サーバ上で動作する SOAP サービス，および J2EE サーバ上で動作する SOAP クライアント（サーブレット，EJB）が該当します。

- **J2EE サーバ上で動作しない場合**

  J2EE サーバ上で動作しない SOAP クライアント（コマンドライン），および開発支援コマンドが該当します。

J2EE サーバ上で動作する場合，および J2EE サーバ上で動作しない場合に分けて，出力されるトレースファイルの種類，出力先，トレースファイル名を示します。

(a)　J2EE サーバ上で動作する場合

表 14–3　出力される SOAP トレース（J2EE サーバ上で動作する場合）

トレースファイルの種類	トレースファイルの出力先	トレースファイル名
Application Server のメッセージログ	<J2EE サーバのログ出力先フォルダ>※1	cjmessage?.log（?：1～16）※1※5
サーバトレース	<J2EE サーバのログ出力先フォルダ>¥WS	サーバ定義ファイル（c4websv.cfg）で設定したファイル名称となります。※2※5
クライアントトレース	<J2EE サーバのログ出力先フォルダ>¥WS	クライアント定義ファイル（c4webcl.properties）で設定したファイル名称となります。※3※5
デフォルトトレース	<J2EE サーバのログ出力先フォルダ>¥WS	c4webcl-default-?.log（?：1～16）※4※5

注※1

マニュアル「アプリケーションサーバ 機能解説 保守／移行編」で記述している J2EE サーバのログ取得の設定，およびトラブルシューティングの資料取得の設定についての説明を参照してください。

注※2

サーバ定義ファイルの設定については，「10.2　サーバ定義ファイルの設定」を参照してください。

注※3

クライアント定義ファイルの設定については，「10.3　クライアント定義ファイルの設定」を参照してください。

注※4

（?：1～16）は，共通定義ファイルの設定によって最大値が異なります。

共通定義ファイルの設定については，「10.4　共通定義ファイルの設定」を参照してください。

注※5

デフォルトでは，出力先ファイルが指定したファイルサイズになると，SOAP トレースの出力先を次のファイルに切り替えます（ラップアラウンドモード）。J2EE サーバ上で動作する場合，SOAP トレースを出力していたファイルを指定した時刻でリネームし，SOAP トレースの出力先ファイル名を変更しないこともできます（シフトモード）。シフトモードの詳細については，マニュアル「アプリケーションサーバ 機能解説 保守／移行編」で記述しているトラブルシューティングのための準備についての説明を参照してください。

(b)　J2EE サーバ上で動作しない場合

表 14–4　出力される SOAP トレース（J2EE サーバ上で動作しない場合）

トレースファイルの種類	トレースファイルの出力先	トレースファイル名
Application Server のメッセージログ	<J2EE サーバのログ出力先フォルダ>※1	cjclmessage?.log（?：1～16）※1
クライアントトレース	<J2EE サーバのログ出力先フォルダ>¥WS	クライアント定義ファイル（c4webcl.properties）で設定したファイル名称となります。※2
デフォルトトレース	<J2EE サーバのログ出力先フォルダ>¥WS	c4webcl-default-?.log（?：1～16）※3
提供コマンドトレース	<Application Server のインストールディレクトリ>¥CC¥client¥logs¥system¥ejbcl¥WS	Java2WSDL コマンド：Java2WSDL-?.log WSDL2Java コマンド：WSDL2Java-?.log Java2WSDD コマンド：Java2WSDD-?.log （?：1～16）※3

注※1

マニュアル「アプリケーションサーバ 機能解説 保守／移行編」で記述している EJB クライアントアプリケーションのシステムログについての説明を参照してください。

注※2

クライアント定義ファイルの設定については，「10.3　クライアント定義ファイルの設定」を参照してください。

注※3

（?：1～16）は，共通定義ファイルの設定により最大値が異なります。

共通定義ファイルの設定については，「10.4　共通定義ファイルの設定」を参照してください。

(2)　JAXR トレース

JAXR トレースファイルの出力先およびトレースファイルの名称を次の表に示します。

表 14–5　出力される JAXR トレース

トレースファイルの種類	トレースファイルの出力先	トレースファイル名
JAXR トレース	システムプロパティ（com.cosminexus.xml.registry.trace.file_path）で設定したディレクトリに出力します。	＜システムプロパティ（com.cosminexus.xml.registry.trace.file_path）で設定したファイル名＞n.log （n:1～16）

注意事項

トレースファイルの出力先ディレクトリ下には，トレースファイル以外のファイル（拡張子が.log 以外のファイル）が生成されます。このファイルは SOAP クライアントまたは SOAP サービスを停止しても削除されないので，適宜削除してください。

14.4.3　トレースファイル出力の重要度

トレースファイル出力の重要度を変更することで，障害発生時に出力する情報量を変更できます。出力する情報量を多くすることで，障害発生の要因を特定しやすくなります。ただし，出力する情報量を多くすると，プログラムの処理性能への影響が大きくなります。

次の表に，指定できる重要度と出力される内容を示します。

表 14–6　トレースファイルの重要度および出力内容

重要度	出力内容
ERROR	例外発生時にトレースファイルを出力します。
WARN	例外発生時または外部の API 呼び出し時にトレースファイルを出力します。
INFO	次の場合にトレースファイルを出力します。 • 例外発生時 • ほかのプログラムの処理の呼び出しと終了 • SOAP 通信基盤が提供する API の開始と終了
DEBUG	次の場合にトレースファイルを出力します。 • 例外発生時 • ほかのプログラムの処理の呼び出しと終了 • SOAP 通信基盤が提供する API の開始と終了 • SOAP 通信基盤が提供する API の呼び出し時の引数

14.4.4　トレースファイルの見積もり方法

クライアントトレース，サーバトレースのトレースファイルのサイズ，および面数の算出方法を次に示します。なお，この算出方法はトレースファイル出力の重要度が INFO の場合のものです。トレースファイル出力の重要度が ERROR，または WARN の場合は，初回起動時に 5KB 程度出力される以外にトレース出力はありません。また，例外発生時には，発生した例外に応じてトレース出力があります。

(1)　RPC 形態の SOAP アプリケーションの場合

サービスのインタフェース，実行時オプションの設定，エンコーディング種別（SOAP エンコーディング，リテラルエンコーディング）に関係なく，1 回のリクエストおよびレスポンスで次に示すトレース量が出力されます。

- **サーバ側：**1.0KB
- **クライアント側：**2.0KB

注意事項

- 初回リクエストは初期化処理をするため，この値より約 2.0KB 多くトレース出力されます。
- このトレース出力量は，ユーザ実装部分は含まれていません。ユーザ実装部分で SOAP の提供する API を発行すると，1API 当たり約 0.4KB のトレース量がトレースファイルに出力されます。

(2)　メッセージング形態の SOAP アプリケーションの場合

送受信している SOAP メッセージの大きさや添付するファイル数によって出力されるトレース量が異なります。次に示す値を参考にして，トレースファイルのサイズおよび面数を算出してください。

表 14-7　1 回の送受信で出力されるトレース量の目安

送受信する内容			トレース出力量	
SOAP ヘッダの子要素数	SOAP ボディの子要素数	添付するファイル数	クライアント側（単位:KB）	サーバ側（単位:KB）
0	0	0	2.0	2.0
1	0	0	4.0	3.0
10	0	0	19.0	18.0
0	1	0	4.0	3.0
0	10	0	19.0	18.0
0	0	1	3.0	2.0
0	0	10	8.0	7.0
1	1	1	6.0	5.0
10	10	10	43.0	42.0

注意事項

- 初回リクエストは初期化処理をするため，この値より約 3.0KB 多くトレース出力されます。
- トレース出力量にはユーザ実装部分は含まれません。ユーザ実装部分で SOAP が提供する API を発行すると，1API 当たり約 0.4KB のトレース量がトレースファイルに出力されます。
- 設定する添付ファイルの数によって，出力されるトレース量は増加します。ただし，設定する添付ファイルのサイズによって出力されるトレースのサイズは変わりません。

14.5　アプリケーションログ

SOAP アプリケーション開発支援機能および SOAP 通信基盤では，SOAP アプリケーションへの要求と SOAP アプリケーションからの応答時の SOAP メッセージ，および添付ファイル付き SOAP メッセージの MIME ヘッダの一部をアプリケーションログとして出力します。アプリケーションログの種類を次に示します。

サーバログ

SOAP アプリケーションを提供するサーバ側で出力するアプリケーションログのことをいいます。

クライアントログ

SOAP アプリケーションを利用するクライアント側で出力するアプリケーションログのことをいいます。

14.5.1　アプリケーションログに出力される内容

アプリケーションログに出力される内容を次に示します。

表 14–8　アプリケーションログの出力内容

項目	内容
日付	出力時の日付（yyyy/mm/dd 形式）が出力されます。
時刻	出力時の時刻（hh:mm:ss 形式）が出力されます。
メッセージの種類	メッセージの種類によって，次のどれかが出力されます。 Request SOAP Message クライアントから SOAP アプリケーションに対して送信されたメッセージであることを意味します。 Response SOAP Message SOAP アプリケーションからクライアントに対する応答メッセージであることを意味します。 Fault Response SOAP Message SOAP アプリケーションからクライアントに対する SOAP Fault メッセージであることを意味します。 上記以外 添付ファイル付き SOAP メッセージの MIME ヘッダであることを示します。
シーケンス番号	シーケンス番号が出力されます。「Seq：<シーケンス番号>」の形式で出力されます。
メッセージまたは引数	メッセージまたは引数が出力されます。メッセージの場合は XML 形式で出力されます。提供する API の場合は引数の文字列表現で出力されます。

アプリケーションログが出力されるタイミングと同時に，トレースファイルにメッセージの種類およびシーケンス番号の文字列が出力されます。これによって，アプリケーションログの情報がどのタイミングで出力されたのかをトレースファイルから解析できます。

アプリケーションログの出力に関する注意事項を示します。

(1) 送受信メッセージの文字コードに関する注意

アプリケーションログに出力される送受信メッセージの文字コードは，OS のデフォルト文字コードになります。送受信されたメッセージの文字コードと，OS のデフォルト文字コードが異なる場合は，送受信されたメッセージの文字コードと，アプリケーションログに出力されるメッセージの文字コードが異なります。

(2) 受信メッセージの文字参照の有無に関する注意

アプリケーションログに出力される受信メッセージの文字参照の有無は，動作定義ファイルの設定に従います。受信されたメッセージの文字参照の有無と，定義ファイルでの文字参照の有無の設定が異なる場合は，アプリケーションログに出力されるメッセージの内容と，受信したメッセージの内容が異なります。

14.5.2 アプリケーションログ出力の重要度

アプリケーションログを出力する場合は，動作定義ファイルに重要度を設定します。次にアプリケーションログ出力の重要度を示します。

表 14-9 アプリケーションログの重要度および出力内容

重要度	出力内容
WARN	アプリケーションログを出力しません。
INFO または DEBUG	主要な API の引数および SOAP メッセージを出力します。

注意事項

アプリケーションログとして出力される情報は，SOAP アプリケーションに対して送受信するメッセージの内容によってはサイズが大きくなり，性能に影響が出ることがあります。通常運用では，出力しないように設定することをお勧めします。

14.6　異常発生時のアプリケーションログの出力

SOAP アプリケーション開発支援機能および SOAP 通信基盤では，SOAP アプリケーションへの要求と SOAP アプリケーションからの応答時の SOAP メッセージを，異常が発生した場合にアプリケーションログとして出力できます。

「異常が発生した場合」とは，次のような場合のことです。

サーバ側

- SOAP エンジンが SOAP Fault を返す場合
- SOAP エンジンが java.lang.Error 相当の例外を J2EE サーバにスローする場合

クライアント側

- SOAP クライアントライブラリが，クライアントプログラムに C4Fault 例外をスローする場合
- SOAP クライアントライブラリが，クライアントプログラムに java.lang.Error 相当の例外をスローする場合
- SOAP クライアントライブラリが，クライアントプログラムに javax.xml.soap.SOAPException 例外をスローする場合

注意事項

SOAP Fault を返す場合や C4Fault 例外を返す場合でも，SOAP メッセージ以外の HTTP メッセージを受信したときは，アプリケーションログが出力されないことがあります。

14.6.1　アプリケーションログに出力される内容

異常が発生した場合にアプリケーションログに出力される内容を次に示します。

表 14-10　異常発生時のアプリケーションログの出力内容

項目	内容
日付	出力時の日付（yyyy/mm/dd 形式）が出力されます。
時刻	出力時の時刻（hh:mm:ss 形式）が出力されます。
メッセージの種類	メッセージの種類によって，次のどちらかが出力されます。 Request SOAP Message(error record) 　異常が発生した場合のリクエストメッセージを示します。 Response SOAP Message(error record) 　異常が発生した場合のレスポンスメッセージを示します。
シーケンス番号	シーケンス番号を出力します。「Seq：<シーケンス番号>」の形式で出力されます。
メッセージ	メッセージは XML 形式で出力されます。ただし，メッセージが破壊されている場合，または文字コードが不正の場合は 16 進文字列形式で出力されます。

アプリケーションログが出力されるタイミングと同時に，トレースファイルにメッセージの種類およびシーケンス番号の文字列が出力されます。これによって，アプリケーションログの情報がどのタイミングで出力されたのかをトレースファイルから解析できます。

異常発生時のアプリケーションログの出力に関する注意事項を示します。

(1) 送受信メッセージの文字コードに関する注意

アプリケーションログに出力される送受信メッセージの文字コードは，OS のデフォルト文字コードになります。送受信されたメッセージの文字コードと，OS のデフォルト文字コードが異なる場合は，送受信されたメッセージの文字コードと，アプリケーションログに出力されるメッセージの文字コードが異なります。

(2) 受信メッセージの文字参照の有無に関する注意

アプリケーションログに出力される受信メッセージの文字参照の有無は，動作定義ファイルの設定に従います。受信されたメッセージの文字参照の有無と，定義ファイルでの文字参照の有無の設定が異なる場合は，アプリケーションログに出力されるメッセージの内容と，受信したメッセージの内容が異なります。

14.6.2 異常発生時のアプリケーションログの出力方法

異常発生時のアプリケーションログを出力する場合は，動作定義ファイルに出力ポイントの指定値を設定します。動作定義ファイルに指定する値と出力内容を次に示します。

表 14–11 動作定義ファイルに指定する値と出力内容

指定値	出力内容
ALL	異常発生時に，送信および受信メッセージをアプリケーションログに出力します。
SEND	異常発生時に，送信メッセージをアプリケーションログに出力します。
RECV	異常発生時に，受信メッセージをアプリケーションログに出力します。
NONE	アプリケーションログを出力しません。

注意事項

- アプリケーションログ出力の重要度の設定とは関係なく，異常発生時のアプリケーションログが設定されます。
- 異常発生時のアプリケーションログを出力する場合，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。

14.7　性能解析トレース

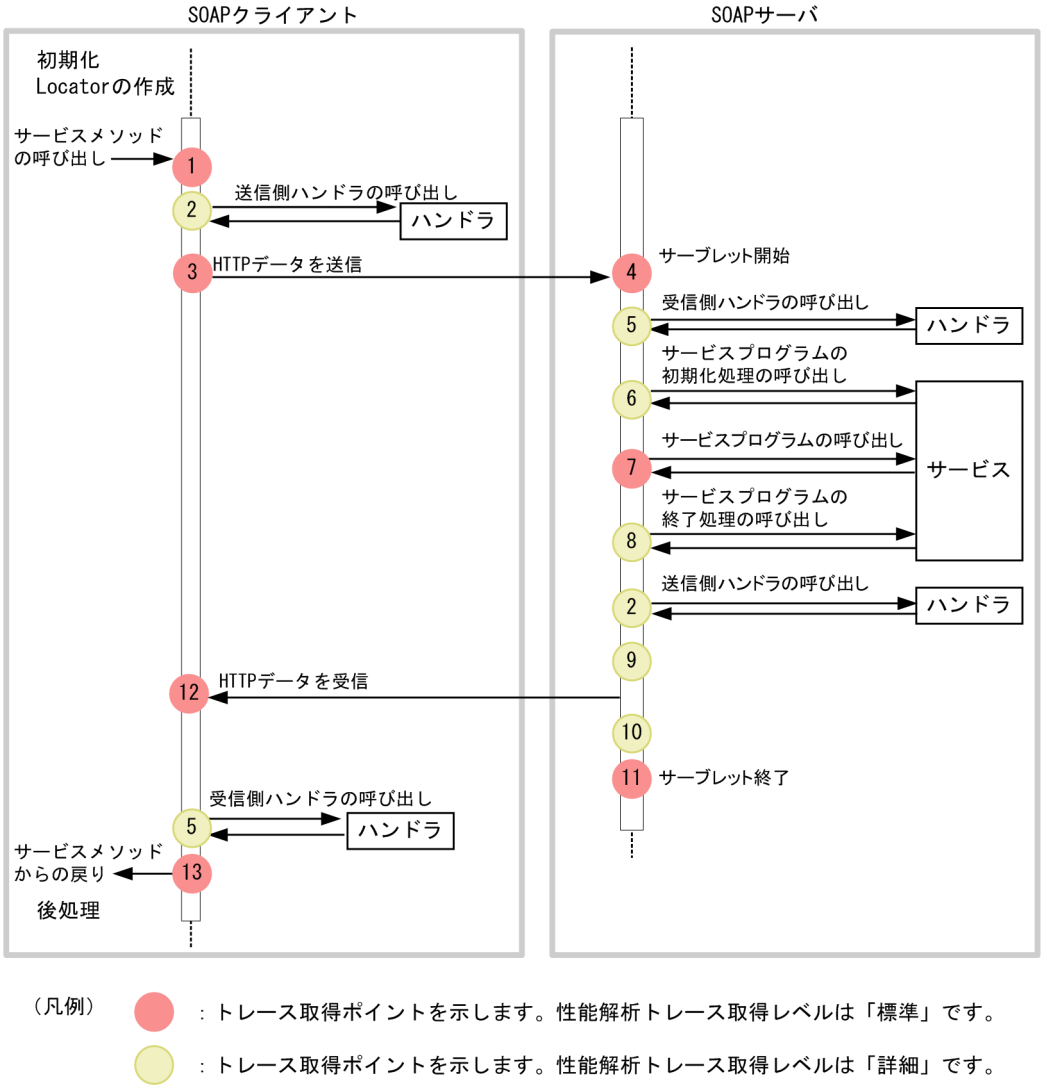
SOAP アプリケーションの性能解析トレースについて説明します。この性能解析トレースは，アプリケーションサーバシステムが提供する性能解析トレースの一部として提供します。

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

14.7.1　トレース取得ポイント

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 14-2　性能解析トレースのトレース取得ポイント



（凡例）　：トレース取得ポイントを示します。性能解析トレース取得レベルは「標準」です。

：トレース取得ポイントを示します。性能解析トレース取得レベルは「詳細」です。

イベント ID，トレース取得ポイント，および性能解析トレース取得レベルを次の表に示します。表の「図中の番号」は，図 14-2 中の番号と対応しています。

表 14–12　トレース取得ポイントの詳細

イベントID	図中の番号	トレース取得ポイント	レベル
0x9000	1	SOAP クライアントの開始	A
0x9022	2	送信側ハンドラの呼び出し※1※2	B
0x9026		ハンドラの呼び出し※1※2	B
0x9023		送信側ハンドラからの戻り※1※2	B
0x9027		ハンドラからの戻り※1※2	B
0x9004	3	クライアントから SOAP メッセージをサービスに送信	A
0x9008	4	SOAP サービスの開始	A
0x9024	5	受信側ハンドラの呼び出し※1※2	B
0x9026		ハンドラの呼び出し※1※2	B
0x9025		受信側ハンドラからの戻り※1※2	B
0x9027		ハンドラからの戻り※1※2	B
0x9010	6	サービスの初期化処理の呼び出し	B
0x9011		サービスの初期化処理からの戻り	B
0x9012	7	サービスの呼び出し	A
0x9013		サービスからの戻り	A
0x9016	8	サービスの終了処理の呼び出し	B
0x9017		サービスの終了処理からの戻り	B
0x900c	9	SOAP サービスからの SOAP メッセージ書き込み開始	B
0x900d	10	SOAP サービスからの SOAP メッセージ書き込み終了	B
0x9009	11	SOAP サービスの終了	A
0x9005	12	クライアントで SOAP メッセージをサービスから受信	A
0x9001	13	SOAP クライアントの終了	A

（凡例）

A：標準

B：詳細

注※1

ハンドラは，Web サービスセキュリティ機能などを使用すると出力されます。

注※2

動作定義ファイルで「prf_trace_level=NONHANDLER」を指定すると出力されません。

14.7.2　性能解析トレースの利用方法

性能解析トレースファイルを使用して性能を解析する場合，CSV 形式のファイルを編集できるアプリケーションプログラムで表示し，目的に合わせてフィルタリングや並べ替えの機能を利用して性能を解析します。

性能解析トレースファイルの利用方法について，SOAP サービスおよびサービスのレスポンスタイムの解析を例として説明します。

(1)　SOAP サービスおよびサービスのレスポンスタイムの解析例

SOAP サービスがクライアントからリクエストを受けてから返却するまでに掛かった時間を解析します。

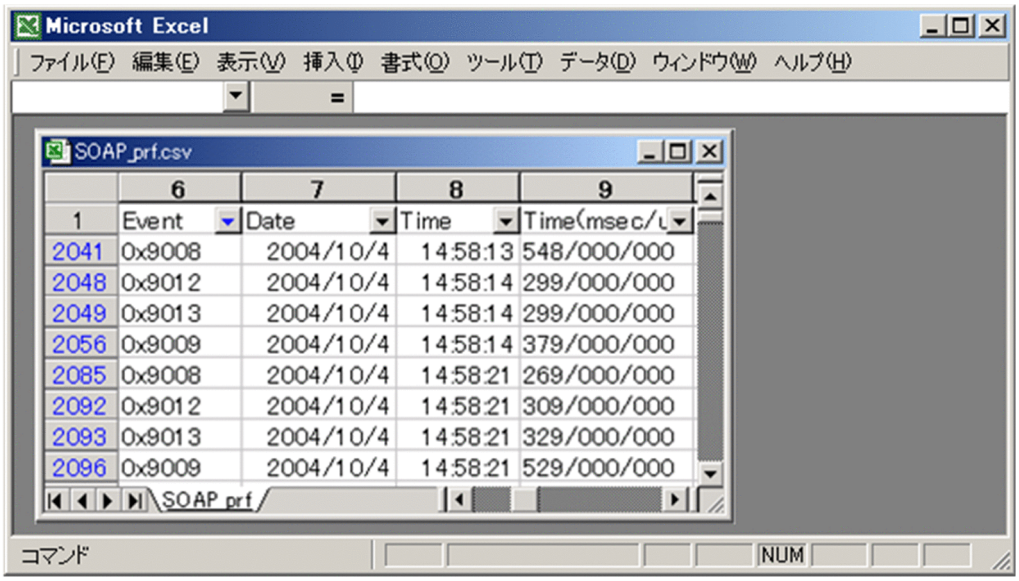
イベント ID「0x9008」，「0x9009」，「0x9012」，および「0x9013」をキーにして，収集した性能解析トレースファイルをフィルタリングします。フィルタリングに使用するイベント ID が示すトレース取得ポイントを次に示します。

表 14–13　フィルタリングに使用するイベント ID が示すトレース取得ポイント

イベント ID	トレース取得ポイント
0x9008	SOAP サービスの開始
0x9009	SOAP サービスの終了
0x9012	サービスの呼び出し
0x9013	サービスからの戻り

イベント ID「0x9008」，「0x9009」，「0x9012」，および「0x9013」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に示します。

図 14–3　性能解析トレースファイルをフィルタリングした例



	6	7	8	9
1	Event	Date	Time	Time(msec/
2041	0x9008	2004/10/4	14:58:13	548/000/000
2048	0x9012	2004/10/4	14:58:14	299/000/000
2049	0x9013	2004/10/4	14:58:14	299/000/000
2056	0x9009	2004/10/4	14:58:14	379/000/000
2085	0x9008	2004/10/4	14:58:21	269/000/000
2092	0x9012	2004/10/4	14:58:21	309/000/000
2093	0x9013	2004/10/4	14:58:21	329/000/000
2096	0x9009	2004/10/4	14:58:21	529/000/000

「0x9008」および「0x9009」のトレース取得時刻から，SOAP サービスのレスポンスタイムを解析できます。

「0x9012」および「0x9013」のトレース取得時刻から，サービスのレスポンスタイムを解析できます。

14.7.3　性能解析トレース使用時の注意事項

性能解析トレースを使用する際の注意事項を次に示します。

- SOAP アプリケーションの性能解析トレースを出力する場合は，性能解析トレースの取得レイヤに「SOAP 通信基盤」の取得レイヤを設定してください。また，J2EE サーバで性能解析トレースを出力できる状態にする必要があります。指定方法については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」を参照してください。
- 性能解析トレースは，性能解析トレース取得レベルによって出力されるトレース情報が異なります。性能解析トレース取得レベルと出力されるトレース情報を次の表に示します。

表 14-14　性能解析トレース取得レベルと出力されるトレース情報

性能解析トレース取得レベル			出力されるトレース情報	
取得レイヤの取得レベル※1	動作定義ファイルの取得レベル※2	取得ポイントの取得レベル※3	ハンドラ部分のトレース情報※4	ハンドラ部分以外のトレース情報※4
標準	ALL	標準	○	○
		詳細	×	×
	NONHANDLER	標準	×	○
		詳細	×	×
詳細	ALL	標準	○	○
		詳細	○	○
	NONHANDLER	標準	×	○
		詳細	×	○

（凡例）

○：性能解析トレースが出力されます。

×：性能解析トレースが出力されません。

注※1

取得レイヤは SOAP 通信基盤の機能レイヤになります。取得レイヤの取得レベルは，cprflevel コマンドで変更できます。cprflevel コマンドについては，マニュアル「アプリケーションサーバ リファレンス コマンド編」を参照してください。

注※2

動作定義ファイルの「prf_trace_level」キーで変更できます。

注※3

取得ポイントの取得レベルは変更できません。取得ポイントの取得レベルについては，「14.7.1　トレース取得ポイント」を参照してください。

注※4

ハンドラ部分のトレース情報とは，イベント ID が 0x9020～0x902F のトレースのことです。

14.8 FAQ

SOAP を利用した通信ができない場合の，考えられる原因や対処方法について示します。

14.8.1 アプリケーションログが出力されません。どのように対処すればよいですか？

アプリケーションログを出力するには，動作定義ファイルの設定が必要になります。「10. 動作定義ファイルおよび実行時オプションの設定項目」を参照して，動作定義ファイルが正しく設定されているかを確認してください。特に次に示す点に注意してください。

- **識別子の内容**

  サーバ側のアプリケーションログは，SOAP アプリケーションごとに出力されます。識別子の内容が，コンテキストルートから「/」を抜いた名称と一致しているか確認してください。

- **アプリケーションログ出力の重要度**

  アプリケーションログを出力するかどうかは，アプリケーションログ出力の重要度の項目によって設定します。設定値が「INFO」または「DEBUG」になっているか確認してください。設定値は大文字と小文字が区別されるため，注意してください。

- **動作定義ファイルの設定後の再起動**

  動作定義ファイルを設定したあとに，SOAP アプリケーションを再起動したかどうかを確認してください。設定した内容が反映されるのは，SOAP アプリケーションの再起動時になります。

- **動作定義ファイルの状態およびディスクの空き容量**

  動作定義ファイルに設定したディレクトリやファイルが書き込みできる状態になっているか確認してください。また，ディスクの空き容量は十分かどうか確認してください。

14.8.2 クライアントから SOAP メッセージが正しく出力されません。どのように対処すればよいですか？

メソッド呼び出しで例外が発生していないか確認してください。また，ほかのメソッドやほかの SOAP サービスで，SOAP メッセージが正しく出力されているかどうかを確認することで，特定のメソッド呼び出しだけの問題かどうか確認できます。

メソッドを呼び出すときに，SOAP クライアントライブラリが動作しているかどうかを確認するには，トレースファイルを利用できます。出力されるトレースファイルのメッセージテキストの次に示すキーワードを参照することで，どこまで動作しているか把握できます。

- exit Service::createCall

  クライアントの通信に必要なクラスが生成されたときに出力します。

- enter Call::invoke(Object[])：RPC 形態の場合
- enter SOAPConnectionImpl::call(SOAPMessage, Object)：メッセージング形態の場合
  通信に必要なパラメタなどを指定したあとに，SOAP メッセージの送信のための処理を開始したときに出力します。
- exit HTTPSender::invoke
  サーバから返信用の SOAP メッセージが戻ってきたときに出力します。

トレースファイルに出力される形式に関しては，「14.4　トレースファイル」を参照してください。

14.8.3　SOAP メッセージがサーバに届きません。原因として何が考えられますか？

クライアントから SOAP メッセージが正しく出力されているのに，サーバに SOAP メッセージが届かない場合は，送信先の指定が間違っているおそれがあります。クライアントからスタブを利用するときに指定する送信先 URL を確認してください。次に送信先 URL の例を示します。

```
http://hostname:8080/WebApp1/services/UserInfo
```

送信先 URL の形式，ポート番号，コンテキストルートの指定内容などが正しいか確認してください。

また，Web サーバが正しく動作しているか確認してください。SOAP による通信ができないだけでなく，HTML ファイルが表示できない場合は，通信路に問題がないかについても確認してください。

14.8.4　SOAP サービスでメッセージを処理できません。どのように対処すればよいですか？

サーバに SOAP メッセージが届いているのに，正しい SOAP メッセージを返せない場合は，送信された SOAP メッセージの内容が正しいか，および送信された SOAP メッセージが正しく処理されているか確認します。

送信された SOAP メッセージの内容が正しいかについては，アプリケーションログの内容を確認してください。送信された SOAP メッセージが，SOAP サービスとして公開しているサービスに対応したものかどうかや，メソッド名，パラメタ名，およびパラメタに指定している内容などを確認してください。

SOAP メッセージが SOAP サービスで正しく処理されているかの確認は，サーバトレースを利用できます。出力されたファイルに記述された次のキーワードを参照することで，どこまで動作しているか把握できます。

- invoke User RPC Service：RPC 形態の場合
- invoke User Messaging Service：メッセージング形態の場合

SOAP 通信基盤から SOAP サービスの実装クラスを呼び出す直前に出力します。

- return User RPC Service：RPC 形態の場合
- return User Messaging Service：メッセージング形態の場合
  SOAP サービスの実装クラスの呼び出しから SOAP 通信基盤に戻ってきたときに出力します。

トレースファイルに出力される形式に関しては，「14.4　トレースファイル」を参照してください。

また，ユーザが作成する SOAP サービスの実装そのものが正しく動作しているかどうかについても確認してください。

14.8.5　サーバから返信用 SOAP メッセージが正しく出力されていません。原因として何が考えられますか？

サーバから返信用 SOAP メッセージが正しく出力されない場合は，SOAP サービスで処理した結果として正しい SOAP メッセージを返せないことが考えられます。「14.8.4　SOAP サービスでメッセージを処理できません。どのように対処すればよいですか？」を参照し，送信されてきた SOAP メッセージの内容が正しいか，およびそのメッセージを正しく処理できているか確認してください。

14.8.6　クライアントへ返信用 SOAP メッセージが届いていません。どのように対処すればよいですか？

SOAP サービスでは正しく処理できているのに，クライアントへ意図した SOAP メッセージが届いていない場合は，サーバトレースおよびクライアントトレースによって，SOAP メッセージの内容が正しいか，およびそのメッセージが正しく処理されているかを確認してください。トレースファイルの確認方法については，次に示す内容を参照してください。

- 「14.8.2　クライアントから SOAP メッセージが正しく出力されません。どのように対処すればよいですか？」
- 「14.8.4　SOAP サービスでメッセージを処理できません。どのように対処すればよいですか？」

14.8.7　クライアントで返信用メッセージを処理できません。どのように対処すればよいですか？

クライアントに返信用 SOAP メッセージが届いているのに，クライアント側で処理できない場合は，クライアントトレースによって，SOAP メッセージの内容が正しいか，およびそのメッセージが正しく処理されているかを確認してください。トレースファイルの確認方法については，「14.8.2　クライアントから SOAP メッセージが正しく出力されません。どのように対処すればよいですか？」を参照してください。

14.8.8　Application Server で提供している SOAP クライアント以外のクライアントから通信できません。原因として何が考えられますか？

Application Server で提供しているクライアント以外のクライアントでは，SOAP 1.1 に対するサポート範囲や解釈の違いから，Application Server で提供する SOAP 通信基盤で扱えない形式の SOAP メッセージを出している可能性があります。SOAP 通信基盤が扱える SOAP 1.1 の範囲については，「12.1　SOAP 1.1 との対応」を参照してください。

Application Server で提供しているクライアント以外のクライアントと，Application Server 上で動作している SOAP サービス間の通信ができない場合，クライアント部分に関してはご使用のクライアントから提供される情報に従って対処してください。

Application Server が提供している SOAP 通信基盤のサポート範囲外の SOAP メッセージか，Application Server 以外のクライアントの動作に問題があるかを切り分けるには，同じ WSDL を利用して Application Server の SOAP クライアントライブラリから Application Server 上で動作している SOAP サービスへの通信ができるかどうかを試すのが有効です。

また，サポート範囲外のメッセージを出していないか，および内容に問題ないかを確認するには，ご使用のクライアントと Application Server の SOAP クライアントライブラリを使用した場合に出力されるメッセージの比較が有効です。Application Server の SOAP クライアントライブラリを使用した場合のメッセージの確認は，アプリケーションログを参照してください。アプリケーションログについては，「14.5 アプリケーションログ」を参照してください。

14.8.9　プロキシサーバを越えて外部の SOAP サービスを利用できますか？

利用できます。外部の SOAP サービスを利用する場合，クライアント定義ファイルのプロキシオプションに必要な情報を設定します。プロキシオプションの設定項目については，「10.6.6　プロキシオプション」を参照してください。クライアント定義ファイルの設定については，「10.3　クライアント定義ファイルの設定」を参照してください。

14.8.10　他社製品と接続するときに HTTP ヘッダ中の SOAPAction 値の設定が必要なケースがあります。Application Server の SOAP クライアントで SOAPAction 値は設定できますか？

設定できます。RPC 形態の SOAP アプリケーションを例に，SOAPAction 値を設定する方法を説明します。

RPC 形態の場合，WSDL 内で SOAPAction 値を指定する個所を直接変更して，その WSDL からスタブのソースコードを自動生成することで，送信メッセージの SOAPAction 値を指定します。Java2WSDL コマンドで WSDL を生成すると，SOAPAction 値はデフォルトで「""」が設定されます。次に，SOAPAction 値の指定個所を抜粋した WSDL の例を示します。

- 自動生成された WSDL の例（デフォルト値）

```
  <wsdl:binding name="UserInfoSoapBinding" type="intf:UserInfo">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="getUserData">
       <soap:operation soapAction="" />
       <wsdl:input>
           <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace
="http://localhost" use="literal" />
       </wsdl:input>
```

- SOAPAction 値を変更した WSDL の例

```
  <wsdl:binding name="UserInfoSoapBinding" type="intf:UserInfo">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="getUserData">
       <soap:operation soapAction="http://localhost" />
       <wsdl:input>
           <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace
="http://localhost" use="literal" />
       </wsdl:input>
```

14.8.11　SOAP サービスを呼び出す時にユーザ認証が必要な場合，どのようにユーザ ID やパスワードを設定すればよいですか？

次のように，呼び出すサービスの URL にユーザ ID とパスワードを指定することで，ユーザ認証が必要な SOAP サービスを呼び出すことができます。

```
http://<ユーザID>:<パスワード>@<ホスト名またはIPアドレス>:<ポート番号>/・・・
```

なお，SOAP 通信基盤では，BASIC 認証だけに対応しています（DIGEST/FORM/CLIENT-CERT 認証には対応していません）。

次に，ユーザ ID とパスワードを指定する実装例を示します。

- RPC 形態の場合

  サービスクラス名：UserInfoServiceLocator

  インタフェースクラス名：UserInfo

  サービスの URL：http://hostname:8080/WebApp1/services/UserInfo

  設定するユーザ ID：user1

  設定するパスワード：pass1

  このような場合，次のように実装します。

```
UserInfoServiceLocator uis = new UserInfoServiceLocator();
java.net.URL endpoint = new java.net.URL("http://user1:pass1@hostname:8080/WebApp1/servic
es/UserInfo");
UserInfo ui = uis.getUserInfo(endpoint);
```

14.8.12　SOAP メッセージで要素の省略を含んだ配列を使用するにはどのようにしたらよいですか？

メッセージング形態の SOAP アプリケーションを利用してください。RPC 形態の SOAP アプリケーションでは，配列の要素を省略した SOAP メッセージを生成することはできません。

配列要素の省略とは，SOAP メッセージ中で，配列に offset を指定した場合，position を指定した場合，配列要素に nil 値を指定した場合，配列のサイズよりも要素の数が少なかった場合です。

他社製品のクライアントから接続する場合などで，配列要素の省略を含んだ SOAP メッセージを受け取る RPC サービスを実装するときの注意点を次に示します。

RPC 呼び出しの引数として，int[]や double[]のように，Java の基本データ型を要素に持つ配列を使用した場合，配列要素が省略されていると，サービスから SOAP Fault が返ります。

表 14–15　基本データ型配列要素が省略されていた場合の SOAP Fault の内容

SOAP Fault の要素	内容
faultcode	{http://c4web.cosminexus.com}:Server.userException
faultstring	KDCCF9000-E C4Fault exception occurred. Detail = java.lang.NullPointerException
faultactor	なし
detail	なし

RPC サービス実装で配列要素の省略を使用しなくてはならない場合は，配列要素に基本データ型のラッパークラスを使用してください。

基本データ型のラッパークラスとは，java.lang.Integer クラスや java.lang.Double クラスなどの，基本データ型の値を格納することのできるクラスです。

配列要素がクラスである場合，省略された要素には null 値が格納されます。

14.8.13　SOAP クライアントから SSL を使用して SOAP サービスに接続するにはどのようにすればよいですか？

SSL を使用するためには，暗号化に使用する鍵を格納するキーストアや証明書を格納するトラストストアを用意し，それらの情報を SOAP クライアント実行時に設定する必要があります。また，SSL 認証の種類によって必要となるストア情報が異なります。次に SSL 認証の種類と，それに必要なストア情報の関係を示します。

表 14–16　SSL 認証の種類とストア情報の関係

認証の種類 ※1	キーストア情報 ※2		トラストストア情報 ※2	
	キーストア	キーストア パスワード	トラストストア	トラストストア パスワード
サーバ認証	－	－	○	○
クライアント認証	○	○	○ ※3	○ ※3
相互認証	○	○	○	○

（凡例）

○：設定します。

－：設定しません。

注※1

認証の種類によって，それぞれ次のような準備が必要です。

サーバ認証

サーバ認証をするためには，サーバ証明書を発行した機関のルート証明書をあらかじめトラストストアに格納しておく必要があります。

クライアント認証

クライアント認証をするためには，クライアント証明書をあらかじめキーストアに格納しておく必要があります。中間証明書が存在する場合は，トラストストアに格納しておきます。

相互認証

サーバ認証およびクライアント認証の両方の準備をする必要があります。

注※2

キーストア情報およびトラストストア情報は，次の各プロパティに設定します。

- キーストア：javax.net.ssl. keyStore
- キーストアパスワード：javax.net.ssl. keyStorePassword
- トラストストア：javax.net.ssl.trustStore
- トラストストアパスワード：javax.net.ssl.trustStorePassword

注※3

中間証明書が存在する場合は設定する必要があります。

Application Server を使用する場合は，usrconf.properties に必要なプロパティを設定します。usrconf.properties については，マニュアル「アプリケーションサーバ リファレンス 定義編(サーバ定義)」を参照してください。

次に使用例を示します。

（例）サーバ認証をする場合

usrconf.properties ファイルに次のプロパティを設定します。

- javax.net.ssl.trustStore=<トラストストアの絶対パス>
- javax.net.ssl.trustStorePassword=<トラストストアパスワード>

（例）相互認証をする場合

usrconf.properties ファイルに次のプロパティを設定します。

- javax.net.ssl.keyStore=<キーストアの絶対パス>
- javax.net.ssl.keyStorePassword=<キーストアパスワード>
- javax.net.ssl.trustStore=<トラストストアの絶対パス>
- javax.net.ssl.trustStorePassword=<トラストストアパスワード>

注意事項

- usrconf.properties 中にストアのパスワードを設定する場合，usrconf.properties の読み取り権限を限定するなど，第三者にパスワードを読み取られないように注意してください。コマンドラインを使用する場合も，同様に注意してください。
- SSL の接続に対して使用可能なプロトコルは，JDK のバージョンに依存します。使用可能なプロトコルについては，JDK のドキュメントを参照してください。また，JDK の https.protocols プロパティによる使用可能なプロトコルの変更はできませんので注意してください。
- SOAP クライアントから，SSL プロトコルに対応した SOAP サーバに接続する場合，エンドポイントアドレスに含まれるホスト名と，証明書のホスト名が一致しているかどうかを検証します。ホスト名が一致していない場合，KDCCP0012-E メッセージの C4Fault が発生します。

  ホスト名の検証を無効にしたい場合は，サーバ定義ファイルまたはクライアント定義ファイルの次のプロパティに false を設定してください。

  サーバ定義ファイル

  c4web.application.<識別子>.hostname_verification.enable

  クライアント定義ファイル

  c4web.application.hostname_verification.enable

  サーバ定義ファイルについては，「10.2　サーバ定義ファイルの設定」を参照してください。クライアント定義ファイルについては，「10.3　クライアント定義ファイルの設定」を参照してください。

14.8.14　性能解析トレースが出力されません。どのように対処すればよいですか？

性能解析トレースを出力するには，動作環境に応じた設定が必要です。「14.7.3　性能解析トレース使用時の注意事項」を参照して，動作環境を正しく設定してください。

付録

付録 A　SOAP アプリケーションの移行

SOAP アプリケーション開発支援機能，および SOAP 通信基盤をバージョンアップした場合，旧バージョンの環境で開発した SOAP アプリケーションを最新バージョンの環境で使用するには，SOAP アプリケーションを移行する必要があります。ここでは，バージョンごとに SOAP アプリケーションの移行手順について説明します。

付録 A.1　互換モードで動作する SOAP アプリケーションを移行する場合の注意事項

バージョン 09-00 以降では，SOAP 通信基盤 互換モードを同梱していません。SOAP 通信基盤 互換モードを利用している場合は，SOAP アプリケーションを移行してください。

(1)　RPC 形態の SOAP アプリケーションの移行

RPC 形態では，WSDL 定義から生成されるスタブ，スケルトン，サービスデプロイ定義ファイル（server-config.xml），DD（web.xml），およびユーザ実装を修正する必要があります。

ここでは，SOAP アプリケーションを新規開発した場合と，既存の Java クラスを利用して開発した場合に分けて移行手順を説明します。

新規開発した場合の移行手順（RPC）

1. WSDL2Java コマンドを使用して WSDL からクライアントのスタブ，サービスのスケルトン，およびサービスデプロイ定義ファイルを再生成します。
2. ユーザ実装部分で SOAP 通信基盤が提供する API を使用している場合は，パッケージ名「com.cosminexus.c4web.*」を「com.cosminexus.cws.*」に変更します。
3. DD（web.xml）中のパッケージ名「com.cosminexus.c4web.*」を「com.cosminexus.cws.*」に変更します。
4. コンパイル後，J2EE サーバに配置するためのアーカイブ（WAR ファイル）を作成します。
   アーカイブの作成については，「3.9　アーカイブ（WAR ファイル）の作成」を参照してください。
5. WAR ファイルをデプロイします。
   これで SOAP アプリケーションの移行は完了です。

既存の Java クラスを利用して開発した場合の移行手順（RPC）

1. WSDL2Java コマンドを使用して WSDL からクライアントのスタブを再生成します。
2. ユーザ実装部分で SOAP 通信基盤が提供する API を使用している場合は，パッケージ名「com.cosminexus.c4web.*」を「com.cosminexus.cws.*」に変更します。
3. Java2WSDD コマンドを使用してサービスデプロイ定義ファイルを再生成します。

4. DD（web.xml）中のパッケージ名「com.cosminexus.c4web.*」を「com.cosminexus.cws.*」に変更します。
5. J2EE サーバに配置するためのアーカイブ（WAR ファイル）を作成します。
   アーカイブの作成については，「3.9　アーカイブ（WAR ファイル）の作成」を参照してください。
6. WAR ファイルをデプロイします。
   これで SOAP アプリケーションの移行は完了です。

(2)　EJB を利用した SOAP アプリケーションの移行

EJB を利用した SOAP アプリケーションを移行するには，WSDL 定義から生成されるスタブ，サービスデプロイ定義ファイル（server-config.xml），DD（web.xml），およびユーザ実装を修正する必要があります。

ここでは，SOAP アプリケーションを既存の EJB を利用して開発した場合の移行手順について説明します。

移行手順

1. WSDL2Java コマンドを使用してクライアントのスタブを再生成します。
2. ユーザ実装部分で SOAP 通信基盤が提供する API を使用している場合は，パッケージ名「com.cosminexus.c4web.*」を「com.cosminexus.cws.*」に変更します。
3. Java2WSDD コマンドを使用してサービスデプロイ定義ファイルを再生成します。
4. DD（web.xml）中のパッケージ名「com.cosminexus.c4web.*」を「com.cosminexus.cws.*」に変更します。
5. J2EE サーバに配置するためのアーカイブ（WAR ファイル）を作成します。
   アーカイブの作成については，「3.9　アーカイブ（WAR ファイル）の作成」を参照してください。
6. WAR ファイルをデプロイします。
   これで SOAP アプリケーションの移行は完了です。

(3)　メッセージング形態の SOAP アプリケーションの移行

メッセージングを利用した SOAP アプリケーションでは，SAAJ を利用した SOAP アプリケーションの形態に変更する必要があります。SAAJ を利用した SOAP アプリケーションの開発例については，「6.　メッセージング形態の SOAP アプリケーションの開発例」を参照してください。

付録 B　添付ファイル使用時に作成される退避ファイル

添付ファイル付きの SOAP メッセージを受信した場合，メモリ使用量を抑制するために，添付ファイルのサイズに応じて退避ファイルが作成されます。ここでは，作成される退避ファイルについて説明します。

付録 B.1　退避ファイルの詳細

ここでは，退避ファイルの格納先，サイズ，および個数について説明します。

(1)　退避ファイルの格納先

退避ファイルの格納先を次の表に示します。

表 B-1　退避ファイルの格納先

分類	説明
出力先ディレクトリ	動作定義ファイルの「c4web.attachment.attachment_temp_directory」または「c4web.attachment.<識別子>.attachment_temp_directory」で設定したディレクトリです。動作定義ファイルの設定については，「10.　動作定義ファイルおよび実行時オプションの設定項目」を参照してください。動作定義ファイルで退避ファイルの出力先ディレクトリを設定していない場合や，設定したディレクトリが使用できない場合，退避ファイルは次の場所に格納されます※。 Windows の場合： <Application Server のインストールフォルダ>¥c4web¥attachments Windows 以外の場合： /opt/Cosminexus/c4web/attachments
ファイル名	ファイル名は，SOAP アプリケーションが動作する形態によって異なります。ファイル名は変更しないでください。 コマンドラインで動作する SOAP アプリケーションの場合： CosmiSOAP_<数字とアンダースコアから構成される文字列>.tmp J2EE サーバ上で動作する SOAP アプリケーションの場合： CosmiSOAP_j2ee_<J2EE サーバ名>_<数字とアンダースコアから構成される文字列>.tmp

注※

動作定義ファイルで退避ファイルの出力先ディレクトリを設定する場合，次のことを確認してください。

・指定したディレクトリが存在すること

・指定したディレクトリにアクセスできること

(2)　退避ファイルのサイズ

退避ファイルと添付ファイルは 1 対 1 で対応しているため，個々の退避ファイルのサイズは，受信する添付ファイルのサイズと同じサイズとなります。

退避ファイルのディスク使用量の見積もりについては，「付録 B.3　退避ファイルのディスク使用量の見積もり方法」を参照してください。

(3)　退避ファイルの数

受信した添付ファイルのうち，退避ファイルを生成する条件に当てはまる添付ファイルの数と同じ数だけ退避ファイルが生成されます。一つの SOAP メッセージに複数の添付ファイルがある場合，一度の SOAP メッセージ受信で複数の退避ファイルが生成されることもあります。

退避ファイルを生成する条件については，「付録 B.2(1)　退避ファイルが生成される条件」を参照してください。

付録 B.2　退避ファイルの生成と削除

ここでは，退避ファイルが生成される条件，および退避ファイルが削除されるタイミングについて説明します。

(1)　退避ファイルが生成される条件

受信したメッセージの添付ファイルのサイズが 16KB より大きい場合，退避ファイルが生成されます。一つの SOAP メッセージに複数の添付ファイルがある場合は，サイズが 16KB より大きい添付ファイルに対してだけ退避ファイルが生成されます。添付ファイルのサイズが 16KB 以下の場合は，退避ファイルは生成されません。

(2)　退避ファイルの削除

退避ファイルは，次の場合に SOAP 通信基盤によって自動的に削除されます。

- 添付ファイルへの参照がすべて破棄された場合

  JavaVM のファイナライズ処理の実行時に，添付ファイルへの参照がなければ，退避ファイルは自動的に削除されます。添付ファイルへの参照が残っている場合，退避ファイルは削除されません。

  退避ファイルの占有期間を短くするためにも，JAF の javax.activation.DataHandler オブジェクトへの参照や，javax.activation.DataHandler オブジェクトから取得できる java.io.InputStream オブジェクトへの参照などは，不要になった時点ですぐに破棄することをお勧めします。
- SOAP サービスの呼び出しが終了する場合

  受信した添付ファイルに対応する退避ファイルは自動的に削除されます。
- JavaVM が終了する場合

  J2EE サーバの停止や，コマンドラインから実行した SOAP クライアントの終了など，JavaVM が終了するタイミングで退避ファイルは自動的に削除されます。

また，API を使用して退避ファイルを削除することもできます。

Management クラスの disconnectClientIDtoCurrentThread メソッドが呼び出された時に，退避ファイルが削除されます。このとき，connectClientIDtoCurrentThread メソッドの呼び出しから disconnectClientIDtoCurrentThread メソッドの呼び出しまでに，現在のスレッドで受信した添付ファイルに対応する退避ファイルが削除されます。

なお，connectClientIDtoCurrentThread メソッドの呼び出しから disconnectClientIDtoCurrentThread メソッドの呼び出しまでに，SOAP サービスを複数回呼び出した場合は，対応するすべての退避ファイルが削除されます。

Management クラスの詳細については，「3.7.2　Management クラスと ClientID クラスの使用」を参照してください。

注意事項

J2EE サーバのプロセスを強制停止した場合など，ユーザが意図しないで SOAP 通信基盤が停止されたときは，退避ファイルが削除されないことがあります。この場合は退避ファイルの格納先を確認し，必要に応じて手動で退避ファイルを削除してください。

付録 B.3　退避ファイルのディスク使用量の見積もり方法

ここでは，退避ファイルのディスク使用量を見積もり方法について説明します。

退避ファイルのディスク使用量の算出式を次に示します。

- SOAP サービスの場合

退避ファイルの最大ディスク使用量＝
〈受信添付ファイルの最大サイズ※1〉×〈受信添付ファイルの最大数※2〉×〈同時実行数※3〉

＜例＞

次の条件の場合，退避ファイルの最大ディスク使用量は，100,000MB となります。

- 受信できる添付ファイルの最大サイズ：100MB
- 受信できる添付ファイルの最大数：100 個
- 同時実行数：10

- SOAP クライアントの場合

退避ファイルの最大ディスク使用量＝
〈受信添付ファイルの最大サイズ※1〉×〈受信添付ファイルの最大数※2〉×〈サービス呼び出し回数※4〉

＜例＞

次の条件の場合，退避ファイルの最大ディスク使用量は，10,000MB となります。

- 受信できる添付ファイルの最大サイズ：100MB
- 受信できる添付ファイルの最大数：100 個
- SOAP サービスを呼び出す回数：1 回

注※1

受信添付ファイルのサイズは，生成される退避ファイルのサイズに影響します。生成される退避ファイルのサイズが大きくなるとディスク使用量が増えるため，注意してください。受信できる添付ファイルの最大サイズは，動作定義ファイルの「c4web.attachment.receive_max_attachment_size」または「c4web.attachment.<識別子>.receive_max_attachment_size」で設定できます。動作定義ファイルの設定については，「10.　動作定義ファイルおよび実行時オプションの設定項目」を参照してください。

注※2

受信添付ファイルの個数は，生成される退避ファイルの個数に影響します。生成される退避ファイルの個数が多くなるとディスク使用量が増えるため，注意してください。受信できる添付ファイルの最大の個数は，動作定義ファイルの「c4web.attachment.receive_max_attachment_count」または「c4web.attachment.<識別子>.receive_max_attachment_count」で設定できます。動作定義ファイルの設定については，「10.　動作定義ファイルおよび実行時オプションの設定項目」を参照してください。

注※3

同時実行数は，生成される退避ファイルの個数に影響します。生成される退避ファイルの個数が多くなるとディスク使用量が増えるため，注意してください。J2EE サーバ上で動作する SOAP サービスの場合，J2EE サーバの同時実行スレッド数の設定や，Web サーバの同時接続数の設定をすることで，同時実行数を制限できます。

J2EE サーバの同時実行スレッド数の設定については，マニュアル「アプリケーションサーバ 機能解説 基本・開発編(Web コンテナ)」を参照してください。Web サーバの同時接続数の設定については，マニュアル「HTTP Server」を参照してください。

注※4

サービス呼び出し回数は，生成される退避ファイルの個数に影響します。生成される退避ファイルの個数が多くなるとディスク使用量が増えるため，注意してください。

サービス呼び出し回数とは，Management クラスの connectClientIDtoCurrentThread メソッドの呼び出しから disconnectClientIDtoCurrentThread メソッドの呼び出しまでに，現在のスレッドで SOAP クライアントが SOAP サービスを呼び出す回数です。

SOAP サービスを繰り返し呼び出す場合，および複数の SOAP サービスを呼び出す場合，すべての呼び出しの回数を合計してください。

付録 C　WS-I Basic Profile に適合したシステムを構築するための注意事項

付録 C.1　WS-I Basic Profile について

WS-I Basic Profile Version 1.0a は，Web サービスの相互接続性を向上させるために，SOAP 仕様および WSDL 仕様を明確にすることを目的として Web Services Interoperability Organization（WS-I）が制定したガイドラインです。このガイドラインには，WSDL 定義を記述するときの記述のあいまいさを軽減したり，SOAP アプリケーションの振る舞いを統一したりする要件が含まれています。このガイドラインに対応すると，同じく対応したほかのシステムとの相互接続性を高めることができます。

SOAP アプリケーション開発支援機能および SOAP 通信基盤では，このガイドラインに沿った SOAP アプリケーションおよび SOAP アプリケーションを利用するクライアントアプリケーションを開発，運用できます。しかし，そのためには SOAP アプリケーションおよびクライアントアプリケーション開発時にもガイドラインを意識する必要があります。

付録 C.2　新しく SOAP アプリケーションを開発する場合

新しく SOAP アプリケーションを開発する場合，SOAP アプリケーション開発支援機能使用時に WS-I Basic Profile に対応させるために必要となる設定は特にありません。ガイドラインを意識して SOAP アプリケーションを開発してください。

付録 C.3　WS-I Basic Profile 1.0a に対応するための注意事項

SOAP 通信基盤によって WS-I Basic Profile 1.0a に対応した SOAP アプリケーションを運用する場合の注意事項について説明します。

(1)　メッセージ送信時の HTTP バージョンについて

SOAP 通信基盤を利用して SOAP クライアントを運用する場合，SOAP メッセージは HTTP/1.0 を使用して送信されます。

(2)　WSDL 定義とメッセージとの整合性の確認順序について

SOAP 通信基盤では，SOAP メッセージを受信した場合，WSDL 定義とメッセージとの整合性を次に示す順に確認してください。矛盾がある場合には，対応する faultcode を持つフォルトを返します。また，メッセージが XML 的に不正だった場合などに，この順番に関係なく SOAP 通信基盤独自の faultcode"Server"を持つフォルトを返す場合があります。

1. SOAP エンベロープの名前空間が正しいかどうか

   正しくなければ VersionMismatch を返します。

2. サービスが処理しなければならないヘッダを処理しているかどうか

処理していなければ MustUnderstand を返します。

3. それ以外の矛盾がある場合

SOAP 通信基盤独自の Server を返します。

(3) faultcode について

サービス実装内で faultcode が必要になった場合は，独自の名前空間を用意して独自の faultcode を生成するか，SOAP1.1 の faultcode を使用してください。そのとき，ドット表記で詳細コードを追加することは避けてください。

（例）

×：SomeFaultCode.moreDetailCode

○：SomeFaultCode

独自の faultcode を用意しても，ドット表記で詳細コードを追加すると，WS-I Basic Profile に対応しない可能性があります。

SOAP メッセージヘッダ中に，自サービスがあて先として指定されているにもかかわらず，対処できない mustUnderstand 属性が"1"であるヘッダがある場合，MustUnderstand の faultcode を持つフォルトを作成してください。MustUnderstand 以外の faultcode を持つフォルトを作成すると，WS-I Basic Profile に対応しない可能性があります。

(4) RPC でリテラルエンコーディングを使用する場合の body 要素の namespace 属性について

リテラルエンコーディングに対応した WSDL 定義を生成する場合は，body 要素には namespace 属性を付加してください。また，その属性値には名前空間を表す絶対 URI を指定してください。Body 要素に namespace 属性がなかったり属性値が絶対 URI でなかったりする場合，WS-I Basic Profile に対応しない可能性があります。

(5) ユーザ定義配列の型名について

SOAP 通信基盤では，開発支援コマンドを使用してユーザ定義クラスの配列を含むソースから WSDL 定義を生成する場合，ユーザ定義型名に SequenceOf というプレフィクスを付加した型名を用意します。形名は変更できますが，名称の選択には注意してください。プレフィクスを ArrayOf などに変更すると，WS-I Basic Profile に対応しない可能性があります。

付録 D　SOAP 通信基盤が返す HTTP ステータスコード

SOAP 通信基盤が返す HTTP ステータスコード，および HTTP ステータスコードを返す条件を次に示します。

表 D–1　SOAP 通信基盤が返す HTTP ステータスコード

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	SOAP サービスの呼び出しが正常終了した場合。
2	400 Bad Request	SOAP サービスが受信した SOAP メッセージが次に示す状態の場合。 • 整形式の XML 文書ではない不正な XML である • SOAP サービスの WSDL に従っていない
3	415 Unsupported Media Type	SOAP サービスが受信した HTTP リクエストの Content-Type ヘッダの値が，次に示す値以外の場合。 • "text/xml" • "multipart/related"
4	500 Internal Server Error	項番 1～項番 3 の場合以外で，SOAP サービスが SOAP フォルトを返す場合。

付録 E　用語解説

マニュアルで使用する用語について

マニュアル「アプリケーションサーバ & BPM/ESB 基盤 用語解説」を参照してください。

索引

記号

A

B

C

D

E

T

U

W

X

あ

い

か

き

く

こ

さ

し

す

せ

そ

た

て

と

な

は

ひ

ふ

ほ

ま

み

め

も

ゆ

よ

ら

り

れ