

# JEUS Webサービスガイド

JEUS v8.0



Copyright © 2016 TmaxSoft Co., Ltd. All Rights Reserved.

## Copyright Notice

Copyright © 2016 TmaxSoft Co., Ltd. All Rights Reserved.

45, Jeongjail-ro, Bundang-gu, Seongnam-si, Gyeonggi-do, 13613, South Korea

## Restricted Rights Legend

All TmaxSoft Software (JEUS®) and documents are protected by copyright laws and international convention. TmaxSoft software and documents are made available under the terms of the TmaxSoft License Agreement and this document may only be distributed or copied in accordance with the terms of this agreement. No part of this document may be transmitted, copied, deployed, or reproduced in any form or by any means, electronic, mechanical, or optical, without the prior written consent of TmaxSoft Co., Ltd.

Nothing in this software document and agreement constitutes a transfer of intellectual property rights regardless of whether or not such rights are registered) or any rights to TmaxSoft trademarks, logos, or any other brand features. This document is for information purposes only. The company assumes no direct or indirect responsibilities for the contents of this document, and does not guarantee that the information contained in this document satisfies certain legal or commercial conditions. The information contained in this document is subject to change without prior notice due to product upgrades or updates. The company assumes no liability for any errors in this document.

このソフトウェア(JEUS®)マニュアルの内容とプログラムは、日本国の著作権法および国際条約によって保護されています。マニュアルの内容とプログラムは、TmaxSoft Co., Ltd.との使用許諾契約書の下でのみ使用することができ、マニュアルは使用許諾契約で許可されている範囲を除いては、配布または複製することができません。TmaxSoftの書面による事前の承諾を得ることなく、このマニュアルの全部または一部を電子的または機械的な方法を問わず、転送、複製、配布したり、または二次的著作物を作成する等の行為を一切禁じます。

このソフトウェアのマニュアルとプログラムの使用許諾契約は、いかなる場合においても、マニュアル及びプログラムと関連する知的財産権(登録の有無を問わず)を譲渡するものと解釈されず、TmaxSoftのブランド、ロゴ、商標等の使用権限を与えるものではありません。マニュアルは、情報を提供する目的でのみ提供しており、これに伴う契約上の直接的ないしは間接的な責任を負わず、マニュアルの内容は法律上もしくは商業的な特定の条件が満たされることを保証しません。マニュアルの内容は、製品のアップグレード及び修正により、その内容が予告なく変更されることがあり、内容上の誤りがないことを保証しません。

## Trademarks

JEUS® is registered trademark of TmaxSoft Co., Ltd.

JEUS®は、TmaxSoft Co., Ltd.の登録商標です。

Java and Solaris are registered trademarks of Oracle Corporation and its subsidiaries and affiliates.

Java、Solarisは、Oracle Corporation及びその子会社、関連会社の登録商標です。

Microsoft, Windows, and Windows NT are registered trademarks or trademarks of Microsoft Corporation.

Microsoft、Windows、Windows NTは、Microsoft Corporationの登録商標または商標です。

HP-UX is a registered trademark of Hewlett Packard Enterprise Company.

HP-UXは、Hewlett Packard Enterprise Companyの登録商標です。

---

AIX is a registered trademark of International Business Machines Corporation.

AIXは、International Business Machines Corporationの登録商標です。

UNIX is a registered trademark of X/Open Company, Ltd.

UNIXは、X/Open Company, Ltd.の登録商標です。

Linux is a registered trademark of Linus Torvalds.

Linuxは、Linus Torvaldsの登録商標です。

Other products and company names are trademarks or registered trademarks of their respective owners.

その他、記載されている会社名、製品名などは、各社の商号、商標または登録商標です。

The names of companies, systems, and products mentioned in this manual may not necessarily be indicated with a trademark symbol (TM, ®).

本マニュアルに記載されている会社名、システム名、製品名などには必ずしも商標表示(TM、®)を付記しておりません。

### **Open Source Software Notice**

Some modules or files of this product are subject to the terms of the following licenses. : APACHE2.0, CDDL1.0, EDL1.0, OPEN SYMPHONY SOFTWARE1.1, TRILEAD-SSH2, Bouncy Castle, BSD, MIT, SIL OPEN FONT1.1

Detailed Information related to the license can be found in the following directory : \${INSTALL\_PATH}/lib/licenses

この製品の一部ファイルまたはモジュールは、APACHE2.0、CDDL1.0、EDL1.0、OPEN SYMPHONY SOFTWARE1.1、TRILEAD-SSH2、Bouncy Castle、BSD、MIT、SIL OPEN FONT1.1のライセンスに準拠します。

### **文書情報**

文書名: JEUS Webサービスガイド

発行日: 2016年10月14日

ソフトウェアバージョン: JEUS v8.0

ガイドバージョン: v2.1.1

---



# 目次

このガイドについて .....	xxiii
<b>第1章 Webサービス .....</b>	<b>1</b>
1.1. 基本概念 .....	1
1.2. Webサービスの標準 .....	2
1.2.1. SOAP標準 .....	3
1.2.2. WSDL標準 .....	4
1.2.3. UDDI標準 .....	4
1.2.4. JAX-WS、JAXB、StAX標準 .....	4
1.2.5. JAX-RPC標準 .....	5
1.2.6. SAAJ標準 .....	5
1.3. SOAPメッセージの交換とエンコーディング .....	5
<b>第2章 JEUSのWebサービス .....</b>	<b>7</b>
2.1. 基本構造 .....	7
2.2. Webサービスの設計 .....	8
2.2.1. Webサービス・バックエンドの選択 .....	8
2.2.2. RPC方式とドキュメント方式の選択 .....	9
2.2.3. Webサービスの実装方式の選択 .....	9
2.2.4. SOAPメッセージ・ハンドラーの作成 .....	11
<b>I. JEUS 8のWebサービス .....</b>	<b>13</b>
<b>第3章 JEUS Webサービスの実装 .....</b>	<b>15</b>
3.1. 概要 .....	15
3.2. JavaクラスWebサービスの実装 .....	15
3.3. EJB Webサービスの実装 .....	20
3.4. WSDLによるWebサービスの実装 .....	25
<b>第4章 Webサービスの作成とデプロイ .....</b>	<b>31</b>
4.1. 概要 .....	31
4.2. JavaクラスWebサービスの作成とデプロイ .....	31
4.3. EJB Webサービスの作成とデプロイ .....	32
4.4. WSDLによるWebサービスの作成とデプロイ .....	32
4.5. エンドポイントのアドレスの決定方式 .....	33
4.5.1. サーブレット・エンドポイント .....	33
4.5.2. EJBエンドポイント .....	35
<b>第5章 Webサービスの呼び出し .....</b>	<b>39</b>
5.1. 概要 .....	39
5.2. 動的プロキシ方式のWebサービスの呼び出し .....	39
5.2.1. クライアント・アーティファクトの作成 .....	39
5.2.2. Java SEクライアント方式 .....	45
5.2.3. Java EEクライアント方式 .....	47

5.3.	ディスパッチ方式のWebサービスの呼び出し .....	49
<b>第6章</b>	<b>標準バインディング宣言とカスタマイズ .....</b>	<b>51</b>
6.1.	概要 .....	51
6.2.	標準バインディングの宣言 .....	51
6.2.1.	外部文書(ファイル)での直接宣言 .....	52
6.2.2.	WSDL文書内での直接宣言 .....	52
6.3.	標準バインディングのカスタマイズ .....	53
6.3.1.	全体的なバインディング .....	54
6.3.2.	パッケージ名のカスタマイズ .....	54
6.3.3.	Wrappedスタイル .....	55
6.3.4.	非同期化 .....	55
6.3.5.	プロバイダー・インターフェース .....	56
6.3.6.	クラス名のカスタマイズ .....	56
6.3.7.	Javaメソッドのカスタマイズ .....	57
6.3.8.	Javaパラメータのカスタマイズ .....	58
6.3.9.	XMLスキーマのカスタマイズ .....	58
6.3.10.	ハンドラー・チェーンのカスタマイズ .....	59
<b>第7章</b>	<b>ハンドラー・フレームワーク .....</b>	<b>61</b>
7.1.	概要 .....	61
7.2.	ハンドラー・チェーンの優先順位 .....	62
7.3.	ハンドラー・クラスの構成 .....	62
7.3.1.	ハンドラー・クラスの宣言 .....	62
7.4.	ハンドラー・クラスの設定 .....	63
7.4.1.	JavaクラスによるWebサービスの構成 .....	64
7.4.2.	WSDLによるWebサービスの構成 .....	64
7.4.3.	クライアントの構成 .....	65
7.5.	ハンドラー・チェーンを使用するWebサービスの例 .....	65
7.6.	Webサービスのハンドラー・フレームワークの実行 .....	68
<b>第8章</b>	<b>プロバイダーとディスパッチ・インターフェース .....</b>	<b>71</b>
8.1.	概要 .....	71
8.2.	サービス・エンドポイントのプロバイダー・インターフェース .....	71
8.2.1.	プロバイダー・インターフェース .....	71
8.2.2.	プロバイダー・インターフェースの例 .....	73
8.2.3.	プロバイダー・インターフェースの例の実行 .....	74
8.3.	クライアントのディスパッチ・インターフェース .....	75
8.3.1.	ディスパッチ・インターフェース .....	75
8.3.2.	ディスパッチ・インターフェースの例 .....	77
8.3.3.	ディスパッチの例の実行 .....	78
8.4.	XML/HTTPバインディング .....	79
8.4.1.	RESTful Webサービス .....	79
8.4.2.	RESTful Webサービスの例 .....	81
8.4.3.	RESTful Webサービスの例の実行 .....	82

<b>第9章 非同期Webサービス .....</b>	<b>85</b>
9.1. 概要 .....	85
9.2. クライアントの非同期オペレーション .....	85
9.2.1. 非同期メソッドを持つサービス・エンドポイント・インターフェース・スタブの使用方 法 .....	85
9.2.2. ディスパッチ・インターフェースの使用法 .....	90
9.3. 非同期Webサービス .....	91
9.3.1. 非同期Webサービスの設定 .....	91
<b>第10章 MIME添付メッセージの送信 .....</b>	<b>93</b>
10.1. 概要 .....	93
10.2. MTOM/XOP .....	93
10.2.1. 基本動作 .....	94
10.2.2. 添付バイナリ・データ・サイズの設定 .....	95
10.2.3. MTOM/XOPの例 .....	96
10.2.4. MTOM/XOPの例の実行 .....	98
10.3. swaRef .....	99
10.3.1. swaRefの使用法 .....	99
10.3.2. swaRefの例 .....	100
10.3.3. swaRefの例の実行 .....	101
10.4. ストリーミング方式による添付ファイルの処理方法 .....	102
<b>第11章 Fast Infosetを使用したWebサービス .....</b>	<b>105</b>
11.1. 概要 .....	105
11.2. Fast Infosetの使用 .....	107
11.2.1. コンテンツ・ネゴシエーション .....	107
11.3. Fast Infosetの例 .....	107
11.4. Fast Infoset Webサービスの実行 .....	108
<b>第12章 JAX-WS JMS転送 .....</b>	<b>111</b>
12.1. 概要 .....	111
12.2. JAX-WS JMS転送の設定 .....	111
12.2.1. JMSサーバーの設定 .....	111
12.2.2. Webサービスの作成 .....	112
12.2.3. WSDLの設定 .....	113
12.2.4. Webサービス・クライアントの作成 .....	113
<b>第13章 Webサービスのポリシー .....</b>	<b>115</b>
13.1. 概要 .....	115
13.2. Webサービスのポリシー(WS-Policy) .....	115
13.3. サーバー・ポリシーの設定 .....	117
13.3.1. WSDLによるWebサービスの構成 .....	117
13.3.2. JavaクラスによるWebサービスの構成 .....	117
13.4. クライアントのポリシー設定 .....	119
<b>第14章 Webサービス・アドレッシング .....</b>	<b>121</b>

14.1.	概要 .....	121
14.2.	サーバーの設定 .....	123
14.2.1.	Javaクラスによる設定 .....	123
14.2.2.	WSDLによる設定 .....	125
14.3.	クライアント設定 .....	125
14.4.	例 .....	126
14.5.	例の実行 .....	126
<b>第15章</b>	<b>高信頼性メッセージング技術 .....</b>	<b>131</b>
15.1.	概要 .....	131
15.2.	サーバーの設定 .....	132
15.2.1.	WSDLによる設定 .....	132
15.2.2.	Javaクラスによる設定 .....	134
15.3.	クライアントの設定 .....	134
15.4.	例 .....	134
15.5.	例の実行 .....	135
<b>第16章</b>	<b>Webサービス・トランザクション .....</b>	<b>137</b>
16.1.	概要 .....	137
16.2.	サーバーの設定 .....	137
16.2.1.	WSDLによる設定 .....	138
16.2.2.	Javaクラスによる設定 .....	139
16.3.	クライアントの設定 .....	139
16.4.	コーディネーター・サービス .....	139
16.5.	Webサービス・トランザクションの例 .....	140
<b>第17章</b>	<b>Webサービスのセキュリティ .....</b>	<b>141</b>
17.1.	概要 .....	141
17.2.	トランスポート・レベルのセキュリティ .....	141
17.3.	メッセージ・レベルのセキュリティ .....	142
17.3.1.	Webサービスのセキュリティ・ポリシー .....	142
17.3.2.	Webサービスのセキュリティ .....	143
17.3.3.	Webサービスのセキュリティ対話 .....	146
17.3.4.	Webサービスの信頼 .....	147
17.4.	メッセージ・レベルのセキュリティ設定 .....	148
17.4.1.	共通の設定 .....	148
17.4.2.	ユーザー名の認証による対称バインドイングの認証機能の強化 .....	164
17.4.3.	相互認証セキュリティ(Mutual Certificates Security) .....	167
17.4.4.	SSLによるSAML認証 .....	169
17.4.5.	セキュリティ対話(Secure Conversation) .....	171
17.4.6.	Webサービスの信頼(WS-Trust) .....	174
17.4.7.	実行 .....	178
17.5.	JAX-RPC(JEUS 5) Webサービスのセキュリティからのマイグレーション方法 ....	180
17.5.1.	暗号化(Encryption) .....	180
17.5.2.	署名(Signature) .....	183



17.5.3.	タイムスタンプ .....	186
17.5.4.	ユーザー名トークン .....	188
17.6.	アクセス制御が設定されているWebサービスの呼び出し .....	191
17.6.1.	ポータブル・アーティファクトの作成 .....	192
17.6.2.	Webサービスのクライアントの作成 .....	192
<b>第18章</b>	<b>Server-Sent Event .....</b>	<b>193</b>
18.1.	概要 .....	193
18.2.	JAX-RSリソースのServer-Sent Events (SSE)のサポート .....	193
18.2.1.	簡単なSSEリソース・メソッド .....	193
18.3.	JAX-RSクライアントでのSSEイベント処理 .....	195
18.3.1.	EventInputを使用してSSEイベントを読み込む .....	195
18.3.2.	EventSourceを使用した非同期SSEの処理 .....	196
<b>第19章</b>	<b>UDDIの使用 .....</b>	<b>197</b>
19.1.	概要 .....	197
19.1.1.	UDDIの使用 .....	197
19.2.	JEUSにおけるUDDIサーバーの運用 .....	198
19.2.1.	UDDIデータストアの作成 .....	199
19.2.2.	UDDIサーバーのデプロイ .....	199
19.2.3.	UDDIサーバーの構成設定 .....	200
19.2.4.	新しいユーザーの追加 .....	200
19.2.5.	UDDIサーバーの実行 .....	202
19.3.	JEUSサーバーでのUDDIエクスプローラの使用 .....	203
19.3.1.	UDDIレジストリーのクエリ .....	204
19.3.2.	UDDIレジストリーの公開 .....	206
19.3.3.	JEUS UDDIエクスプローラの設定 .....	212
19.4.	UDDIクライアントの作成 .....	213
19.4.1.	UDDIクライアントの作成 .....	213
19.4.2.	UDDIクライアントのコンパイル .....	215
19.4.3.	UDDIクライアントの実行 .....	215
19.5.	XMLデジタル署名の使用方法 .....	216
19.5.1.	デジタル署名 .....	216
19.5.2.	UDDIクライアントにおけるXML署名の作成方法 .....	217
19.5.3.	UDDIクライアントにおけるXML署名の検証 .....	218
19.6.	UDDIサブスクリプションの使用方法 .....	219
19.6.1.	基本概念 .....	219
19.6.2.	UDDIサブスクリプションの作成方法 .....	219
19.6.3.	UDDIサブスクリプションの例 .....	220
19.6.4.	UDDIサブスクリプション・クライアント・プログラミング .....	223
19.6.5.	Eメール通知を受けるためのUDDIサーバーの設定 .....	225
19.7.	UDDI WSDLの公開の使用方法 .....	225
19.7.1.	UDDI WSDLの公開 .....	225
19.7.2.	wsdl2uddiの使用 .....	226

<b>第20章 JEUS WebサービスのXML</b>	<b>229</b>
20.1. 概要	229
20.2. JAXB(XMLバインディングのためのJavaアーキテクチャー)	229
20.2.1. バインディング・コンパイラ(XJC)関連のプログラミング	230
20.2.2. Schemagen関連プログラミング	233
20.3. JAXP(XMLを扱うためのJava標準API)	236
20.3.1. StAX(JavaストリーミングのXMLパーサ)	236
<b>II. JAX-RPCサービスのサポート</b>	<b>237</b>
<b>第21章 JAX-RPC Webサービスの実装</b>	<b>239</b>
21.1. 概要	239
21.2. JavaクラスによるWebサービスの実装	239
21.2.1. 実装例	239
21.2.2. Javaクラスの作成規則	240
21.3. EJB Webサービスの実装	241
21.3.1. 実装例	241
21.3.2. EJB Webサービスの作成規則	243
21.4. WSDLによるWebサービスの実装	243
21.5. SAAJの使用	279
<b>第22章 JAX-RPC Webサービスの作成とデプロイ</b>	<b>245</b>
22.1. JavaクラスWebサービスの作成とデプロイ	245
22.1.1. サービス設定ファイルの作成	245
22.1.2. WSDLファイルとJAX-RPCマッピング・ファイルの作成	246
22.1.3. WebサービスのDDファイルの作成	247
22.1.4. パッケージングとデプロイ	249
22.2. EJB Webサービスの作成とデプロイ	252
22.2.1. サービス設定ファイルの作成	253
22.2.2. WSDLファイルとJAX-RPCマッピング・ファイルの作成	253
22.2.3. WebサービスのDDファイルの作成	255
22.2.4. パッケージングとデプロイ	256
<b>第23章 JAX-RPC Webサービスの呼び出し</b>	<b>261</b>
23.1. JAX-RPC Webサービスの呼び出し(Java SEクライアント)	261
23.1.1. スタブ・クライアント	261
23.1.2. DIIクライアント	265
23.2. JAX-RPC Webサービスの呼び出し(Java EEクライアント)	268
23.2.1. Java EEクライアント・プログラミング・モデル	268
23.2.2. Java EEクライアント・プログラミングの手順	269
23.2.3. Java EEクライアントの作成と例	271
<b>第24章 JAX-RPC WebサービスのSOAPメッセージ・ハンドラーの作成</b>	<b>279</b>
24.1. SAAJの使用	279
24.1.1. SOAPメッセージの作成	279
24.1.2. SAAJ文書の作成と使用	280

24.1.3.	SAAJを使用したSOAPメッセージの送信 .....	281
24.2.	SOAPメッセージ・ハンドラーの作成 .....	281
24.2.1.	メッセージ・ハンドラーの作成 .....	282
24.2.2.	メッセージ・ハンドラーとハンドラー・チェーンの設計 .....	283
24.2.3.	ハンドラー・インターフェースの実装 .....	284
24.2.4.	Java EE WebサービスのDDファイルの作成 .....	285
24.2.5.	クライアントにおけるSOAPメッセージ・ハンドラーの使用 .....	286
24.2.6.	ファイルを送受信するWebサービスとクライアントの例 .....	286
<b>第25章</b>	<b>JAX-RPC Webサービス設定ファイルの作成 .....</b>	<b>295</b>
25.1.	JAX-RPC WebサービスのDDファイルの作成 .....	295
25.2.	Webサービスのマッピング・ファイルの作成 .....	300
25.2.1.	JAX-RPCマッピング・ファイルの内容 .....	300
25.2.2.	JAX-RPCマッピング・ファイルの作成 .....	301
<b>第26章</b>	<b>JAX-RPC Webサービスのデータ型 .....</b>	<b>307</b>
26.1.	概要 .....	307
26.2.	JavaとXML型のマッピング .....	308
26.2.1.	組み込み型マッピング .....	308
26.2.2.	配列 .....	310
26.2.3.	ユーザー定義型：JAX-RPC Value型 .....	310
26.3.	JAX-RPC Value型の使用 .....	311
26.3.1.	JAX-RPC Value型を使用するWebサービスの作成 .....	311
26.3.2.	JAX-RPC Value型を使用するWebサービス・クライアントの作成 .....	312
26.4.	Holderクラス .....	314
26.4.1.	組み込み型Holderクラス .....	314
26.4.2.	ユーザー定義型のためのHolderクラスの作成 .....	316
26.5.	ExceptionとSOAPフォルト .....	319
26.6.	MIME型のDataHandler型へのマッピング .....	320
26.6.1.	Wsdl2javaでのdataHandlerOnlyオプションの使用 .....	320
26.7.	Doc/Literalでのデータ・バインディングの不使用 .....	321
26.7.1.	Wsdl2javaでのnoDataBindingオプションの使用 .....	321
<b>第27章</b>	<b>JAX-RPC Webサービスのセキュリティ .....</b>	<b>325</b>
27.1.	概要 .....	325
27.2.	トランスポート・レベルのセキュリティ .....	325
27.3.	メッセージ・レベルのセキュリティ .....	326
27.3.1.	Webサービスのセキュリティの適用 .....	326
27.3.2.	Webサービスのセキュリティ・アーキテクチャー .....	327
27.3.3.	JEUS Webサービスのセキュリティ設定 .....	328
27.3.4.	パスワード・コールバック・クラスの作成 .....	330
27.3.5.	JEUS Webサービス・サーバーのセキュリティ適用例 .....	331
27.3.6.	JEUS Webサービス・クライアントのセキュリティ適用例 .....	335
27.3.7.	セキュリティAPIを利用したJEUS Webサービス・クライアントの作成 .....	337
27.4.	アクセス制御の設定 .....	343

27.4.1.	JavaクラスWebサービスのアクセス制御の設定 .....	343
27.4.2.	EJB Webサービスのアクセス制御の設定 .....	344
27.4.3.	アクセス制御が設定されているWebサービスの呼び出し .....	346
<b>用語集</b>	.....	<b>349</b>
<b>索引</b>	.....	<b>351</b>

## 図目次

[図 2.1]	JEUS Webサービスの構造 .....	7
[図 7.1]	メッセージ・コンテキスト間の関連付け .....	61
[図 7.2]	ハンドラー・フレームワーク .....	62
[図 11.1]	Fast Infosetの直列化とパーシング .....	106
[図 17.1]	WS-Trust .....	198
[図 19.1]	情報階層と主要XMLタグの名前 .....	198
[図 19.2]	JEUS UDDIレジストリーの初期画面 .....	203
[図 19.3]	UDDI検索画面 .....	204
[図 19.4]	UDDIレジストリーで検索した結果画面 .....	205
[図 19.5]	登録したエントリー画面 .....	206
[図 19.6]	プロバイダーの修正画面 .....	207
[図 19.7]	サービスの追加画面 .....	208
[図 19.8]	tModelの保存結果画面 .....	209
[図 19.9]	バインディング・テンプレートの追加画面 .....	210
[図 19.10]	バインディング・テンプレートの保存結果画面 .....	211
[図 19.11]	JEUS UDDIエクスプローラでのユーザー管理画面 .....	212
[図 19.12]	JEUS UDDIエクスプローラでのユーザー登録画面 .....	213
[図 19.13]	UDDIサブスクリプションの流れ .....	219
[図 19.14]	WSDL descriptionとUDDI Data Structureのマッピング .....	226
[図 19.15]	UDDIに公開されたwsdl .....	227
[図 24.1]	添付を持つSOAPの構造 .....	279
[図 27.1]	JEUS Webサービスのセキュリティー・アーキテクチャー .....	327



## 表目次

[表 10.1]	JAXB 2.0 specification of mime:expectedContentType to Java type mapping .....	94
[表 19.1]	uddi.properties .....	200
[表 19.2]	PUBLISHER表の列 .....	202
[表 21.1]	Required Mappings : Java to MIME .....	244
[表 26.1]	組み込み型のXML/Java型のマッピング .....	308
[表 26.2]	組み込み型Holderクラス .....	314





## 例目次

[例 3.1]	<< AddNumbersImpl.java >> .....	16
[例 3.2]	<< build.xml >> .....	17
[例 3.3]	<< AddNumbersImplService.wsdl >> .....	18
[例 3.4]	<< AddNumbers.java >> .....	19
[例 3.5]	<< AddNumbersResponse.java >> .....	20
[例 3.6]	<< AddNumbersImpl.java >> .....	20
[例 3.7]	<< build.xml >> .....	22
[例 3.8]	<< AddNumbersImplService.wsdl >> .....	23
[例 3.9]	<< AddNumbers.java >> .....	24
[例 3.10]	<< AddNumbersResponse.java >> .....	24
[例 3.11]	<< AddNumbers.wsdl >> .....	25
[例 3.12]	<< build.xml >> .....	27
[例 3.13]	<< AddNumbersPortType.java >> .....	28
[例 3.14]	<< AddNumbers.java >> .....	29
[例 3.15]	<< AddNumbersResponse.java >> .....	29
[例 3.16]	<< AddNumbersImpl.java >> .....	30
[例 4.1]	<< web.xml >> .....	33
[例 4.2]	<< AddNumbersImpl.java >> .....	34
[例 4.3]	<< AddNumbersImpl.java >> .....	34
[例 4.4]	<< AddNumbersImpl.java >> .....	35
[例 4.5]	<< AddNumbersImpl.java >> .....	35
[例 4.6]	<< AddNumbersImpl.java >> .....	36
[例 4.7]	<< AddNumbersImpl.java >> .....	36
[例 4.8]	<< AddNumbersImpl.java >> .....	37
[例 4.9]	<< AddNumbersImpl.java >> .....	37
[例 4.10]	<< AddNumbersImpl.java >> .....	38
[例 5.1]	<< http://host:port/AddNumbers/AddNumbersImplService?wsdl >> .....	40
[例 5.2]	<< build.xml >> .....	41
[例 5.3]	<< AddNumbersImplService.java >> .....	42
[例 5.4]	<< AddNumbersImpl.java >> .....	44
[例 5.5]	<< AddNumbers.java >> .....	44
[例 5.6]	<< AddNumbersResponse.java >> .....	45
[例 5.7]	<< AddNumbersClient.java >> .....	46
[例 5.8]	<< AddNumbersClient.java >> .....	48
[例 6.1]	<< custom-client.xml >> .....	52
[例 6.2]	<< custom-client.xml >> .....	52
[例 6.3]	<< AddNumbers.wsdl >> .....	53
[例 6.4]	<< custom-client.xml >> .....	54
[例 6.5]	<< custom-client.xml >> .....	54
[例 6.6]	<< custom-client.xml >> .....	55

[例 6.7]	<< custom-client.xml >> .....	55
[例 6.8]	<< custom-client.xml >> .....	56
[例 6.9]	<< custom-client.xml >> .....	56
[例 6.10]	<< custom-client.xml >> .....	57
[例 6.11]	<< custom-client.xml >> .....	57
[例 6.12]	<< custom-client.xml >> .....	57
[例 6.13]	<< custom-client.xml >> .....	58
[例 6.14]	<< custom-client.xml >> .....	58
[例 6.15]	<< custom-client.xml >> .....	58
[例 6.16]	<< custom-client.xml >> .....	59
[例 6.17]	<< custom-client.xml >> .....	59
[例 7.1]	<< MyLogicalHandler.java >> .....	63
[例 7.2]	<< MySOAPHandler.java >> .....	63
[例 7.3]	<< MyLogicalHandler.java >> .....	63
[例 7.4]	<< MyServiceImpl.java >> .....	64
[例 7.5]	<< handlers.xml >> .....	64
[例 7.6]	<< LoggingHandler.java >> .....	65
[例 7.7]	<< handlers.xml >> .....	66
[例 7.8]	<< AddNumbersImpl.java >> .....	67
[例 7.9]	<< AddNumbersClient.java >> .....	67
[例 7.10]	<< custom-client.xml >> .....	67
[例 8.1]	<< AddnumbersImpl.java >> .....	73
[例 8.2]	<< AddNumbersClient.java >> .....	77
[例 8.3]	<< AddNumbersImpl.java >> .....	81
[例 8.4]	<< AddNumbersClient.java >> .....	82
[例 9.1]	<< custom-schema.xml >> .....	86
[例 9.2]	<< build.xml >> .....	86
[例 9.3]	<< AddNumbersClient.java >> .....	87
[例 9.4]	<< AddNumbersClient.java >> .....	88
[例 9.5]	<< AddNumbersClient.java >> .....	89
[例 9.6]	<< AddNumbersImpl.java >> .....	91
[例 9.7]	<< web.xml >> .....	91
[例 10.1]	<< hello.wsdl >> .....	97
[例 10.2]	<< hello.wsdl >> .....	100
[例 10.3]	<< AttachmentApp.java >> .....	103
[例 11.1]	<< AddNumbersClient.java >> .....	108
[例 12.1]	<< domain.xml >> .....	111
[例 12.2]	<< AddNumbersImpl.java >> .....	112
[例 12.3]	<< AddNumbers.wsdl >> .....	113
[例 12.4]	<< AddNumbersClient.java >> .....	113
[例 13.1]	<< service-config.xml >> .....	118
[例 14.1]	<< AddnumbersImpl.java >> .....	123
[例 14.2]	<< AddnumbersImpl.java >> .....	123

[例 14.3]	<< Addnumbers.wsdl >> .....	124
[例 14.4]	<< AddNumbersImpl.java >> .....	126
[例 15.1]	<< AddNumbers.wsdl >> .....	132
[例 15.2]	<< service-config.xml >> .....	134
[例 15.3]	<< service-config.xml >> .....	135
[例 16.1]	<< AddNumbers.wsdl >> .....	138
[例 16.2]	<< AddnumbersImpl.java >> .....	139
[例 16.3]	<< AddNumbersClient.jsp >> .....	140
[例 17.1]	<< jeus-webservices-config.xsd >> .....	149
[例 17.2]	<< jeus-webservices-config.xsd >> .....	149
[例 17.3]	<< jeus-webservices-config.xsd >> .....	150
[例 17.4]	<< jeus-webservices-config.xsd >> .....	151
[例 17.5]	<< jeus-webservices-config.xsd >> .....	152
[例 17.6]	<< jeus-webservices-config.xsd >> .....	153
[例 17.7]	<< jeus-webservices-config.xsd >> .....	153
[例 17.8]	<< jeus-webservices-config.xsd >> .....	154
[例 17.9]	<< jeus-webservices-config.xsd >> .....	155
[例 17.10]	<< jeus-webservices-config.xsd >> .....	156
[例 17.11]	<< jeus-webservices-config.xsd >> .....	157
[例 17.12]	<< jeus-webservices-config.xsd >> .....	158
[例 17.13]	<< jeus-webservices-config.xsd >> .....	159
[例 17.14]	<< jeus-webservices-config.xsd >> .....	163
[例 17.15]	<<service-config.xml>> .....	164
[例 17.16]	<< UsernamePasswordValidator.java >> .....	165
[例 17.17]	<<wsit-client.xml>> .....	166
[例 17.18]	<< UsernamePasswordCallbackHandler.java >> .....	166
[例 17.19]	<< service-config.xml >> .....	167
[例 17.20]	<< wsit-client.xml >> .....	168
[例 17.21]	<< service-config.xml >> .....	169
[例 17.22]	<< wsit-client.xml >> .....	170
[例 17.23]	<< SamlCallbackHandler.java >> .....	170
[例 17.24]	<<service-config.xml>> .....	171
[例 17.25]	<<wsit-client.xml>> .....	172
[例 17.26]	<<service-config.xml>> .....	174
[例 17.27]	WS-SecurityPolicy設定 : <<sts.wsdl>> .....	175
[例 17.28]	<<wsit-client.xml>> .....	177
[例 17.29]	<< jeus-webservices-dd.xml >> .....	180
[例 17.30]	<< service-config.xml >> .....	181
[例 17.31]	<< jeus-webservices-dd.xml >> .....	183
[例 17.32]	<< service-config.xml >> .....	184
[例 17.33]	<< jeus-webservices-dd.xml >> .....	186
[例 17.34]	<< service-config.xml >> .....	187
[例 17.35]	<< jeus-webservices-dd.xml >> .....	189

[例 17.36]	<< service-config.xml >> .....	190
[例 17.37]	<< authorization.txt >> .....	191
[例 17.38]	<< AddNumberClient.java >> .....	192
[例 18.1]	簡単なSSEリソース・メソッド .....	193
[例 18.2]	EventInputを使用してSSEイベントを読み込む .....	195
[例 18.3]	EventSourceを使用してSSEイベントを読み込む .....	196
[例 19.1]	<<jeus-web-dd.xml>> .....	199
[例 19.2]	<<web.xml>> .....	200
[例 20.1]	<< build.xml >> .....	232
[例 20.2]	<< Main.java >> .....	232
[例 20.3]	<< Main.java (続き) >> .....	233
[例 20.4]	<< Main.java (続き) >> .....	233
[例 20.5]	<< build.xml >> .....	234
[例 20.6]	<< Main.java >> .....	235
[例 20.7]	<< Main.java (続き) >> .....	236
[例 21.1]	<< HelloIF.java >> .....	241
[例 21.2]	<< Hello.java >> .....	242
[例 21.3]	<< HelloHome.java >> .....	242
[例 21.4]	<< HelloEJB >> .....	242
[例 21.5]	<< build.xml >> .....	243
[例 22.1]	<< webservices.xml >> .....	248
[例 22.2]	<< jeus-webservices-dd.xml >> .....	248
[例 22.3]	<< web.xml >> .....	249
[例 22.4]	<< jeus-web-dd.xml >> .....	250
[例 22.5]	<< application.xml >> .....	251
[例 22.6]	<< service-config.xml >> .....	253
[例 22.7]	<< webservices.xml >> .....	255
[例 22.8]	<< jeus-webservices-dd.xml >> .....	256
[例 22.9]	<< ejb-jar.xml >> .....	257
[例 22.10]	<< jeus-ejb-dd.xml >> .....	257
[例 22.11]	<< application.xml >> .....	259
[例 23.1]	<< build.xml >> .....	261
[例 23.2]	<<ProxyClient.java>> .....	264
[例 23.3]	<< DIIClient.java >> .....	267
[例 23.4]	<< web.xml >> .....	270
[例 23.5]	<< build.xml >> .....	271
[例 23.6]	<< helloClient.jsp >> .....	272
[例 23.7]	<< web.xml >> .....	273
[例 23.8]	<< jeus-web-dd.xml >> .....	274
[例 23.9]	<< helloClient.jsp >> .....	274
[例 23.10]	<< web.xml >> .....	276
[例 24.1]	メッセージ・ハンドラーを定義する<port-component>の例 .....	285
[例 24.2]	<< ServerAttachmentHandler.java >> .....	286

[例 24.3]	<< FileTransfer.java >> .....	289
[例 24.4]	<< FileTransferIF.java >> .....	289
[例 24.5]	<< webservices.xml >> .....	290
[例 24.6]	<< web.xml >> .....	290
[例 24.7]	<< jeus-webservices-dd.xml >> .....	291
[例 24.8]	<< ClientAttachmentHandler.java >> .....	291
[例 24.9]	<< Client.java >> .....	293
[例 25.1]	<< webservices.xml >> .....	295
[例 25.2]	DocLitEchoService : <<webservices.xml >> .....	297
[例 25.3]	DocLitEchoService : << web.xml >> .....	298
[例 25.4]	AddressBookService : <<webservices.xml >> .....	298
[例 25.5]	AddressBookService : << web.xml >> .....	298
[例 25.6]	FileAttachmentService : <<webservices.xml >> .....	299
[例 25.7]	JAX-RPCマッピング・ファイルの構造 .....	301
[例 25.8]	<< AddressBookService.java >> .....	303
[例 26.1]	<<Calculator.java>> .....	311
[例 26.2]	<<CalcData.java>> .....	311
[例 26.3]	<<CalculatorIF.java>> .....	312
[例 26.4]	<<CalcClient.java>> .....	313
[例 26.5]	<< Calculator.java >> .....	314
[例 26.6]	<< CalcClient.javaのrun()メソッド >> .....	315
[例 26.7]	<< CalcData.java >> .....	316
[例 26.8]	<< CalcDataHolder.java >> .....	317
[例 26.9]	<< Calculator.java >> .....	319
[例 27.1]	<<jeus-webservices-dd.xml>> .....	329
[例 27.2]	<<jeus-web-dd.xml>> .....	329
[例 27.3]	<< PWCallback.java >> .....	330
[例 27.4]	<< Ping.java >> .....	333
[例 27.5]	<< PingImpl.java >> .....	333
[例 27.6]	<< PingPWCallback.java >> .....	333
[例 27.7]	<< jeus-webservices-dd.xml >> .....	334
[例 27.8]	<< pingClient.jsp >> .....	336
[例 27.9]	<< jeus-web-dd.xml >> .....	336
[例 27.10]	<< pingClient.jsp >> .....	338
[例 27.11]	<< accounts.xml >> .....	343
[例 27.12]	<< jeus-web-dd.xml >> .....	343
[例 27.13]	<< web.xml >> .....	344
[例 27.14]	<< accounts.xml >> .....	345
[例 27.15]	<< jeus-ejb-dd.xml >> .....	345
[例 27.16]	<< ejb-jar.xml >> .....	345
[例 27.17]	<< jeus-webservices-dd.xml >> .....	346



# このガイドについて

## 対象読者

本書は、基本的なWebサービスの概念を説明し、Java EEプラットフォームでプログラミングが可能な開発者であれば、Webサービスを知らない状態でもWebサービスの作成ができるようになるためのガイドです。JEUS<sup>®</sup>(以下、JEUS) WebエンジンおよびJEUS EJBコンテナでWebサービス・アプリケーションを開発、デプロイ、および保守する開発者を対象としています。

## 前提知識

本書を理解するためには、以下の事項をあらかじめ把握しておく必要があります。

- サープレットの開発とパッケージングについての基本的な知識  
(<http://www.oracle.com/technetwork/java/index.html>)
- EJBについての基本的な知識
- XML、SOAPプロトコル、およびWSDLについての知識

JEUSの基本的な使用方法と製品を理解するには、以下のガイドについてあらかじめ熟知することをお勧めします。

- JEUS 紹介ガイド
- JEUS インストール&スタートガイド

本書のすべてのサンプルと環境構成は、UNIXスタイルに準拠します。Microsoft Windows<sup>™</sup>(以下、Windows)など他の環境で作業を行う場合は、次のような事項を考慮してください。

たとえば、Windowsプラットフォームでは、ディレクトリー区切り子をUNIXスタイルのスラッシュ(/)からWindowsスタイルのバックスラッシュ(\)に変えて使用してください。また、環境変数もWindowsスタイル(%%)に変更して使用してください。

本書で触れているJEUS\_HOMEは、JEUSがインストールされているディレクトリーです。

## 制限事項

本書の内容は、サープレットやEJBなどの基本的なJava EE仕様については取り上げていません。関連内容については関連ガイドを参照してください。

---

## 参考

ソースコードに関連するXMLファイルは、JEUSをインストール後、samplesディレクトリーに位置します。

---



## 本書の構成

本書は、計26章で構成されています。

- [「第1章 Webサービス」](#)

Webサービスの基本概念について説明します。

- [「第2章 JEUSのWebサービス」](#)

JEUSでサポートするWebサービスの概念とサポート方式について説明します。

- [「第3章 JEUS Webサービスの実装」](#)

Webサービスを実装するための方法について説明します。

- [「第4章 Webサービスの作成とデプロイ」](#)

JavaクラスとEJBをエンドポイントで持つWebサービスの生成とデプロイについて説明します。

- [「第5章 Webサービスの呼び出し」](#)

Webサービスを呼び出すためのクライアントの作成と呼び出し方法について説明します。

- [「第6章 標準バインディング宣言とカスタマイズ」](#)

WSDLバインディングのカスタマイズ宣言について説明します。

- [「第7章 ハンドラー・フレームワーク」](#)

Webサービスのハンドラ・フレームワークについて説明します。

- [「第8章 プロバイダーとディスパッチ・インターフェース」](#)

XMLメッセージ・レベルでメッセージを扱う方法について説明します。

- [「第9章 非同期Webサービス」](#)

非同期Webサービスについて説明します。

- [「第10章 MIME添付メッセージの送信」](#)

MIMEメッセージを使用して、より効果的にWebサービス・バイナリデータを扱うための方法について説明します。

- [「第11章 Fast Infosetを使用したWebサービス」](#)

Fast Infosetを使用するWebサービスについて説明します。

- [「第12章 JAX-WS JMS転送」](#)

JMSトランスポートを使用するWebサービスについて説明します。

- [「第13章 Webサービスのポリシー」](#)

Webサービスでポリシー機能を設定し、運用する方法について説明します。

- [「第14章 Webサービス・アドレッシング」](#)

Webサービスでアドレッシング機能を設定して運用する方法について説明します。

- [「第15章 高信頼性メッセージング技術」](#)

信頼性(reliable)のあるWebサービスについて説明します。

- [「第16章 Webサービス・トランザクション」](#)

Webトランザクションについて説明します。

- [「第17章 Webサービスのセキュリティ」](#)

JAX-WS Webサービスのセキュリティ設定について説明します。

- [「第18章 Server-Sent Event」](#)

JAX-RSベースのServer-Sent Eventsについて説明します。

- [「第19章 UDDIの使用」](#)

UDDIの概要と運用および使用について説明します。

- [「第20章 JEUS WebサービスのXML」](#)

JEUS WebサービスがサポートするXMLを扱うための様々な方法について説明します。

- [「第21章 JAX-RPC Webサービスの実装」](#)

JAX-RPC Webサービスを実装する方法について説明します。

- [「第22章 JAX-RPC Webサービスの作成とデプロイ」](#)

JavaクラスとEJBをエンドポイントで持つJAX-RPC Webサービスの生成とデプロイ方法について説明します。

- [「第23章 JAX-RPC Webサービスの呼び出し」](#)

JAX-RPC Webサービスを呼び出すためのクライアントの作成と呼び出し方法について説明します。

- 「第24章 JAX-RPC WebサービスのSOAPメッセージ・ハンドラーの作成」

JAX-RPC Webサービスのメッセージ・ハンドラとSAAJを使用したSOAPメッセージの処理について説明します。

- 「第25章 JAX-RPC Webサービス設定ファイルの作成」

JAX-RPC Webサービスの標準Webサービスのデプロイメント記述子とJAX-RPCマッピング・ファイルの作成方法について説明します。

- 「第26章 JAX-RPC Webサービスのデータ型」

JAX-RPCの仕様に基づき、データ型に関連する、向上したWebサービス・プログラミングについて説明します。

- 「第27章 JAX-RPC Webサービスのセキュリティ」

JAX-RPC Webサービスのセキュリティ設定方法について説明します。

## 表記上の規則

表記	意味
<<AaBbCc123>>	プログラム・ソースコードのファイル名
<Ctrl>+C	CtrlキーとCキーを同時に押す
[Button]	GUIのボタン、メニュー名
太字	強調
「」、『』（鍵カッコ）	関連文書、あるいはガイド内の他の章および節の表示
「入力項目」	画面UI上の入力項目
ハイパーリンク	メール・アカウント、Webサイト
>	メニューの実行順
+----	下位ディレクトリー/ファイル有り
----	下位ディレクトリー/ファイル無し
<b>参考</b>	参照/注意事項
<b>注</b>	注意事項
[図 1.1]	図の名前
[表 1.1]	表の名前
AaBbCc123	Javaコード、XMLドキュメント
[ <i>command argument</i> ]	オプション・パラメータ
< xyz >	「<」と「>」の間の内容は実際に使用される特定の名前または値で置き換えられる
	構文の中の相互に排他的な選択項目の選択肢を示す 例) A B: AとBのいずれかを選択
...	パラメータ、値、または他の情報が繰り返される
\${ }	環境変数

## システム要件

	要求事項
プラットフォーム	Solaris 9, 10, 11
	HP-UX 11.x, 11i, 11iV2
	IBM AIX 5L, 6L, AIX 7L
	MS Windows 2008, 2012, Vista, 7, 8
ハードウェア	最小2GB以上、推奨20GBのハードディスク容量
	推奨1GB以上のメモリー容量
JDK	JDK 7, JDK 8

## 関連文書

ガイド	説明
JEUS 紹介ガイド	JEUSサーバーについて全般的に紹介し、JEUSのアーキテクチャーを含む各構成要素について記述しています
JEUS インストール&スタートガイド	JEUSについて紹介し、JEUSのインストールおよび開始方法について記述しています
JEUS サーバガイド	JEUSシステムおよびサーバーの概要とシステムの管理方法について記述しています
JEUS Webエンジンガイド	JEUS Webエンジンの管理方法、Java EE WARアーカイブとサーブレット/JSPの管理およびデプロイ方法について記述しています
JEUS EJBガイド	JEUS EJBエンジンおよびEJBモジュールのデプロイについて記述しています
JEUS WebAdminガイド	JEUSのWeb管理ツールであるWebAdminを利用したJEUSの設定および制御、モニタリング、クラスタリング、リソースの設定および管理について記述しています

## 参考文献

- SOAP 1.1 Specification (<http://www.w3.org/TR/soap11>)
- SOAP 1.2 Specification (<http://www.w3.org/TR/soap12>)
- WSDL 1.1 Specification (<http://www.w3.org/TR/wsdl>)
- UDDI 2.0 Specification ([http://uddi.org/pubs/ProgrammersAPI\\_v2.htm](http://uddi.org/pubs/ProgrammersAPI_v2.htm), <http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv2> )
- UDDI 3.0 Specification ([http://uddi.org/pubs/uddi\\_v3.htm](http://uddi.org/pubs/uddi_v3.htm))
- JAX-WS 2.2 Specification (<http://jcp.org/en/jsr/detail?id=224>)
- JAX-RPC 1.1 Specification (<http://jcp.org/en/jsr/detail?id=101>)
- SAAJ 1.2 Specification (<http://jcp.org/en/jsr/detail?id=67>)
- SAAJ 1.3 Specification (<http://jcp.org/en/jsr/detail?id=67>)
- JAXR 1.0 Specification (<http://jcp.org/en/jsr/detail?id=93>)
- WS-I Basic Profile 1.0 (<http://www.ws-i.org/Profiles/BasicProfile-1.0.html>)
- Servlet 2.4 Specification (<http://jcp.org/en/jsr/detail?id=154>)
- Servlet 2.5 Specification (<http://jcp.org/en/jsr/detail?id=154>)
- EJB 2.1 Specification (<http://jcp.org/en/jsr/detail?id=153>)
- EJB 3.0 Specification (<http://jcp.org/en/jsr/detail?id=220>)
- JEUS Web Service Annotations and Feature

Webサービスのエンドポイントで利用できるJEUS Webサービス・アノテーションは以下のとおりです。

– Service Scope

- `@jeus.webservices.jaxws.api.ApplicationScope`

デプロイされたWebサービス・エンドポイントは、1つのインスタンスを持ちます。この値がデフォルト値になります。

- `@jeus.webservices.jaxws.api.SessionScope`

HTTPセッションごとに1つのWebサービス・エンドポイントのインスタンスを作成します。他のHTTPセッションは他のインスタンスを持ちます。HTTP送信を使用する場合にのみ使用できます。

- @jeus.webservices.jaxws.api.RequestScope

リクエスト・メッセージがある度に新しいWebサービス・エンドポイントのインスタンスを作成します。

– Transport

- @jeus.webservices.jaxws.api.JMSWebService

JMSベースの送信が可能なWebサービス・エンドポイントです。

Webサービス・クライアントで利用できるJEUS Webサービス・フィーチャーは以下のとおりです。

– jeus.webservices.jaxws.api.LocalTransportFeature

クライアントのWebサービス・ポートがローカル送信を使用し、Webサービス・エンドポイントと通信します。Webサービス・クライアントとエンドポイントが同じJVMに存在する場合にのみ有効です。

- その他の設定およびAPIに関連する詳細内容は以下の文書を参照してください。

– 位置: JEUS\_HOME/docs/reference-book/reference\_wsdl2uddi.html

– xjcコンソール・ツール

位置: JEUS\_HOME/docs/reference-book/reference\_xjc.html

– schemagenコンソール・ツール

位置: JEUS\_HOME/docs/reference-book/reference\_schemagen.html

– tcpmonコンソール・ツール

位置: JEUS\_HOME/docs/reference-book/reference\_tcpmon.html

– Antタスク

位置: JEUS\_HOME/docs/reference-book/webservice\_appendix\_ant\_task.html

– XML Reference - jeus-webservices-config.xml

XML Reference - jeus-webservices-dd.xml

XML Reference - jeus-webservices-dd-client.xml

位置: JEUS\_HOME/docs/reference/schema/index.html

## お問合せ先

### Korea

TmaxSoft Co., Ltd.  
45, Jeongjail-ro, Bundang-gu,  
Seongnam-si, Gyeonggi-do, 13613  
South Korea  
Tel: +82-31-8018-1000  
Fax: +82-31-8018-1115  
Email: [info@tmax.co.kr](mailto:info@tmax.co.kr)  
Web (Korean): <http://www.tmaxsoft.com>  
TechNet: <http://technet.tmaxsoft.com>

### USA

TmaxSoft Inc.  
101 North Wacker Drive, Suite 2014,  
Chicago, IL 60606  
U.S.A  
Tel: +1-312-525-8330  
Email: [info@tmaxsoft.com](mailto:info@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/us\\_en/home](http://www.tmaxsoft.com/us_en/home)

### Japan

TmaxSoft Japan Co., Ltd.  
5F Sanko Bldg, 3-12-16 Mita, Minato-Ku, Tokyo, 108-0073  
Japan  
Tel: +81-3-5765-2550  
Fax: +81-3-5765-2567  
Email: [info@tmaxsoft.co.jp](mailto:info@tmaxsoft.co.jp)  
Web (Japanese): <http://www.tmaxsoft.co.jp>



## China

Beijing TmaxSoft System Software Co., Ltd.  
Room103, No.2 Huizhong Building, Seven Street Shangdi,  
Haidian District, Beijing, 100085  
P.R.China  
Tel: +86-10-6298-8827  
Email: [info@tmaxsoft.com.cn](mailto:info@tmaxsoft.com.cn)  
Web (Chinese): [http://www.tmaxsoft.com/cn\\_en/home\\_cn\\_en](http://www.tmaxsoft.com/cn_en/home_cn_en)

## Brazil

Tmax Brasil Sistemas e Serviços Ltda.  
Av. Copacabana, 177, sala 32~35 Empresarial 18 do Fortel  
Alphaville Barueri, Sao Paulo, 06472-001  
Brazil  
Tel: +55-11-4191-3100  
Fax: +55(11) 4191-3705 (extension#112)  
Email: [info.bra@tmaxsoft.com](mailto:info.bra@tmaxsoft.com)  
Web (Portuguese): [http://www.tmaxsoft.com/br\\_en/home\\_br\\_en](http://www.tmaxsoft.com/br_en/home_br_en)

## Russia

Tmax Rus L.L.C.  
Leninsky prospekt, 113/1 (Park Place Moscow),  
Office 318e, Moscow, 117198  
Russia  
Tel: +7(495)970-01-35  
Email: [info.rus@tmaxsoft.com](mailto:info.rus@tmaxsoft.com)  
Web (Russian): [http://www.tmaxsoft.com/ru\\_ru/home\\_ru\\_ru](http://www.tmaxsoft.com/ru_ru/home_ru_ru)

## Singapore

Tmax Singapore Pte. Ltd.  
430 Lorong 6, Toa Payoh #10-02,  
OrangeTee Building, 319402  
Singapore  
Tel: +65-6259-7223  
Fax: +65-6258-7112  
Email: [info.sg@tmaxsoft.com](mailto:info.sg@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/sg\\_en/home\\_sg\\_en](http://www.tmaxsoft.com/sg_en/home_sg_en)

## United Kingdom

TmaxSoft UK Ltd.  
215 Knyvett House, Watermans Business Park,  
The Causeway, Staines TW18 3BAB  
United Kingdom  
Tel: +44-1784-895005  
Email: [info.uk@tmaxsoft.com](mailto:info.uk@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/gb\\_en/home\\_gb\\_en](http://www.tmaxsoft.com/gb_en/home_gb_en)

## Canada

TmaxSoft Canada, Inc.  
2425 Matheson Blvd East, 8th floor,  
Unit 824 Mississauga, ON, L4W 5K4  
Canada  
Tel: +1-905-361-2888  
Email: [info.canada@tmaxsoft.com](mailto:info.canada@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/ca\\_en/home\\_ca\\_en](http://www.tmaxsoft.com/ca_en/home_ca_en)

## Australia

TmaxSoft Proprietary Limited  
L32, 101 Miller Street, North Sydney 2060  
Australia  
Tel: +91-9845-330-704  
Email: [info.aus@tmaxsoft.com](mailto:info.aus@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/au\\_en/home\\_au\\_en](http://www.tmaxsoft.com/au_en/home_au_en)

## India

TmaxSoft Technologies Private Limited  
Sobha Alexander Plaza, 3rd Floor,  
16/2 Commissariat Road, Bangalore-560025  
India  
Tel: +91-9845-330-704  
Email: [info.india@tmaxsoft.com](mailto:info.india@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/in\\_en/home\\_in\\_en](http://www.tmaxsoft.com/in_en/home_in_en)

## Turkey

TmaxSoft Co., Ltd. Turkey Liaison Office  
Windowist Tower. Eski Buyukdere Cad. No:26,  
Maslak 34467 Istanbul  
Turkey  
Tel: +90-544-553-6045  
Email: [cslee@tmaxsoft.com](mailto:cslee@tmaxsoft.com)  
Web (English): [http://www.tmaxsoft.com/tr\\_en/home\\_tr\\_en](http://www.tmaxsoft.com/tr_en/home_tr_en)



# 第1章 Webサービス

本章では、WebサービスについてXML文書での基本概念と標準について説明します。

## 1.1. 基本概念

Webサービスは、アプリケーションがプラットフォームとプログラミング言語とは独立した形式で互いに通信できるようにする標準化された技術で、標準XMLメッセージングを通じてネットワークに接続できるオペレーションを記述するソフトウェア・インターフェースです。Webサービスはインターネットにのみ接続されている場合、サービス権限を持っているどのユーザーに対してでもビジネスを公開および使用できるよう、メッセージング・プロトコル、プログラミング標準、サービス検出のための利便的な環境などを定義しています。

また、WebサービスはインターネットのURIでアクセス可能なアプリケーション・プログラムです。WebサービスのインターフェースとバインディングはXML文書で定義および記述されます。1つのWebサービスは、インターネット上で公開された他のWebサービスとの相互連動が可能だけでなく、既存のバックエンド・アプリケーション・プログラムとも連動が可能です。

従来のアプリケーション・プログラム・アーキテクチャは、2つのカテゴリに属していました。1つはメインフレームをベースに作動する単一化されたシステムで、もう1つはデスクトップで作動するクライアント・サーバー (client-server) ベースのシステムです。これらのシステムはすべて正常に作動しますが、このようなアーキテクチャ上で動作するプログラムは、そのシステムでのみ作動が可能になるよう調整されているという限界があります。このようなアーキテクチャ内でのプログラムは非常に閉鎖的でアクセスが難しいため、Webに存在する無数のユーザーには役に立ちません。

そのため、ソフトウェア業界はサービス指向アーキテクチャ(SOA : Service-Oriented Architecture)に移行するようになり、このベース上でのアプリケーション・プログラムはWebでの動作により相互適用できるようになりました。アプリケーション・プログラムは、大規模なソフトウェア・システムをより小さくモジュール化された複数のサブシステムに構成させ、このように小さくモジュール化されたサブソフトウェア・システムは、それぞれ多様な技術で実装されるようになりました。また、1つのコンピュータに存在しなくてもよく、再使用できるようになりました。XMLとHTTPのような標準Webプロトコルを使用して、すべてのインターネット・ユーザーが簡単にアクセスできるようにしました。

このようなサービス指向技術は、既に数年前から、RMI、COM、CORBAのような多様な形式の技術で実装されてきているため、最近になって生まれたものではありません。ところが、このような技術はベンダーや実装された技術に従属的であるという短所があります。しかし、Webサービスはこのような短所を克服します。

Webサービスは次のような特徴を持っています。

- Web上でアクセス可能です。
- 自ら記述します。

Webサービスが自身の役割と機能、属性について記述することによって、Webサービス・クライアントがサービスについて理解可能になります。

WSDL(Web Service Description Language)というファイルにサービスを記述し、これを公開することで、他のアプリケーション・プログラムでサービスを使用可能にします。

- HTTPなどの標準のインターネット・プロトコルによって送信されるXMLメッセージを介して、Webサービス・クライアントと通信します。Webサービス・クライアントは、アプリケーション・プログラムであることもあり、他のWebサービスであることもあります。
- リクエスト-レスポンスあるいはワンウェイ方式で作動し、同期あるいは非同期通信で呼び出されることがあります。このような作動方式と通信方式には関係なく、WebサービスとWebサービス・クライアント間で交換される根本的な単位はメッセージです。

Webサービスには以下のようなメリットがあります。

- インターネット公開標準をサポートします。

Webサービスは、SOAP(Simple Object Access Protocol)、WSDL、UDDI(Universal Description, Discovery and Integration)のような公開標準を決め、これを基にして相互運用がされるため、Webサービスをサポートするアプリケーション・プログラムは相互作用に問題はありません。

- プラットフォームと言語に依存しません。
- HTTPのようなWebプロトコルを使用するため、ファイアウォールのような障害にも問題なく、アプリケーション・プログラムに対するアクセスが簡単です。

## 1.2. Webサービスの標準

J2EE(Java 2 Platform, Enterprise Edition)の仕様は、バージョン1.4でWebサービスの重要性が反映され、トランザクションのサポートや、データベースの接続、ライフサイクル管理などのサービスが、アプリケーション・プログラムのソースコードを修正することなくサポートされるようになりました。

Java EE(Java Enterprise Edition, J2EEから名称変更)バージョン5から新しい仕様が追加され、同時に既存の仕様の機能を整理および向上させ、より成長したWebサービスを提供できるようになりました。

現在JEUS Webサービスは、以下の標準に従っています。

- EWS(Implementing Enterprise Web Services)標準
- JAX-WS(Java API for XML-based Web Services)標準
- JAX-RPC(Java API for XML-based RPC)標準
- JAXB(Java Architecture for XML Binding)標準

- SAAJ(SOAP with Attachments API for Java)標準
- Streaming API for XML標準
- Web Service Metadata for the Java Platform標準
- XML, XML Namespace, XML Infoset, XML Schema標準
- SOAP, MTOM, WS-Addressing標準
- WS-ReliableMessaging, WS-Coordination, WS-AtomicTransaction標準
- WSDL, WS-Policy, WS-MetadataExchange標準
- WS-Security Policy, WS-Security, WS-Trust, WS-SecureConversation標準
- UDDI(Universal Description, Discovery and Integration)標準

次の節では、この中からいくつかの重要な標準について紹介します。

### 1.2.1. SOAP標準

SOAP(Simple Object Access Protocol)は、分散環境での情報交換を目的とする軽量のXMLベースのプロトコルです。

SOAPは以下の情報交換方式をサポートします。

- RPC(Remote Procedure Call)方式

RPC方式の情報交換は、要求-応答プロセスを許容し、サービス・エンドポイントはプロシージャ中心のメッセージを受け取り、適切な応答メッセージを作成して返します。

- メッセージ(Message-oriented)方式

送信者がサービスの応答をすぐに要求しなくてもいい場合にも使用可能で、主にビジネスと様々な形式の文字型を交換する必要がある場合に使用されます。メッセージ方式の情報交換はドキュメント方式の情報交換と呼ばれることもあります。

SOAPプロトコルは、以下のような要素で構成されます。

- SOAPメッセージを記述するエンベロープ

エンベロープは、誰がどのようにメッセージを処理するのかについての情報を持っており、メッセージのボディ部分を囲んでいます。

- アプリケーション・プログラムに使用されるデータ型のオブジェクトを説明するエンコード・ルール
- リモート・プロシージャの呼び出しと応答を示すのに必要な遵守事項

SOAPには以下のような特徴があります。

- プロトコルに依存しません。
- 実装言語に依存しません。
- プラットフォームとOSに依存しません。
- SOAP XMLメッセージにMIME型のような形式で追加的なメッセージを加えて送信できます。

### 1.2.2. WSDL標準

WSDL(Web Service Description Language)はWebサービス記述言語ともいい、RPCベースとメッセージ・ベースのサービスを記述するXML形式を意味します。WSDLファイルは、開発ツールやサービス開発者あるいはデプロイ者(サービス提供者)によって作成されます。インターネットに公開されると、そのときからサービスは他のクライアントに知られるようになり、これはクライアントがサービスにアクセスできる環境を構築できるようにします。

クライアント・プログラマー(サービス・コンシューマ)は、公開されたWSDLを使用して、提供されるWebサービスについての情報を取得し、その情報を基にしてWebサービスにアクセスできるようにプロキシやテンプレートなどを作成できます。

### 1.2.3. UDDI標準

UDDI(Universal Description, Discovery and Integration)標準は、Webサービスに関する情報を公開および検索するための規格です。サービス提供者は、UDDIというサービス・コンシューマに既に知らされているオンライン・ストレージにサービスを格納し、サービス・コンシューマはそのストレージにアクセスすることでサービス一覧を検索できます。

### 1.2.4. JAX-WS、JAXB、StAX標準

JAX-WS(Java API for XML-based Web Services)とは、Sun Microsystemsが提供するWebサービスAPIを定義する規格です。これはJAX-RPC標準を発展させたもので、XMLのバインディングのためのJAXB標準と標準ストリーミング・パーサのためのStAX標準、機能が向上した新しいSSAJ標準をベースに統合しました。このようなAPIを採択することによって、J2EE 1.4仕様が要求していた多くの記述子ファイルをすべてJavaSE 5のアノテーション機能に代替できるようになりました。



## 1.2.5. JAX-RPC標準

JAX-RPC(Java API for XML-based RPC)標準とは、Sun Microsystemsが提供するWebサービスのAPIを定義する規格です。Webサービスの当面の課題の1つは、相互互換性を持つサービスの開発です。こういった問題を解決するために、WS-I、OASIS、W3C、そしてSOAPBuildersなど多様な標準化団体が多大な努力をしてきました。

JAX-RPCは、このような相互互換性の問題を解決し、Webサービスの実装に対する柔軟なプログラミングAPIを提供するために、J2EE 1.4で標準として追加されました。JAX-RPCは、ランタイム・フレームワークによって、サービス・ユーザーと開発者に相互互換性の負担を減らしました。これは、後にJAX-WSというWebサービスのためのフレームワークに発展します。

## 1.2.6. SAAJ標準

SAAJ(SOAP with Attachments API for Java)標準は、SOAPプロトコルへの直接アクセスが可能なプログラミング・インターフェースを提供します。低レベルのプロトコル処理に慣れていれば、SOAPメッセージを無理なく作成できます。

## 1.3. SOAPメッセージの交換とエンコーディング

SOAPは、RPC方式とドキュメント方式の情報交換をサポートします。SOAPメッセージは、RPC方式であれドキュメント方式であれ、SOAPメッセージの要素に定義されているEncodingStyle属性によって定義されているエンコーディング方式を使用します。

本節では、このようなSOAPメッセージの特性について記述します。

- SOAPメッセージの構成要素

SOAPメッセージはSOAPエンベロープを持ち、エンベロープはヘッダーとボディの2つの下位要素を持ちます。ヘッダーの下位要素はヘッダー・ブロックで、各ヘッダー・ブロックはデータの論理的な結合といえます。SOAPボディはSOAPメッセージの必須要素で、SOAPメッセージが最終的に伝えようとする内容が格納されている場所です。

- SOAPメッセージのエンコーディング

SOAPは、RPC方式とドキュメント方式の情報交換をサポートします。

- RPC方式

RPC方式とは、リモート・プロシージャの呼び出し(RPC)方式のメッセージ交換方式を指します。クライアントとサーバーが定義されているプログラミング・モデルで作成される必要があり、クライアントはパラメータを持つメソッドを呼び出し、サーバーは応答として1つの値を返します。RPC方式のメッセージ交換方式では、SOAPはメッセージ・ボディの形式を別途定義します。

- ドキュメント方式

ドキュメント方式ではXML文書が交換され、各要素の意味はサーバーとクライアントが解釈を行います。つまり、SOAPメッセージのボディー構造についての制約事項はSOAPで規定しておらず、アプリケーション・プログラムまたは別途決められたXMLスキーマによって、SOAPのボディーに属しているXML文書の構造が決定されます。

## 第2章 JEUSのWebサービス

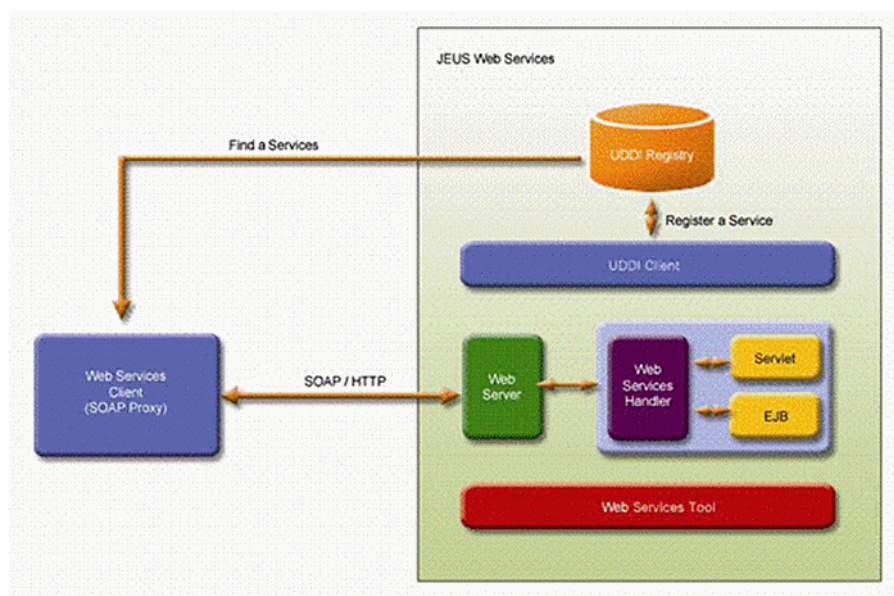
本章では、JEUS Webサービスと、サポートされている仕様の概念について説明します。また、JEUS Webサービスのための設定ファイルとツール、システム変数について説明します。

### 2.1. 基本構造

Java EE 7互換Webアプリケーション・サーバー(WAS)であるJEUSは、Java EE 7で要求するJAX-WS Webサービスをサポートします。JAX-WS Webサービスの最も重要な特徴は、設定ファイルなしで自由に実装できるPOJO(Plain Old Java Object)方式のWebサービスの実装です。JEUS WebサービスはJava EE 7の仕様に準拠するベンダーとのWebサービスの相互運用性が確保されます。

JEUS Webサービスの構造は以下のとおりです。

[図 2.1] JEUS Webサービスの構造



- アノテーションとXML設定ファイル

JEUS Webサービスはビジネス・ロジックでEJBとJavaクラスを使用します。EJBはEJBコンテナの内部で実行され、JavaクラスはWebコンテナの内部で実行されます。

JEUS Webサービスは、JavaSE 5でサポートするアノテーションを使用してWebサービスに関連する情報を設定するJAX-WS Webサービスの実装を基本的にサポートします。また、下位互換性のためにこのような情報をwebservices.xmlのようなWebサービス・デプロイメント記述子(以下、DD)に設定するJAX-RPC Webサービスもサポートしています。

- ツールとユーティリティ

JEUS Webサービスでは、WSDLファイルとJavaクラスの作成および管理のために、コマンドライン・ツールとApache Antツールを使用します。各ツールの使用方法については、本書を参照してください。

Antツール機能を使用するためには、Apache Ant 1.6.1以上のバージョンをインストールする必要があります。<http://ant.apache.org>でApache Antをダウンロードできます。

- システム環境変数

JEUS Webサービスのための特別なシステム環境変数はなく、『JEUS Webエンジンガイド』と『JEUS EJBガイド』で記述されているシステム環境変数のみで充分です。

## 2.2. Webサービスの設計

本節では、Webサービスを設計するプロセスにおいて考慮が必要な事項について説明します。

### 2.2.1. Webサービス・バックエンドの選択

Webサービス・コンポーネントは、WebコンテナやEJBコンテナで実行されるJava EEコンポーネントです。

Webサービスのバックエンドは、次のオブジェクトを公開します。

- Javaクラスファイル

JavaクラスファイルはWebコンテナで実行されます。JavaクラスはEJBを作成するよりスピーディーかつ簡単です。

永続性(Persistence)、セキュリティ、トランザクションなどのようなEJBコンテナで提供する機能がそれほど重要でない場合は、JavaクラスのWebサービス・バックエンドを選択することをお勧めします。

- ステートレス・セッションEJB

EJBはトランザクションが重要なシステムで選択できるプログラミング・モデルで、ステートレス・セッションEJBはEJBコンテナで実行されます。

データベースにアップデートや削除などの作業を多く行うシステムでは、ビジネス・ロジックをEJBで実装する場合に、効率的なトランザクション管理ができます。

上記のようにトランザクションが重要ではない場合でも、EJBバックエンドはWebサービス・バックエンドとして選択されることがあり、CMP(Container Managed Persistence)がその例です。CMPについての詳細はEnterprise Java Beansのガイドを参照してください。

既存のビジネス・ロジックは必要に応じて、EJB(ステートレスセッションBeanまたはCMP)で構成されます。ステートレス・セッションBeanでビジネス・ロジックが実装されている状態で既存のEJBロジックをWebサービスで公開したい場合には、ステートレス・セッションBeanのWebサービス・バックエンドを選択することをお勧めします。

通常、EJB Webサービスはステートレス・セッションBeanのみをバックエンドで許可するため、CMPはWebサービスと一見関係がなさそうに見えるかもしれませんが、ステートレス・セッションBeanがCMPで実装されているビジネス・ロジックへのインターフェースの役割をし、ステートレス・セッションBeanがWebサービスに公開されると、CMPビジネス・ロジックをWebサービスに公開されます。既存のCMPビジネス・ロジックの長所をそのまま生かしながら、Webサービスの機能を追加する場合に、ステートレス・セッションBeanのWebサービス・バックエンドを選択することをお勧めします。

## 2.2.2. RPC方式とドキュメント方式の選択

JEUS WebサービスはRPC方式とドキュメント方式をサポートします。

- RPC方式

WSDL 1.1仕様によると、RPC方式はSOAPメッセージがWebサービス・オペレーションの引数と戻り値を含み、ドキュメント方式はSOAPメッセージがXML文書を含みます。また、RPC方式はリモート・プロシージャの呼び出し(RPC)方式のメッセージング交換方式を示し、クライアントとサーバーが定義されたプログラミング・モデルで作成される必要があります。クライアントは引数を持つメソッドを呼び出し、サーバーは応答として1つの値を返します。

RPC方式では引数の数に制限はありません。

- ドキュメント方式

XML文書が交換され、各要素の意味はサーバーとクライアントが解釈を行います。つまり、SOAPメッセージのボディー構造に対する制約事項はSOAPで規定されておらず、アプリケーション・プログラムまたは別途決められたXMLスキーマによってSOAPボディーに属しているXML文書の構造が決定されます。

JEUS Webサービスで提供するドキュメント方式には、標準ドキュメント方式とWRAPPED方式があります。

区分	説明
標準ドキュメント方式	それぞれのWebサービス・オペレーションは1つの引数(XML文書)のみを持つことができます
WRAPPED方式	様々な引数を1つの複合データ型で囲んで1つの引数を作り、SOAPメッセージに添付するため、Webサービス・オペレーションは複数の引数を持つことができます

## 2.2.3. Webサービスの実装方式の選択

一般的にWebサービスの実装は、サービス・エンドポイントまたはWSDLから開始できます。これは開発者の好みと開発環境によって選択され、JEUS WebサービスでJAX-WS方式とJAX-RPC方式で実装することができます。

- サービス・エンドポイントから開始

サービスの実装がシンプルかつ簡単で、直感的に実装が可能です。

- WSDLからの開始

実装言語に中立的な相互運用が可能なサービスの実装に適合します。

## サービス・エンドポイントから開始

サービス・エンドポイントを実装してからWSDLを作成する方法です。この方法は、サービスの実装を容易かつ直感的に実現します。開発者が慣れた開発環境、つまりJava環境で他のプラットフォームとの相互運用性を心配することなく、サービス・エンドポイントとサービス実装体を開発できます。その次に、作成されたJavaコードからプラットフォームに独立的なWSDLを生成します。

このように作成されたWSDLがプラットフォームに独立的に定義されるために、JAX-WSとJAX-RPCはサービス・エンドポイントからWSDLを作成する方法を提供します。この方式は、Java開発者にはWebサービス開発を始める最も簡単な方法です。

- JAX-WS方式

JAX-WS方式での留意事項は、サービス設定に関する情報を記述しなくてもいいという点です。JAX-WS仕様のために、JEUS Webサービスの実装ではサービス設定ファイルを別途作成しなくても、wsngenツール・キットによってWebサービスを実装できます。

JAX-WS方式のWebサービス実装についての詳細は「[第3章 JEUS Webサービスの実装](#)」を参照してください。

- JAX-RPC方式

JAX-RPC方式での留意事項は、サービス設定に関する情報を記述する必要があるという点です。JAX-RPC仕様は、JavaからWSDLへのマッピングについての詳しい定義をしています。しかし、サービス・エンドポイントのインターフェースだけでは完全なWSDLを作成できません。サービス・エンドポイントのインターフェースはWSDLのサービスとバインディングについての情報を提供できません。JAX-RPC仕様のために、JEUS Webサービスの実装ではサービス設定ファイルを別途作成し、この情報をJava-to-WSDLツール・キットに提供する必要があります。

JAX-RPC方式のWebサービス実装についての詳細は「[第21章 JAX-RPC Webサービスの実装](#)」を参照してください。

## WSDLからの開始

WSDLを先に作成し、これからWebサービスを作成する方法です。

Webサービスの目標の1つは、プラットフォーム間の相互運用性です。この方法は開発言語に中立的で、相互運用性の中心的なサービスを作成しやすい方法です。Webサービスの実装から作成されたWSDLはプラットフォームの特性が反映されます。つまり、Java寄りです。しかし、WSDLから始まったWebサービスの

記述は、Webサービスの記述に使用されているXMLタイプと用語において、より中立的です。つまり、Java環境に慣れていない開発者にも使いやすい一般的なコマンド法としてWSDLを作成できます。

しかしこの方式は、開発者がWSDLについてより深く理解する必要があります。JAX-WS Webサービスの実装については「[第3章 JEUS Webサービスの実装](#)」を参照してください。また、JAX-RPC Webサービスの実装については「[第21章 JAX-RPC Webサービスの実装](#)」を参照してください。

## 2.2.4. SOAPメッセージ・ハンドラーの作成

SOAPメッセージのヘッダーやボディー、あるいはアタッチメントを直接扱う場合、メッセージ・ハンドラーの作成が必要です。

- JAX-WS方式

JAX-WSハンドラーのフレームワークを利用できます。詳細については「[第7章 ハンドラー・フレームワーク](#)」を参照してください。

- JAX-RPC方式

Webサービスの場合、SOAPメッセージのJAX-RPC標準ハンドラーを作成し、SAAJ(SOAP with Attachments API for Java)を利用して直接メッセージを扱うことができます。詳細については「[第24章 JAX-RPC WebサービスのSOAPメッセージ・ハンドラーの作成](#)」を参照してください。





# Part I. JEUS 8のWebサービス

JEUS 8のWebサービスは、Java EE 7のWebサービスの標準であるJAX-WS Webサービスをサポートします。JAX-WS Webサービスの最も重要な特徴は、設定ファイルなしで自由に実装できるPOJOスタイルのWebサービスの実装です。このようなPOJOスタイルのWebサービスの実装はJava SE5のアノテーションの機能によって可能となりました。

このパートでは、JAX-WS Webサービスを実装し、JEUSで駆動させる方法、JAX-WS Webサービスを呼び出すクライアントの作成方法、バインディング・カスタマイズ宣言、ハンドラ、メッセージをXMLレベルで扱う方法、非同期、MIME添付でより効率的にメッセージを送信する方法、より迅速なWebサービスの実装について説明し、最後に相互運用性のためのWebサービス、UDDI、JAXR、そしてXMLを扱う多様な方法について記述します。

この後の章を通じて、JAX-WS WebサービスをJEUS 8 Webサービスがサポートする方法について詳しく説明します。



## 第3章 JEUS Webサービスの実装

本章では、Javaクラス、EJB、WSDLでJEUS Webサービスを実装する方法について説明します。

### 3.1. 概要

JEUS 8 Webサービスは、Java EE 7 Webサービスの標準である**JAX-WS方式**のWebサービスをサポートします。

Java EE 7 Webサービスの標準であるJAX-WS Webサービスは、JAXB(XML文書とJavaオブジェクト間のデータ・バインディングのための標準)、SAAJ(XML SOAPメッセージを実装および修正できるようにする標準API。JAX-RPC Webサービス・モデルでは、メッセージ・ハンドラーを実装するために、これを開発者が直接使用しました。しかし、JAX-WS Webサービス・モデルではメッセージ・ハンドラーのための基本フレームワークを提供し、SAAJはその後で動作します)と共にWebサービスの中心コンポーネントに該当し、既存のJAX-RPC Webサービスに代わるためのものとして開発されました。

JAXBが開発されて発表される前にJAX-RPCが発表されたため、JAXBが行うデータ・バインディング機能も基本的に含んでいました。しかし、XMLに関連する標準において、様々な機能の追加と共にJAX-RPC Webサービスがこれをメンテナンスするのが次第に難しくなり、データ・バインディングに集中するJAXBの機能が向上するに従って、それ以上JAX-RPCがデータ・バインディング機能を維持する必要がなくなりました。これによって、JAX-WSが既存のJAX-RPC Webサービスと一緒に関与していたデータ・バインディング機能は、JAXBが独立的に管理するようになります。JAXBの詳細については、後で説明します。

JAX-WSの内容は基本的にJSR 224(<http://jcp.org/jsr/detail/224.jsp>)で記述されています。JAX-WSはJava SE 5のアノテーション機能によって、既存の複数のWebサービスのDeployment Descriptor(以下、DD)ファイルをアノテーションで処理、POJO(Plain Old Java Object)方式のWebサービスを実装できるようになりました。

本章では、Javaクラス・ファイル、そしてWSDLからJAX-WS Webサービスを実装する方法について説明します。JavaクラスでWebサービスを実装する際は、サービス・エンドポイントの実装クラスのみでアノテーションと**wsgen**ツールを使用し、ポータブル・アーティファクトを取得します。そして、WSDLからWebサービスを実装する際は、作成したWSDLファイルから**wsimport**ツールを使用し、サービス・エンドポイント・インターフェースとポータブル・アーティファクトを作成後、これを実装する実装Javaクラスを作成します。

### 3.2. JavaクラスWebサービスの実装

Javaエンドポイント実装クラスからWebサービスを実装する際、このJavaエンドポイント実装クラスにはJAX-WSによって規定されたいくつかの規則があります。それは以下のとおりです。

- `javax.jws.WebService`アノテーションを伴う必要があります。

- メソッドはjavax.jws.WebMethodアノテーションを伴う必要があります。
- すべてのメソッドはサービス固有の特定例外と共にjava.rmi.RemoteExceptionを返すことができます。
- すべてのメソッドのパラメータと戻り値のタイプは、JAXBのJavaからXMLスキーマへのマッピング定義に明示されている必要があります。
- メソッドのパラメータまたは戻り値のタイプはjava.rmi.Remoteインターフェースを直・間接的に実装するものであってはいけません。

上の規則に従う簡単なJavaエンドポイント実装クラスの例は以下のとおりです。

### [例 3.1] << Addnumbersimpl.java >>

```
package fromjava.server;

@WebService
public class AddNumbersImpl {

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

上記のように、@WebServiceというアノテーションが追加されます。このように、アノテーションとしてWebServiceを設定してエンドポイント実装ロジックを構成すると、基本的なサーバー側のWebサービス・プログラミングはすべて完了します。このようにJavaエンドポイント実装クラスを実装すると、次にJEUS 8 Webサービスで提供するツールを使用してポータブル・アーティファクトを作成します。このようなポータブル・アーティファクトは、メソッドの呼び出しまたは応答をJavaオブジェクトに変換またはXML文書に変換するのに使用されるいくつかのJava Beanクラスとサービス固有の特定例外クラスで構成されます。このようなポータブル・アーティファクトを作成するには、JEUS 8 Webサービスが提供するwsngenツールを使用します。

以下は、wsngenの使用方法についての説明です。

```
$ wsngen -help

Usage: WSGEN [options] <SEI>

where [options] include:
  -classpath <path>          specify where to find input class files
  -cp <path>                 same as -classpath <path>
  -d <directory>            specify where to place generated output files
  -extension                  allow vendor extensions - functionality
                              not specified by the specification.
                              Use of extensions may result in
                              applications that are not portable or
                              may not interoperate with other implementations
  -help                      display help
```

<code>-keep</code>	keep generated files
<code>-r &lt;directory&gt;</code>	resource destination directory, specify where to place resource files such as WSDLs
<code>-s &lt;directory&gt;</code>	specify where to place generated source files
<code>-verbose</code>	output messages about what the compiler is doing
<code>-version</code>	print version information
<code>-wsdl[:protocol]</code>	generate a WSDL file. The protocol is optional. Valid protocols are [soap1.1, Xsoap1.2], the default is soap1.1. The non standard protocols [Xsoap1.2] can only be used in conjunction with the <code>-extension</code> option.
<code>-servicename &lt;name&gt;</code>	specify the Service name to use in the generated WSDL Used in conjunction with the <code>-wsdl</code> option.
<code>-portname &lt;name&gt;</code>	specify the Port name to use in the generated WSDL Used in conjunction with the <code>-wsdl</code> option.

Examples:

```
wsgen -cp . example.Stock
wsgen -cp . example.Stock -wsdl
      -servicename {http://mynamespace}MyService
```

## 参考

JEUS 8 Webサービスは wsgenのAntタスクもサポートします。コンソール・コマンドとAntタスク項目の詳細については『*JEUS リファレンスガイド*』の「5.5.3. wsgen」と『*JEUS リファレンスガイド*』の「5.6.1. wsgen」「5.8.1. インストール」を参照してください。

wsgenツールを使用して、上で作成したJavaエンドポイント実装クラスを使用し、以下のようにポータブル・アーティファクトを作成します。

### [例 3.2] << build.xml >>

```
...
<target name="build_server" depends="init">
  <antcall target="do-compile">
    <param name="javac.excludes" value="fromjava/client/" />
  </antcall>
  <antcall target="wsgen">
    <param name="sib.file" value="fromjava.server.AddNumbersImpl" />
  </antcall>
  <antcall target="do-package-war" />
</target>
...
```

wsgenツールを使用して、上のAddnumbersImpl Javaエンドポイント実装クラスよりポータブル・アーティファクトとWSDLファイルを作成します。以下はその結果です。

```
AddNumbersImplService.wsdl
AddNumbersImplService_schema1.xsd
fromjava/server/jaxws/AddNumbers.class
fromjava/server/jaxws/AddNumbersResponse.class
```

AddNumbersとAddNumbersResponseファイルは、各JAXBがaddNumbersメソッドに対する要求と応答をJavaオブジェクトまたはXML文に変換するために使用するJava Beanです。

AddNumbersImplService.wsdlファイルは、このWebサービスのWSDLファイルを示し、AddNumbersImplService\_schema1.xsdスキーマ・ファイルはこのWebサービスによって使用されるデータ型を定義したもので、WSDL文書の内部で使用されています。

以下は、各ファイルの実装された内容です。

### [例 3.3] << AddNumbersImplService.wsdl >>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions targetNamespace="http://server.fromjava/"
  name="AddNumbersImplService" xmlns:tns="http://server.fromjava/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://server.fromjava/"
        schemaLocation="AddNumbersImplService_schema1.xsd" />
    </xsd:schema>
  </types>
  <message name="addNumbers">
    <part name="parameters" element="tns:addNumbers" />
  </message>
  <message name="addNumbersResponse">
    <part name="parameters" element="tns:addNumbersResponse" />
  </message>
  <portType name="AddNumbersImpl">
    <operation name="addNumbers">
      <input message="tns:addNumbers" />
      <output message="tns:addNumbersResponse" />
    </operation>
  </portType>
  <binding name="AddNumbersImplPortBinding" type="tns:AddNumbersImpl">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
      style="document" />
    <operation name="addNumbers">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
  </binding>
</definitions>
```

```

        </output>
    </operation>
</binding>
<service name="AddNumbersImplService">
    <port name="AddNumbersImplPort"
        binding="tns:AddNumbersImplPortBinding">
        <soap:address location="REPLACE_WITH_ACTUAL_URL" />
    </port>
</service>
</definitions>

```

上記のように、WSDLファイルの<message>要素に対し、サーバーのポータブル・アーティファクトが作成されたことを確認できます。このようなポータブル・アーティファクトは、実際にJAX-WSエンジンとエンドポイント・クラス間のメッセージの送信をサポートします。

2つの<message>要素のポータブル・アーティファクトJavaクラスは、AddNumbersとAddNumbersResponseです。

以下は、クライアントから呼び出されるメッセージのAddNumbersに対するJava Beanクラスです。

#### [例 3.4] << AddNumbers.java >>

```

@XmlRootElement(name = "addNumbers", namespace = "http://server.fromjava/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addNumbers", namespace = "http://server.fromjava/",
    propOrder = {"arg0", "arg1" })
public class AddNumbers {

    @XmlElement(name = "arg0", namespace = "")
    private int arg0;

    @XmlElement(name = "arg1", namespace = "")
    private int arg1;

    public int getArg0() {
        return this.arg0;
    }

    public void setArg0(int arg0) {
        this.arg0 = arg0;
    }

    public int getArg1() {
        return this.arg1;
    }

    public void setArg1(int arg1) {
        this.arg1 = arg1;
    }
}

```

以下は、サービスから返されるメッセージのAddNumbersResponse <message>要素に対するJava Beanクラスです。

**[例 3.5] << AddNumbersResponse.java >>**

```
@XmlRootElement(name = "addNumbersResponse", namespace = "http://server.fromjava/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addNumbersResponse", namespace = "http://server.fromjava/")
public class AddNumbersResponse {

    @XmlElement(name = "return", namespace = "")
    private int _return;

    public int get_return() {
        return this._return;
    }

    public void set_return(int _return) {
        this._return = _return;
    }
}
```

### 3.3. EJB Webサービスの実装

本節では、EJBコンテナで動作するステートレス・セッションBeanを使用して実装するEJB Webサービス・プログラミング・モデルについて説明します。

JEUS 8で提供するEJB 3.0プログラミング・モデルは、サービス・エンドポイント実装クラスの作成を容易にするための様々な機能を提供しています。javax.ejb.SessionBeanまたはjavax.ejb.EntityBeanインターフェースを実装せずにアノテーションで明示し、Session Beanの場合はコンポーネント・インターフェースあるいはホーム・インターフェースを実装する必要がなくなります。

---

#### 参考

JEUS 8で提供する機能の詳細については『JEUS EJBガイド』を参照してください。

---

JEUS 8で提供するEJB 3.0機能を使用してBeanクラスを実装した後、これをJAX-WSサービス・エンドポイント実装クラスとして使用するためには、Beanクラスに@WebServiceアノテーションを設定します。上記規則に従ったEJBエンドポイント実装クラスの簡単な例は以下のとおりです。

**[例 3.6] << Addnumbersimpl.java >>**

```
package fromejb.server;

@Stateless
@WebService
public class AddNumbersImpl {
```



```

    public AddNumbersImpl() {

    }

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}

```

上記のように、@Statelessアノテーションと@WebServiceアノテーションが追加されます。このように、アノテーションで「WebService」と設定し、エンドポイント実装ロジックを構成すると、基本的なサーバーのWebサービス・プログラミングはすべて完了します。

このようにEJBエンドポイント実装クラスを実装すると、次にJEUS 8 Webサービスで提供するツールを使用してポータブル・アーティファクトを作成します。このようなポータブル・アーティファクトは、メソッドの呼び出しまたは応答を、Javaオブジェクトに変換あるいはXML文書に変換するのに使用されるいくつかのJava Beanクラスとサービス固有の特定例外クラスで構成されます。このようなポータブル・アーティファクトを作成するには、JEUS 8 Webサービスが提供する**wsgen**ツールを使用します。

以下は、**wsgen**の使用方法についての説明です。

```

$ wsgen -help

Usage: WSGEN [options] <SEI>

where [options] include:
    -classpath <path>          specify where to find input class files
    -cp <path>                 same as -classpath <path>
    -d <directory>            specify where to place generated output files
    -extension                  allow vendor extensions - functionality
                                not specified by the specification.
                                Use of extensions may result in
                                applications that are not portable or
                                may not interoperate with other implementations
    -help                      display help
    -keep                      keep generated files
    -r <directory>            resource destination directory, specify
                                where to place resource files such as WSDLs
    -s <directory>            specify where to place generated source files
    -verbose                   output messages about what the compiler is doing
    -version                   print version information
    -wsdl[:protocol]          generate a WSDL file. The protocol is optional.
                                Valid protocols are [soap1.1, Xsoap1.2],
                                the default is soap1.1.
                                The non standard protocols [Xsoap1.2]
                                can only be used in conjunction with the
                                -extension option.
    -servicename <name>       specify the Service name to use in the generated
                                WSDL Used in conjunction with the -wsdl option.

```

```
-portname <name>          specify the Port name to use in the generated
                           WSDL Used in conjunction with the -wsdl option.
```

Examples:

```
wsgen -cp . example.Stock
wsgen -cp . example.Stock -wsdl
      -servicename {http://mynamespace}MyService
```

---

## 参考

JEUS 8 WebサービスはwsgenのAntタスクもサポートします。コンソール・コマンドとAntタスク項目の詳細については『*JEUS リファレンスガイド*』の「5.5.3. wsgen」と『*JEUS リファレンスガイド*』の「5.6.1. wsgen」「5.8.1. インストール」を参照してください。

---

wsgenツールを使用して、上で作成したJavaエンドポイント実装クラスを使用し、以下のようにポータブル・アーティファクトを作成します。

### [例 3.7] << build.xml >>

```
...

<target name="build_server" depends="init">
  <antcall target="do-compile">
    <param name="javac.excludes" value="fromejb/client/" />
  </antcall>
  <antcall target="wsgen">
    <param name="sib.file" value="fromejb.server.AddNumbersImpl" />
  </antcall>
  <antcall target="do-package-jar" />
</target>

...
```

wsgenツールを使用して、上のAddNumbersImpl EJBエンドポイント実装クラスからポータブル・アーティファクトとWSDLファイルを作成した結果は以下のとおりです。

```
AddNumbersImplService.wsdl
AddNumbersImplService_schema1.xsd
fromjava/server/jaxws/AddNumbers.class
fromjava/server/jaxws/AddNumbersResponse.class
```

AddNumbersとAddNumbersResponseファイルは、各JAXBがaddNumbersメソッドに対する要求と応答をJavaオブジェクトまたはXML文に変換するために使用するJava Beanです。

AddNumbersImplService.wsdlファイルはこのWebサービスのWSDLファイルを示し、AddNumbersImplService\_schema1.xsdスキーマ・ファイルはこのWebサービスによって使用されるデータ型を定義したもので、WSDL文書の内部で使用されています。

以下は、各ファイルの実装された内容です。

### [例 3.8] << AddNumbersImplService.wsdl >>

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="http://server.fromejb/"
  name="AddNumbersImplService" xmlns:tns="http://server.fromejb/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://server.fromejb/"
        schemaLocation="AddNumbersImplService_schema1.xsd" />
    </xsd:schema>
  </types>
  <message name="addNumbers">
    <part name="parameters" element="tns:addNumbers" />
  </message>
  <message name="addNumbersResponse">
    <part name="parameters" element="tns:addNumbersResponse" />
  </message>
  <portType name="AddNumbersImpl">
    <operation name="addNumbers">
      <input message="tns:addNumbers" />
      <output message="tns:addNumbersResponse" />
    </operation>
  </portType>
  <binding name="AddNumbersImplPortBinding" type="tns:AddNumbersImpl">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
      style="document" />
    <operation name="addNumbers">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
  </binding>
  <service name="AddNumbersImplService">
    <port name="AddNumbersImplPort"
      binding="tns:AddNumbersImplPortBinding">
      <soap:address location="REPLACE_WITH_ACTUAL_URL" />
    </port>
  </service>
</definitions>
```

上記のように、WSDLファイルの<message>要素に対し、サーバーのポータブル・アーティファクトが作成されたことが確認できます。このようなポータブル・アーティファクトは実際にJAX-WSエンジンとエンドポイント・クラス間のメッセージの送信をサポートします。

2つの<message>要素のポータブル・アーティファクトJavaクラスは、AddNumbersとAddNumbersResponseです。

以下は、クライアントから呼び出されるメッセージのAddNumbers <message>に対するJava Beanクラスです。

**[例 3.9] << AddNumbers.java >>**

```
@XmlRootElement(name = "addNumbers", namespace = "http://server.fromejb/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addNumbers", namespace = "http://server.fromejb/", propOrder = {
    "arg0", "arg1" })
public class AddNumbers {

    @XmlElement(name = "arg0", namespace = "")
    private int arg0;

    @XmlElement(name = "arg1", namespace = "")
    private int arg1;

    public int getArg0() {
        return this.arg0;
    }

    public void setArg0(int arg0) {
        this.arg0 = arg0;
    }

    public int getArg1() {
        return this.arg1;
    }

    public void setArg1(int arg1) {
        this.arg1 = arg1;
    }
}
```

以下は、サービスから返されるメッセージのAddNumbersResponse <message>要素に対するJava Beanクラスです。

**[例 3.10] << AddNumbersResponse.java >>**

```
@XmlRootElement(name = "addNumbersResponse", namespace = "http://server.fromejb/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addNumbersResponse", namespace = "http://server.fromejb/")
public class AddNumbersResponse {

    @XmlElement(name = "return", namespace = "")
    private int _return;
}
```

```

    public int get_return() {
        return this._return;
    }

    public void set_return(int _return) {
        this._return = _return;
    }
}

```

## 3.4. WSDLによるWebサービスの実装

WSDLからWebサービスを実装する際は、JEUS 8 Webサービスで提供するツールである**wsimport**を使用して、まずサービス・エンドポイント・インターフェースを作成する必要があります。

以下は、本節で使用するWSDLファイルの例です。

### [例 3.11] << AddNumbers.wsdl >>

```

<?xml version="1.0" encoding="UTF-8"?>

<definitions name="AddNumbers" targetNamespace="urn:AddNumbers"
    xmlns:impl="urn:AddNumbers"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

    <types>
        <xsd:schema elementFormDefault="qualified"
            targetNamespace="urn:AddNumbers"
            xmlns="http://www.w3.org/2001/XMLSchema">
            <complexType name="addNumbersResponse">
                <sequence>
                    <element name="return" type="xsd:int" />
                </sequence>
            </complexType>
            <element name="addNumbersResponse"
                type="impl:addNumbersResponse" />
            <complexType name="addNumbers">
                <sequence>
                    <element name="arg0" type="xsd:int" />
                    <element name="arg1" type="xsd:int" />
                </sequence>
            </complexType>
            <element name="addNumbers" type="impl:addNumbers" />
        </xsd:schema>
    </types>

    <message name="addNumbers">
        <part name="parameters" element="impl:addNumbers" />
    </message>

```

```

<message name="addNumbersResponse">
  <part name="result" element="impl:addNumbersResponse" />
</message>

<portType name="AddNumbersPortType">
  <operation name="addNumbers">
    <input message="impl:addNumbers" name="add" />
    <output message="impl:addNumbersResponse"
      name="addResponse" />
  </operation>
</portType>

<binding name="AddNumbersBinding"
  type="impl:AddNumbersPortType">
  <soap:binding
    transport="http://schemas.xmlsoap.org/soap/http"
    style="document" />
  <operation name="addNumbers">
    <soap:operation soapAction="" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>

<service name="AddNumbersService">
  <port name="AddNumbersPort" binding="impl:AddNumbersBinding">
    <soap:address location="REPLACE_WITH_ACTUAL_URL" />
  </port>
</service>
</definitions>

```

上記のようにWSDLファイルを実装した場合、次にJEUS 8 Webサービスで提供する**wsimport**ツールを使用してサービス・エンドポイント・インターフェースとポータブル・アーティファクトを生成します。

**wsimport**の使用方法は以下のとおりです。

```

$ wsimport -help

Usage: wsimport [options] <WSDL_URI>

where [options] include:
  -b <path>                specify jaxws/jaxb binding files or additional schemas
                           (Each <path> must have its own -b)
  -B<jaxbOption>           Pass this option to JAXB schema compiler
  -catalog <file>          specify catalog file to resolve external

```

	entity references supports TR9401, XCatalog, and OASIS XML Catalog format.
-d <directory>	specify where to place generated output files
-extension	allow vendor extensions - functionality not specified by the specification. Use of extensions may result in applications that are not portable or may not interoperate with other implementations
-help	display help
-httpproxy:<host>:<port>	specify a HTTP proxy server (port defaultsto 8080)
-keep	keep generated files
-p <pkg>	specifies the target package
-quiet	suppress wsimport output
-s <directory>	specify where to place generated source files
-target <version>	enerate code as per the given JAXWS spec version e.g. 2.0 will generate compliant code for JAXWS 2.0 spec
-verbose	output messages about what the compiler is doing
-version	print version information
-wsdllocation <location>	@WebServiceClient.wsdlLocation value

#### Examples:

```
wsimport stock.wsdl -b stock.xml -b stock.xjb
wsimport -d generated http://example.org/stock?wsdl
```

---

## 参考

JEUS 8 WebサービスはwsimportのAntタスクもサポートします。コンソール・コマンドとAntタスク項目の詳細については『*JEUS リファレンスガイド*』の「5.5.4. wsimport」と『*JEUS リファレンスガイド*』の「5.6.2. wsimport」を参照してください。

---

wsimportツールを使用して、上で作成したWSDLファイルを使用し、以下のようにサービス・エンドポイント・インターフェースとポータブル・アーティファクトを作成します。

### [例 3.12] << build.xml >>

```
...

<target name="build_server" depends="init">
  <mkdir dir="${build.classes.dir}" />
  <antcall target="wsimport">
    <param name="package.name" value="fromwsdl.server" />
    <param name="binding.file" value="" />
    <param name="wsdl.file" value="${src.web}/WEB-INF/wsdl/AddNumbers.wsdl" />
  </antcall>
  <antcall target="do-compile">
    <param name="javac.excludes" value="fromwsdl/client/" />
  </antcall>
  <antcall target="do-package-war" />
</target>
```

...

上のwsimport Antタスクの結果で作成されるサービス・エンドポイント・インターフェースおよびポータブル・アーティファクトは以下のとおりです。

```
fromwsdl/server/AddNumbers.class
fromwsdl/server/AddNumbersPortType.class
fromwsdl/server/AddNumbersResponse.class
fromwsdl/server/AddNumbersService.class
fromwsdl/server/ObjectFactory.class
fromwsdl/server/package-info.class
```

AddNumbersとAddNumbersResponseファイルは、各JAXBがaddNumbersメソッドに対する要求と応答をJavaオブジェクトまたはXML文に交換するのに使用するJava Beanです。

AddNumbersPortTypeファイルはサービス・エンドポイント・インターフェースを示し、AddNumbersServiceファイルはクライアントでプロキシの用途で使用するJavaクラスです。ObjectFactoryクラスとpackage-infoクラスはJAXBが作成したファイルです。

各ファイルの内容は以下のとおりです。wsimportツールを使用して上のWSDLファイルから取得したサービス・エンドポイント・インターフェースであるAddNumbersPortTypeクラスを実装します。

#### [例 3.13] << AddNumbersPortType.java >>

```
@WebService(name = "AddNumbersPortType", targetNamespace = "urn:AddNumbers")
@XmlSeeAlso( { ObjectFactory.class } )
public interface AddNumbersPortType {

    @WebMethod
    @WebResult(targetNamespace = "urn:AddNumbers")
    @RequestWrapper(localName = "addNumbers",
        targetNamespace = "urn:AddNumbers",
        className = "fromwsdl.server.AddNumbers")
    @ResponseWrapper(localName = "addNumbersResponse",
        targetNamespace = "urn:AddNumbers",
        className = "fromwsdl.server.AddNumbersResponse")
    public int addNumbers(
        @WebParam(name = "arg0", targetNamespace = "urn:AddNumbers")
        int arg0,
        @WebParam(name = "arg1", targetNamespace = "urn:AddNumbers")
        int arg1);
}
```

上のように作成されたサービス・エンドポイント・インターフェースは、動的バインディング(dynamic binding)またはランタイムに必要なアノテーションを持っています。このようなアノテーションは、XML文書からJavaオブジェクト、またはJavaオブジェクトからXML文書への変換に必要です。



---

## 注

このサービス・エンドポイント・インターフェースは、再びWSDLとスキーマ・ファイルを生成するのに使用できます。しかし、このように生成されたWSDLとスキーマ・ファイルはサービス・エンドポイント・インターフェースを取得するために使用されたオリジナルのWSDLとスキーマ・ファイルと一致しない場合があります。

---

また、JAXBがaddNumbersメソッドに対する要求と応答をJavaオブジェクトまたはXML文に変換するために使用するJava BeanクラスであるAddNumbersクラスとAddNumbersResponseクラスは以下のとおりです。

### [例 3.14] << AddNumbers.java >>

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addNumbers", propOrder = { "arg0", "arg1" })
public class AddNumbers {

    protected int arg0;

    protected int arg1;

    public int getArg0() {
        return arg0;
    }
    public void setArg0(int value) {
        this.arg0 = value;
    }
    public int getArg1() {
        return arg1;
    }
    public void setArg1(int value) {
        this.arg1 = value;
    }
}
```

### [例 3.15] << AddNumbersResponse.java >>

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addNumbersResponse", propOrder = { "_return" })
public class AddNumbersResponse {

    @XmlElement(name = "return")
    protected int _return;

    public int getReturn() {
        return _return;
    }
    public void setReturn(int value) {
        this._return = value;
    }
}
```

この2つのJava Beanクラスは、上で作成したWSDLファイルの<message>要素であるAddNumbers、AddNumbersResponseに該当することが分かります。

次に、このようにWSDLから生成されたサービス・エンドポイント・インターフェースとポータブル・アーティファクトのうち、実際にサービスに関するビジネス・ロジックを格納しているサービス・エンドポイント実装クラスを作成します。この実装Javaクラスを作成するには、まずサービス・エンドポイント・インターフェースの名前を明示するアノテーションをこの実装Javaクラスに提供する必要があります。また、作成したWSDLファイルの位置とWSDLファイルの名前空間(Target Name Space)、サービス名、ポート名を設定します。

**[例 3.16] << AddNumbersImpl.java >>**

```
package fromwsdl.server;

@javax.jws.WebService(
    endpointInterface = "fromwsdl.server.AddNumbersPortType",
    wsdlLocation="WEB-INF/wsdl/AddNumbers.wsdl",
    targetNamespace = "urn:AddNumbers",
    serviceName = "AddNumbersService",
    portName = "AddNumbersPort"
)
public class AddNumbersImpl {
    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

上記のように、@WebServiceというアノテーションに、実装Javaクラスのサービス・エンドポイント・インターフェース名を、endpointInterface、wsdlLocation、targetNamespace、serviceName、portNameという変数値で指定しています。

## 第4章 Webサービスの作成とデプロイ

本章では、JAX-WS WebサービスをJavaクラスとWSDLファイルから作成してデプロイし、パッケージングする方法について説明します。また、JAX-WS WebサービスをJEUS 8で動作させる方法について説明します。

### 4.1. 概要

基本的に、Webサービスの実装方法にかかわらず、デプロイの過程は同一です。JAX-WS Webサービスは、POJO方式のWebサービス実装を目指すため、DDの数がJAX-RPC Webサービスに比べて大幅に減少します。

JEUS 5 JAX-RPC方式のWebサービスの実装で一般的に使用されていたDDなしでプログラミングが可能で、非常に簡単かつ速く、利便的にWebサービスを開発できるようにします。(Description-free programming)

### 4.2. JavaクラスWebサービスの作成とデプロイ

Javaクラスから実装したWebサービスをWARファイルで構成することは、サービス・エンドポイント実装クラスとその他のJavaクラス(ポータブル・アーティファクト)を複数DDファイルと一緒にWAR形式にパッケージングすることを意味します。

以下は、前章でJavaクラスから実装したWebサービスをWAR形式で構成する様子をディレクトリ別に示したものです。

```
META-INF/MANIFEST.MF
WEB-INF/classes/AddNumbersImplService_schema1.xsd
WEB-INF/classes/AddNumbersImplService.wsdl
WEB-INF/classes/fromjava/server/AddNumbersImpl.class
WEB-INF/classes/fromjava/server/jaxws/AddNumbers.class
WEB-INF/classes/fromjava/server/jaxws/AddNumbersResponse.class
```

上のWARファイルは、サービス・エンドポイント実装クラスとポータブル・アーティファクト・クラスと一緒に一意のDDであるweb.xmlから構成されます。web.xmlファイルはJAX-WS Webサービスの実装に必要ありませんが、ここではエンドポイントのアドレスを取得するためにweb.xmlを使用しました。エンドポイントのアドレスの詳細については「[4.5. エンドポイントのアドレスの決定方式](#)」を参照してください。

このように作成されたWebサービスのWARファイルはJEUS 8で提供する方式でデプロイされ、アクセスできます。作成されたWARファイルをJEUS 8にデプロイするには、コンソールで以下のコマンドを実行します。

```
$ ant build deploy
```

正常にデプロイされた場合、このサービスにアクセスできる実際のアドレスは以下のとおりです。

```
http://host:port/AddNumbers/AddNumbersImplService
```

## 4.3. EJB Webサービスの作成とデプロイ

Javaクラスから実装したWebサービスをJARファイルで構成することは、サービス・エンドポイント実装クラスと、その他のJavaクラス(ポータブル・アーティファクト)を複数のDDファイルと一緒にJAR形式にパッケージングすることを意味します。

以下は、前章でEJBステートレス・セッションBeanから実装したWebサービスをJAR形式で構成する様子をディレクトリー別に示したものです。

```
META-INF/MANIFEST.MF
fromejb/server/AddNumbersImpl.class
fromejb/server/jaxws/AddNumbers.class
fromejb/server/jaxws/AddNumbersResponse.class
```

上のJARファイルは、サービス・エンドポイント実装クラスとポータブル・アーティファクト・クラスから構成されます。エンドポイントのアドレスの詳細については「[4.5. エンドポイントのアドレスの決定方式](#)」を参照してください。

このように作成されたWebサービスのJARファイルはJEUS 8で提供する方式でデプロイされ、アクセスできます。作成されたJARファイルをJEUS 8にデプロイするには、コンソールで以下のコマンドを実行します。

```
$ ant build deploy
```

正常にデプロイされた場合、このサービスにアクセスできる実際のアドレスは以下のとおりです。

```
http://host:port/AddNumbersImplService/AddNumbersImpl
```

## 4.4. WSDLによるWebサービスの作成とデプロイ

WSDLから実装したWebサービスをWARファイルで構成することは、サービス・エンドポイント・インターフェースとそれを実装するサービス・エンドポイント実装クラス、その他のポータブル・アーティファクトを複数のDDファイルと一緒にWAR形式にパッケージングすることを意味します。

以下は、前章でWSDLから実装したWebサービスをWAR形式で構成する様子をディレクトリー別に示したものです。

```
META-INF/MANIFEST.MF
WEB-INF/wsdl/AddNumbers.wsdl
WEB-INF/classes/fromjava/server/AddNumbers.class
WEB-INF/classes/fromjava/server/AddNumbersImpl.class
WEB-INF/classes/fromjava/server/AddNumbersPortType.class
WEB-INF/classes/fromjava/server/AddNumbersResponse.class
WEB-INF/classes/fromjava/server/AddNumbersService.class
WEB-INF/classes/fromjava/server/ObjectFactory.class
WEB-INF/classes/fromjava/server/package-info.class
```

上のWARファイルは、サービス・エンドポイント・インターフェースのAddNumbersPortTypeと、これを実装するAddNumbersImpl、そして複数のポータブル・アーティファクト・クラスと一緒に一意のDDであるweb.xmlファイルで構成されたものです。web.xmlファイルはJAX-WS Webサービスの実装に必要ありませんが、ここではエンドポイントのアドレスを取得するためにweb.xmlを使用しました。エンドポイントのアドレスの詳細については「[4.5. エンドポイントのアドレスの決定方式](#)」を参照してください。

このように作成されたWebサービスのWARファイルはJEUS 8で提供する方式でデプロイされ、Webサービスで公開できます。作成されたWARファイルをJEUS 8にデプロイするには、コンソールで以下のコマンドを実行します。

```
$ ant build deploy
```

正常にデプロイされた場合、このサービスにアクセスできる実際のアドレスは以下のとおりです。

```
http://host:port/AddNumbers/AddNumbersService
```

## 4.5. エンドポイントのアドレスの決定方式

JAX-WS Webサービスは、web.xmlを始め、webservices.xmlなどの既存JEUS 5のJAX-RPC Webサービスで必要としていたDDファイルがなくても(Descriptor-free)、JEUS 8にデプロイが可能です。

本節では、サブレット・ベースのWebサービスとEJBベースのWebサービスについて、このようなJAX-WSがJEUS 8にデプロイされるときに、実際にこのサービスへアクセスできるアドレスの決定方式について説明します。

### 4.5.1. サブレット・エンドポイント

サブレット・エンドポイントの場合、URL決定の優先順位は以下のとおりです。

#### 1. web.xmlファイルの<url-pattern>を使用する場合

web.xmlファイルの<url-pattern>を使用する場合、入力された値が実際にこのWebサービスにアクセスできるURLアドレスです。これは@EndpointDescriptionエンドポイントのアドレスのデフォルト値を上書きします。

以下は、web.xmlファイルとAddNumbersImplと同じエンドポイント・クラスの例です。

#### [例 4.1] << web.xml >>

```
<web-app>
  <display-name>fromwsdl</display-name>
  <description>fromwsdl</description>
  <servlet>
    <servlet-name>fromwsdl</servlet-name>
    <display-name>fromwsdl</display-name>
    <servlet-class>fromwsdl.server.AddNumbersImpl</servlet-class>
    <load-on-startup>1</load-on-startup>
```

```

</servlet>
<servlet-mapping>
    <servlet-name>fromwsdl</servlet-name>
    <url-pattern>/addnumbers</url-pattern>
</servlet-mapping>
</web-app>

```

#### [例 4.2] << AddNumbersImpl.java >>

```

@WebService(serviceName="AddNumbers")
@EndpointDescription(endpointUrl="MyService")
public class AddNumbersImpl {

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}

```

このWebサービスにアクセスできる実際のURLアドレスは以下のとおりです。

```
http://server:port/context/addnumbers
```

## 2. @EndpointDescriptionアノテーションを使用する場合

@EndpointDescriptionというアノテーションにendpointUrlという変数値が決まっている場合、この値が実際にこのWebサービスにアクセスできるURLアドレスです。これは@WebServiceアノテーションのserviceNameエンドポイントのアドレス値を上書きします。

以下はエンドポイント・クラスの例です。

#### [例 4.3] << AddNumbersImpl.java >>

```

@WebService(serviceName="AddNumbers")
@EndpointDescription(endpointUrl="MyService")
public class AddNumbersImpl {

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}

```

このWebサービスにアクセスできる実際のURLアドレスは以下のとおりです。

```
http://server:port/context/MyService
```

## 3. @WebServiceアノテーションのserviceName属性を使用する場合

@WebServiceというアノテーションにserviceNameという変数値が決まっている場合、この値が実際にこのWebサービスにアクセスできるURLアドレスです。これはエンドポイントのアドレスのデフォルト値を上書きします。

以下はエンドポイント・クラスの例です。

**[例 4.4] << AddNumbersImpl.java >>**

```
@WebService(serviceName="AddNumbers")
public class AddNumbersImpl {

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

このWebサービスにアクセスできる実際のURLアドレスは以下のとおりです。

```
http://server:port/context/AddNumbers
```

#### 4. エンドポイントのアドレスのデフォルト値+「Service」

@WebServiceというアノテーション以外に何も設定されていない場合、エンドポイント・クラス名+「Service」がこのWebサービスのエンドポイントのアドレスのデフォルト値となります。

以下はエンドポイント・クラスの例です。

**[例 4.5] << AddNumbersImpl.java >>**

```
@WebService
public class AddNumbersImpl {

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

このWebサービスにアクセスできる実際のURLアドレスは以下のとおりです。

```
http://server:port/context/AddNumbersImplService
```

## 4.5.2. EJBエンドポイント

EJBエンドポイントの場合、URL決定の優先順位はサーブレット・エンドポイントの場合と同じで、コンテキストの決定方式のみ異なります。

コンテキストの決定優先順位は以下のとおりです。

## 1. @EndpointDescriptionアノテーションを使用する場合

- エンドポイント

@EndpointDescriptionというアノテーションにendpointUrlという変数値が決まっている場合、この値が実際にこのWebサービスにアクセスできるURLアドレスです。これは@WebServiceアノテーションのnameエンドポイントのアドレス値を上書きします。

以下はエンドポイント・クラスの例です。

### [例 4.6] << AddNumbersImpl.java >>

```
@WebService(name="AddNumbersService")
@EndpointDescription(endpointUrl="MyService")
@Stateless
public class AddNumbersImpl {
    public AddNumbersImpl() {
    }
    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

このWebサービスにアクセスできる実際のURLアドレスは以下のとおりです。

```
http://server:port/context/MyService
```

- コンテキスト

@jeus.webservices.annotation.EndpointDescriptionアノテーションにcontextPath属性を使用する場合、コンテキストのデフォルト値を上書きします。

以下はエンドポイント・クラスの例です。

### [例 4.7] << AddNumbersImpl.java >>

```
@WebService(name="Hello", serviceName="AddNumbersService")
@jeus.webservices.annotation.EndpointDescription(contextPath="EJBService",
    endpointUrl="MyEndpoint")
@Stateless
public class AddNumbersImpl {
    public AddNumbersImpl() {
    }
    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

このWebサービスにアクセスできる実際のURLアドレスは以下のとおりです。

```
http://server:port/EJBService/MyEndpoint
```



## 2. @WebServiceアノテーションのserviceName属性を使用する場合

- エンドポイント

@WebServiceというアノテーションにnameという変数値が決まっている場合、この値が実際にこのWebサービスにアクセスできるURLアドレスです。これは、エンドポイントのアドレスのデフォルト値を上書きします。

以下はエンドポイント・クラスの例です。

**[例 4.8] << AddNumbersImpl.java >>**

```
@WebService(name="AddNumbersService")
@Stateless
public class AddNumbersImpl {

    public AddNumbersImpl() {

    }

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

このWebサービスにアクセスできる実際のURLアドレスは以下のとおりです。

```
http://server:port/context/AddNumbersService
```

- コンテキスト

@WebServiceアノテーションにserviceName属性を使用する場合、コンテキストのデフォルト値を上書きします。

以下はエンドポイント・クラスの例です。

**[例 4.9] << AddNumbersImpl.java >>**

```
@WebService(name="Hello", serviceName="AddNumbersService")
@Stateless
public class AddNumbersImpl {

    public AddNumbersImpl() {

    }

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

このWebサービスにアクセスできる実際のURLアドレスは以下のとおりです。

```
http://server:port/AddNumbersService/Hello
```

### 3. コンテキストのデフォルト値

- エンドポイント

@WebServiceというアノテーション以外に何も設定されていない場合、エンドポイント・クラス名がこのWebサービスのエンドポイントのアドレスのデフォルト値です。

以下はエンドポイント・クラスの例です。

**[例 4.10] << AddNumbersImpl.java >>**

```
@WebService
@Stateless
public class AddNumbersImpl {

    public AddNumbersImpl() {

    }

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

このWebサービスにアクセスできる実際のURLアドレスは以下のとおりです。

```
http://server:port/context/AddNumbersImpl
```

- コンテキスト

コンテキストのデフォルト値は、そのWebサービスのWSDL文書のServiceNameです。また、このServiceNameはエンドポイント・クラス名にServiceを加えた値がデフォルト値として設定されます。

したがって、上のエンドポイント・クラスのWebサービスへは以下のURLアドレスでアクセスできます。

```
http://server:port/AddNumbersImplService/AddNumbersImpl
```

# 第5章 Webサービスの呼び出し

本章では、Webサービスの呼び出し方式について説明します。

## 5.1. 概要

1つのJAX-WS Webサービス・クライアント・アプリケーションは、以下の2つの方式でリモートWebサービス・エンドポイントにアクセスできます。

- 動的プロキシ(Dynamic Proxy)方式
- ディスパッチ(Dispatch)方式

## 5.2. 動的プロキシ方式のWebサービスの呼び出し

動的プロキシ方式のWebサービスの呼び出しは、Java SE方式とJava EE方式に分けられます。両方ともクライアント・アーティファクトを作成することを前提としています。

本節では、クライアント・アーティファクトを作成する方法と、Java SE方式そしてJava EE方式のWebサービスの呼び出しについて詳しく説明します。

### 5.2.1. クライアント・アーティファクトの作成

Webサービスのクライアントの呼び出しは、そのWebサービスがJavaクラスから作成されるか、WSDLファイルから作成されるかにかかわらず、そのWebサービスが配布するWSDLファイルを使用してクライアント・アーティファクトを作成した後に呼び出すJavaクラスの作成を通じて行われます。したがって本節では、前章でJavaクラスから作成したWebサービスを呼び出すクライアントのためのクライアント・アーティファクトを作成します。

Webサービスのクライアント・アーティファクト(プロキシ)は、そのサービスが提供するWSDLからJEUS 8 Webサービスで提供するツールである**wsimport**を使用して取得したサービス・インターフェースクラスとサービスエンドポイント・インターフェースクラスで構成されます。

コンソールでwsimportを使用してクライアント・アーティファクトを取得する方法は以下のとおりです。

```
$ wsimport -help
Usage: wsimport [options] <WSDL_URI>
```

where [options] include:

-b <path>	specify jaxws/jaxb binding files or additional schemas (Each <path> must have its own -b)
-B<jaxbOption>	Pass this option to JAXB schema compiler
-catalog <file>	specify catalog file to resolve external entity references supports TR9401, XCatalog, and OASIS XML Catalog format.
-d <directory>	specify where to place generated output files
-extension	allow vendor extensions - functionality not specified by the specification. Use of extensions may result in applications that are not portable or may not interoperate with other implementations
-help	display help
-httpproxy:<host>:<port>	specify a HTTP proxy server (port defaults to 8080)
-keep	keep generated files
-p <pkg>	specifies the target package
-quiet	suppress wsimport output
-s <directory>	specify where to place generated source files
-target <version>	generate code as per the given JAXWS spec version e.g. 2.0 will generate compliant code for JAXWS 2.0 spec
-verbose	output messages about what the compiler is doing
-version	print version information
-wsdllocation <location>	@WebServiceClient.wsdlLocation value

Examples:

```
wsimport stock.wsdl -b stock.xml -b stock.xjb
wsimport -d generated http://example.org/stock?wsdl
```

---

## 参考

JEUS 8 WebサービスはwsimportのAntタスクもサポートします。コンソール・コマンドとAntタスク項目の詳細については『*JEUS リファレンスガイド*』の「5.5.4. wsimport」と『*JEUS リファレンスガイド*』の「5.6.2. wsimport」を参照してください。

---

以下のようなリモートWebサービスのWSDLファイルを使用し、その下のwsimportのAntタスクを使用してポータブル・アーティファクトを作成します。

### [例 5.1] << http://host:port/AddNumbers/AddNumbersImplService?wsdl >>

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

```

<types>
  <xsd:schema>
    <xsd:import namespace="http://server.fromjava/"
      schemaLocation="AddNumbersImplService_schema1.xsd" />
  </xsd:schema>
</types>
<message name="addNumbers">
  <part name="parameters" element="tns:addNumbers" />
</message>
<message name="addNumbersResponse">
  <part name="parameters" element="tns:addNumbersResponse" />
</message>
<portType name="AddNumbersImpl">
  <operation name="addNumbers">
    <input message="tns:addNumbers" />
    <output message="tns:addNumbersResponse" />
  </operation>
</portType>
<binding name="AddNumbersImplPortBinding" type="tns:AddNumbersImpl">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
    style="document" />
  <operation name="addNumbers">
    <soap:operation soapAction="" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>
<service name="AddNumbersImplService">
  <port name="AddNumbersImplPort"
    binding="tns:AddNumbersImplPortBinding">
    <soap:address location="REPLACE_WITH_ACTUAL_URL" />
  </port>
</service>
</definitions>

```

#### [例 5.2] << build.xml >>

```

...

<target name="build_client" depends="do-deploy-success, init">
  <antcall target="wsimport">
    <param name="package.name" value="fromjava.client" />
    <param name="binding.file" value="" />
    <param name="wsdl.file"
      value="http://localhost:8088/AddNumbers/AddNumbersImplService?wsdl" />
  </antcall>

```

```

    <antcall target="do-compile">
        <param name="javac.excludes" value="fromjava/server/" />
    </antcall>
</target>

...

```

上のwsimport Antタスクで呼び出すWebサービスのWSDLファイルを使用してポータブル・アーティファクトを作成すると、以下のファイルが作成されます。

```

fromwsdl/client/AddNumbers.class
fromwsdl/client/AddNumbersImpl.class
fromwsdl/client/AddNumbersImplService.class
fromwsdl/client/AddNumbersResponse.class
fromwsdl/client/ObjectFactory.class
fromwsdl/client/package-info.class

```

AddNumbersとAddNumbersResponseファイルは、各JAXBがaddNumbersメソッドに対する要求と応答をJavaオブジェクトまたはXML文書に変換するために使用するJava Beanです。

AddNumbersImplファイルはサービス・エンドポイント・インターフェースを示します。AddNumbersImplServiceファイルはクライアントでプロキシ用途で使用するサービス・インターフェースのJavaクラスです。ObjectFactoryクラスとpackage-infoクラスはJAXBが作成したファイルです。

以下は、wsimportツールを使用して上のリモートWebサービスのWSDLファイルより取得したサービス・インターフェース・クラスのAddNumbersImplServiceクラスです。

#### [例 5.3] << AddNumbersImplService.java >>

```

@WebServiceClient(name = "AddNumbersImplService",
    targetNamespace = "http://server.fromjava/",
    wsdlLocation = "http://localhost:8088/AddNumbers/AddNumbersImplService?wsdl")
public class AddNumbersImplService extends Service {

    private final static URL ADDNUMBERSIMPLSERVICE_WSDL_LOCATION;
    private final static WebServiceException ADDNUMBERSIMPLSERVICE_EXCEPTION;
    private final static QName ADDNUMBERSIMPLSERVICE_QNAME =
        new QName("http://server.fromjava/", "AddNumbersImplService");

    static {
        URL url = null;
        WebServiceException e = null;
        try {
            url = new URL(
                "http://localhost:8088/AddNumbers/AddNumbersImplService?wsdl");
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
        ADDNUMBERSIMPLSERVICE_WSDL_LOCATION = url;
        ADDNUMBERSIMPLSERVICE_EXCEPTION = e;
    }
}

```

```

    }

    public AddNumbersImplService() {
        super(__getWsdllLocation(), ADDNUMBERSIMPLSERVICE_QNAME);
    }

    public AddNumbersImplService(WebServiceFeature... features) {
        super(__getWsdllLocation(), ADDNUMBERSIMPLSERVICE_QNAME, features);
    }

    public AddNumbersImplService(URL wsdlLocation) {
        super(wsdlLocation, ADDNUMBERSIMPLSERVICE_QNAME);
    }

    public AddNumbersImplService(URL wsdlLocation, WebServiceFeature... features) {
        super(wsdlLocation, ADDNUMBERSIMPLSERVICE_QNAME, features);
    }

    public AddNumbersImplService(URL wsdlLocation, QName serviceName) {
        super(wsdlLocation, serviceName);
    }

    public AddNumbersImplService(URL wsdlLocation, QName serviceName,
                                   WebServiceFeature... features) {
        super(wsdlLocation, serviceName, features);
    }

    @WebEndpoint(name = "AddNumbersImplPort")
    public AddNumbersImpl getAddNumbersImplPort() {
        return super.getPort(new QName("http://server.fromjava/",
                                         "AddNumbersImplPort"),
                              AddNumbersImpl.class);
    }

    @WebEndpoint(name = "AddNumbersImplPort")
    public AddNumbersImpl getAddNumbersImplPort(WebServiceFeature... features) {
        return super.getPort(new QName("http://server.fromjava/",
                                         "AddNumbersImplPort"),
                              AddNumbersImpl.class, features);
    }

    private static URL __getWsdllLocation() {
        if (ADDNUMBERSIMPLSERVICE_EXCEPTION != null) {
            throw ADDNUMBERSIMPLSERVICE_EXCEPTION;
        }
        return ADDNUMBERSIMPLSERVICE_WSDL_LOCATION;
    }
}

```

上のサービス・インターフェースは、クライアントからプロキシ・オブジェクトを取得する場合に使用されます。

クライアントでは、上のクラスでAddNumbersImplService構築子を通じて取得したAddNumbersImplServiceオブジェクトよりgetAddNumbersImplPort()メソッドを通じてサービス・エンドポイント・インターフェースであるAddNumbersImplを取得できます。

以下は、wsimportツールを使用して、上のリモートWebサービスのWSDLファイルから取得したサービス・エンドポイント・インターフェースであるAddNumbersImplクラスです。

**[例 5.4] << AddNumbersImpl.java >>**

```
@WebService(name = "AddNumbersImpl", targetNamespace = "http://server.fromjava/")
@XmlSeeAlso( { ObjectFactory.class })
public interface AddNumbersImpl {

    @WebMethod
    @WebResult(targetNamespace = "")
    @RequestWrapper(localName = "addNumbers",
        targetNamespace = "http://server.fromjava/",
        className = "fromjava.client.AddNumbers")
    @ResponseWrapper(localName = "addNumbersResponse",
        targetNamespace = "http://server.fromjava/",
        className = "fromjava.client.AddNumbersResponse")
    public int addNumbers(@WebParam(name = "arg0", targetNamespace = "")
        int arg0, @WebParam(name = "arg1", targetNamespace = "")
        int arg1);
}
```

上のように生成されたサービス・エンドポイント・インターフェースは、動的バインディングやランタイム時にJavaオブジェクトまたはXML文書に変換する際に必要なアノテーションを持っています。(このサービス・エンドポイント・インターフェースはWSDLとスキーマ・ファイルを再作成するのに使用されますが、このサービス・エンドポイント・インターフェースを取得するために使用されたWSDLやスキーマファイルと一致しない場合があります)

また、JAXBがaddNumbersメソッドに対する要求と応答をJavaオブジェクトまたはXML文書に変換するために使用するJava BeanクラスであるAddNumbersクラスとAddNumbersResponseクラスは以下のとおりです。

**[例 5.5] << AddNumbers.java >>**

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addNumbers", propOrder = { "arg0", "arg1" })
public class AddNumbers {

    protected int arg0;

    protected int arg1;

    public int getArg0() {
        return arg0;
    }
}
```



```

    public void setArg0(int value) {
        this.arg0 = value;
    }

    public int getArg1() {
        return arg1;
    }

    public void setArg1(int value) {
        this.arg1 = value;
    }
}

```

#### [例 5.6] << AddNumbersResponse.java >>

```

@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addNumbersResponse", propOrder = { "_return" })
public class AddNumbersResponse {

    @XmlElement(name = "return")
    protected int _return;

    public int getReturn() {
        return _return;
    }

    public void setReturn(int value) {
        this._return = value;
    }
}

```

上の2つのJava Beanクラスは、リモートWebサービスのWSDLファイルのメッセージ要素であるAddNumbers、AddNumbersResponseに該当することを確認できます。

## 5.2.2. Java SEクライアント方式

本節では、「[5.2.1. クライアント・アーティファクトの作成](#)」から取得したクライアント・ポータブル・アーティファクトを使用して、実際にリモートのWebサービスを呼び出すロジックを格納しているクライアント・クラスをJava SE方式で作成し、呼び出します。

### 5.2.2.1. Java SEクライアント・プログラムの作成

Java SEクライアント・クラスの実装は簡単です。上で取得したサービス・インターフェースのAddNumbersImplServiceオブジェクトを1つ作成し、そのオブジェクトからサービス・エンドポイント・インターフェースのAddNumbersImplを実装したオブジェクトを取得します。実際にこのオブジェクトは直接動的プロキシを通じてリモートのWebサービスを呼び出すロジックを格納しています。このようにサービス・エンドポイ

ント・インターフェースを実装するオブジェクトを取得した場合、これから実際のWebサービス呼び出すメソッドを呼び出します。

以下は、上のwsimport Antタスクを通じて取得したポータブル・アーティファクトを使用してリモートWebサービス呼び出すコードの一部です。

**[例 5.7] << AddNumbersClient.java >>**

```
public class AddNumbersClient {

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

        int number1 = 10;
        int number2 = 20;

        System.out.println("#####");
        System.out.println("### JAX-WS Webservicess examples - fromjava ###");
        System.out.println("#####");
        System.out.println("Invoking addNumbers(" + number1 + ", " + number2 + ")");
        int result = port.addNumbers(number1, number2);
        System.out.println("Result: " + result);
    }
}
```

## 5.2.2.2. Java SEクライアント・プログラムの呼び出し

本節では、上記で実装したクラスおよびその他の設定ファイルを使用してWebサービスを実装し、JEUS 8にデプロイした後にクライアントを実行する方法について説明します。

リモートのWebサービスにアクセスするクライアント・プログラムの呼び出し方式は、サービスをJavaで実装する場合もWSDLで実装する場合も同一です。

以下のように、Javaで実装するとfromjavaディレクトリーで、WSDLファイルで実装するとfromwsdlディレクトリーでサービスを作成してJEUS 8にデプロイします。

```
$ ant build deploy
```

上のプロセスがすべて実行され、サービスが正常にデプロイされたら、クライアントをビルドします。Java SEクライアントでwsimportのプロセスを経るため、サービスのデプロイがすべて終了した際にクライアントの構成が可能です。

以下のようにクライアントを作成し、サービスを呼び出します。

```
$ ant run
...

run:
    [java] #####
    [java] ##### JAX-WS Webservicess examples - fromjava #####
```

```
[java] #####
[java] Invoking addNumbers(10, 20)
[java] Result: 30

...

BUILD SUCCESSFUL
```

呼び出し結果にて、クライアントが正常にこのサービス呼び出し、値を取得したことを確認できます。

## 5.2.3. Java EEクライアント方式

本節では、前章でWSDL文書を通じて作成したWebサービスを呼び出すクライアントをJava EE方式で作成し、呼び出します。作成前に「[5.2.1. クライアント・アーティファクトの作成](#)」で説明した方式でクライアント・ポータブル・アーティファクトを取得したと仮定します。

### 5.2.3.1. Java EEクライアント・プログラムの作成

Java EEクライアント方式で重要な点は、既存のJAX-RPC環境のJava EE Webサービス・クライアント構成では、サーブレットの場合はweb.xmlに、EJBの場合はejb-jar.xmlに<service-ref>を追加してWebサービス・クライアント・プロキシの作成に必要な情報をJNDIに登録できますが、JAX-WS Webサービス環境でのJava EEクライアント構成は@WebServiceRefアノテーションを設定することで、それと同じ作業が可能であるということです。

以下は、@WebServiceRefアノテーションを設定する一例です。

```
...
@WebServiceRef(wsdlLocation="http://host:port/TmaxService/TmaxService?wsdl")
static TmaxServiceImplService tsvc;
...
```

上で取得したサービス・インターフェースであるAddNumbersServiceタイプを使用して、クライアントJavaクラスのメンバー変数として@WebServiceRefアノテーションと一緒に宣言します。そうすると、このメンバー変数はランタイム中にクライアント・クラスが初期化される際にsetメソッドがなくても、サーブレットの場合はサーブレット・コンテナ、EJBの場合はEJBコンテナで自動的にこの値を挿入します。

これからサービス・エンドポイント・インターフェースのAddNumbersPortTypeを実装したオブジェクトを取得します。実際にこのオブジェクトは直接動的プロキシを通じてリモートのWebサービスを呼び出すロジックを格納しています。このようにサービス・エンドポイント・インターフェースを実装するオブジェクトを取得した場合、実際のWebサービスを呼び出すメソッドを呼び出します。

以下は、上のwsimport Antタスクを通じて取得したポータブル・アーティファクトを使用して、リモートのWebサービスを呼び出すクライアント・プログラムです。

#### [例 5.8] << AddNumbersClient.java >>

```
public class AddNumbersClient extends HttpServlet {

    @WebServiceRef
    static AddNumbersImplService svc;

    protected void doGet(HttpServletRequest arg0, HttpServletResponse arg1)
        throws ServletException, IOException {
        AddNumbersImpl port = svc.getAddNumbersImplPort();

        int number1 = 10;
        int number2 = 20;

        System.out.println("#####");
        System.out.println("### JAX-WS Webservicex examples - fromjava ###");
        System.out.println("#####");
        System.out.println("Invoking addNumbers(" + number1 + ", " + number2 + ")");
        int result = port.addNumbers(number1, number2);
        System.out.println("#####");
        System.out.println("### JAX-WS Webservicex examples - fromjava ###");
        System.out.println("#####");
        System.out.println("Result: " + result);
    }
}
```

### 5.2.3.2. Java EEクライアント・プログラムの呼び出し

上記で実装したクラスおよびその他の設定ファイルを使用してWebサービスを実装し、JEUS 8にデプロイした後にJava EEクライアントを実行できます。リモートWebサービスにアクセスするクライアント・プログラムの呼び出し方式は、サービスがJavaで実装する場合もWSDLで実装する場合も同一です。

以下のように、Javaで実装するとfromjavaディレクトリで、WSDLファイルで実装するとfromwsdlディレクトリでサービスを作成してJEUS 8にデプロイします。

```
$ ant build deploy
```

上のプロセスがすべて実行され、サービスが正常にデプロイされたら、Java EEクライアントをビルドします。Java EEクライアントでwsimportのプロセスを経るため、サービスのデプロイがすべて終了した際にクライアントの構成が可能です。

以下のようにクライアントを作成し、サービスを呼び出します。

```
$ ant run
...

#####
### JAX-WS Webservicex examples - fromjava ###
#####
Invoking addNumbers(10, 20)
```

```
...

#####
### JAX-WS Webservices examples - fromjava ###
#####
Result: 30

...
```

上のように入力すると、Webブラウザにてクライアント・サーブレットが正常にこのサービス呼び出し、サーバー上のログを通じて値を確認できます。

## 5.3. ディスパッチ方式のWebサービスの呼び出し

ディスパッチ方式のWebサービスの呼び出しは、XML構成を`java.lang.transform.Source`あるいは`javax.xml.soap.SOAPMessage`、つまりXMLレベルで扱う開発者のためのものです。ディスパッチ方式のWebサービスの呼び出しメッセージ・モードとPayloadモードで使用でき、XML/HTTPバインディング(`javax.activation.DataSource`)でREST Webサービスを作成する際に使用できます。

詳細については「[第8章 プロバイダーとディスパッチ・インターフェース](#)」を参照してください。



# 第6章 標準バインディング宣言とカスタマイズ

本章では、WSDL文書でJavaクラスを使用した標準バインディング宣言方法およびカスタマイズの詳細について説明します。

## 6.1. 概要

前章で説明したとおり、基本的なJAX-WS WebサービスはJavaクラスまたはWSDL文書で実装できます。

より拡張可能で多様な機能を提供するWebサービスを実装するために、様々な機能を追加したり、多様な設定をしたりすることができます。その方法の1つとして、このような機能や設定をJavaクラスにアノテーションを記述することで、wsngenツールを使用して取得できます。また別の方法としては、WSDL文書に直接あるいは間接的に機能を追加または設定し、wsimportツールを使用して取得できます。

本章では、WSDL文書でwsimportツールを使用してWebサービスを構成する際や、クライアントを構成する際に使用できるバインディングのカスタマイズについて説明します。より詳しい機能についての紹介と設定方法、およびJavaクラスからwsngenツールを使用してWebサービスを実装する方法については、次章で説明します。

JEUS Webサービスは、WSDL文書からJavaクラスへのバインディング宣言およびカスタマイズ(主にwsimportツールで作業する場合)を、JAX-WSで要求する標準化された方式でサポートします。既存のJAX-RPC方式のWebサービスではこのような標準が規定されていなかったため、ベンダー間で相互運用できないWebサービスが生成される問題がありました。

このようなJEUS 8 WebサービスのWSDL文書でJavaクラスの標準化されたバインディング宣言およびカスタマイズは、Webサービスを実装するのに以下の2つの役割をします。

- ほぼすべてのWSDLコンポーネントから、サービス・エンドポイント・インターフェース・クラス、メソッド名、パラメータ名、例外クラスなど、Java言語へのマッピングをカスタマイズできます。
- 非同期化、プロバイダー、Wrapper方式、付加的なヘッダーのような機能をカスタマイズできます。

## 6.2. 標準バインディングの宣言

すべてのバインディングに関連した要素は<http://java.sun.com/xml/ns/jaxws>名前空間に属し、その名前空間に属するプレフィックスであるjaxwsをつけてjaxws:bindingsという要素でバインディングを宣言します。

バインディングを宣言する位置によって2種類に分けることができます。

- 外部文書を通じてバインディングを宣言する方法

- バインディングするWSDL文書内に直接バインディング宣言を含める方法

## 6.2.1. 外部文書(ファイル)での直接宣言

外部文書(ファイル)で直接宣言する方法は、主にWebサービスを使用するクライアントによって、wsimportツールのパラメータにWSDL文書の位置と一緒に渡されます。ここで使用されるWSDL文書の位置の値は、以下の例のように、バインディング宣言文書のwsdlLocationという要素値で追加します。

### [例 6.1] << custom-client.xml >>

```
<jaxws:bindings
  wsdlLocation="http://localhost:8080/AddNumbers/addnumbers?WSDL"
  jaxws:xmlns="http://java.sun.com/xml/ns/jaxws">
  ...
</jaxws:bindings>
```

WSDL文書のコンポーネントのバインディングを設定するには、上記例で宣言した最上位要素の子要素として追加します。宣言する対象コンポーネントは、nodeという要素内にXPath文法を使用して配置します。また、そのコンポーネントに対して、宣言するバインディングをその下にある子要素として追加します。

以下は、WSDL文書のコンポーネントのバインディングを設定するために要素を追加した例です。

### [例 6.2] << custom-client.xml >>

```
<jaxws:bindings
  wsdlLocation="http://localhost:8088/AddNumbers/addnumbers?WSDL"
  jaxws:xmlns="http://java.sun.com/xml/ns/jaxws">
  <jaxws:bindings node="wsdl:definitions"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    <jaxws:package name="customize.client"/>
    ...
  </jaxws:bindings>
```

上記例でこのバインディング宣言は、WSDL文書内の「wsdl:definitions」に対して作成されるJavaクラスに対し、そのパッケージ名をcustomize.clientにカスタマイズするという意味です。

このように作成された外部バインディング宣言文書は、wsimportツールで以下のように実行できます。

```
$ wsimport -b custom-client.xml http://localhost:8088/AddNumbers/addnumbers?WSDL
```

## 6.2.2. WSDL文書内での直接宣言

WSDL文書内で直接宣言する方法は、前述の外部文書(ファイル)で直接宣言する方法とは以下のような違いがあります。

- WSDL文書内で拡張要素として<jaxws:bindings>要素を使用します。



- WSDL文書内で宣言する場合、ノード属性は使用しません。
- <jaxws:bindings>要素内で子要素として<jaxws:bindings>要素を使用しません。
- <jaxws:bindings>要素は、それを囲んでいるWSDLコンポーネントにのみ影響を与えます。

**[例 6.3] << AddNumbers.wsdl >>**

```
<wsdl:portType name="AddNumbersImpl">
  <jaxws:bindings xmlns:jaxws="http://java.sun.com/xml/ns/jaxws">
    <jaxws:class name="MathUtil"/>
    ...
  </jaxws:bindings>
  <wsdl:operation name="addNumber">
    ...
  </wsdl:operation>
</wsdl:portType>
```

上記例は、WSDL文書内でのwsdl:portTypeに対するカスタマイズを示します。

<jaxws:bindings>という拡張要素は、このwsdl:portTypeより作成されるサービス・エンドポイント・インターフェース・クラス名のカスタマイズを示しますが、標準デフォルト値で作られるクラス名（ここではAddNumbersImpl）をMathUtilでカスタマイズすることを示します。

## 6.3. 標準バインディングのカスタマイズ

本節では、以下の宣言されたバインディングのカスタマイズについて詳しく説明しています。

- 全体的なバインディング
- パッケージ名のカスタマイズ
- Wrappedスタイル
- 非同期化
- プロバイダー・インターフェース
- クラス名のカスタマイズ
- Javaメソッドのカスタマイズ
- Javaパラメータのカスタマイズ
- XMLスキーマのカスタマイズ
- ハンドラー・チェーンのカスタマイズ

### 6.3.1. 全体的なバインディング

全体的なバインディングは、外部文書ファイルで定義されたバインディング宣言に有効で、`jaxws:bindings@wsdlLocation`に明示したWSDL文書の`wsdl:definition`全体に適用されます。

```
<jaxws:package name="..." />
<jaxws:enableWrapperStyle />
<jaxws:enableAsyncMapping />
```

全体的なバインディングに該当する要素は以下のとおりです。

以下は、実際に使用される例です。

#### [例 6.4] << custom-client.xml >>

```
<jaxws:bindings
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  wsdlLocation="http://localhost:8080/AddNumbers/addnumbers?WSDL"
  xmlns="http://java.sun.com/xml/ns/jaxws">
  <package name="customize.client" />
  <enableWrapperStyle>true</enableWrapperStyle>
  <enableAsyncMapping>false</enableAsyncMapping>
</jaxws:bindings>
```

### 6.3.2. パッケージ名のカスタマイズ

デフォルト値として`wsimport`ツールを使用してWSDL文書からJavaクラスを作成する場合、クラスのパッケージ名はそのWSDLファイル文書の名前空間によって決められます。

このようなクラスのパッケージ名をカスタマイズして他の値に指定するためには、`jaxws:package`要素でバインディング・カスタマイズ宣言をします。`wsimport`ツールを使用して作成する場合、`-p`オプションを使用してパッケージ名を変更できます。これは、`jaxws:package`要素でバインディング・カスタマイズしたものを上書きします。

#### [例 6.5] << custom-client.xml >>

```
<jaxws:bindings
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  wsdlLocation="http://localhost:8080/AddNumbers/addnumbers?WSDL"
  xmlns="http://java.sun.com/xml/ns/jaxws">
  <package name="customize.client" />
  ...
</jaxws:bindings>
```

上記例は以下のように表現することもできます。

**[例 6.6] << custom-client.xml >>**

```
<jaxws:bindings
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  wsdlLocation="http://localhost:8080/jaxws-external-customize/addnumbers?WSDL"
  xmlns="http://java.sun.com/xml/ns/jaxws">
  <bindings node="wsdl:definitions">
    <package name="customize.client" />
  ...
</jaxws:bindings>
```

### 6.3.3. Wrappedスタイル

wsimportツールは、デフォルト値としてWSDL文書内のwsdl:portTypeで定義された抽象オペレーションに対し、Wrappedスタイルの規則を適用します。このようなWrappedスタイルは、バインディング・カスタマイズによって可能とならないようにできます。

**[例 6.7] << custom-client.xml >>**

```
<jaxws:bindings
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  wsdlLocation="http://localhost:8080/AddNumbers/addnumbers?WSDL"
  xmlns="http://java.sun.com/xml/ns/jaxws">
  <enableWrapperStyle>true</enableWrapperStyle>
  <bindings node="wsdl:definitions/"
    wsdl:portType[@name='AddNumbersImpl']/
    wsdl:operation[@name='addNumbers']">
    <enableWrapperStyle>false</enableWrapperStyle>
  ...
</jaxws:bindings>
```

上記のように、jaxws:enable WrapperStyle要素として設定でき、wsdl:definitions、wsdl:portType、wsdl:operation要素の下でそれぞれ使用できます。

- wsdl:definitionsの下で使用される場合、すべてのwsdl:portType属性のすべてのwsdl:operations要素に適用されます。
- wsdl:portTypeの下で使用される場合、そのポートタイプのすべてのwsdl:operationsに適用されます。
- wsdl:operationの下に適用される場合、該当オペレーションに対してのみ適用されます。

### 6.3.4. 非同期化

クライアント・アプリケーションは、jaxws:enableAsyncMappingbinding要素を宣言することによって、wsimportする場合に非同期ポーリングもしくはコールバック方式のオペレーションを作成します。非同期クライアント・アプリケーションについては、「[第9章 非同期Webサービス](#)」を参照してください。

適用されるWSDL文書内のコンポーネントおよび適用範囲は、上で説明したWrappedスタイルの規則と同じです。

**[例 6.8] << custom-client.xml >>**

```
<jaxws:bindings
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  wsdlLocation="http://localhost:8080/AddNumbers/addnumbers?WSDL"
  xmlns="http://java.sun.com/xml/ns/jaxws">
  <enableAsyncMapping>false</enableAsyncMapping>
  <bindings node="wsdl:definitions/"
    wsdl:portType[@name='AddNumbersImpl']/
    wsdl:operation[@name='addNumbers']">
    <enableAsyncMapping>true</enableAsyncMapping>
    ...
  </jaxws:bindings>
```

### 6.3.5. プロバイダー・インターフェース

本節では、プロバイダー・インターフェースをバインディング宣言を通じて設定する方法について説明します。詳細については「[第8章 プロバイダーとディスパッチ・インターフェース](#)」を参照してください。

WSDL文書内の特定ポート番号をプロバイダー・インターフェースに設定したい場合、wsdl:portノードをXPath形式にしてjaxws:providerバインディングを設定します。デフォルト値は「設定しない」です。これは、WebサービスをWSDLから作成する場合に有効です。

### 6.3.6. クラス名のカスタマイズ

WSDL文書内のwsdl:portType、wsdl:fault、soap:headerfault、wsdl:serverはJavaクラスに作成されますが、ここでjaxws:classバインディング宣言を行うと、希望するクラス名に変更できます。本節では、このようなWSDL文書内のコンポーネントに対する各Javaクラスについて説明します。

- サービス・エンドポイント・インターフェース・クラス

以下は、サービス・エンドポイント・インターフェース・クラス名に対するバインディング・カスタマイズの例です。

**[例 6.9] << custom-client.xml >>**

```
<jaxws:bindings node="wsdl:definitions/wsdl:portType[@name='AddNumbersImpl']">
  <jaxws:class name="TmaxUtil" />
</jaxws:bindings>
```

上記例でのサービス・エンドポイント・インターフェース・クラス名は、デフォルト値の「AddNumbersImpl」ではなく「TmaxUtil」となります。

- 例外クラス

以下は、例外クラス名に対するバインディング・カスタマイズの例です。

**[例 6.10] << custom-client.xml >>**

```
<jaxws:bindings node="wsdl:definitions/  
  wsdl:portType[@name='AddNumbersImpl']/  
  wsdl:operation[@name='addNumbers']/  
  wsdl:fault[@name='AddNumbersException']">  
  <jaxws:class name="TmaxException" />  
</jaxws:bindings>
```

上記例での例外クラス名は、デフォルト値の「AddNumbersException」ではなく「TmaxException」となります。

- サービス・クラス

以下は、サービス・クラス名に対するバインディング・カスタマイズの例です。

**[例 6.11] << custom-client.xml >>**

```
<jaxws:bindings node="wsdl:definitions/  
  wsdl:service[@name='AddNumbersService']">  
  <jaxws:class name="TmaxService" />  
</jaxws:bindings>
```

上記例でのサービス・クラス名は、デフォルト値の「AddNumbersService」ではなく「TmaxService」となります。

## 6.3.7. Javaメソッドのカスタマイズ

サービス・エンドポイントのメソッド名あるいはサービス・クラスのポートを取得するためのメソッド名をカスタマイズする場合、jaxws:methodバインディング宣言を使用します。

- サービス・エンドポイント・インターフェース・メソッド

以下は、サービス・エンドポイント・インターフェースのメソッド名を変更するバインディング宣言の例です。

**[例 6.12] << custom-client.xml >>**

```
<jaxws:bindings node="wsdl:definitions/  
  wsdl:portType[@name='AddNumbersImpl']/  
  wsdl:operation[@name='addNumbers']">  
  <jaxws:method name="add" />  
</jaxws:bindings>
```

wsimportツールは、サービス・エンドポイント・インターフェースのメソッド名として、デフォルト値の「addNumbers」メソッド名の代わりに「add」というメソッド名を持ちます。

- ポートにアクセスするためのサービス・クラスのメソッド

以下は、ポートにアクセスするためのサービス・クラスのメソッド名を変更するバインディング宣言の例です。

**[例 6.13] << custom-client.xml >>**

```
<jaxws:bindings node="wsdl:definitions/  
  wsdl:service[@name='AddNumbersService']/  
  wsdl:port[@name='AddNumbersImplPort']">  
  <jaxws:method name="getTmaxUtil" />  
</jaxws:bindings>
```

wsimportツールは、ポートにアクセスするためのサービス・クラスのメソッド名として、デフォルト値の「getAddNumbersImplPort」の代わりに「getTmaxUtil」というメソッド名を持ちます。

## 6.3.8. Javaパラメータのカスタマイズ

jaxws:parameterバインディング宣言は、作成されるJavaメソッドのパラメータ名を変更する際に使用します。wsdl:operationあるいはwsdl:portTypeのメソッド・パラメータを変更できます。

**[例 6.14] << custom-client.xml >>**

```
<jaxws:bindings node="wsdl:definitions/  
  wsdl:portType[@name='AddNumbersImpl']/  
  wsdl:operation[@name='addNumbers']">  
  <jaxws:parameter part="definitions/  
    message[@name='addNumbers']/  
    part[@name='parameters']"  
    element="tns:number1" name="num1"/>  
</jaxws:bindings>
```

上記のように、Javaパラメータのカスタマイズ宣言は、wsdl:operationのメソッド・パラメータを「number1」から「num1」に変更されたことを確認できます。

## 6.3.9. XMLスキーマのカスタマイズ

WSDL文書内で宣言されたXMLスキーマは、jaxws:bindings要素の配下で標準JAXBバインディング要素を使用することによって変更できます。

**[例 6.15] << custom-client.xml >>**

```
<jaxws:bindings node="wsdl:definitions/  
  wsdl:types/  
  xsd:schema[@targetNamespace='http://tmaxsoft.com']">  
  <jaxb:schemaBindings>  
    <jaxb:package name="fromwsdl.server"/>  
  </jaxb:schemaBindings>  
</jaxws:bindings>
```

```
</jaxb:schemaBindings>
</jaxws:bindings>
```

また、WSDL文書によってインポートされるXMLスキーマ・ファイルもバイnding・カスタマイズできます。この場合、JAXB標準拡張バイnding宣言ファイルを使用します。

**[例 6.16] << custom-client.xml >>**

```
<jxb:bindings
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:jxb="http://java.sun.com/xml/ns/jaxb"
  version="1.0">
  <jxb:bindings
    schemaLocation=
      "http://localhost:8088/AddNumbers/schema1.xsd"
    node="/xsd:schema">
    <jxb:schemaBindings>
      <jxb:package name="fromjava.client"/>
    </jxb:schemaBindings>
  </jxb:bindings>
  ...
```

外部標準JAXBバイnding宣言ファイルは、wsimportツールの-bスイッチによって渡されます。

## 6.3.10. ハンドラー・チェーンのカスタマイズ

jaxws:bindingsバイnding宣言は、ハンドラーの追加やカスタマイズにも使用されます。ハンドラーについての詳細は「[第7章 ハンドラー・フレームワーク](#)」を参照してください。

バイnding・カスタマイズ宣言にハンドラー事項を追加あるいは変更する際は、ハンドラー・チェーン設定に関するスキーマ(JAR 181)に明示されたように、jaxws:bindingsにハンドラー・チェーン設定を行います。

**[例 6.17] << custom-client.xml >>**

```
<jaxws:bindings
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:jxb="http://java.sun.com/xml/ns/jaxb"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  wsdlLocation="http://localhost:8080/jaxws-fromwsdlhandler/addnumbers?WSDL"
  xmlns:jaxws="http://java.sun.com/xml/ns/jaxws"
  xmlns:javaee="http://java.sun.com/xml/ns/javaee">
  <jaxws:bindings node="wsdl:definitions">
    <javaee:handler-chain>
      <javaee:handler-chain-name>
        LoggingHandlers
      </javaee:handler-chain-name>
      <javaee:handler>
        <javaee:handler-name>Logger</javaee:handler-name>
```

```
        <javaee:handler-class>
            fromwsdlhandler.common.LoggingHandler
        </javaee:handler-class>
    </javaee:handler>
</javaee:handler-chain>
</jaxws:bindings>
</jaxws:bindings>
```

このような外部バインディング宣言文書とWSDL文書を、wsimportツールを使用してポータブル・アーティファクトを作成すると、ハンドラー設定ファイルが作成されます。また、作成されたサービス・エンドポイント・インターフェース・クラスには、JAX-WSランタイムがハンドラーを検索するための@javax.jws.HandlerChainアノテーションが追加されます。



# 第7章 ハンドラー・フレームワーク

本章では、Webサービスのハンドラー・フレームワークについての基本概念と構成、その設定方法を説明します。

## 7.1. 概要

JAX-WS Webサービスは、ハンドラーのために容易なプラグイン形式のフレームワークを提供します。これは、JEUS 8 Webサービスのランタイム・システム機能を向上させます。

ハンドラーのタイプには以下の2つがあります。

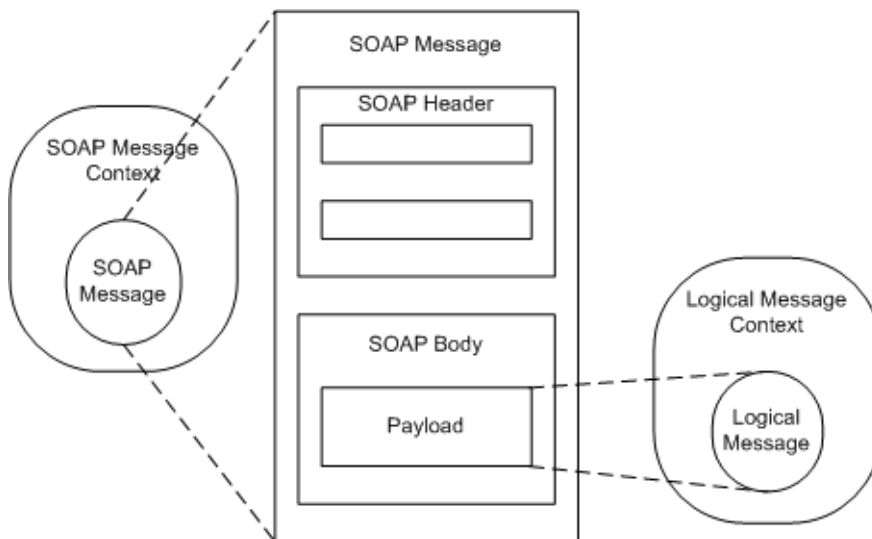
- 論理ハンドラー(Logical Handler)

プロトコルに関係なく、メッセージ格納領域(payload)にアクセス可能なハンドラーです。

- SOAPハンドラー(SOAP Handler)

ヘッダーを含むSOAPメッセージ全体にアクセス可能なハンドラーです。

[図 7.1] メッセージ・コンテキスト間の関連付け



ハンドラー・フレームワークは、以下の基準に従って使用されます。

- SOAPメッセージ全体を必要とする場合、SOAPハンドラーを使用します。
- SOAPメッセージのXML文書の格納領域のみを必要とする場合、論理ハンドラーを使用します。

その他の場合はエンドポイント・クラスで処理するようにWebサービスを構成し、特別にJavaオブジェクトを必要とする場合はJEUS EJBでサポートするインターセプター(Interceptor)を使用します。JEUS EJBインターセプターについては『JEUS EJBガイド』を参照してください。

## 7.2. ハンドラー・チェーンの優先順位

ハンドラー・チェーンは以下の図のように、Wireから送信されるメッセージの場合、すべての論理ハンドラーがSOAPハンドラーより先に処理されます。一方、受信するメッセージの場合は、すべてのSOAPハンドラーが論理ハンドラーより先に処理されます。

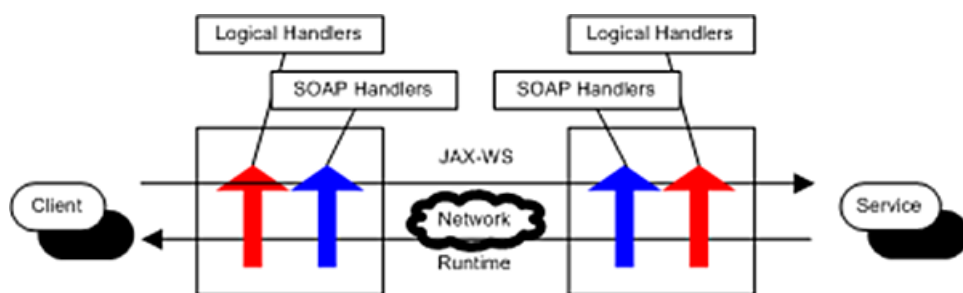
---

### 注

クライアントまたはサービス・エンドポイント側では、論理ハンドラーをSOAPハンドラーより先に設定しているため、サービスの生成や呼び出し時には、すべての論理ハンドラーはSOAPハンドラーより先に処理されます。

---

[図 7.2] ハンドラー・フレームワーク



## 7.3. ハンドラー・クラスの構成

本節では、ハンドラー・クラスを構成する方法と、ハンドラー・クラスで使用するメッセージ・コンテキスト・クラスについて説明します。

### 7.3.1. ハンドラー・クラスの宣言

ユーザーはハンドラー・クラスを構成するために、論理ハンドラーまたはSOAPハンドラーのインターフェースを実装するクラスを作成します。

以下は、各ハンドラーを構成した例です。

**[例 7.1] << MyLogicalHandler.java >>**

```
public class MyLogicalHandler implements
    LogicalHandler<LogicalMessageContext> {
    public boolean handleMessage(
        LogicalMessageContext messageContext) {
        LogicalMessage msg = messageContext.getMessage();
        return true;
    }
}
```

**[例 7.2] << MySOAPHandler.java >>**

```
public class MySOAPHandler implements
    SOAPHandler<SOAPMessageContext> {
    public boolean handleMessage(SOAPMessageContext messageContext) {
        SOAPMessage msg = messageContext.getMessage();
        return true;
    }
}
```

上記のように、論理ハンドラーとSOAPハンドラーは両方ともハンドラー・インターフェースを実装しています。ハンドラー・インターフェースはhandleMessage()とhandleFault()メソッドを持っています。

これらのメソッドは、メッセージ・コンテキスト・クラスを継承するオブジェクトがパラメータに渡されます。そのオブジェクトは、現在ハンドラーに渡されたメッセージが受信したものであるのか送信するものであるのかを区分します。また、このようなユーザー・ハンドラー・クラスは、以下のように@PostConstructアノテーションと@PreDestroyアノテーションを使用できます。

**[例 7.3] << MyLogicalHandler.java >>**

```
public class MyLogicalHandler implements
    LogicalHandler<LogicalMessageContext> {
    @PostConstruct
    public void methodA() {}

    @PreDestroy
    public void methodB() {}
}
```

上記のように宣言されたMyLogicalHandlerで、@PostConstructアノテーションで宣言されたメソッドである「methodA」はこのハンドラーが作成された後、@PreDestroyアノテーションで宣言されたメソッドである「methodB」はこのハンドラーがなくなる前に呼び出されます。

## 7.4. ハンドラー・クラスの設定

本節では、構成したユーザー・ハンドラーをWebサービスに適用する方法について説明します。

## 7.4.1. JavaクラスによるWebサービスの構成

JavaクラスからWebサービスを構成する際、サービス・エンドポイント実装クラスに@HandlerChainアノテーションで設定し、wsгенツールを使用してWebサービスを構成します。

### [例 7.4] << MyServiceImpl.java >>

```
@WebService
@HandlerChain( file="handlers.xml")
public class MyServiceImpl {
    ...
}
```

上記のように、@HandlerChainアノテーションに設定したhandlers.xmlは以下のとおりです。サーバーのハンドラー設定はhandlers.xmlというメタデータ・ファイルを使用して設定されます。

### [例 7.5] << handlers.xml >>

```
<?xml version="1.0" encoding="UTF-8"?>
<jws:handler-chains xmlns:jws="http://java.sun.com/xml/ns/javaee">
  <jws:handler-chain>
    <jws:handler>
      <jws:handler-class>fromjava.handler.TmaxHandler</jws:handler-class>
    </jws:handler>
  </jws:handler-chain>
</jws:handler-chains>
```

## 7.4.2. WSDLによるWebサービスの構成

WSDLからWebサービスを構成する際は、Webサービスを構成するWSDL文書に間接的にバインディング・カスタマイズを設定し、wsimportツールを使用してWebサービスを構成します。

以下は、外部ファイルを使用してWSDL文書に間接的にバインディングをカスタマイズする例です。

```
<bindings xmlns="http://java.sun.com/xml/ns/jaxws">
  <handler-chains xmlns="http://java.sun.com/xml/ns/javaee">
    <handler-chain>
      <handler>
        <handler-class>fromwsdl.handler.TmaxHandler</handler-class>
      </handler>
    </handler-chain>
  </handler-chains>
</bindings>
```

上のように、バインディング・カスタマイズ・ファイルに<handler-chains>が追加されています。

以下のように、<handler-chains>の下に<handler-chain>を複数設定することもできます。

```

<handler-chains xmlns="http://java.sun.com/xml/ns/javaee">
  <handler-chain>
    <service-name-pattern xmlns:tm="http://tmaxsoft.com">
      tm:Tmax*Service
    </service-name-pattern>
    <handler>...</handler>
  </handler-chain>

  <handler-chain>
    <port-name-pattern xmlns:tm="http://tmaxsoft.com">
      tm:TmaxPort
    </port-name-pattern>
    <handler>...</handler>
  </handler-chain>

  <handler-chain>
    <protocol-bindings>##SOAP11_HTTP</protocol-bindings>
    <handler>...</handler>
  </handler-chain>
</handler-chains>

```

上のように複数の<handler-chain>が1つの<handler-chains>で構成される場合、或るハンドラーにサービス名やポート名、あるいはプロトコルのような属性を付与できます。

### 7.4.3. クライアントの構成

上のサービスの構成のうち、WSDLからWebサービスを構成する方法と同一です。

## 7.5. ハンドラー・チェーンを使用するWebサービスの例

本節では、ログを出力するユーザーSOAPハンドラーを実装し、Webサービスに使用する簡単な例について説明します。

論理ハンドラー・クラスに属するjavax.xml.ws.handler.LogicalHandlerあるいはSOAPハンドラーに属するjavax.xml.ws.handler.SOAPHandlerクラスは、抽象インターフェースのjavax.xml.ws.handler.Handlerを継承しています。この2つのクラスを実装することで、希望するハンドラーを作成することができます。

ここで実装するハンドラー・クラスの「LoggingHandler」は、SOAPハンドラーのjavax.xml.ws.handler.soap.SOAPHandlerを実装するクラスです。

#### [例 7.6] << LoggingHandler.java >>

```

public class LoggingHandler implements SOAPHandler<SOAPMessageContext> {

    public Set<QName> getHeaders() {
        return null;
    }
}

```

```

public void close(MessageContext messageContext) {

}

public boolean handleFault(SOAPMessageContext smc) {
    return true;
}

public boolean handleMessage(SOAPMessageContext smc) {
    Boolean inboundProperty =
        (Boolean) smc.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);

    System.out.println("/n#####");
    System.out.println("### JAX-WS Webservies examples - handler ###");
    System.out.println("#####");

    if (inboundProperty.booleanValue()) {
        System.out.println("/nClient message:");
    } else {
        System.out.println("/nServer message:");
    }

    SOAPMessage message = smc.getMessage();
    try {
        message.writeTo(System.out);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return true;
}
}

```

上のように、このハンドラーに入るメッセージがサーバーで受信されたメッセージであるのか、あるいはクライアントから送信されたメッセージであるのかをチェックして出力します。

## サービス・クラス

以下は、この例のサービス部分のJavaクラスです。サービス部分で@HandlerChainアノテーションを使用してサーバーのハンドラー部分を設定しています。

### [例 7.7] << handlers.xml >>

```

<?xml version="1.0" encoding="UTF-8"?>
<handler-chains xmlns="http://java.sun.com/xml/ns/javaee">
    <handler-chain>
        <handler>
            <handler-class>fromjavahandler.common.LoggingHandler</handler-class>
        </handler>
    </handler-chain>
</handler-chains>

```

```

        </handler>
    </handler-chain>
</handler-chains>

```

以下は、@HandlerChainに登録されているファイルのhandlers.xmlの例です。

#### [例 7.8] << AddNumbersImpl.java >>

```

@WebService
@HandlerChain(file = "handlers.xml")
public class AddNumbersImpl {

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}

```

## クライアント・クラス

以下は、この例のクライアント側のJavaクラスです。クライアント部分ではWSDL文書を使用してクライアントを作成する際に使用するwsimportツールにバインディング・ユーザー宣言を追加します。

#### [例 7.9] << AddNumbersClient.java >>

```

public class AddNumbersClient {

    public static void main(String[] args) {
        AddNumbersImpl port = new AddNumbersImplService().getAddNumbersImplPort();
        port.addNumbers(10, 20);
    }
}

```

以下は、このようなバインディング・ユーザー宣言の例です。

#### [例 7.10] << custom-client.xml >>

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bindings xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  wsdlLocation="http://localhost:8088/AddNumbers/addnumbers?wsdl"
  xmlns="http://java.sun.com/xml/ns/jaxws">
  <bindings node="wsdl:definitions"
    xmlns:jws="http://java.sun.com/xml/ns/javaee">
    <jws:handler-chains>
      <jws:handler-chain>
        <jws:handler>
          <jws:handler-class>
            fromjavahandler.common.LoggingHandler
          </jws:handler-class>

```

```

        </jws:handler>
    </jws:handler-chain>
</jws:handler-chains>
</bindings>
</bindings>

```

## 7.6. Webサービスのハンドラー・フレームワークの実行

本節では、上で実装したクラスとその他の設定ファイルを使用して、ハンドラー・フレームワークを実行する方法について説明します。その他のサービス・エンドポイント・インターフェースの実装クラスおよびその他の設定ファイルは、前章で説明した例の内容と同一です。

以下のように、ハンドラー・フレームワークを設定したサービスを作成してJEUSにデプロイします。

```
$ ant build deploy
```

サービスを正常にデプロイされた後、クライアントをビルドして呼び出します。クライアントでwsimportのプロセスを経るので、サービスのデプロイがすべて完了してからクライアントを構成することができます。

以下のように、ハンドラー・フレームワークを設定したクライアントを作成し、サービス呼び出します。コンソールで入力すると、LoggingHandlerによって、サービスおよびクライアント両方の画面にメッセージの送受信がすべて表示されます。

```

$ ant run

...

run:

[java] #####
[java] ### JAX-WS Webservicess examples - handler ###
[java] #####

[java] Client message:
[java] <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Body><ns2:addNumbers xmlns:ns2="http://server.fromjavahandler/"><arg0>10</arg0>
<arg1>20</arg1></ns2:addNumbers></S:Body></S:Envelope>
[java] #####
[java] ### JAX-WS Webservicess examples - handler ###
[java] #####

[java] Server message:
[java] <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/><S:Body><ns2:addNumbersResponse xmlns:ns2="http://server.fromjavahandler/">
<return>30</return></ns2:addNumbersResponse></S:Body></S:Envelope>

...

```



BUILD SUCCESSFUL



# 第8章 プロバイダーとディスパッチ・インターフェース

本章では、サービス・エンドポイントのプロバイダー・インターフェースと、クライアントのディスパッチ・インターフェースについて説明します。

## 8.1. 概要

Webサービスのサービスとクライアント間の通信はXMLベースのメッセージを使用します。このとき、JAX-WS サービス・エンドポイント・インターフェースは、JavaオブジェクトからXMLベースのメッセージあるいはXMLベースのメッセージからJavaオブジェクトへの複雑な相互変換過程を隠すハイレベルの抽象化を提供します。しかしこの場合は、このようなサービスとクライアント間のXMLベースのメッセージ交換をメッセージ・レベルで(SOAPメッセージ・ハンドラーのような複雑なプログラミング・モデルのサポートなく)処理することを開発者が好む場合があります。

JEUS 8 Webサービスは、サービス・エンドポイントの場合、**プロバイダー**というインターフェースとクライアントの場合、**ディスパッチ**というインターフェースを提供します。これによって交換されるメッセージをメッセージ・レベルでアクセスできるようにサポートします。プロバイダーとディスパッチのインターフェースはWebサービスの開発者やそのサービスを消費するクライアントの開発者にメッセージをXMLメッセージ・レベルで処理できるようにします。

本章では、この2つのインターフェースについて説明します。

## 8.2. サービス・エンドポイントのプロバイダー・インターフェース

本節では、プロバイダー・インターフェースについて説明します。

### 8.2.1. プロバイダー・インターフェース

プロバイダー・インターフェースは、Webサービス・エンドポイントがXML文書をXMLメッセージ・レベルで取り扱う場合に使用します。そのため、エンドポイントはメッセージあるいはメッセージ格納領域に低レベルのGeneric(JavaSE 5の新機能) APIを通じてこれにアクセスできます。

プロバイダー・インターフェースの使用方法是以下のとおりです。

1. @javax.xml.ws.WebServiceProviderアノテーションを持っている必要があります。

2. 該当クラスは、Provider<javax.xml.transform.Source>、Provider<javax.xml.soap.SOAPMessage>、Provider<javax.activation.DataSource>を実装する必要があります。

- Provider<javax.xml.transform.Source>を実装してメッセージ格納領域を使用

メッセージ格納領域をSourceオブジェクトとして使用するには、@ServiceModeアノテーションをService.Mode.PAYLOAD値に設定します。

これは、@WebServiceProviderアノテーションさえ宣言されていれば自動で設定されるデフォルト値であるため、省略が可能です。

```
@WebServiceProvider
public class ProviderImpl implements Provider<Source> {
    public Source invoke(Source source) {
        ...
    }
}
```

- Provider<javax.xml.soap.SOAPMessage>を実装してメッセージを使用

メッセージをSOAPMessageオブジェクトとして使用するには、@ServiceModeアノテーションをService.Mode.Message値に設定します。

```
@WebServiceProvider
@ServiceMode(value=Service.Mode.MESSAGE)
public class ProviderImpl implements Provider<SOAPMessage> {
    public SOAPMessage invoke(SOAPMessage msg) {
        ...
    }
}
```

- Provider<javax.activation.DataSource>を実装してメッセージを使用

メッセージをSourceオブジェクトとして使用するには、@ServiceModeアノテーションをService.Mode.Message値に設定します。要求メッセージがSOAPメッセージの場合、Attachmentを除いたSOAP Part部分のみがSourceオブジェクトに渡されます。戻り値がnull値の場合、このWebサービスはOne-Way方式のWebサービスであることを意味します。

```
@WebServiceProvider
@ServiceMode(value=Service.Mode.MESSAGE)
public class ProviderImpl implements Provider<Source> {
    public Source invoke(Source source) {
        ...
        return null;
    }
}
```

3. エンドポイントがメッセージ(Service.Mode.MESSAGE)にアクセスするのか、あるいは格納領域(Service.Mode.PAYLOAD)にアクセスするのかについては、@javax.xml.ws.ServiceModeアノテーションが担当します。

@javax.xml.ws.ServiceModeアノテーションがない場合は格納領域のみを担当します。

実際にWebサービスのエンドポイントを構成するときに、プロバイダー・インターフェースを実装するサービス実装クラスは、エンドポイント・インターフェースに対するいかなる情報も提供できません。したがって、必ずWSDLからWebサービスを作成する方式を採用する必要があります。

また、プロバイダー・インターフェースを実装するサービス実装クラスでWebサービスを構成する場合、WSDLから取得したポータブル・アーティファクトのうち、あるポートはプロバイダー・インターフェースを実装するサービス実装クラスによって実際に使用されません。したがって、wsimportツールを使用してWSDLからポータブル・アーティファクトを作成するにあたり、性能上の論点が大い場合は、このようなポートが作成されないように外部バインディング宣言を使用できます。

## 8.2.2. プロバイダー・インターフェースの例

以下は、Provider<Source>インターフェースを実装してメッセージの格納領域部分を作成するためのサービス実装クラスの例です。

### [例 8.1] << AddNumbersImpl.java >>

```
@ServiceMode(value = Service.Mode.PAYLOAD)
@WebServiceProvider(wsdlLocation = "WEB-INF/wsdl/AddNumbers.wsdl",
    targetNamespace = "http://tmaxsoft.com",
    serviceName = "AddNumbersService", portName = "AddNumbersPort")
public class AddNumbersImpl implements Provider<Source> {
    public Source invoke(Source source) {
        try {
            DOMResult dom = new DOMResult();
            Transformer trans = TransformerFactory.newInstance().newTransformer();
            trans.transform(source, dom);
            Node node = dom.getNode();
            Node root = node.getFirstChild();
            Node first = root.getFirstChild();
            int number1 = Integer.decode(first.getFirstChild().getNodeValue());
            Node second = first.getNextSibling();
            int number2 = Integer.decode(second.getFirstChild().getNodeValue());
            int sum = number1 + number2;

            String body =
                "<ns:addNumbersResponse xmlns:ns='http://tmaxsoft.com/'><ns:return>"
                + sum + "</ns:return></ns:addNumbersResponse>";
            Source sumsource =
                new StreamSource(new ByteArrayInputStream(body.getBytes()));
```

```

        return sumsource;
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
}

```

上記例では、AddNumbersImplというクラスは、@WebServiceProviderというアノテーションと一緒に Provider<Source>を実装し、@ServiceModeアノテーションでメッセージ格納領域を作成しています。

### 8.2.3. プロバイダー・インターフェースの例の実行

上記で実装したクラスおよびその他の設定ファイルを使用してプロバイダー・インターフェースを実装するWebサービスを実行する方法は以下のとおりです。

以下のように、プロバイダー・インターフェースを実装するWebサービスを作成してJEUSにデプロイします。

```
$ ant deploy
```

上のプロセスがすべて実行され、サービスが正常にデプロイされたら、クライアントをビルドします。クライアントでwsimportのプロセスを経るため、サービスのデプロイがすべて終了したときにクライアントの構成が可能です。

以下のように、クライアントを作成してサービスを呼び出します。画面にメッセージの送受信を表示し、プロバイダー・インターフェースを実装するWebサービスを確認できます。

```

$ ant run

...

run:
    [java] #####
    [java] ### JAX-WS Webservices examples - Provider ###
    [java] #####
    [java] Testing Provider webservices...
    [java] Success!

...

BUILD SUCCESSFUL

...

---[HTTP request]---
Host: localhost:8088
Content-length: 199
Content-type: text/xml; charset=utf-8
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg, *; q=.2,

```

```

**; q=.2
Connection: keep-alive
Soapaction: ""
User-agent: JAX-WS RI 2.2 - JEUS 8
<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><addNumbers xmlns="http://tmaxsoft.com"><arg0>10</arg0><arg1>20</arg1></addNumbers></S:Body></S:Envelope>
-----
---[HTTP response 200]---
<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><ns:addNumbersResponse xmlns:ns="http://tmaxsoft.com"><ns:return>30</ns:return></ns:addNumbersResponse></S:Body></S:Envelope>
-----
...

```

## 8.3. クライアントのディスパッチ・インターフェース

本節では、クライアントのディスパッチ・インターフェースについて説明します。

### 8.3.1. ディスパッチ・インターフェース

サービス・エンドポイントのプロバイダー・インターフェースと対称になる概念で、クライアントでもアプリケーションがXML文書をメッセージ・レベルで使用できます。

サーバー側のプロバイダー・インターフェースと同様に、クライアントのディスパッチ・インターフェースを実装するクライアント・アプリケーションも、WSDLから取得したポータブル・アーティファクトのうち、あるポートはディスパッチ・インターフェースを実装するクラスによって実際に使用されません。したがって、wsimportツールを使用してWSDLからポータブル・アーティファクトを作成するにあたり、性能上の論点が大きき場合は、このようなポートが作成されないように外部バインディング宣言を使用できます。

ディスパッチ・インターフェースの使用方法は以下のとおりです。

1. javax.xml.transform.Source、javax.xml.soap.SOAPMessage、javax.activation.DataSourceを実装します。

サービス・エンドポイントのプロバイダー・インターフェースがProvider<Source>を実装してクライアントのメッセージから格納領域部分を作成したり、Provider<SOAPMessage>あるいはProvider<Source>を実装してクライアントのメッセージを作成するのと同様に、クライアントのディスパッチ・インターフェースは、SOAPMessage、SourceもしくはJAXBオブジェクトでメッセージを構成できます。

2. Dispatchオブジェクトを作成するためのサービス・オブジェクトを作成(動的なサービス・オペレーションの呼び出し)します。

サービス(javax.xml.ws.Service)は、動的なサービスを作成するためのファクトリーの役割をします。動的なサービスの作成とは、WSDLなしでServiceオブジェクトを作成するために実際のサービスのバインディング情報を動的に割り当てて作成するためのものです。

以下は、動的なサービスを作成するために使用するコードの例です。

```
Service service = Service.createService(QName serviceName);
Service service = Service.createService(URL wsdlLocation,
    QName serviceName);
```

また、このように作成されたServiceオブジェクトに対し、Dispatchオブジェクトがバインドされている特定のポートやエンドポイントを次のようにaddPort()メソッドを通じて追加できます。このように追加されたポートを通じて、クライアント・アプリケーションはディスパッチ・インターフェースを使用して呼び出すことができます。

上のServiceオブジェクトをwsdlLocationの情報を通じて動的に作成する場合、以下のメソッドは動作しないことに注意してください。既にwsdlLocationの情報を通じてサービスはポートを構成しているためです。

```
service.addPort(QName portName, String SOAPBinding.SOAP11HTTP_BINDING,
    String endpointAddress);
service.addPort(QName portName, String SOAPBinding.SOAP12HTTP_BINDING,
    String endpointAddress);
service.addPort(QName portName, String HTTPBinding.HTTP_BINDING,
    String endpointAddress);
```

上記例は、それぞれの順番どおり、SOAP 1.1、SOAP 1.2、XML/HTTPのバインディングをQNameとエンドポイント・アドレスを使用してポートを構成する方式です。

WSDLファイルから作成されるServiceオブジェクトには上のオペレーションは無効です。その理由は、WSDLファイルからwsimportツールを使用して作成する際のサービスは、このようなポート・バインディングあるいはQName、エンドポイント・アドレスのような情報を、既に作成されているポータブル・アーティファクトの中でServiceオブジェクトが分かっているためです。

以下は、一般的なWSDLファイルからwsimportツールを使用して作成されたポータブル・アーティファクトのうち、Serviceオブジェクトを作成するサンプル・コードです。

```
Service service = new AddNumbersService();
```

3. Dispatchオブジェクトを作成します。Serviceオブジェクトを動的に、あるいはWSDLファイルから作成した場合、Dispatchオブジェクトは以下のサービス・クラスの2つのメソッドを通じて作成されます。

```
Dispatch dispatch = service.createDispatch(QName portName,
    Class clazz, Service.Mode mode);
Dispatch dispatch = service.createDispatch(QName portName,
    JAXBContext jaxbcontext, Service.Mode mode);
```

Dispatchオブジェクトは、javax.xml.transform.SourceあるいはJAXBデータ・バインディング・オブジェクトを使用できます。この2つの場合、ServiceオブジェクトからService.Mode.PAYLOADあるいは



Service.Mode.MESSAGEモードとcreateDispatchメソッドを使用して、Dispatchオブジェクトを作成します。

また、Dispatchオブジェクトがjavax.xml.soap.SOAPMessageを使用する場合は、ServiceオブジェクトからService.Mode.MESSAGEモードとcreateDispatchメソッドを使用してDispatchオブジェクトを作成します。

## 8.3.2. ディスパッチ・インターフェースの例

以下は、リモートWebサービスのWSDL文書からディスパッチ・インターフェースを実装したクライアントWebサービスの例です。

### [例 8.2] << AddNumbersClient.java >>

```
public class AddNumbersClient {

    public static void main(String[] args) {
        try {
            Service service = new AddNumbersService();
            String request = "<addNumbers xmlns='http://tmaxsoft.com/'>
                <arg0>10</arg0><arg1>20</arg1></addNumbers>";
            QName portQName = new QName("http://tmaxsoft.com", "AddNumbersPort");
            Dispatch<Source> sourceDispatch = service.createDispatch(portQName,
                Source.class, Service.Mode.PAYLOAD);

            System.out.println("#####");
            System.out.println("### JAX-WS Webservices examples - Dispatch ###");
            System.out.println("#####");
            System.out.println("Testing Dispatch webservices...");

            Source result =
                sourceDispatch.invoke(new StreamSource(new StringReader(request)));

            DOMResult dom = new DOMResult();
            Transformer trans = TransformerFactory.newInstance().newTransformer();
            trans.transform(result, dom);
            Node node = dom.getNode();
            Node root = node.getFirstChild();
            Node first = root.getFirstChild();
            int sum = Integer.decode(first.getFirstChild().getNodeValue());
            if (sum == 30) {
                System.out.println("Success!");
            } else {
                System.out.println("Fail!");
            }
        } catch (ProtocolException jex) {
            jex.printStackTrace();
        } catch (TransformerException e) {
            e.printStackTrace();
        }
    }
}
```

```

    }
}
}

```

上記例では、AddNumbersClientというクラスはWSDLから取得したServiceオブジェクトに対し、createDispatch()メソッドを使用してDispatch<Source>タイプのsourceDispatchオブジェクトを取得してプログラミングしています。

### 8.3.3. ディスパッチの例の実行

上記で実装したクラスおよびその他の設定ファイルを使用してディスパッチ・インターフェースを実装するWebサービスを実行する方法は以下のとおりです。

ディスパッチ・インターフェースを実装するWebサービスを作成してJEUSにデプロイします。

```
$ ant deploy
```

上のプロセスがすべて実行され、正常にデプロイされたら、クライアントをビルドします。クライアントでwsimportのプロセスを経るため、サービスのデプロイがすべて終了したときにクライアントの構成が可能です。

以下のように、クライアントを作成してサービスを呼び出します。画面にメッセージの送受信を表示し、ディスパッチ・インターフェースを実装するWebサービスを確認できます。

```

$ ant run

...

run:
    [java] #####
    [java] ### JAX-WS Webservicess examples - Dispatch ###
    [java] #####
    [java] Testing Dispatch webservicess...
    [java] Success!

...

BUILD SUCCESSFUL

...

---[HTTP request]---
Host: localhost:8088
Content-length: 199
Content-type: text/xml; charset=utf-8
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Soapaction: ""

```

```

User-agent: JAX-WS RI 2.2 - JEUS 8
<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><addNumbers xmlns="http://tmaxsoft.com"><arg0>10</arg0><arg1>20</arg1></addNumbers></S:Body></S:Envelope>
-----
---[HTTP response 200]---
<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><addNumbersResponse xmlns="http://tmaxsoft.com"><return>30</return></addNumbersResponse></S:Body></S:Envelope>
-----
...

```

## 8.4. XML/HTTPバインディング

Webサービスではメッセージを送受信するときに、SOAPメッセージではなく、XML文書そのものをHTTP経由で送信する場合があります。このようなWebサービスをRESTful Webサービスといいます。基本的に、RESTful Webサービスのサービスとクライアントは、プロバイダーとディスパッチ・インターフェースを使用します。プロバイダーとディスパッチ・インターフェースについては「[8.2. サービス・エンドポイントのプロバイダー・インターフェース](#)」と「[8.3. クライアントのディスパッチ・インターフェース](#)」を参照してください。

本節では、このようなRESTful WebサービスをサポートするためのJEUS WebサービスのXML/HTTPバインディングについて説明します。

### 8.4.1. RESTful Webサービス

RESTful Webサービスの主な特徴は以下のとおりです。

- HTTPのGETリクエストもエンドポイントに渡すことができます。
- 標準MessageContext.QUERY\_STRINGとMessageContext.PATH\_INFOを通じて、必要なHTTPリクエストのクエリ文字列(Query String)とパス(path)情報を取得します。

RESTful Webサービスは通常、WSDL文書を使用してWebサービスとクライアントを構成するのではなく、あらかじめ約束されたXML文書のスキーマを使用して、WebサービスのプロデューサーとコンシューマはWebサービスを構成します。Webサービスの供給者はwsгенツールを使用してサービス・クラスを構成し、クライアントはポータブル・アーティファクトのサポートなく独立的にアプリケーションを構成します。

JAX-WSは、プロバイダーとディスパッチ・インターフェースを使用することでXML/HTTPバインディングをサポートし、プロデューサーとコンシューマはRESTful Webサービスを容易に構成できます。

### サービス・エンドポイントの構成

「[8.2. サービス・エンドポイントのプロバイダー・インターフェース](#)」では、プロバイダー・インターフェースを実装するために、サービス・エンドポイントがProvider<Source>またはProvider<SOAPMessage>を実装する

クラスに作成して、バインディングされたソースまたはSOAPMessageオブジェクトを使用して、メッセージ格納領域あるいは全体を取り扱うようにできることを説明しました。

プロバイダー・エンドポイントはバインディング識別子を使用して他のバインディングに設定できますが、このようなバインディング識別子はエンドポイント・クラスが@BindingTypeアノテーションを設定することで可能になります。このようなバインディング識別子がない場合、基本バインディングであるSOAP1.1/HTTPが宣言されます。

以下は、このようなバインディング識別子を使用してXML/HTTPバインディングを宣言した例です。

```
@ServiceMode(value=Service.Mode.MESSAGE)
@BindingType(value=HTTPBinding.HTTP_BINDING)
public class ProviderImpl implements Provider<Source> {
    public Source invoke(Source source) {
        ...
    }
}
```

上のように、サービス・エンドポイント・クラスは受信するメッセージをXML/HTTPでバインディングすると宣言したため、Provider<SOAPMessage>ではなく、Provider<Source>を実装しています。

## クライアントの構成

RESTful Webサービスのクライアントの構成は、「[8.3.1. ディスパッチ・インターフェース](#)」の方法2で説明したように、動的なServiceオブジェクトを作成し、HTTPBinding.HTTP\_BINDINGでポートを作成して、以下のようプログラミングします。

```
Service service = Service.createService(QName serviceName);
service.addPort(QName portName, String HTTPBinding.HTTP_BINDING,
    String endpointAddress);
```

このように作成されたServiceオブジェクトを使用してDispatchオブジェクトを作成し、必要に応じてこのServiceオブジェクトにPOSTまたはGET方式でリクエストを指定し、必要なクエリ・ストリング(MessageContext.QUERY\_STRING)とパス(MessageContext.PATH\_INFO)情報を登録してサービスを呼び出します。

以下は、このようなクライアント・コードを構成する例です。

```
...

Dispatch<Source> d = service.createDispatch(portQName, Source.class,
    Service.Mode.MESSAGE);
Map<String, Object> requestContext = d.getRequestContext();
requestContext.put(MessageContext.HTTP_REQUEST_METHOD, new String("GET"));
requestContext.put(MessageContext.QUERY_STRING, queryString);
requestContext.put(MessageContext.PATH_INFO, path);
Source result = d.invoke(null);

...
```

---

## 注

invokeメソッドに何のパラメータも渡さないことに注意してください。

---

### 8.4.2. RESTful Webサービスの例

本節では、RESTful Webサービスとクライアントの例について説明します。

#### RESTful Webサービスの例

以下は、RESTful Webサービスでサービスに該当するJavaクラスの例です。

##### [例 8.3] << AddNumbersImpl.java >>

```
@WebServiceProvider(serviceName = "AddNumbersService")
@ServiceMode(value = Service.Mode.MESSAGE)
@BindingType(value = HTTPBinding.HTTP_BINDING)
public class AddNumbersImpl implements Provider<Source> {

    @Resource(type = Object.class)
    protected WebServiceContext wsContext;

    public Source invoke(Source source) {
        try {
            MessageContext mc = wsContext.getMessageContext();
            String query = (String) mc.get(MessageContext.QUERY_STRING);
            StringTokenizer st = new StringTokenizer(query, "&/");
            st.nextToken();
            int number1 = Integer.parseInt(st.nextToken());
            st.nextToken();
            int number2 = Integer.parseInt(st.nextToken());
            int sum = number1 + number2;
            String body =
                "<ns:addNumbersResponse xmlns:ns=/\"urn:AddNumbers/\"><ns:return>"
                + sum + "</ns:return></ns:addNumbersResponse>";
            Source resultsrc =
                new StreamSource(new ByteArrayInputStream(body.getBytes()));
            return resultsrc;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

上記では、RESTful Webサービスの宣言は、BindingTypeアノテーションを使用して HTTPBinding.HTTP\_BINDINGを設定し、リソース・アノテーションを使用してWebServiceContextタイプの

メンバー変数のwsContextを設定しています。wsContextはランタイムにJAX-WS RIが自動で注入されます。

このクラスの大部分が、WebServiceContextからMessageContextを取得し、これを使用してクエリ・ストリング値を取得後、sendSourceメソッドを使用してパーシングおよび返すソースを直接作成して、これを戻り値として返します。

## RESTful Webサービスのクライアントの例

以下は、RESTful Webサービスでクライアントに該当するJavaクラスの例です。

### [例 8.4] << AddNumbersClient.java >>

```
public class AddNumbersClient {

    public static void main(String[] args) throws Exception {
        String endpointAddress = "http://localhost:8088/AddNumbers/addnumbers";
        URL url = new URL(endpointAddress + "?num1=10&num2=20");
        System.out.println("#####");
        System.out.println("### JAX-WS Webservices examples - RESTful ###");
        System.out.println("#####");
        System.out.println("Testing RESTful webservices...");
        InputStream in = url.openStream();
        StreamSource source = new StreamSource(in);

        try {
            ByteArrayOutputStream bos = new ByteArrayOutputStream();
            StreamResult sr = new StreamResult(bos);
            Transformer trans = TransformerFactory.newInstance().newTransformer();
            Properties oprops = new Properties();
            oprops.put(OutputKeys.OMIT_XML_DECLARATION, "yes");
            trans.setOutputProperties(oprops);
            trans.transform(source, sr);
            System.out.println(bos.toString());
            bos.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

上記のように、RESTful Webサービスのクライアントは、WSDLから作成するポータブル・アーティファクト・コードは表示しません。URLオブジェクトを使用して直接作成したソースをストリーム形式で表示します。

### 8.4.3. RESTful Webサービスの例の実行

本節では、実装したクラスおよびその他の設定ファイルを使用してRESTful Webサービスを実行する方法は以下のとおりです。

以下のように、RESTful Webサービスを作成してJEUSにデプロイします。

```
$ ant build deploy
```

上のプロセスがすべて実行され、サービスが正常にデプロイされたら、クライアントをビルドします。

RESTful Webサービスのためのクライアントを作成してサービスを呼び出します。以下のようにコンソールで入力すると、メッセージの送受信を表示します。

```
$ ant run

...

run:
  [java] #####
  [java] ### JAX-WS Webservices examples - RESTful ###
  [java] #####
  [java] Testing Restful webservices...
  [java] <ns:addNumbersResponse xmlns:ns="urn:AddNumbers"><ns:return>30</ns:r
eturn></ns:addNumbersResponse>

...

BUILD SUCCESSFUL

...

---[HTTP request]---
Host: localhost:8088
Content-type: application/x-www-form-urlencoded
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
User-agent: JAX-WS RI 2.2 - JEUS 8

-----
---[HTTP response 200]---
<ns:addNumbersResponse xmlns:ns="urn:AddNumbers"><ns:return>30</ns:return></ns:a
ddNumbersResponse>
-----

...
```





# 第9章 非同期Webサービス

本章では、クライアントの非同期オペレーションと非同期プロバイダーを使用した非同期Webサービスについて説明します。

## 9.1. 概要

サービスとクライアント間のWebサービスの呼び出しにおいて、クライアントはサーバーの応答を受信するまでそのスレッドをブロックし、待機しています。このような非効率性を改善するために、JAX-WS Webサービスは以下のような非同期オペレーションを構成します。

- クライアント側でJAX-WS APIを使用した非同期オペレーション

JAX-WS Webサービスのクライアントを構成するためには、呼び出し対象のサービスのWSDLファイルを利用します。サービスWSDLファイルをバインディング・カスタマイズ宣言を通じて、静的な非同期メソッドを持つサービス・エンドポイント・インターフェース・スタブを作り、それを実現するクライアント・クラスを構成することによって、クライアントの非同期オペレーションを構成する方法です。

- サービス側でJEUSが提供する非標準的な非同期オペレーション

Webサービス・エンドポイントを、Servlet 3.0の非同期処理方式で動作する非同期Webサービスで構成する方法です。JAX-WS標準は、サービス側の非同期オペレーションを規定していません。

## 9.2. クライアントの非同期オペレーション

本節では、クライアント側でJAX-WS APIを使用した非同期オペレーションについて説明します。

### 9.2.1. 非同期メソッドを持つサービス・エンドポイント・インターフェース・スタブの使用法

非同期化wsdl:operationは、ポーリング(Polling)とコールバック(Callback)のメソッドでマッピングされます。ポーリング・メソッドは`javax.xml.ws.Response`インターフェースを、Callbackメソッドは`javax.xml.ws.AsyncHandler`インターフェースを返します。まず、サービスのWSDLから、このような非同期メソッドを持つサービス・エンドポイント・インターフェース・スタブを取得するために使用する非同期化バインディング宣言について説明します。

### 9.2.1.1. 非同期化バインディングの宣言

WSDLで明示されているwsdl:operation要素を非同期化されたマッピングに使用するには、JEUS Webサービスの wsimportのようなツールを使用し、非同期化wsdl:operationマッピングに基づいたサービス・エンドポイント・インターフェースを作成する必要があります。

本節では、このような非同期化wsdl:operationマッピングを作成する方法について説明します。

非同期化wsdl:operationのマッピングを作成するには、wsimportツールに基づいたバインディング・カスタマイズ設定が必要です。以下は、バインディング設定ファイルであるcustom-schema.xmlの一例です。

#### [例 9.1] << custom-schema.xml >>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bindings xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  wsdlLocation="http://localhost:8088/AddNumbers/addnumbers?wsdl"
  xmlns="http://java.sun.com/xml/ns/jaxws">
  <bindings node="wsdl:definitions">
    <enableAsyncMapping>true</enableAsyncMapping>
  </bindings>
</bindings>
```

上記のように、wsdl:definitionsに対してバインディングを設定すると、このWSDL文書内のすべてのwsdl:operation要素に対して非同期化設定が行われます。

以下は、custom-schema.xmlファイルを使用するbuild.xmlの一部分です。

#### [例 9.2] << build.xml >>

```
...

<target name="build_client" depends="do-deploy-success, init">
  <antcall target="wsimport">
    <param name="package.name" value="async.client" />
    <param name="binding.file" value="-b ${src.conf}/custom-client.xml" />
    <param name="wsdl.file"
      value="http://localhost:8088/AddNumbers/addnumbers?wsdl" />
  </antcall>
  <antcall target="do-compile">
    <param name="javac.excludes" value="fromjava/server/" />
  </antcall>
</target>

...
```

以下は、バインディング・カスタマイズ宣言で、wsimportツールを使用してポータブル・アーティファクトを作成すると、作成された非同期メソッドを持つサービス・エンドポイント・インターフェースです。

```

public int addNumbers(int number1, int number2)
    throws java.rmi.RemoteException;
public Response<AddNumbersResponse> addNumbers(int number1, int number2);
public Future<?> addNumbers(int number1, int number2,
                            AsyncHandler<AddNumbersResponse>);

```

上記のように、Response<AddNumbersResponse>とFuture<?>を戻り値として持つメソッドが2つ作成されます。これは、それぞれポーリング方式とコールバック方式のメソッドです。以降で、これらを使用してクライアントJavaクラスをどのように構成するかについて説明します。

### 9.2.1.2. 非同期化クライアントの構成

非同期化クライアントは、ポーリング・メソッドあるいはCallbackメソッドを使用して構成できます。

#### ポーリング・メソッドを使用するクライアントの構成方法

以下は、wsimportツールによって取得した非同期サービス・エンドポイント・インターフェースのポーリング・メソッドです。

```

public Response<AddNumbersResponse> addNumbers(int number1, int number2);

```

以下は、このようなポーリング・メソッドでマッピングされたメソッドを使用して実装したクライアントWebサービスの例の一部です。クライアント・アプリケーションは、サービス・エンドポイント・インターフェースの非同期ポーリング・メソッドを呼び出し、いつ結果値が返されるのかを確認します。

```

javax.xml.ws.Response<AddNumbersResponse> resp = port.addNumbersAsync(10, 20);
while(!resp.isDone()){
}
System.out.println(resp.get().getReturn());
...

```

上記のようにポーリング・メソッドでマッピングされたメソッドは、javax.xml.ws.Responseタイプのオブジェクトを返します。これは、java.util.concurrent.Future<T>から継承されたisDone()メソッドを使用することによって、このオペレーションが完了し、結果を返す時点を決定的にできます。

以下は、ポーリング・メソッドをサポートするクライアント・アプリケーションのサンプル・コードの一部です。

#### [例 9.3] << AddNumbersClient.java >>

```

public class AddNumbersClient {

    ...

    public static void main(String[] args) {
        try {
            AddNumbersImpl port = new AddNumbersService().getAddNumbersImplPort();

            // Asynchronous polling
            Response<AddNumbersResponse> resp = port.addNumbersAsync(10, 20);

```

```

        Thread.sleep(2000);
        AddNumbersResponse output = resp.get();
        System.out.println("#####");
        System.out.println("### JAX-WS Webservices examples - polling ###");
        System.out.println("#####");
        System.out.printf("call webservices in an Asynchronous Polling way...");
        System.out.printf("result : %d/n", output.getReturn());

        ...

    } catch (Exception e) {
        e.printStackTrace();
    }
}

...

}

```

## Callbackメソッドを使用するクライアントの構成方法

wsimportツールから取得した非同期サービス・エンドポイント・インターフェースのCallbackメソッドは以下のとおりです。

```
public Future<?> addNumbers(int number1, int number2, AsyncHandler<AddNumbersResponse>);
```

Callbackメソッドでマッピングされたメソッドは、クライアント開発者がjavax.xml.ws.AsyncHandlerパラメータを追加で実装したハンドラー・オブジェクトを提供します。

以下は、このようなAsyncHandlerを実装したハンドラー・オブジェクトのサンプル・コードです。

### [例 9.4] << AddNumbersClient.java >>

```

public class AddNumbersClient {
    ...
    static class AddNumbersCallbackHandler implements
        AsyncHandler<AddNumbersResponse> {

        private AddNumbersResponse output;

        public void handleResponse(Response<AddNumbersResponse> response) {
            try {
                output = response.get();
            } catch (ExecutionException e) {
                e.printStackTrace();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

```

```

        AddNumbersResponse getResponse() {
            return output;
        }
    }
}

```

上のようにAsyncHandlerを追加で実装したハンドラー・オブジェクトは、ランタイムにサーバーからそのWebサービス・オペレーションの結果を取得できるときにhandleResponseというメソッドを呼び出し、クライアント・アプリケーションはgetResponse()メソッドを通じて結果値を得ることができます。

以下は、上で実装したハンドラー・オブジェクトを利用してサービス・エンドポイント・インターフェースの非同期Callbackメソッドを使用するクライアント・アプリケーションのサンプル・コードの一部です。クライアント・アプリケーションは、サービス・エンドポイント・インターフェースの非同期Callbackメソッドを呼び出し、結果値が得られる時点を確認できます。

```

AddNumbersCallbackHandler callbackHandler = new AddNumbersCallbackHandler();
Future<?> resp = port.addNumbersAsync(number1, number2, callbackHandler);
while(!resp.isDone()){
}
System.out.println(callbackHandler.getResponse().getReturn());

```

上のように、Callbackメソッドでマッピングされたメソッドは、javax.util.concurrent.Futureタイプのオブジェクトを返します。これは、isDone()メソッドを通じて、このオペレーションが完了し、結果を返す時点を決定的にします。

以下は、サービス・エンドポイント・インターフェースの非同期Callbackメソッドを利用するクライアント・アプリケーションのサンプル・コードの一部です。

#### [例 9.5] << AddNumbersClient.java >>

```

public class AddNumbersClient {

    public static void main(String[] args) {
        try {
            AddNumbersImpl port = new AddNumbersService().getAddNumbersImplPort();
            ...

            // Asynchronous callback
            AddNumbersCallbackHandler callbackHandler = new AddNumbersCallbackHandler();
            Future<?> response = port.addNumbersAsync(10, 20, callbackHandler);
            Thread.sleep(2000);

            output = callbackHandler.getResponse();
            System.out.println("#####");
            System.out.println("### JAX-WS Webservices examples - callback ###");
            System.out.println("#####");
            System.out.printf("call webservices in an Asynchronous Callback way...");
            System.out.printf("result: %d/n", output.getReturn());
        } catch (Exception e) {

```

```

        e.printStackTrace();
    }
}
...
}

```

### 9.2.1.3. 非同期化クライアントの実行

本節では、実装したクラスおよびその他の設定ファイルを使用してハンドラー・フレームワークを実行する方法について説明します。

以下のように、非同期化オペレーションを設定したWebサービスを作成し、JEUSにデプロイします。

```
$ ant build deploy
```

上のプロセスがすべて実行され、サービスが正常にデプロイされたら、クライアントをビルドします。

ハンドラー・フレームワークを設定したクライアントを作成し、クライアントからサービスを呼び出します。

コンソールで以下のように入力すると、ポーリング方式とコールバック方式の、2回の応答を正常に受信したことが確認できます。

```

$ ant run

...

run:
    [java] #####
    [java] ### JAX-WS Webservicess examples - polling ###
    [java] #####
    [java] call webservicess in an Asynchronous Polling way...result : 30
    [java] #####
    [java] ### JAX-WS Webservicess examples - callback ###
    [java] #####
    [java] call webservicess in an Asynchronous Callback way...result: 30

...

BUILD SUCCESSFUL

```

### 9.2.2. ディスパッチ・インターフェースの使用方法

ディスパッチ・インターフェースを使用してクライアントの非同期オペレーションを使用することは、基本的に非同期メソッドを持つサービス・エンドポイント・インターフェース・スタブを利用するのと同じ概念です。ただし、作成されたDispatchオブジェクトで提供するinvokeAsyncメソッドを使用します。

以下は、invokeAsyncメソッドを使用する例です。

```
Response<T> response = dispatch.invokeAsync(T);
Future<?> response = dispatch.invokeAsync(T, AsyncHandler);
```

上のようにinvokeAsync(T)はポーリング方式の非同期をサポートするメソッドで、invokeAsync(T, AsyncHandler)はコールバック方式の非同期をサポートするメソッドです。AsyncHandlerはコールバック方式で使用するためにユーザー・ハンドラーを実装したものです。

## 9.3. 非同期Webサービス

JEUS JAX-WSは、Servlet 3.0の非同期処理ベースの非同期Webサービスの構成方法を提供します。Webサービス・エンドポイントは、要求を受けて応答を送るまで同期化され、サーブレット・コンテナが割り当てた要求処理スレッドを占有し、サービスでの処理時間が長くなります。これによって他の要求が待機する状況が頻繁に発生することがあります。この場合、処理時間が長いサービスでは別途のスレッドを割り当て、要求処理スレッドはコンテナに返し、待機要求を効率的に処理します。

本節では、非同期Webサービスの設定方法について説明します。

### 9.3.1. 非同期Webサービスの設定

JEUSは、JAX-WS Webサービスを非同期Webサービスで構成するために以下の方法を提供します。

- **@jeus.webservices.jaxws.api.AsyncWebServiceアノテーションを使用する場合**

以下は、アノテーションを使用してWebサービスを非同期Webサービスに設定する例です。

**[例 9.6] << AddNumbersImpl.java >>**

```
@WebService(serviceName="AddNumbers")
@AsyncWebService
public class AddNumbersImpl {

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

- **web.xmlの<async-supported>を使用する場合**

既の実装されたWebサービスは、Servlet 3.0がサポートするweb.xmlのasync-supportedを使用して非同期Webサービスに設定できます。

**[例 9.7] << web.xml >>**

```
<web-app>
  <servlet>
    <servlet-name>AddNumbers</servlet-name>
    <servlet-class>fromwsdl.server.AddNumbersImpl</servlet-class>
```

```
        <async-supported>true</async-supported>
    </servlet>
    <servlet-mapping>
        <servlet-name>AddNumbers</servlet-name>
        <url-pattern>/addnumbers</url-pattern>
    </servlet-mapping>
</web-app>
```



# 第10章 MIME添付メッセージの送信

本章では、MIME添付を使用したメッセージの送信方式について説明します。

## 10.1. 概要

Webサービスの実装において、クライアントまたはサーバーからWebサービスのパラメータまたは戻り値はSOAPメッセージのボディー部分に格納領域形式で含まれており、このメッセージがネットワークを通じて送信されます。このとき、パラメータまたは戻り値が写真や音楽ファイルのようにサイズが大きいバイナリ・データの場合は、そのデータ・サイズが大きければ大きいほど効率は落ちます。

JAX-WS WebサービスはMTOM(Message Transmission and Optimization Mechanism)とXOP(XML Binary Optimized Packaging)を通じて、`xs:base64Binary`または`xs:hexBinary`のような要素で定義されたバイナリ・データが最適化されて送信されます。また、`swaRef`というスキーマをサポートし、このスキーマで定義されたXML要素のコンテンツはすべてMIME添付形式で送信する機能を提供します。

JEUS 8 JAX-WS Webサービスは、`MessageContext`プロパティを使用して添付ファイルをストリーミング方式で送受信できます。この機能は、大容量添付ファイルを受信する際に有効です。

## 10.2. MTOM/XOP

MTOMは、XOPと共に、どのようにXML binary data such as `xs:base64Binary`または`xs:hexBinary`のようなXMLバイナリ・データがネットワーク上で最適化されて送信されるのかについて定義しています。

`xs:base64Binary`のようなXML型は、通常SOAPエンベロープに含まれて送信されますが、このようなタイプのデータ、写真や音楽のようにデータ・サイズが大きければ大きいほど効率は落ちます。このような問題を解決するために、MTOMはバイナリ・データのサイズが規定の値より大きい場合、MIME添付形式でXOPパッケージングしてメッセージを送信します。このように、`xs:base64Binary`または`xs:hexBinary`型の要素値がMIME添付形式でXOPパッケージングされる場合、そのMIME添付のContent-Typeは`xmime:expectedContentType`属性に指定することができ、これはJAXBでサポートするタイプ・マッピングの適用を受けます。

`xs:base64Binary`あるいは`xs:hexBinary`型のスキーマ要素は、`xmime:expectedContentType`属性と一緒に使用した場合、JAXBでサポートするタイプ・マッピングの適用を受けます。この要素値のサイズがある上限を超えると、以下のようなMIME形式でMIME添付に含まれます。`xmime:expectedType`属性を使用しないと、普通の`byte[]`でマッピングされ、この要素値のサイズがある上限を超えると、Content-Typeに一般的な`application/octet-stream`のような値でMIME添付に含まれます。

このような`xmime:expectedContentType`のJava型としてのJAXB型マッピングは以下のとおりです。

[表 10.1] JAXB 2.0 specification of mime:expectedContentType to Java type mapping

MIME Type	Java Type
image/gif	java.awt.Image
image/jpeg	java.awt.Image
text/plain	java.lang.String
text/xml or application/xml	javax.xml.transform.Source
*/*	javax.activation.DataHandler

したがって、<element name="image" type="base64Binary"/>のような要素はbyte[]型でマッピングされますが、<element name="image" type="base64Binary" mime:expectedContentType="image/jpeg" xmlns:mime="http://www.w3.org/2005/05/xmlmime"/>のような要素はjava.awt.Image型でマッピングされます。

## 10.2.1. 基本動作

MTOM/XOP環境を使用できるよう、WSDL文書のwsdl:typeにxs:base64Binaryまたはxs:hexBinaryのようなタイプで定義された要素がmime:expectedContentType属性で定義されます。定義された属性はwsimportツールを使用してJAXBの特定タイプ・マッピングでサービス・インタフェースおよびポータブル・アーティファクトが作成された場合、クライアントおよびサーバーではMTOM/XOP環境を動作させることができます。

## サーバー側でのMTOMの動作

サーバー側でMTOMを動作させるためには、サービス・エンドポイント実装クラスに@javax.xml.ws.soap.MTOMアノテーションを追加します。

```
@javax.xml.ws.soap.MTOM
@WebService(endpointInterface = "com.tmax.mtom.Server")
public class ServerImpl implements Server {
    ...
}
```

以下のように、サービス・エンドポイントの実装クラスに@BindingTypeアノテーションを追加させ、MTOMを動作させる方法もあります。

```
@BindingType(value=javax.xml.ws.SOAPBinding.SOAP11HTTP_MTOM_BINDING)
@BindingType(value=javax.xml.ws.SOAPBinding.SOAP12HTTP_MTOM_BINDING)
```

## クライアント側でのMTOMの動作

クライアント側でMTOMを動作させるには、サービスからProxyあるいはDispatchオブジェクトをjavax.xml.ws.soap.MTOMFeatureというパラメータを通じて取得します。

```
Server port = new ServerService().getServerPort(new MTOMFeature());
javax.xml.ws.Service.createDispatch(..., new javax.xml.ws.soap.MTOMFeature())
```

以下は、ProxyまたはDispatchがMTOMに設定されているかを確認するメソッドです。

```
Server port = new ServerService().getServerPort();
SOAPBinding binding = (SOAPBinding)((BindingProvider)port).getBinding();
boolean mtomEnabled = binding.isMTOMEnabled();
binding.setMTOMEnabled(true);
```

## 10.2.2. 添付バイナリ・データ・サイズの設定

JAX-WS Webサービスは、xs:base64Binaryとxs:hexBinary型の要素のJavaオブジェクトについて、そのサイズが1Kbyte以上の場合、クライアントまたはサーバーからの送信時にMIME添付形式でXOPエンコーディングされてメッセージにパッケージングされます。そうでない場合、SOAPメッセージ内にそのまま埋め込みます。

XOPエンコーディングされてメッセージにパッケージングされる場合、元の要素には<xop:Include href=...>と同じ値が設定されますが、この要素のhref属性に設定される値は添付のContent-Id値です。また、スキーマのxs:base64Binary、xs:hexBinaryで定義されたタイプのxmime:expectedContentTypes属性値が添付のContent-Type値として設定されます。

添付形式で扱われるバイナリ・データのサイズの設定方法は以下のとおりです。

- サーバーでは@MTOMアノテーションで設定します。

```
@javax.xml.ws.soap.MTOM(threshold=3000)
@WebService (endpointInterface = "com.tmax.mtom.Server")
public class ServerImpl implements Server {
    ...
}
```

- クライアントではMTOMFeatureクラスを通じて設定します。

```
Server port = new ServerService().getServerPort(new MTOMFeature(3000));
javax.xml.ws.Service.createDispatch(..., new javax.xml.ws.soap.MTOMFeature())
```

以下は、WSDLのwsdl:typeのスキーマの例です。

```
<element name="Detail" type="types:DetailType"/>
<complexType name="DetailType">
  <sequence>
    <element name="Photo" type="base64Binary"/>
    <element name="image" type="base64Binary"
      xmime:expectedContentTypes="image/jpeg"/>
  </sequence>
</complexType>
```

```
</sequence>
</complexType>
```

WSDLのwsdl:typeのスキーマが上記の場合、ネットワークに送信されるメッセージは以下のとおりです。

```
Content-Type: Multipart/Related;
start-info="text/xml";
type="application/xop+xml";
boundary="-----_Part_0_1744155.1118953559416"
Content-Length: 3453
SOAPAction: ""

-----=_Part_1_4558657.1118953559446

Content-Type: application/xop+xml;
type="text/xml"; charset=utf-8
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
        <Detail xmlns="http://example.org/mtom/data">
            <Photo>RHVrZQ==</Photo>
            <image>
                <xop:Include
                    xmlns:xop="http://www.w3.org/2004/08/xop/include"
                    href="cid:5aeaa450-17f0-4484-b845-a8480c363444@example.org">
                </xop:Include>
            </image>
        </Detail>
    </soapenv:Body>
</soapenv:Envelope>

-----=_Part_1_4558657.1118953559446

Content-Type: image/jpeg
Content-ID: <5aeaa450-17f0-4484-b845-a8480c363444@example.org>
α JFIF          ♀♂♂♀↓↑ ❄️ →▼▲ →└┐ $. ' " ,#└┐(7),
01444▼'9=82<.342   C  ♀♂♀↑ ↑ 2!└ :22222222222222222222222222222222
22222222222 222222 └ ) ~ "      — ▼                      ♂
—                               }           ↓!1A Qa"q❄️?#B⬆️$R ≡$3bré
↑ ↓→%&'()*456789:CDEFGHIJSTUVWXYZcdefghijstuvwxzyzääåäçêëèÆôöðùÿÖÜ
óúñÑªº¿ ¸ | ÷     ¶|||——+ ™  ↵  Γ π Σ σ μ τ Φ Θ Ω ±
÷ ° · —
```

### 10.2.3. MTOM/XOPの例

以下は、MTOM/XOPが適用されたWSDLファイルの例です。

### [例 10.1] << hello.wsdl >>

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:types="http://tmaxsoft.com/mtom/data"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://tmaxsoft.com/mtom"
  targetNamespace="http://tmaxsoft.com/mtom" name="mtom">
  <wsdl:types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://tmaxsoft.com/mtom/data"
      xmlns:xmime="http://www.w3.org/2005/05/xmlmime"
      elementFormDefault="qualified">
      <complexType name="DetailType">
        <sequence>
          <element name="image" type="base64Binary"
            xmime:expectedContentTypes="image/jpeg" />
        </sequence>
      </complexType>
      <element name="Detail" type="types:DetailType" />
      <element name="DetailResponse" type="types:DetailType" />
    </schema>
  </wsdl:types>
  <wsdl:message name="HelloIn">
    <wsdl:part name="data" element="types:Detail" />
  </wsdl:message>
  <wsdl:message name="HelloOut">
    <wsdl:part name="data" element="types:DetailResponse" />
  </wsdl:message>
  <wsdl:portType name="Hello">
    <wsdl:operation name="Detail">
      <wsdl:input message="tns:HelloIn" />
      <wsdl:output message="tns:HelloOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="HelloBinding" type="tns:Hello">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="Detail">
      <soap:operation />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="HelloService">
```

```

        <wsdl:port name="HelloPort" binding="tns:HelloBinding">
            <soap:address location="REPLACE_WITH_ACTUAL_URL" />
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>

```

上記のように、DetailType要素のスキーマはbase64Binary型の要素のイメージが定義されており、これはMTOMで管理できます。またイメージ・タイプは、xmime:expectedContentTypes="image/jpeg"で属性が宣言されており、MTOMで添付として扱う場合、そのContent-Typeはimage/jpegになります。

## 10.2.4. MTOM/XOPの例の実行

本節では、上で実装したクラスと設定ファイルを使用してハンドラー・フレームワークを実行する方法について説明します。

以下のように、MTOMを設定したサービスを作成してJEUSにデプロイします。

```
$ ant deploy
```

上のプロセスがすべて実行され、サービスが正常にデプロイされたら、クライアントをビルドします。クライアントでwsimportのプロセスを経るため、サービスのデプロイがすべて終了した際にクライアントの構成が可能です。

以下のようにMTOMを設定したクライアントを生成し、サービスを呼び出します。コンソールで以下のように入力すると、クライアントとサービスが正常にメッセージを送受信していることを確認できます。

```

$ ant run

...

Host: localhost:8088
Content-length: 4848
Content-type: multipart/related;type="application/xop+xml";boundary="uuid:23ba193c-bd1a-4323-abdd-339bf5c05cd1";start-info="text/xml"
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Soapaction:
User-agent: JAX-WS RI 2.2 - JEUS 8
--uuid:23ba193c-bd1a-4323-abdd-339bf5c05cd1
Content-Type: application/xop+xml;charset=utf-8;type="text/xml"
Content-Transfer-Encoding: binary

<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><Detail xmlns="http://tmaxsoft.com/mtom/data"><image><Include xmlns="http://www.w3.org/2004/08/xop/include" href="cid:dc0fa852-3d46-4bac-9ad4-889d59c6198a@example.jaxws.sun.com"/></image></Detail></S:Body></S:Envelope>
--uuid:23ba193c-bd1a-4323-abdd-339bf5c05cd1

```

```
Content-Type: image/jpeg
Content-Id: <dc0fa852-3d46-4bac-9ad4-889d59c6198a@example.jaxws.sun.com>
Content-Transfer-Encoding: binary

??JFIF

...
```

## 10.3. swaRef

JAX-WS Webサービスは、WSDLのwsdl:type内に要素をwsi:swaRefというスキーマ・タイプで定義し、Webサービスを構成するときに送信されるメッセージは要素値をMIME添付形式で含めます。

実際にそのSOAPボディー・メッセージの要素内には、その添付へのリファレンス形式をとります。このようなwsi:swaRefスキーマの要素は、javax.activation.DataHandler形式のJavaクラスにマッピングされます。

### 10.3.1. swaRefの使用方法

wsi:swaRef型のXML要素はDataHandler Javaクラスにマッピングされ、実際のネットワークには添付形式で送信されます。

たとえば、以下のようなXMLスキーマがWSDLに定義されていると仮定します。

```
<element name="claimForm" type="wsi:swaRef"
  xmlns:wsi="http://ws-i.org/profiles/basic/1.1/xsd"/>
```

XMLスキーマが定義されているWSDL文書を通じてwsimportツールでWebサービスを構成し、ネットワーク上にメッセージを送信する際のメッセージは以下のとおりです。

```
Content-Type: Multipart/Related;
start-info="text/xml";
type="application/xop+xml";
boundary="-----_Part_4_32542424.1118953563492"
Content-Length: 1193
SOAPAction: ""

-----_Part_5_32550604.1118953563502

Content-Type: application/xop+xml;
type="text/xml"; charset=utf-8
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Body>
<claimForm xmlns="http://example.org/mtom/data">
cid:b0a597fd-5ef7-4f0c-9d85-6666239f1d25@example.jaxws.sun.com
</claimForm>
</soapenv:Body>
```

```

</soapenv:Envelope>

-----=_Part_5_32550604.1118953563502

Content-Type: application/xml
Content-ID:
    <b0a597fd-5ef7-4f0c-9d85-6666239f1d25@example.jaxws.sun.com>
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation= "http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/application_1_4.xsd"
    version="1.4">
    <display-name>Simple example of application</display-name>
    <description>Simple example</description>
    <module>
        <ejb>ejb1.jar</ejb>
    </module>
    <module>
        <ejb>ejb2.jar</ejb>
    </module>
    <module>
        <web>
            <web-uri>web.war</web-uri>
            <context-root>web</context-root>
        </web>
    </module>
</application>

```

## 10.3.2. swaRefの例

swaRefが適用されたWSDLファイルの例は以下のとおりです。

### [例 10.2] << hello.wsdl >>

```

<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:types="http://tmaxsoft.com/swaref/data"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://tmaxsoft.com/swaref"
    targetNamespace="http://tmaxsoft.com/swaref" name="swaref">
    <wsdl:types>
        <schema xmlns="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://tmaxsoft.com/swaref/data"
            xmlns:xmime="http://www.w3.org/2005/05/xmlmime"
            elementFormDefault="qualified"
            xmlns:ref="http://ws-i.org/profiles/basic/1.1/xsd">

```



```

        <import namespace="http://ws-i.org/profiles/basic/1.1/xsd"
            schemaLocation="wsi-swa.xsd" />
        <element name="claimForm" type="ref:swaRef" />
        <element name="claimFormResponse" type="ref:swaRef" />
    </schema>
</wsdl:types>
<wsdl:message name="claimFormIn">
    <wsdl:part name="data" element="types:claimForm" />
</wsdl:message>
<wsdl:message name="claimFormOut">
    <wsdl:part name="data" element="types:claimFormResponse" />
</wsdl:message>
<wsdl:portType name="Hello">
    <wsdl:operation name="claimForm">
        <wsdl:input message="tns:claimFormIn" />
        <wsdl:output message="tns:claimFormOut" />
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="HelloBinding" type="tns:Hello">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="claimForm">
        <soap:operation />
        <wsdl:input>
            <soap:body use="literal" />
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal" />
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:service name="HelloService">
    <wsdl:port name="HelloPort" binding="tns:HelloBinding">
        <soap:address location="REPLACE_WITH_ACTUAL_URL" />
    </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

上記のように、claimFormとclaimFormResponse要素はref:swaRefで、外部スキーマであるwsi-swa.xsdを使用してswaRefのデータ型で定義されているため、その後ネットワーク上にメッセージが送信される場合は添付形式で送信されます。

### 10.3.3. swaRefの例の実行

本節では、上で実装したクラスとその他の設定ファイルを使用してハンドラー・フレームワークを実行する方法について説明します。

以下のように、swaRefを設定したサービスを作成してJEUSにデプロイします。

```
$ ant build deploy
```

上のプロセスがすべて実行され、サービスが正常にデプロイされたら、クライアントをビルドします。クライアントでwsimportのプロセスを経るため、サービスのデプロイがすべて終了した際にクライアントの構成が可能です。

以下のようにswaRefを設定したクライアントを作成し、サービスを呼び出します。コンソールで以下のように入力すると、クライアントとサービスが正常にメッセージを送受信していることを確認できます。

```
$ ant run
...

Host: localhost:8088
Content-length: 2672
Content-type: multipart/related; type="text/xml"; boundary="uuid:26973efe-2e29-436a-8fce-3fa58b6fd91f"
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Soapaction:
User-agent: JAX-WS RI 2.2 - JEUS 8
--uuid:26973efe-2e29-436a-8fce-3fa58b6fd91f
Content-Type: text/xml

<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><claimForm xmlns="http://tmaxsoft.com/swaref/data">cid:40b15647-3c0b-4449-90ad-29b1667e940b@example.jaxws.sun.com</claimForm></S:Body></S:Envelope>
--uuid:26973efe-2e29-436a-8fce-3fa58b6fd91f
Content-Id:<40b15647-3c0b-4449-90ad-29b1667e940b@example.jaxws.sun.com>
Content-Type: text/xml
Content-Transfer-Encoding: binary

<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:types="http://tmaxsoft.com/swaref/data"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://tmaxsoft.com/swaref"
  targetNamespace="http://tmaxsoft.com/swaref" name="swaref">
...
```

## 10.4. ストリーミング方式による添付ファイルの処理方法

JEUS 8 JAX-WS Webサービスは、添付ファイルを含むSOAPメッセージを送受信する際に、添付ファイルをメモリにロードすることなく、ストリーミング方式で送受信できます。この機能は、添付ファイルが大容量の場合にWebサービス・エンドポイントの呼び出し性能を向上させ、特に添付ファイルがJVM Heapサイズより大きい場合にOutOfMemoryエラーなく添付ファイルの送受信を可能にします。

本節では、Webサービス・クライアントがOutbound SOAPメッセージに含まれた添付ファイルをストリーミング方式で送信する方法について説明します。

Webサービス・クライアントがChunked transfer-encodingを使用してOutbound要求メッセージを送信するように設定します。サービス・ポートにWebサービス・オペレーションを呼び出す前に(つまり、Outbound要求メッセージを送信する前に)、サービス・ポートから取得した要求MessageContextに次のプロパティを設定します。

- Property Key: "com.sun.xml.ws.transport.http.client.streaming.chunk.size"
- Property Value: chunked data size

このプロパティは、JAX-WS HTTPトランスポートがHTTPリクエストをChunked Streaming方式で送信します。JEUS 8サーバーはChunked Streaming方式をサポートします。

**[例 10.3] << AttachmentApp.java >>**

```
Hello port = new HelloService().getHelloPort(  
    new jeus.webservices.jaxws.api.transport.http.AttachmentFeature());  
  
Map<String, Object> reqCnt = ((BindingProvider)port).getRequestContext();  
reqCnt.put("com.sun.xml.ws.transport.http.client.streaming.chunk.size",  
    new Integer(4*1024));
```

---

**注**

すべてのHTTPサーバーがこの方式をサポートしてはいないことに注意してください。

---



# 第11章 Fast Infosetを使用したWebサービス

本章では、JEUS Webサービスが新しくサポートするFast Infoset Webサービスと、その実装および使用方法について説明します。

## 11.1. 概要

Fast Infoset標準は、XMLの代わりに使用できる効果的なXML Infosetのバイナリ形式を明示しています。

Fast Infoset(ITU-T Rec. X.891 | ISO/IEC 24824-1 Fast Infoset)の仕様は以下によって標準化されています。

- ITU-T(国際電気通信連合の電気通信標準化部門)

<http://www.itu.int/ITU-T/studygroups/com17/index.asp>

- ISO(International Standards Organization)

[http://www.iso.org/iso/home/standards\\_development/list\\_of\\_iso\\_technical\\_committees.htm](http://www.iso.org/iso/home/standards_development/list_of_iso_technical_committees.htm)

XML文と同様に、このようなXML Infosetのバイナリ・フォーマットのインスタンスを**Fast Infoset文**といいます。XML文とFast Infoset文は実質的な形式を持ち、内部的にXML Infosetを持っています。このようなFast Infoset文は直列化あるいはパーシングを迅速に行うことができ、XML文よりもサイズが小さいです。したがって、Fast Infoset文は、XML文の処理速度やサイズが問題となる場合に使用されます。

## Fast Infosetの設計

Fast Infoset文は、XML文と同様に、直列化あるいはパーシングされます。

- 直列化

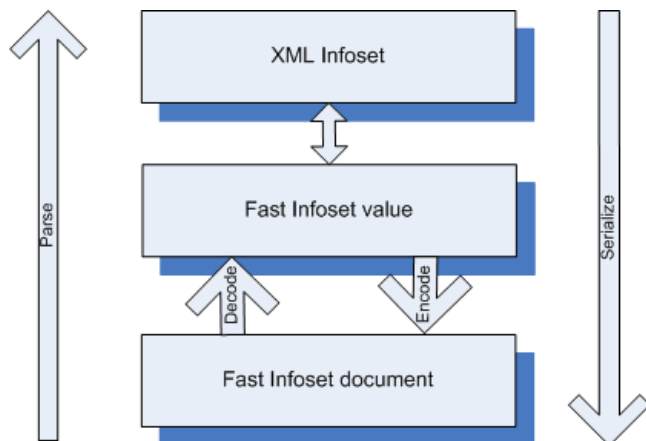
XML文から取得したDOM文あるいはSAXイベントに該当するXML InfosetからFast Infosetを取得し、再度このFast InfosetからFast Infoset文を取得します。

- パーシング

Fast Infoset文からFast Infosetを取得し、再度このFast InfosetからXML Infosetを取得します。Fast Infosetの仕様は、このような文書の効果的な直列化あるいはパーシング、そして文書サイズの縮小の最適化を保証します。

以下の図は、このような直列化またはパーシングのプロセスを示しています。

**[図 11.1] Fast Infosetの直列化とパーシング**



## Fast Infosetのメリット

Fast Infoset文はXML文に比べてパーシングと直列化が速く、文書のサイズが小さくなるメリットがあります。

その他にも、以下のようなメリットがあります。

- XML文で使用されるような終了タグ(end-tag)が存在しません。
- escaping文字データが存在しません。
- Fast Infoset文をFast Infosetに変換するデコーダは長さを事前に把握しています。
- 繰り返される文字列をインデクシングします。
- 名前空間(namespace)情報をインデクシングします。
- バイナリ・コンテンツを含むことができます。
- 類似する単語(vocabulary)を持つ文書の状態を保存します。

Fast Infoset文のバイナリ形式のメリットは以下のとおりです。

- 要素(Element)アイテムまたは文字情報が、より小さいビットでエンコーディングされます。
- Fast Infosetのエンコーダーとデコーダーは、明確に定義されたバウンダリによって、実装が容易で効率的です。
- インデクシング情報を通じて直列化あるいはパーシングするにあたり、対象のサイズが小さくなるため、速いです。

- ストリーミング機能の活用に適切です。

## 11.2. Fast Infosetの使用

本節では、Fast Infoset機能を使用するための方法であるコンテンツ・ネゴシエーション(Content Negotiation)について説明します。

### 11.2.1. コンテンツ・ネゴシエーション

JAX-WS WebサービスでFast Infoset機能を使用するには、標準HTTPヘッダーであるAcceptあるいはContent-TypeにFast Infoset機能の使用可否を登録します。このようなFast Infosetの使用可否が常にWebサービスを呼び出すクライアントによって決定されるという点、そしてHTTPヘッダーを通じてこのような決定がされるという点で、**コンテンツ・ネゴシエーション**と呼ばれます。

クライアントがHTTPヘッダーに情報を挿入するのによって決定されるWebサービスのFast Infosetで、クライアントが一番目のメッセージを送信時に、XMLでエンコーディングされたSOAPメッセージを送信します。このとき、クライアントがHTTP AcceptヘッダーにMIME-Type application/fastinfoset情報を加えると、その次からのメッセージ送信では(サーバーがFast Infosetをサポートする場合)、すべてFast InfosetでエンコーディングされたSOAPメッセージを送信します。つまり、クライアントがXMLでエンコーディングされたSOAPメッセージをHTTP AcceptヘッダーにMIME type application/fastinfosetで送信すると、Fast Infosetが可能なサーバーはFast Infosetでエンコーディングされたメッセージを返します。この後にクライアントが同じスタブ・オブジェクトを通じてメッセージを送信した場合、クライアントとサーバーはFast Infosetでエンコーディングされたメッセージを送受信することになります。これを**Negotiation Pessimistic方式**といいます。

JAX-WS WebサービスでFast Infoset機能を使用するためには以下の方法で設定します。

- クライアントの属性を設定します。

```
((BindingProvider) stubOrDispatch).getRequestContext().put(
    com.sun.xml.ws.client.ContentNegotiation.PROPERTY, "pessimistic");
```

- クライアントが動作するVMのシステム変数を設定します。

```
java -Dcom.sun.xml.ws.client.ContentNegotiation=pessimistic
```

## 11.3. Fast Infosetの例

JAX-WS WebサービスでFast Infoset機能を使用するには、クライアント・アプリケーションでクライアント属性を設定する方法とクライアントが動作するVMのシステム変数を設定する方法があります。

本節では、クライアント・アプリケーションを通じてクライアントの属性を設定する方法について説明します。

---

## 参考

サーバーでは他のオプションを使用した機能は提供していません。Fast Infoset属性はすべてクライアントによって決定されます。

---

以下は、Fast Infoset属性のためのクライアント・アプリケーションの例です。

### [例 11.1] << AddNumbersClient.java >>

```
public class AddNumbersClient {
    public static void main(String[] args) {
        AddNumbersImpl port = new AddNumbersImplService().getAddNumbersImplPort();
        ((BindingProvider) port).getRequestContext().put(
            ContentNegotiation.PROPERTY, "pessimistic");

        int number1 = 10;
        int number2 = 20;

        System.out.println("#####");
        System.out.println("### JAX-WS Webservices examples - fastinfoset ###");
        System.out.println("#####");
        System.out.println("Testing Fast Infoset webservices...");
        int result = port.addNumbers(number1, number2);
        if (result == 30) {
            System.out.println("Success!");
        }
    }
}
```

上記のようにFast Infoset機能を使用するには、サービス・インターフェースから取得したプロキシ(Stub)オブジェクトを使用し、pessimistic属性をリクエスト・コンテキストのContentNegotiationで設定します。

## 11.4. Fast Infoset Webサービスの実行

本節では、上で実装したクラスおよびその他の設定ファイルを使用してFast Infoset Webサービスを実行する方法について説明します。その他のサービス・エンドポイント・インターフェースの実装クラスおよびその他の設定ファイルは前章で説明した例の内容と同一です。

以下のように、Fast Infoset Webサービスを設定したサービスを作成し、JEUSにデプロイします。

```
$ ant build deploy
```

上のプロセスがすべて実行され、サービスが正常にデプロイされたら、クライアントをビルドし、呼び出します。クライアントでwsimportのプロセスを経るため、サービスのデプロイがすべて終了した際にクライアントの構成が可能です。

以下のように、Fast Infoset Webサービスを設定したクライアントを作成し、サービスに呼び出します。



```

$ ant run

...

run:
    [java] #####
    [java] ### JAX-WS Webservicess examples - fastinfofet ###
    [java] #####
    [java] Testing Fast Infofet webservicess...
    [java] Success!

...

BUILD SUCCESSFUL

```

```

$ ant run

---[HTTP request]---
Host: localhost:8088
Content-length: 218
Content-type: text/xml; charset=utf-8
Accept: application/fastinfofet, text/xml, multipart/related, text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Soapaction: ""
User-agent: JAX-WS RI 2.2 - JEUS 8
<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><ns2:addNumbers xmlns:ns2="http://server.fastinfofet/"><arg0>10</arg0><arg1>20</arg1></ns2:addNumbers></S:Body></S:Envelope>
-----
---[HTTP response 200]---
?

```



# 第12章 JAX-WS JMS転送

本章では、WebサービスでのJAX-WS JMS転送の使用方法について説明します。

## 12.1. 概要

Webサービス・クライアント・アプリケーションは、JEUS Webサービスを呼び出すとき、送信プロトコルとしてHTTPを使用します。しかし、JEUS JAX-WSでは、JMS転送を使用してWebサービスを呼び出すことができます。

JAX-WS JMS転送は、JEUS MQサーバーの接続・ファクトリーとデスティネーションの設定を使用します。JAX-WS JMS転送を使用するWebサービスは@jeus.webservices.jaxws.api.JMSWebServiceアノテーションをWebサービス・エンドポイントに追加します。

JAX-WS JMS転送を使用するWebサービスがデプロイされると、公開されたWSDLには2つのwsdl:portが定義されます(HTTPベースのポートとJMSベースのポート)。Webサービス・クライアント・アプリケーションは、使用したいタイプのポートを選択してWebサービスを呼び出すことができます。

## 12.2. JAX-WS JMS転送の設定

本節では、開発者がJAX-WS WebサービスとJMSの使用に慣れているという前提でJAX-WS JMS転送を設定する方法について説明します。

### 12.2.1. JMSサーバーの設定

JAX-WS JMS転送は、JMSサーバーが提供する接続・ファクトリーとデスティネーションを使用して動作します。デスティネーションにQueueタイプを使用します。JEUSドメイン・サーバーのdomain.xmlファイルに<connection-factory>、<destination>を設定します。設定方法の詳細については『JEUS MQガイド』を参照してください。

以下は、JAX-WS JMS転送の設定についての例です。

#### [例 12.1] << domain.xml >>

```
<jms-engine xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  ...
  <connection-factory>
    <type>queue</type>
    <name>QueueConnectionFactory</name>
    <service>jmstest</service>
  </connection-factory>
</jms-engine>
```

```

        </connection-factory>
        ...
    </jms-engine>
    <jms-resource>
        ...
        <destination>
            <type>queue</type>
            <name>ExamplesQueue</name>
            <multiple-receiver>true</multiple-receiver>
        </destination>
        ...
    </jms-resource>

```

## 12.2.2. Webサービスの作成

HTTPの代わりにJMS転送を使用するには、@jeus.webservices.jaxws.api.JMSWebServiceアノテーションをWebサービス・エンドポイントに追加します。

@JMSWebServiceアノテーションが設定されたWebサービス・エンドポイントは以下のとおりです。

### [例 12.2] << AddNumbersImpl.java >>

```

@WebService
@jeus.webservices.jaxws.api.JMSWebService(
    connectionFactory = "QueueConnectionFactory",
    destination = "ExamplesQueue",
    portName = "AddNumbersJMSPort")
public class AddNumbersImpl {
    ...
}

```

JMSWebServiceの属性は以下のとおりです。

属性	説明
connectionFactory	J M S サーバーの接続・ファクトリー名です。domain.xml#<connection-factory>/<name>値と一致する必要があります
destination	JMSサーバーのデスティネーション名です。domain.xml#<destination>/<name>値と一致する必要があります
portName	fromJavaモデルではpublished WSDLに公開されるポート名です。  @JMSWebService#portName属性が明示されていない場合、JAX-WS JMS転送のためのportNameは"JMSTransport" + default_PortNameに設定されます。たとえば、portNameを設定していない場合は"JMSTransportAddNumbersImplPort"となります

### 12.2.3. WSDLの設定

Webサービス・エンドポイントをWSDLで作成する場合、デプロイするアプリケーション(WARまたはJAR)に含まれているWSDLには、HTTPを使用するwsdl:portだけでなく、JMS転送のためのwsdl:portが設定されている必要があります。

以下は、WSDLにJAX-WS JMS転送のためのwsdl:portを設定する例です。

#### [例 12.3] << AddNumbers.wsdl >>

```
<definitions name="AddNumbers" targetNamespace="urn:AddNumbers" ... >
  ...
  <service name="AddNumbersService">
    <port name="AddNumbersPort" binding="impl:AddNumbersBinding">
      <soap:address location="REPLACE_WITH_ACTUAL_URL" />
    </port>
    <port name="AddNumbersJMSPort" binding="impl:AddNumbersBinding">
      <soap:address location="REPLACE_WITH_ACTUAL_URL" />
    </port>
  </service>
</definitions>
```

wsdl:portに設定されているJMS転送のための<port name>は、Webサービス・エンドポイントの@JMSWebService#portName属性値とも一致する必要があります。

### 12.2.4. Webサービス・クライアントの作成

JAX-WS JMS転送を使用するWebサービス・クライアントは、既存のHTTPベースのポートの代わりにJMSベースのポートを使用してWebサービスを呼び出すことができます。

Webサービスの公開された(Published)WSDLは、JMS転送のためのwsdl:portを持っています。このWSDLによりJEUS Webサービスのwsimportツールとして作成されたサービス(extends javax.xml.ws.Service)クラスは、既存のHTTPベースのポートを取得するためのメソッドだけでなく、JMSベースのポートを取得するためのメソッドも持っています。

Webサービス・クライアント・アプリケーションは、以下のように、使用するタイプのポートを選択してWebサービスを呼び出すことができます。

#### [例 12.4] << AddNumbersClient.java >>

```
//AddNumbersPortType port = new AddNumbersService().getAddNumbersPort();
AddNumbersPortType port = new AddNumbersService().getAddNumbersJMSPort();
...
```



# 第13章 Webサービスのポリシー

本章では、Webサービスのポリシー設定についての基本的な知識と共に、簡単なシナリオについて説明します。Webサービスのポリシー設定の詳細については、後の章でそれぞれ説明します。

## 13.1. 概要

JEUS Webサービスは、Webサービスのポリシー(WS-Policy)をサポートします。Webサービスのポリシーは、Webサービスが持っている多様な機能(JEUS Webサービスでは、WS-Addressing、WS-RM、WS-TX、WS-Securityなど)のポリシーを公開するための標準詳細です。

JEUS WebサービスでのWebサービスのポリシーのシナリオは、サーバー・ポリシーとクライアント・ポリシーに分けられます。サーバーはWSDLを通じてWebサービスのポリシーを公開することができ、クライアントはWebサービスのポリシーに合う機能を自動で構成します。

Webサービスのポリシー設定については、「第14章 Webサービス・アドレッシング」、「第15章 高信頼性メッセージング技術」、「第16章 Webサービス・トランザクション」、「第17章 Webサービスのセキュリティー」の内容を参照してください。

## 13.2. Webサービスのポリシー(WS-Policy)

本節では、一般的なWebサービスのポリシー(WS-Policy)について説明します。

一般的なWebサービスのポリシーの特徴は以下のとおりです。

- Webサービスのポリシーの詳細は、柔軟に表現でき、拡張性のある設定になっています。
- Webサービスのポリシーは、1つ以上の「ポリシー・アサーション(policy assertion)」を使用して表現されます。

---

### 参考

標準 Web サービス の ポ リ シ ー に 関 す る ス キ ー マ の 内 容 は <http://schemas.xmlsoap.org/ws/2004/09/policy/ws-policy.xsd>を参照してください。

---

## Webサービスのポリシーのフレームワーク(Framework)

Webサービスのポリシーのフレームワークについて説明します。

- ポリシー・コンテナ(Policy Container)

Webサービス・ポリシー・フレームワークの主要コンポーネントは「Policy」という要素で表現されているポリシー・コンテナです。この要素はID値の割り当てを受け、それを参照・再利用することができます。この要素は、アサーション(assertion)またはアサーションの組み合わせで構成されます。このようなアサーションは、ポリシー演算子(Operator)で構成されます。

- ポリシー演算子(Operator)

Webサービスのポリシーの詳細は、2つの演算子と1つの属性を定義しています。

- ExactlyOne Operator

この演算子は、下位要素にアサーションまたは演算子が複数存在する場合、そのうち1つだけを選択してポリシーとして取得します。

以下は、ExactlyOne Operatorを使用した例です。

```
<wsp:Policy>
  <wsp:ExactlyOne>
    <wsse:SecurityToken>
      <wsse:Token
        ...
      <wsse:
        ...
    </wsp:Policy>
```

- All Operator

この演算子は、下位要素にアサーションまたは演算子が複数存在する場合、それを組み合わせてポリシーとして取得します。

以下は、Optional Operatorを使用した例です。

```
<wsp:Policy>
  <wsp:All>
    <wsse:SecurityToken>
      <wsse:Token
        ...
    </wsp:Policy>
```

- Optional Operator

この演算子は、下位要素にアサーションまたは演算子がこの属性で宣言されている場合、選択的に取得します。

以下は、Optional Operatorを使用した例です。

```
<wsp:Policy>
  <wsse:Integrity wsp:optional="true">
```



```
...
</wsp:Policy>
```

## 13.3. サーバー・ポリシーの設定

サーバーのWebサービスのポリシー設定のシナリオは、WSDLでWebサービスを構成するシナリオと、JavaクラスでWebサービスを構成するシナリオに分けられます。

### 13.3.1. WSDLによるWebサービスの構成

WSDL文書を利用し、Webサービスのポリシー設定が適用されたWebサービスを構成するシナリオは以下のとおりです。

1. WSDL文書を作成します。
2. WSDL文書にWebサービスのポリシーを設定します。
3. wsimportツールを使用してJava Beanオブジェクトを作成します。
4. サービス実装クラスを作成します。
5. パッケージングされたサービスをJEUSサーバーにデプロイします。

### ディレクトリー構造

WSDL文書でWebサービスのポリシー設定が適用されたJEUS Webサービスをパッケージングします。以下は、サーバーに該当するWebサービスをWSDL文書を利用して構成する場合のディレクトリーです。

```
war_root
|- WEB-INF
    |- classes
        |- ... (SEI, JAX-WS artifacts, Handler, Validator)
    |- wsdl
        |- addnumbers.wsdl
```

### 13.3.2. JavaクラスによるWebサービスの構成

JavaクラスでWebサービスのポリシーが適用されているWebサービスを作成するには、-policy機能を利用して、**wsgen**ツールにwsit-endpoint.xmlファイルを作成する必要があります。

```
$ wsgen fromjava.server.AddNumbersImpl -d web/WEB-INF -policy service-config.xml
```

以下は、service-config.xmlを示しています。設定内容の詳細は、各Webサービス機能について説明した章を参照してください。ここでは、以下で強調した部分を中心に記述します。

**[例 13.1] << service-config.xml >>**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <policy>
    <!-- エンドポイント全体に適用する設定です -->
    <endpoint-policy-subject>
      <addressing-policy>
        <using-addressing>true</using-addressing>
      </addressing-policy>
      <!-- エンドポイントのオペレーション(メソッド)に適用する設定です -->
      <operation-policy-subject>
        <!-- 以下のオペレーション(メソッド)に適用する設定です -->
        <operation-java-name>addNumbers</operation-java-name>
        <!-- クライアント要求メッセージに適用する設定です -->
        <input-message-policy-subject>
          .....
        </input-message-policy-subject>
        <!-- サーバーの応答メッセージに適用する設定です -->
        <output-message-policy-subject>
          .....
        </output-message-policy-subject>
      </operation-policy-subject>
    </endpoint-policy-subject>
  </policy>
</web-services-config>
```

つまり、wsгенの-policy機能を利用してWebサービスのためのJavaクラスおよびwsit-endpoint.xmlファイルを取得し、以下のようなシナリオでWebサービスのポリシーが適用されたWebサービスを作成します。

1. サービス実装クラスを作成します。
2. jeus-webservices-config.xsdスキーマを利用してservice-config.xmlファイルを構成します。
3. 構成したサービス実装クラスを利用して、wsгенツールでWebサービスを作成する場合、「-policy」オプションを利用してwsit-endpoint.xmlファイルを作成します。
4. wsit-endpoint.xmlファイルをパッケージングするWEB-INFフォルダの下に位置させます。
5. パッケージングされたサービスをJEUSサーバーにデプロイします。

## ディレクトリーの構造

JavaクラスでWebサービスのポリシーが適用されたWebサービスをパッケージングします。以下は、WebサービスをJavaクラスで構成する場合のディレクトリーです。

```
war_root
| - WEB-INF
|   | - classes
|   |   | - ... (SEI, JAX-WS artifacts, Handler, Validator)
|   | - wsit-Endpoint.xml
```

## 13.4. クライアントのポリシー設定

クライアント側のWebサービスのポリシー設定は、Webサービス・セキュリティのようなシナリオ以外は必要ありません。JEUS Webサービスはランタイム時にリモートWebサービスのWSDLに設定されているWebサービスのポリシー設定を理解し、自動的にそのポリシーに適合した環境を提供するためです。しかし、Webサービス・セキュリティのような特定のシナリオで追加設定が必要な場合が存在します。

Webサービスのポリシーが設定されているWebサービスに対して追加設定が必要な場合、クライアントを構成するシナリオは以下のとおりです。

1. wsimportツールを使用してクライアントJava Beanオブジェクトを構成します。
2. リモートのWSDL文書をアクセス可能なレポジトリーにwsit-client.xmlという名前で保存します。

---

### 参考

JEUS Webサービスはランタイム時にリモートのWSDLに設定されているWebサービスのポリシー設定を通じてクライアント環境を提供するため、リモートのWSDLに設定されているWebサービスのポリシー設定についての内容は削除しても構いません。

---

3. wsit-client.xmlにクライアントで必要とする追加のWebサービスのポリシーを設定します。
4. JARパッケージングの場合、wsit-client.xmlをパッケージングするclasses/META-INFディレクトリーに位置させます。  
  
WARパッケージングの場合、wsit-client.xmlファイルをパッケージングするWEB-INFディレクトリーの下に位置させます。
5. パッケージングされたサービスをJEUSサーバーにデプロイします。

## ディレクトリー構造

Webサービスのポリシー設定を適用したJEUS Webサービスをパッケージングします。以下は、一般的にコンテナで実行されるWebサービス・クライアントです。

```
war_root
|- WEB-INF
    |- classes
        |- ... (client classes, JAX-WS artifacts, Handler, Validator)
        |- META-INF
            |- wsit-client.xml
|- index.jsp
```

以下は、EJBコンテナで実行されるWebサービス・クライアントまたは独立アプリケーションとして実行されるWebサービス・クライアントです。

```
jar_root
|- classes
    |- ... (client classes, JAX-WS artifacts, Handler, Validator)
    |- META-INF
        |- wsit-client.xml
```

# 第14章 Webサービス・アドレッシング

本章では、トランスポートに独立的なWebサービス・アドレッシングと、その簡単な例を挙げて、Webサービス・アドレッシングの使用方法について説明します。

## 14.1. 概要

Webサービス・アドレッシングとは、トランスポートに独立的なWebサービス・メッセージにアドレスなどの情報を表記する方式です。基本的なWebサービスはStatelessのメッセージの交換であるため、Webサービスのアドレッシングを利用すると、状態の把握が可能なWebサービスのサポートが可能になります。実際のWebサービス・アドレッシングは、以降で説明するWS-RM、WS-Security、WS-secure Conversation、WS-Trustなど、多数の「WS-\*」スペックのベースとなる技術です。

Webサービス・アドレッシングは、Webサービスが実行しているトランスポートがHTTPであれSMTPであれ、関係なく動作します。Webサービス・アドレッシングを設定せずに、アプリケーション・レベルで1つのメッセージに情報を表記する場合のメッセージは以下のとおりです。

```
POST /AddNumbers/addnumbers HTTP 1.1/POST
Host: tmaxsoft.com
SOAPAction: http://tmaxsoft.com/AddNumbers/addnumbers

<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
            xmlns:tmax="http://tmaxsoft.com/">
  <S:Header>
    <tmax:MessageID>
      uuid:e197db59-0982-4c9c-9702-4234d204f7f4
    </tmax:MessageID>
  </S:Header>
  <S:Body>
    ...
  </S:Body>
</S:Envelope>
```

上記のWebサービス・アドレッシングを設定していないメッセージが示しているように、アプリケーションがメッセージに割り当てたIDは、そのアプリケーション・レベルで使用するために設定したものであるため、他のアプリケーションで再使用できません。また、トランスポートをHTTPからSMTPなどに変換すると、トランスポート・ヘッダーにある情報もマッピング・ルールに従って変換する必要があるなど、難しい問題が発生します。そこで、W3CではWebサービス・アドレッシングのためにMAP(Message Addressing Properties)を定義して、SOAPメッセージあるいはWSDL文書へのバインディングを規定しています。

以下は、メッセージ・レベルでMAPの情報が格納されているWebサービス・アドレッシングを設定したメッセージの例です。

```
POST /AddNumbers/addnumbers HTTP 1.1/POST
Host: tmaxsoft.com
SOAPAction: http://tmaxsoft.com/AddNumbers/addnumbers

<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://www.w3.org/2005/08/addressing/">
  <S:Header>
    <wsa:MessageID>
      uuid:e197db59-0982-4c9c-9702-4234d204f7f4
    </wsa:MessageID>
    <wsa:To>
      http://tmaxsoft.com/AddNumbers/addnumbers
    </wsa:To>
    <wsa:Action>
      http://tmaxsoft.com/AddNumbers/addnumbers
    </wsa:Action>
  </S:Header>
  <S:Body>
    ...
  </S:Body>
</S:Envelope>
```

Webサービス・アドレッシングを設定したメッセージのSOAPメッセージは、<wsa:MessageID>、<wsa:To>、<wsa:Action>のようなWebサービス・アドレッシング関連要素で構成されています。これらは、MAP(Message Addressing Properties)のSOAPメッセージへのバインディング形式です。

以下は、各要素についての説明です。

要素	説明
<wsa:MessageID>	メッセージを一意で区分できるように絶対URIで設定した値です
<wsa:To>	メッセージの受信アドレスを表示するために絶対URIで設定した値です
<wsa:Action>	メッセージの意味を表示するために絶対URIで設定した値です

このようなWebサービス・アドレッシング・メッセージは、すべての情報がトランスポートやアプリケーションと独立的にプロセッシングできる固有の形式になっています。たとえば、このようなメッセージがHTTPではないSMTPのようなトランスポートで渡される必要がある場合、<wsa:To>ヘッダー値を「mailto:purchasing@example.com」のように変更します。

本節では、サーバーとクライアントでWebサービス・アドレッシングを設定する方法と例、実行方法について説明します。

## 14.2. サーバーの設定

サーバーのWebサービス・アドレッシングは、JavaクラスあるいはWSDLによって設定します。本節では、各方法について説明します。

### 14.2.1. Javaクラスによる設定

JavaクラスでWebサービスを構成する場合、Webサービス・アドレッシングを設定するには、以下のように `javax.jws.WebService` アノテーションと共に `javax.xml.ws.soap.Addressing` アノテーションを追加します。

**[例 14.1] << AddnumbersImpl.java >>**

```
@Addressing
@WebService
public class AddNumbersImpl {
    ...
}
```

上記のように `javax.xml.ws.soap.Addressing` アノテーションを利用してWebサービス・アドレッシングを設定すると、サービス・エンドポイントは以下のように動作します。

- 生成されるWSDL文書の `<wsdl:binding>` 要素に標準 `<wsaw:UsingAddressing>` 要素が作成されます。
- 受信するメッセージのすべてのWebサービス・アドレッシング・ヘッダーを解析し、文法チェックを行います。
- 文法が正しくない場合はエラー・メッセージを渡します。
- `<wsa:Action>` ヘッダー値が該当オペレーションの期待値と一致しない場合、エラー・メッセージを渡します。
- 渡されるすべてのメッセージには、Webサービス・アドレッシング・ヘッダーに関する情報が含まれています。

さらに、Webサービス・アドレッシングは、サービス・エンドポイント・インターフェースのメソッド(WSDL文書の operation要素)に対し、アクションMAP(Message Addressing Property)を直接設定できます。

**[例 14.2] << AddnumbersImpl.java >>**

```
@Addressing
@WebService
public class AddNumbersImpl {

    @Action(
        input = "http://tmaxsoft.com/input",
        output = "http://tmaxsoft.com/output",
        fault = {
```

```

        @FaultAction(className = AddNumbersException.class,
            value = "http://tmaxsoft.com/fault")
    }
)
public int addNumbers(int number1, int number2)
    throws AddNumbersException {

    ...

}
}

```

上記のように設定されたメソッドは、WSDLに変換される時、以下のようなWSDL文書のoperation要素にバインディングされます。

**[例 14.3] << Addnumbers.wsdl >>**

```

...

<operation name="addNumbers">
    <input wsaw:Action="http://tmaxsoft.com/input"
        message="tns:addNumbers" />
    <output wsaw:Action="http://tmaxsoft.com/output"
        message="tns:addNumbersResponse" />
    <fault wsaw:Action="http://tmaxsoft.com/fault"
        message="tns:AddNumbersException"
        name="AddNumbersException" />
</operation>

...

<binding name="AddNumbersImplPortBinding" type="tns:AddNumbersImpl">
    <wsaw:UsingAddressing />
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
        style="document" />
    <operation name="addNumbers">
        <soap:operation soapAction="" />
        <input>
            <soap:body use="literal" />
        </input>
        <output>
            <soap:body use="literal" />
        </output>
    </operation>
    <operation name="addNumbers2">
        <soap:operation soapAction="" />
        <input>
            <soap:body use="literal" />
        </input>
        <output>

```



```

        <soap:body use="literal" />
    </output>
</operation>
</binding>
...

```

## 14.2.2. WSDLによる設定

WSDLでWebサービスを構成する場合、上記のJavaクラスでWSDLを作成した場合と同様に、標準拡張要素である<wsaw:UsingAddressing>を利用します。このように構成されたWebサービスは、JEUSで提供するwsimportツールを利用して簡単にアドレッシングをサポートするWebサービスを作成できます。

## 14.3. クライアント設定

クライアント側のWebサービス・アドレッシング設定は、クライアントを作成するために参照するWSDL文書の<wsaw:UsingAddressing>が含まれているか否かによって、**wsimport**ツールにより自動的に決まります。

クライアントが別途でWebサービス・アドレッシング機能をアプリケーション・レベルで実行している場合、Webサービス・アドレッシングが設定されているサービスからクライアントを構成する際にWebサービス・アドレッシング機能を動作させない場合があります。

以下は、クライアントでWebサービスのアドレッシング機能を動作させない場合に使用する方法です。

```

new AddNumbersImplService().getAddNumbersImplPort(
    new javax.xml.ws.AddressingFeature(false));

```

上記のようにクライアントからプロキシ(エンドポイント・インターフェース)を取得する場合、javax.xml.ws.AddressingFeature値をfalseに設定すると、クライアントはメッセージの送信時にサービスのWSDLに規定しているWebサービス・アドレッシング機能が動作していない状態でメッセージを送信します。

また、サービスのWSDLにWebサービスのアドレッシング機能を示すwsaw:UsingAddressingが含まれていない場合、**wsimport**ツールで取得したポータブル・アーティファクトは、Webサービスの機能のみを実行します。しかし、クライアントでWebサービス・アドレッシング機能を使用する場合があります。このような場合は、以下のようにjavax.xml.ws.Serviceがサポートするメソッドを使用するか、またはjavax.xml.ws.soap.AddressingFeatureを使用します。

```

<T> Dispatch<T> createDispatch(javax.xml.namespace.QName,
    java.lang.Class<T>, Service.Mode, WebServiceFeature...)
Dispatch<java.lang.Object> createDispatch(javax.xml.namespace.QName,
    javax.xml.bind.JAXBContext, Service.Mode, WebServiceFeature...)
<T> T getPort(java.lang.Class<T>, WebServiceFeature...)
<T> T getPort(javax.xml.namespace.QName, java.lang.Class<T>,
    WebServiceFeature...)

new AddNumbersImplService().getAddNumbersImplPort(
    new javax.xml.ws.AddressingFeature());

```

## 14.4. 例

以下は、サービス・エンドポイントの実装クラスの例です。

**[例 14.4] << AddNumbersImpl.java >>**

```
@Addressing
@WebService
public class AddNumbersImpl {

    @Resource
    WebServiceContext wsc;

    @Action(input = "http://tmaxsoft.com/input",
            output = "http://tmaxsoft.com/output")
    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }

    public int addNumbers2(int number1, int number2){
        return number1 + number2;
    }
}
```

上記のように、アドレッシング・アノテーションとWSDL文書のoperation要素にバインディングされるアクション・アノテーションでWebサービス・アドレッシングが構成されています。

## 14.5. 例の実行

上のように構成したWebサービス・アドレッシングを設定したエンドポイント・クラスやその他のファイルを使用してサービスを構成し、JEUS 8にデプロイする方法は以下のとおりです。

```
$ ant deploy
```

上記プロセスがすべて実行されてサービスが正常にデプロイされると、クライアントを実行します。

以下のように、クライアント・コンソールとサーバー・コンソール画面にて、メッセージが渡されることを確認できます。

```
$ ant run

...

run:
    [java] #####
    [java] ### JAX-WS Webservices examples - addressing ###
    [java] #####
    [java] basic name mapping result: 20
```

```

[java] default name mapping result: 20

BUILD SUCCESSFUL

...
...

---[HTTP request]---
Host: localhost:8088
Content-length: 626
Content-type: text/xml; charset=utf-8
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg,
*; q=.2, */*; q=.2
Connection: keep-alive
Soapaction: "http://tmaxsoft.com/input"
User-agent: JAX-WS RI 2.2 - JEUS 8
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <S:Header>
    <wsa:To>http://localhost:8088/AddNumbers/addnumbers</wsa:To>
    <wsa:Action>http://tmaxsoft.com/input</wsa:Action>
    <wsa:ReplyTo xmlns:wsa="http://www.w3.org/2005/08/addressing">
      <wsa:Address>
        http://www.w3.org/2005/08/addressing/anonymous
      </wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID>uuid:880f3891-d07b-4ad1-bbc1-8dce8flaedef</wsa:MessageID>
  </S:Header>
  <S:Body>
    <ns2:addNumbers xmlns:ns2="http://server.wsaddressing/">
      <arg0>10</arg0><arg1>10</arg1>
    </ns2:addNumbers>
  </S:Body>
</S:Envelope>
---[HTTP response 200]---
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <S:Header>
    <wsa:To>http://www.w3.org/2005/08/addressing/anonymous</wsa:To>
    <wsa:Action>http://tmaxsoft.com/output</wsa:Action>
    <wsa:MessageID>uuid:815cb296-a2f3-45df-80c5-5a2c1ca836ca</wsa:MessageID>
    <wsa:RelatesTo>uuid:880f3891-d07b-4ad1-bbc1-8dce8flaedef</wsa:RelatesTo>
  </S:Header>
  <S:Body>
    <ns2:addNumbersResponse xmlns:ns2="http://server.wsaddressing/">
      <return>20</return>
    </ns2:addNumbersResponse>
  </S:Body>

```

```

</S:Envelope>
-----

...

---[HTTP request]---
Host: localhost:8088
Content-length: 663
Content-type: text/xml; charset=utf-8
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg,
*; q=.2, */*; q=.2
Connection: keep-alive
Soapaction: "http://server.wsaddressing/AddNumbersImpl/addNumbers2Request"
User-agent: JAX-WS RI 2.2 - JEUS 8
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <S:Header>
    <wsa:To>http://localhost:8088/AddNumbers/addnumbers</wsa:To>
    <wsa:Action>
      http://server.wsaddressing/AddNumbersImpl/addNumbers2Request
    </wsa:Action>
    <wsa:ReplyTo xmlns:wsa="http://www.w3.org/2005/08/addressing">
      <wsa:Address>
        http://www.w3.org/2005/08/addressing/anonymous
      </wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID>uuid:bf65c920-9129-495a-b9dc-8cb8efb9c2a6</wsa:MessageID>
  </S:Header>
  <S:Body>
    <ns2:addNumbers2 xmlns:ns2="http://server.wsaddressing/">
      <arg0>10</arg0><arg1>10</arg1>
    </ns2:addNumbers2>
  </S:Body>
</S:Envelope>
---[HTTP response 200]---
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <S:Header>
    <wsa:To>http://www.w3.org/2005/08/addressing/anonymous</wsa:To>
    <wsa:Action>
      http://server.wsaddressing/AddNumbersImpl/addNumbers2Response
    </wsa:Action>
    <wsa:MessageID>uuid:41975002-46e8-4219-89a6-ed6e72899fa8</wsa:MessageID>
    <wsa:RelatesTo>uuid:bf65c920-9129-495a-b9dc-8cb8efb9c2a6</wsa:RelatesTo>
  </S:Header>
  <S:Body>
    <ns2:addNumbers2Response xmlns:ns2="http://server.wsaddressing/">
      <return>20</return>
    </ns2:addNumbers2Response>
  </S:Body>
</S:Envelope>

```

```
        </ns2:addNumbers2Response>
    </S:Body>
</S:Envelope>
-----

...

```



# 第15章 高信頼性メッセージング技術

本章では、高信頼性メッセージング技術(WS-Reliable Messaging)についての概念と設定方法について説明します。

## 15.1. 概要

高信頼性メッセージングは、より信頼性のあるWebサービスのために良質のサービス(Quality of Service、QOS)を提供することが目的です。信頼性は、1つの時点から他の時点までメッセージを送信できる能力により測定されます。高信頼性メッセージングは、Webサービスの両末端からのアプリケーションメッセージの送信を保証します。

高信頼性メッセージング技術とは、メッセージが送信できなかった場合にそれをリカバリーする技術です。メッセージが途中でなくなった場合、メッセージを送信したシステムは、メッセージを受信するシステムから確認(Acknowledge)メッセージが来るまでメッセージを再送信します。また、メッセージを受信するシステムでは、メッセージが順序どおりに渡されなかった場合、メッセージを並び替えて順序どおりにWebサービス・エンドポイントに渡します。

高信頼性メッセージングのスペックは以下のとおりです。

- WS-Reliable Messaging
- WS-Coordination
- WS-AtomicTransactions

以下のような問題が発生した場合は、高信頼性メッセージングの使用を控えてください。

- コミュニケーションの失敗により、利用できないネットワークまたはコネクション・ドロップが発生した場合
- アプリケーション・メッセージが送信途中でなくなった場合
- 適切な順序で送信される必要のあるアプリケーション・メッセージが順序どおりに渡されていない場合

また、高信頼性メッセージングの使用を考慮する際、以下のメリットとデメリットを参考にしてください。

- 高信頼性メッセージングは、ソースから目的地まで1回のみ送信されます。順序どおりに送信されるオプションを選択すると、そのメッセージは順序どおりに送信されます。

- 高信頼性メッセージングでWebサービスのメッセージを送信する場合、それによりWebサービス全体の性能が低下することがあります。特に順序どおりに送信するオプションを利用した場合にその現象が目立ちます。
- 高信頼性メッセージングを使用しないクライアントは、高信頼性メッセージングを使用するWebサービスと相互運用できません。

## 15.2. サーバーの設定

サーバーのWS-Reliable Messagingは、WSDLあるいはJavaクラスによって設定します。

### 15.2.1. WSDLによる設定

WSDLでWS-Reliable Messagingを実装するには、Webサービス・アドレッシングの場合と同様に、WSDL文書にWebサービスのポリシーを設定し、**wsimport**ツールを使用してWebサービスを作成します。

WSDLファイルにWS-Reliable MessagingをWebサービス・ポリシーを設定する方法は以下のとおりです。

- 基本的に、ポリシー・アサーションに以下の設定を行います。

```
<wsrm:RMAssertion xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm/policy" />
```

- 順序どおりの送信(inorder)を保証するためには、ポリシー・アサーションに以下の設定を行います。

```
<rm:Ordered xmlns:rm="http://sun.com/2006/03/rm" />
```

以下は、上記のように構成したWSDLファイルの例です。

#### [例 15.1] << AddNumbers.wsdl >>

```
<?xml version="1.0" ?>
<definitions name="AddNumbers" targetNamespace="http://example.org"
  xmlns:tns="http://example.org" xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm/policy"
  xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
  <wsp:UsingPolicy />
  <wsp:Policy wsu:Id="AddNumbers_policy">
    <wsp:ExactlyOne>
      <wsp:All>
        <wsaw:UsingAddressing />
```



```

        <wsrm:RMAssertion>
            <wsrm:InactivityTimeout Milliseconds="600" />
            <wsrm:AcknowledgementInterval Milliseconds="200" />
        </wsrm:RMAssertion>
    </wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
<types>
    <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified"
        targetNamespace="http://example.org">
        <element name="addNumbersResponse" type="tns:addNumbersResponse" />
        <complexType name="addNumbersResponse">
            <sequence>
                <element name="return" type="xsd:int" />
            </sequence>
        </complexType>
        <element name="addNumbers" type="tns:addNumbers" />
        <complexType name="addNumbers">
            <sequence>
                <element name="arg0" type="xsd:int" />
                <element name="arg1" type="xsd:int" />
            </sequence>
        </complexType>
    </xsd:schema>
</types>
<message name="addNumbers">
    <part name="parameters" element="tns:addNumbers" />
</message>
<message name="addNumbersResponse">
    <part name="result" element="tns:addNumbersResponse" />
</message>
<portType name="AddNumbersPortType">
    <operation name="addNumbers">
        <input message="tns:addNumbers" name="add" />
        <output message="tns:addNumbersResponse" name="addResponse" />
    </operation>
</portType>
<binding name="AddNumbersBinding" type="tns:AddNumbersPortType">
    <wsp:PolicyReference URI="#AddNumbers_policy" />
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
        style="document" />
    <operation name="addNumbers">
        <soap:operation soapAction="" />
        <input>
            <soap:body use="literal" />
        </input>
        <output>
            <soap:body use="literal" />
        </output>
    </operation>
</binding>

```

```

        </operation>
    </binding>
    <service name="AddNumbersService">
        <port name="AddNumbersPort" binding="tns:AddNumbersBinding">
            <soap:address location="REPLACE_WITH_ACTUAL_URL" />
        </port>
    </service>
</definitions>

```

## 15.2.2. Javaクラスによる設定

JavaクラスでWS-Reliable Messagingを設定するには、以下のように**wsgen**ツールの-policy機能を利用してwsit-endpoint.xmlを予め取得する必要があります。

```
$ wsgen fromjava.server.AddNumbersImpl -d web/WEB-INF -policy service-config.xml
```

以下は、service-config.xmlの内容です。

### [例 15.2] << service-config.xml >>

```

<?xml version="1.0" encoding="UTF-8"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <policy>
        <endpoint-policy-subject>
            <addressing-policy/>
            <rm-policy>
                <inactivityTimeout>600000</inactivityTimeout>
                <acknowledgementInterval>1000</acknowledgementInterval>
            </rm-policy>
        </endpoint-policy-subject>
    </policy>
</web-services-config>

```

## 15.3. クライアントの設定

WS-Reliable Messagingのために、クライアント側で追加設定を行う必要はありません。JEUS Webサービスは、クライアントのランタイム時にリモートWebサービスのWSDLのWS-Reliable Messagingのポリシーを解析し、自動的にWS-Reliable Messagingのための環境を提供します。

## 15.4. 例

Javaクラスによる実装は、wsit-endpoint.xmlというデプロイメント記述子ファイルがWARまたはEARパッケージのWEB-INFフォルダに追加されること以外は、JEUS 8 Webサービスと同じです。

以下は、service-config.xmlファイルの例です。

### [例 15.3] << service-config.xml >>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <policy>
    <endpoint-policy-subject>
      <addressing-policy/>
      <rm-policy>
        <inactivityTimeout>600000</inactivityTimeout>
        <acknowledgementInterval>1000</acknowledgementInterval>
      </rm-policy>
    </endpoint-policy-subject>
  </policy>
</web-services-config>
```

## 15.5. 例の実行

上のservice-config.xmlファイルを使用し、wsгенでWebサービスを作成するには、以下のように入力します。

```
$ ant build
```

Webサービスが正常に生成されると、以下のような構造になります。

```
web
|
+-- WEB-INF
    |
    +-- wsit-fromjava.server.AddNumbersImpl.xml
    +-- classes
        |
        +-- fromjava.server.AddNumbersImpl
```

wsit-fromjava.server.AddNumbersImpl.xmlファイルは、WARまたはEARパッケージングのWEB-INFフォルダに追加されています。

サービスをデプロイする方法は、以下のとおりです。

```
$ ant deploy
```

Webサービスが正常にデプロイされると、それを使用するクライアントを以下のように作成して、サービスを呼び出します。

```
$ ant run

...

run:
    [java] #####
```

```
[java] ### JAX-WS Webservices examples - wsit ###  
[java] #####  
[java] Testing wsit webservices...  
[java] Success!  
  
...  
  
BUILD SUCCESSFUL
```

# 第16章 Webサービス・トランザクション

本章では、Webサービス・トランザクションの概念と設定方法について説明します。

## 16.1. 概要

トランザクションとは、信頼性のある分散アプリケーションを実装する際に基本となる概念です。トランザクションは、アプリケーションに参加するすべての参加者が互いに同意した結果を得ることができるようにするメカニズムです。

トランザクションは別名「ACID」という以下の属性を含んでいます。

区分	説明
原子性(Atomicity)	トランザクションは個別の作業単位であり、トランザクションを構成する操作はすべて成功するか、または失敗しなければなりません
一貫性(Consistency)	トランザクションの前後でデータの整合性が保たれ、矛盾の無い状態が継続されます
独立性(Isolated)	トランザクション実行中の処理過程が外部から隠蔽され、他の処理などに影響を与えません
永続性(Durability)	トランザクションが完了したら、その結果は記録され、システム障害などが生じてでも失われることはありません

Webサービス環境は、アプリケーションの動作と結果を制御するために、従来のトランザクション・メカニズムによって提供される同一のコーディネーション動作が必要です。また、柔軟な方法で複数のサービスから得られた処理結果のプロセッシング・コーディネーションをハンドリングする機能も必要です。これには、より自由な形式のトランザクションが必要です。

JEUS 8 Webサービスでは、以下のトランザクションのスペックに対応します。

- WS-Coordination
- WS-AtomicTransaction

上記のトランザクション仕様は、連動している動作についてのWSDL定義を提供します。

## 16.2. サーバーの設定

サーバーのWebサービス・トランザクションは、WSDLあるいはJavaクラスによって設定します。

## 16.2.1. WSDLによる設定

Webサービス・トランザクションをWSDLで実装するには、Webサービス・アドレッシングの場合と同様に、WSDL文書でWebサービス・ポリシーの設定を行い、wsimportツールを使用してWebサービスを作成します。

Webサービス・ポリシーの設定に従って、WSDLファイルにWebサービス・トランザクションを適切に設定するには、PolicyAssertionに以下の設定を行います。

```
<wsat200410:ATAssertion xmlns:ns1="http://schemas.xmlsoap.org/ws/2002/12/policy" />
```

Webサービス・トランザクションのポリシーを設定したWSDLファイルは以下のとおりです。

### [例 16.1] << AddNumbers.wsdl >>

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="AddNumbersService" targetNamespace="http://server.fromwsdl/"
  xmlns:tns="http://server.fromwsdl/" xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
    oasis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsp:Policy xmlns:wsat200410="http://schemas.xmlsoap.org/ws/2004/10/wsat"
    wsu:Id="AddNumbersPortBinding_addNumbers_WSAT_Policy" wsp:Name="">
    <wsat200410:ATAssertion
      xmlns:ns1="http://schemas.xmlsoap.org/ws/2002/12/policy"
      wsp:Optional="true" ns1:Optional="true" />
    </wsp:Policy>
  <types>...</types>
  <message>...</message>
  <portType>...</portType>

  <binding name="AddNumbersPortBinding" type="tns:AddNumbers">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"
  />

  <operation name="addNumbers">
    <wsp:PolicyReference URI="#AddNumbersPortBinding_addNumbers_WSAT_Policy" />
    <soap:operation soapAction="addNumbers" />
    <input>
      <wsp:PolicyReference URI="#AddNumbersPortBinding_addNumbers_WSAT_Policy" />

      <soap:body use="literal" />
    </input>
    <output>
      <wsp:PolicyReference URI="#AddNumbersPortBinding_addNumbers_WSAT_Policy" />

      <soap:body use="literal" />
    </output>
  </operation>
</definitions>
```

```
</output>
</operation>
</binding>
<service>...</service>
<definitions>
```

## 16.2.2. Javaクラスによる設定

JavaクラスでWebサービス・トランザクションを設定するには、Webサービス・エンドポイントの実装クラスに、以下のアノテーションを設定します。

```
@com.sun.xml.ws.api.tx.at.Transactional(version=com.sun.xml.ws.api.tx.at.Transactional.Version.WSAT10)
```

以下は、JavaクラスでWebサービス・トランザクションを設定した例です。

### [例 16.2] << AddnumbersImpl.java >>

```
@WebService
@com.sun.xml.ws.api.tx.at.Transactional(
    version=com.sun.xml.ws.api.tx.at.Transactional.Version.WSAT10)
public class AddNumbersImpl {
    ...
}
```

## 16.3. クライアントの設定

Webサービス・トランザクションのためのクライアント側での追加設定は必要ありません。JEUS Webサービスは、リモートWebサービスのWSDLのWebサービス・トランザクションのポリシーを解釈して、自動でWebサービス・トランザクションのための環境を提供します。

## 16.4. コーディネーター・サービス

Webサービス・トランザクションを使用するためには、トランザクション参加者間のトランザクション・アクティビティを調整するコーディネーター・サービスをデプロイする必要があります。コーディネーター・サービスはサーバーとクライアントの両方に必要です。WebサービスとWebサービス・クライアントが同じサーバーで動作した場合、1つのコーディネーター・サービスのみデプロイします。

コーディネーター・サービスのためのwstx-services.earは、以下のパスのディレクトリーに位置します。

```
JEUS_HOME/lib/systemapps
```

## 16.5. Webサービス・トランザクションの例

Javaクラスでの実装方法は、@com.sun.xml.ws.api.tx.at.Transactional annotationを設定すること以外は、基本的なJEUS 8 Webサービスと同じです。

Webサービス・クライアントにはJTA(Java Transaction API)を使用してプログラミングします。

### [例 16.3] << AddNumbersClient.jsp >>

```
InitialContext ctx = new InitialContext();
UserTransaction utx = (UserTransaction) ctx.lookup("java:comp/UserTransaction");

AddNumbersImplService service = new AddNumbersImplService();
AddNumbersImpl port = service.getAddNumbersImplPort();

utx.begin();

int result = port.addNumbers(number1, number2);

utx.commit();
```



# 第17章 Webサービスのセキュリティ

本章では、トランスポート・レベルとメッセージ・レベルのセキュリティについて紹介し、JEUS Webサービスでこのようなセキュリティの適用を行う方法について説明します。

## 17.1. 概要

Webサービスにセキュリティを適用する方法は以下の2つです。

- トランスポート・レベルのセキュリティ

SSLを使用し、クライアントとWebサービス間の接続セキュリティを保証します。

トランスポート・レベルのセキュリティを適用した場合、SSLを利用してクライアントとJEUSサーバー間の接続を安全に行えます。しかし、この方法のメリットは安全な接続のみです。クライアントとJEUSサーバーの間にルータやメッセージ・キューのような媒介がある場合、その媒介はSOAPメッセージを暗号化されていない読みやすいテキスト文書の形式で持つことができます。また、トランスポート・レベルのセキュリティはメッセージ全体を扱うため、メッセージの一部分に対するセキュリティの適用は不可能です。

- メッセージ・レベルのセキュリティ

SOAPメッセージを電子署名または暗号化します。

メッセージ・レベルのセキュリティはSSLのセキュリティのメリットを持ちつつ、柔軟性も提供します。メッセージ・レベルのセキュリティとは、メッセージの送信時に1つでも媒介が存在するとセキュリティが維持されるEnd-to-Endセキュリティで、接続自体のセキュリティが保証されるというよりはSOAPメッセージを署名・暗号化することを意味します。また、部分的な署名と暗号化が可能であるというメリットがあります。

## 17.2. トランスポート・レベルのセキュリティ

Webサービスのトランスポート・レベルのセキュリティとは、Webサービス・クライアントのアプリケーション・プログラムとWebサービス間の接続を、SSLを利用して安全に行うことです。

全体的な手順は以下のとおりです。

1. JEUSサーバーのSSLを設定します。

Webサービスの開発部分は追加作業が必要ありません。JEUSサーバーでのSSL設定は『JEUS Webエンジンガイド』を参照してください。

2. 以下の手順でクライアントのアプリケーション・プログラムのSSLを設定します。

- a. 証明書を取得します。(Internet Explorerなどを使用して証明書をローカル・ディレクトリーに保存します)
- b. 取得した証明書をキーストアに保存します。
- c. wsimportを使用してWSDLでスタブを作成する場合や、クライアントからWebサービスを呼び出すためにクライアントを実行する場合、システム・プロパティ値を以下のように設定します。

```
-Djavax.net.ssl.trustStore=keystore_name
```

```
-Djavax.net.ssl.trustStorePassword=keystore_password
```

- d. wsimportツールを使用するためには、以下の環境変数を追加で設定する必要があります。

```
set WSIMPORT_OPTS=-Djavax.net.ssl.trustStore=keystore_name  
-Djavax.net.ssl.trustStorePassword=keystore_password
```

## 17.3. メッセージ・レベルのセキュリティ

以下は、JEUS Webサービスが実装しているWebサービスのセキュリティに関する明細書です。

- Webサービスのセキュリティ・ポリシー(WS-Security Policy)
- Webサービスのセキュリティ(WS-Security)
- Webサービスのセキュリティ対話(WS-SecureConversation)
- Webサービスの信頼(WS-Trust)

本節では、各明細書の意味とシナリオについて説明します。

### 17.3.1. Webサービスのセキュリティ・ポリシー

Webサービスのセキュリティ・ポリシーの明細には、メッセージがどのようにセキュリティ化して移動するかを説明するアサーションなどを定義します。このようなアサーションは、トークン、暗号化技法、使用メカニズム、トランスポート・セキュリティなどのポリシーを包括する柔軟性を持ちます。このようなWebサービスのセキュリティ・ポリシーは、Webサービスのメッセージ・レベルのセキュリティの実装時に中心的な要素となります。

JEUS Webサービスでは、多様なWebサービスのメッセージ・レベルのセキュリティのためのシナリオ別の例を提供します。シナリオ別の例のWebサービスのセキュリティ・ポリシーを適用したポリシー・ファイルは、Webサービスのメッセージ・セキュリティの適用に役立ちます。

## 17.3.2. Webサービスのセキュリティ

Webサービスのセキュリティの明細は、メッセージのXML署名を使用したデータの整合性とXML暗号化を使用したデータの秘密性を実現し、安全なWebサービスの実装に使用される拡張されたSOAP要素を定義しています。

以下は、Webサービスのセキュリティの明細を使用したWebサービスのセキュリティのシナリオです。

- ユーザー名の認証による対称バインディングの認証機能の強化
- 相互認証のセキュリティ
- SSLによるSAMLの認証

### 17.3.2.1. ユーザー名の認証による対称バインディングの認証機能の強化

これは、対称バインディングによるメッセージ・セキュリティの例です。前述と同様に、対称バインディングでは、クライアントとサーバーが暗号化キーと署名に同一の証明書情報を使用します。この例では、その証明書情報としてサーバーの証明書を使用します。この場合、クライアントはサーバーの認証を受ける方法がないため、付加的にユーザー名トークンを使用します。

以下は、ユーザー名の認証のシナリオとキースタアの設定に関する説明です。

#### ● シナリオ

以下は、ユーザー名の認証のシナリオです。

1. クライアントは対称鍵を作成し、それを利用して要求メッセージを暗号化・署名します。このとき、サーバーの公開鍵を利用し、対称鍵をメッセージに追加して送信します。また、追加でユーザー名とパスワード情報も一緒に送信します。
2. サーバーは、サーバーの秘密鍵を利用して対称鍵を解読し、それを使用してクライアントの要求メッセージを復号化し、署名します。追加でユーザー名とパスワードを使用してクライアントを認証します。
3. サーバーは、またその対称鍵を使用して、応答メッセージを署名および暗号化します。このとき、対称鍵を別途クライアントに送信しません。
4. クライアントは、持っている対象鍵を使用してサーバーの応答メッセージを解析し、署名を確認します。

#### ● キースタアの設定

以下はキースタアの設定情報です。

区分	説明
クライアント	サーバーの公開鍵が格納されているキースタアです

区分	説明
サーバー	サーバーの秘密鍵が格納されているキーストアです

### 17.3.2.2. 相互認証セキュリティ

これは、非対称バインディング、相互証明書によるメッセージ・セキュリティの例です。対称バインディングとは異なり、非対称バインディングではクライアントとサーバーが暗号化キーと署名に異なる証明書情報を使用します。自分の秘密鍵で署名して証明書情報を渡すことで、互いに署名を検証する際に相手が誰かを認証できます。

以下は、相互認証セキュリティのシナリオとキーストアの設定についての説明です。

#### ● シナリオ

以下は、相互認証セキュリティのシナリオです。

1. クライアントは、クライアントの秘密鍵を利用してメッセージに署名し、1つの対称鍵を作成してメッセージを暗号化します。そして、その対称鍵をサーバーの公開鍵で暗号化し、メッセージと共にサーバーに要求メッセージを送信します。このとき、認証を行うクライアントの証明書を一緒に送信します。
2. サーバーは、サーバーの秘密鍵を使用して対称鍵を取得し、暗号化されたメッセージを解析します。証明書で認証を行い、クライアントの公開鍵で署名を検証します。
3. サーバーはサーバーの秘密鍵を利用してメッセージに署名し、1つの対称鍵を作成してメッセージを暗号化します。そして、その対称鍵をクライアントの公開鍵で暗号化し、メッセージと共にクライアントに応答メッセージを送信します。このとき、認証を行うサーバーの証明書は一緒に送信しません。
4. クライアントは、クライアントの秘密鍵を使用して対称鍵を取得し、暗号化されたメッセージを解析します。サーバーの公開鍵で署名を検証します。

#### ● キーストアの設定

以下はキーストアの設定情報です。

区分	説明
クライアント	サーバーの公開鍵が格納されているキーストアです
サーバー	サーバーの秘密鍵が格納されているキーストアです

### 17.3.2.3. SSLによるSAML認証

本節では、SAML(Security Assertions Mark-up Language)トークンをSOAPメッセージ・ヘッダーに追加して送信する方法について説明します。

実際のWebサービスでSAMLトークンを使用したサービスを行うには、SAMLトークンを作成するSAMLトークン・フレームワークが必要です。ここではハンドラーを利用し、SAMLトークンをテキスト上で作成して送信する例を通じて、SAMLトークンをSOAPメッセージ・ヘッダーに追加する方法について説明します。

Webサービスの発展とユーザーの活用頻度の増加により、多様なWebアプリケーション間のセキュリティおよび認証情報の交換の必要性が高まっているため、交換標準が必要です。このような標準を提供するための言語がSAML(Security Assertions Mark-up Language)です。

SAMLで提供する機能は以下のとおりです。

- ユーザー・セキュリティ情報のXMLフォーマットを提供し、さらにその情報を要求・送信するためのフォーマットも提供します。
- SOAPなどのプロトコルでメッセージを使用する方法を定義します。
- Web SSOのように、一般的な特定の使用例について詳しいメッセージ交換方法を指定します。
- ユーザーの身元をエクスポートせず、ユーザー属性を決める機能などの多様な個人情報保護メカニズムをサポートします。
- UNIX、Microsoft Windows、X.509、LDAP、DCE、XCMLなど、広く使用されている技術で提供するフォーマットでID情報を処理する方法を提供します。
- メタデータ・スキーマを数式化し、参加するシステムでサポートするSAMLオプションと通信できる機能を提供します。

以下は、SSLを使用したSAML認証を使用するためのシナリオです。

#### ● シナリオ

1. クライアントはSAMLトークンをSOAPメッセージ・ヘッダーに追加し、SSL設定によりメッセージを送信します。
2. サーバーはSOAPメッセージ・ヘッダーに存在するSAMLトークンを解析して、応答メッセージを送信します。

---

## 参考

すべてのクライアントとサーバーのメッセージ交換は、JEUSサーバーのSSL設定により行われます。

---

### 17.3.3. Webサービスのセキュリティ対話

Webサービスのセキュリティ対話の明細は、サービス提供者とクライアント間のセキュリティ・コンテキストの作成および共有を定義します。セキュリティ・コンテキストは、メッセージの交換時に発生するオーバーヘッドを減らすために使用します。この明細は、より良いメッセージ・レベルのセキュリティと、効率的な多重メッセージ交換を提供するための標準を定義します。

この明細に従っているJEUS Webサービスは、多重メッセージ交換のためのセキュリティ方式が定義される基本メカニズムを提供します。また、効率的なキーや新しいキー・マテリアルでコンテキストを構成します。このようなアクセス方法は、全体的な性能とメッセージ交換のセキュリティ性を向上させます。

対称バインディングでは、メッセージを送受信するたびに暗号化鍵を作成する必要があります。したがって、メッセージを複数回送受信するシナリオでは適合していません。この例では、セキュリティ対話を利用して最初のメッセージ送受信を行う前に、ハンドシェイクを通じて非対称バインディングでSecureContextを樹立します。その後、実際にメッセージを送受信するとき、先ほど樹立したSecureContextを対称バインディングと共に署名・暗号化に使用します。認証は、最初に非対称バインディングでSecureContextを樹立するときに行います。

以下は、Webサービスのセキュリティ対話のシナリオとキーストアの設定についての説明です。

#### ● シナリオ

以下は、Webサービスのセキュリティ対話のシナリオです。

1. クライアントは自分の証明書を利用して、証明書とSecureContext情報を署名し、サーバーの公開鍵で暗号化してサーバーに送信します。(非対称バインディング)
2. サーバーは自分の秘密鍵でクライアントのメッセージを復号化し、クライアントの署名を検証して認証した後、SecureContext樹立メッセージで応答します。(非対称バインディング)
3. その後、クライアントとサーバーはメッセージの送受信時に、このSecureContext情報を利用して暗号化および署名を行います。(対称バインディング)

#### ● キーストアの設定

以下はキーストアの設定情報です。

区分	説明
クライアント	クライアントの秘密鍵とサーバーの公開鍵が格納されているキーストアです
サーバー	サーバーの秘密鍵とクライアントの公開鍵が格納されているキーストアです

### 17.3.4. Webサービスの信頼

本節では、WS-Trustを認証に使用する例について説明します。互いに異なるトークン、つまり信頼できない関係の証明書は、STSによりSAMLトークンという形式に変換され、要求者を信頼できない応答者がSTSを通じて認証することができます。

Webサービスのセキュリティを使用するメッセージ交換方式では、サービスとクライアント間のセキュリティ情報を共有するために使用するセキュリティ・トークンに合意する必要があります。予め合意しなかった場合は、メッセージ交換を行う前に、信頼関係を構築する必要があります。これをWebサービスの信頼(Web Service Trust)といいます。JEUS WebサービスはWebサービスの信頼をサポートします。

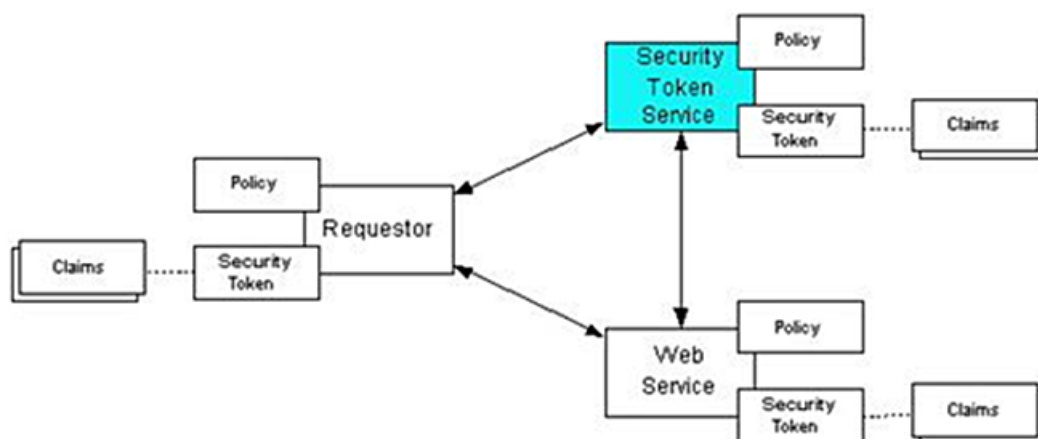
Webサービスの信頼は、サーバーとクライアントが異なる形式のセキュリティ・トークンまたは異なるドメイン上のセキュリティ・トークンを使用して認証が行えない場合に使用します。このような理由で認証が行えない場合のため、クライアントとサーバ間の認証を仲介する他のWebサービスが存在します。このようなWebサービスをそのサービスを**STS(Security Token Service)**といいます。

Webサービスの信頼は以下の関係を持ちます。

- クライアントとサーバーは直接的な信頼関係がありません。
- クライアントとSTSは信頼関係にあります。
- STSとサーバーは信頼関係にあります。

以下の図のように、リクエスターはWebサービスと通信する前に、Webサービスが自分を認証できるようにサポートするセキュリティ・トークンをSTSから動的に割り当てられます。割り当てられたセキュリティ・トークンを通じて、リクエスターはWebサービスに送信するメッセージにSTSから割り当てられたセキュリティ・トークンを追加して送信します。それにより、Webサービスはリクエスターを認証(信頼)できます。

[図 17.1] WS-Trust



\*出典 <http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html>

以下は、Webサービスの信頼のシナリオとキーストアの設定についての説明です。

#### ● シナリオ

以下は、Webサービスの信頼のシナリオです。

1. クライアントは、自分の証明書、ユーザー名、パスワードと共に、実際にメッセージを送信するサーバーに対するSAML証明書の発給をSTSに要求します。
2. STSは、クライアントを認証した後、SAML認証書を発給します。
3. クライアントは、SAML証明書と共にサーバーにメッセージを送信します。
4. サーバーは、SAML証明書を利用してクライアントを信頼し、ビジネス・ロジックを実行します。

#### ● キーストアの設定

以下は、キーストアの設定情報です。

区分	説明
クライアント	1. クライアントの秘密鍵とサーバーの公開鍵が格納されているキーストアです 2. クライアントの秘密鍵とSTSサーバーの公開鍵が格納されているキーストアです
サーバー	サーバーの秘密鍵とクライアントの公開鍵が格納されているキーストアです
STSサーバー	サーバーの秘密鍵とクライアントの公開鍵が格納されているキーストアです

## 17.4. メッセージ・レベルのセキュリティ設定

本節では、JEUS Webサービスのメッセージ・レベルのセキュリティのシナリオ設定について説明します。その後、各シナリオの設定方法について記述します。実際のシナリオの動作については、例を参照してください。

---

#### 参考

各シナリオ別メッセージのセキュリティ設定例のWebサービスとクライアントのビジネス・ロジックはすべて同一です。

---

### 17.4.1. 共通の設定

JEUS Webサービスのメッセージ・レベルのセキュリティのシナリオで共通して適用される設定は、サーバーとクライアントに区分されます。



### 17.4.1.1. サーバーの設定

以下は、endpoint-policy-subject、operation-policy-subject、input(output)-message-policy-subjectに共通のセキュリティ設定について説明します。

- **keystore、truststore要素の設定**

security-policy下位の要素であるkeystore、truststore要素は、security-policy要素のsecurity-bindingのセキュリティ設定によって能動的に設定する必要があります。

keystore-filenameに含まれるキーストア・ファイルの設定は、絶対パスまたは相対パスを含む実際のキーストア・ファイルの位置を指定します。相対パスで指定する場合は、サービス実装クラスが含まれるclassesディレクトリー下位のMETA-INFディレクトリーにキーストア・ファイルが位置する必要があります。

**[例 17.1] << jeus-webservices-config.xsd >>**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0">
    .....
    <xs:complexType name="keyTruststoreType">
        <xs:choice>
            <xs:element name="keystore-file" type="keyTruststoreFileType" />
            <xs:element name="keystore-callbackhandler" type="xs:string" />
        </xs:choice>
    </xs:complexType>
    <xs:complexType name="keyTruststoreFileType">
        <xs:sequence>
            <xs:element name="alias" type="xs:string" />
            <xs:element name="key-type" type="xs:string" minOccurs="0"
default="JKS" />
            <xs:element name="keystore-password" type="xs:string" />
            <xs:element name="keystore-filename" type="xs:string" />
        </xs:sequence>
    </xs:complexType>
    .....
</xs:schema>
```

- **include-token属性の設定**

include-token属性は、メッセージにトークンを含めるか否かを設定します。

X509トークンで使用される例は以下のとおりです。

**[例 17.2] << jeus-webservices-config.xsd >>**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
```

```

targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0">
    .....
    <xs:complexType name="x509TokenType">
        <xs:sequence>
            <xs:element name="include-token" type="xs:boolean" minOccurs="0"
                default="false" />
        </xs:sequence>
    </xs:complexType>
    .....
</xs:schema>

```

---

### 参考

それ以外の要素の設定についての詳細は「jeus-webservices-config.xsd」スキーマおよび『JEUS リファレンスガイド』、サンプル例を参照してください。

---

## JavaクラスによるWebサービスの実装

JavaクラスでWebサービスのメッセージ・セキュリティを実装するには、以下のように**wsgen**ツールの-policy機能を使用してWebサービスを構成します。

```
$ wsgen fromjava.server.AddNumbersImpl -d web/WEB-INF -policy service-config.xml
```

-policyの引数として使用されるservice-config.xmlを構成するには、service-config.xmlのためのJEUSセキュリティ・ポリシー・スキーマであるjeus-webservices-config.xsdについての知識が必要です。

JEUSセキュリティ・ポリシー・スキーマは以下のように、endpoint-policy-subject、operation-policy-subject、input(output)-message-policy-subject要素にセキュリティの設定を行うことができます。

### [例 17.3] << jeus-webservices-config.xsd >>

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0">
    .....
    <xs:element name="web-services-config" type="web-services-configType" />
    <xs:complexType name="web-services-configType">
        <xs:choice>
            .....
            <xs:element name="policy" type="policy-configType"
                maxOccurs="unbounded" />
        </xs:choice>
    </xs:complexType>
    <xs:complexType name="policy-configType">

```

```

<xs:sequence>
  <xs:element name="endpoint-policy-subject"
    type="endpointPolicySubjectType" maxOccurs="unbounded" />
</xs:sequence>
</xs:complexType>
<xs:complexType name="endpointPolicySubjectType">
  <xs:sequence>
    .....
    <xs:element name="security-policy"
      type="endpointSecurityPolicyType" minOccurs="0" />
    <xs:element name="operation-policy-subject"
      type="operationPolicySubjectType" minOccurs="0"
      maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="operationPolicySubjectType">
  <xs:sequence>
    <xs:element name="security-policy"
      type="operationSecurityPolicyType" minOccurs="0" />
    .....
    <xs:element name="input-message-policy-subject"
      type="messagePolicySubjectType" minOccurs="0" />
    <xs:element name="output-message-policy-subject"
      type="messagePolicySubjectType" minOccurs="0" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="messagePolicySubjectType">
  <xs:sequence>
    <xs:element name="security-policy" type="messageSecurityPolicyType"
      minOccurs="0" />
  </xs:sequence>
</xs:complexType>
.....
</xs:schema>

```

この3つの要素のセキュリティー設定の内容は以下のとおりです。

- **endpoint-policy-subject要素のセキュリティー設定**

サービス実装クラス単位でセキュリティーを設定します。セキュリティー・バインディング(非対称、対称)設定や署名、暗号化、セキュリティー・トークン、WS-Securityバージョンなど、メッセージのセキュリティーに関する多様な設定を行います。

**[例 17.4] << jeus-webservices-config.xsd >>**

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
  targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  attributeFormDefault="unqualified" version="7.0">

```

```

.....
<xs:complexType name="endpointSecurityPolicyType">
  <xs:sequence>
    <xs:element name="security-binding" type="securityBindingType"
      minOccurs="0" />
    <xs:element name="token" type="supportingTokenType" minOccurs="0" />
    <xs:element name="protection" type="protectionType" minOccurs="0" />
    .....
  </xs:sequence>
</xs:complexType>
.....
</xs:schema>

```

endpoint-policy-subject要素のセキュリティー設定は、サービス実装クラス単位で以下のように3つに分類されます。

- security-binding

以下のように、security-binding要素のセキュリティー設定は、transport-binding、symmetric-binding、asymmetric-bindingのうち1つを選択できます。

**[例 17.5] << jeus-webservices-config.xsd >>**

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
  targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  attributeFormDefault="unqualified" version="7.0">
  .....
  <xs:complexType name="securityBindingType">
    <xs:sequence>
      <xs:choice>
        <xs:element name="transport-binding"
          type="transportBindingType" />
        <xs:element name="symmetric-binding"
          type="symmetricBindingType" />
        <xs:element name="asymmetric-binding"
          type="asymmetricBindingType" />
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
  .....
</xs:schema>

```

- transport-binding要素のセキュリティー設定

transport-binding要素のセキュリティー設定は、Webサービスのセキュリティー(WS-Security)の明細以外の送信(HTTPSなど)を使用してメッセージを保護する場合に使用します。

### [例 17.6] << jeus-webservices-config.xsd >>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0">
    .....
    <xs:complexType name="transportBindingType">
        <xs:sequence>
            <xs:element name="transport-token" type="tokenType" />
            <xs:element name="algorithm-suite" type="algorithmSuiteType"
minOccurs="0" default="Basic128" />
            <xs:element name="layout" type="layoutType" minOccurs="0"
default="Lax" />
            <xs:element name="timestamp" type="xs:boolean" minOccurs="0"
default="true" />
        </xs:sequence>
    </xs:complexType>
    .....
</xs:schema>
```

- symmetric-binding要素のセキュリティ設定

symmetric-binding要素のセキュリティ設定は、Webサービスのセキュリティ(WS-Security)の明細を使用し、対称バインディング(Symmetric Binding)でメッセージを保護する場合に使用します。

メッセージを暗号化・復号化する場合に使用する対称鍵は、SOAPメッセージのヘッダー部分の追加的な要素(EncryptedKey)下位に暗号化され、メッセージと共に送信されます。対称バインディングとは、この対称鍵を暗号化・復号化するときに使用するトークンが同じであることを意味します。また、署名と署名の検証が同じトークンによって行われることを意味します。下位のprotection-token要素は、追加でメッセージの暗号化および署名に使用されるトークンが同じであることを意味します。

### [例 17.7] << jeus-webservices-config.xsd >>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0">
    .....
    <xs:complexType name="symmetricBindingType">
        <xs:sequence>
            <xs:element name="protection-token" type="tokenType" />
            <xs:element name="algorithm-suite" type="algorithmSuiteType"
minOccurs="0" default="Basic128" />
            <xs:element name="layout" type="layoutType" minOccurs="0"
default="Lax" />
            <xs:element name="timestamp" type="xs:boolean" minOccurs="0"
default="true" />
        </xs:sequence>
    </xs:complexType>
    .....
</xs:schema>
```

```

        default="true" />
        <xs:element name="encrypt-signature" type="xs:boolean"
        minOccurs="0" default="false" />
        <xs:element name="encrypt-before-siging" type="xs:boolean"
        minOccurs="0" default="false" />
    </xs:sequence>
</xs:complexType>
.....
</xs:schema>

```

- asymmetric-binding要素のセキュリティー設定

asymmetric-binding要素のセキュリティー設定は、Webサービスのセキュリティー(WS-Security)の明細を使用して非対称バインディング(Asymmetric Binding)でメッセージを保護する場合に使用します。

メッセージを暗号化・復号化する場合に使用する対称鍵は、SOAPメッセージのヘッダー部分の追加的な要素(EncryptedKey)下位に暗号化され、メッセージと共に送信されます。非対称バインディングとは、この対称鍵を暗号化を暗号化・復号化するときに使用するトークンが異なることを意味します。また、署名と署名の検証が異なるトークンによって行われることを意味します。

**[例 17.8] << jeus-webservices-config.xsd >>**

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0">
    .....
    <xs:complexType name="asymmetricBindingType">
        <xs:sequence>
            <xs:element name="initiator-token" type="tokenType" />
            <xs:element name="recipient-token" type="tokenType" />
            <xs:element name="algorithm-suite" type="algorithmSuiteType"
            minOccurs="0" default="Basic128" />
            <xs:element name="layout" type="layoutType" minOccurs="0"
            default="Strict" />
            <xs:element name="timestamp" type="xs:boolean" minOccurs="0"
            default="true" />
            <xs:element name="encrypt-before-siging" type="xs:boolean"
            minOccurs="0" default="false" />
        </xs:sequence>
    </xs:complexType>
    .....
</xs:schema>

```

以下は、下位の要素についての説明です。

要素	説明
initiator-token	<p>クライアントがメッセージを署名するときに使用するトークンであるとともに、サーバーから受信した暗号化されたメッセージを復号化する際に使用するトークンであることを意味します。</p> <p>サーバーがメッセージを暗号化するときに使用するトークンであるとともに、クライアントから受信したメッセージの署名を検証する際に使用するトークンであることを意味します</p>
recipient-token	<p>サーバーがメッセージを署名するときに使用するトークンであるとともに、クライアントから受信した暗号化されたメッセージの復号化に使用するトークンであることを意味します。</p> <p>クライアントがメッセージを暗号化するときに使用するトークンであるとともに、サーバーから受信したメッセージの署名を検証する際に使用するトークンであることを意味します</p>

#### – token

以下は、token要素のセキュリティー設定に関する例で、下位の要素のうち1つを選択できます。

#### [例 17.9] << jeus-webservices-config.xsd >>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0">
    .....
    <xs:complexType name="supportingTokenType">
        <xs:choice>
            <xs:element name="supporting-token" type="tokenType" />
            <xs:element name="signed-supporting-token" type="tokenType" />
            <xs:element name="endorsing-supporting-token" type="tokenType" />
            <xs:element name="signed-endorsing-supporting-token"
type="tokenType" />
        </xs:choice>
    </xs:complexType>
    .....
</xs:schema>
```

以下は、下位の要素についての説明です。

要素	説明
supporting-token	ここで設定したトークン値は、メッセージのヘッダー部分に含まれ、他のメッセージを暗号化・署名するときに使用します

要素	説明
signed-supporting-token	supporting-tokenと同じです。追加でこのトークンも署名されます
endorsing-supporting-token	再度署名するときに使用するトークンを設定する場合に使用します。Signature要素全体を署名します
signed-endorsing-supporting-token	endorsing-supporting-tokenと同じです。追加でこのトークンも署名を利用して署名を行います(相互署名)

#### – protection

以下は、protection要素のセキュリティー設定の例で、下位の要素のうち1つを選択します。

#### [例 17.10] << jeus-webservices-config.xsd >>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0">
    .....
    <xs:complexType name="protectionType">
        <xs:sequence>
            <xs:element name="signed-part" type="protectionPartType"
minOccurs="0" />
            <xs:element name="encrypted-part" type="protectionPartType"
minOccurs="0" />
            <xs:element name="signed-element" type="xs:string"
minOccurs="0" />
            <xs:element name="encrypted-element" type="xs:string"
minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    .....
</xs:schema>
```

以下は、下位の要素についての説明です。

要素	説明
signed-part	署名(検証)する部分の設定に使用します
encrypted-part	暗号化あるいは復号化する部分の設定に使用します
signed-element	署名(検証)する部分の設定に使用します。特定要素を部分署名する場合に有効です。  signed-element要素の設定時にdisable-streaming-security要素の値をtrueにする必要があります。詳細についてはサンプルを参照してください



要素	説明
encrypted-element	暗号化あるいは復号化する部分の設定に使用します。特定要素を部分暗号化する場合に有効です。  encrypted-element要素の設定時にdisable-streaming-security要素の値をtrueにする必要があります。詳細についてはサンプルを参照してください

#### ● operation-policy-subject要素のセキュリティ設定

サービス実装クラスのメソッド単位でセキュリティ設定をします。署名および暗号化、これをサポートするセキュリティ・トークンを追加で設定できます。

operation-policy-subject要素のセキュリティ設定は、サービス実装クラスのメソッド単位のセキュリティ設定で、以下の2つに分類されます。

#### [例 17.11] << jeus-webservices-config.xsd >>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0">
    .....
    <xs:complexType name="operationSecurityPolicyType">
        <xs:sequence>
            <xs:element name="supporting-token" type="supportingTokenType"
minOccurs="0" />
            <xs:element name="protection" type="protectionType" minOccurs="0" />
        </xs:sequence>
    </xs:complexType>
    .....
</xs:schema>
```

以下は、下位の要素についての説明です。

要素	説明
supporting-token	この設定の内容はendpoint-policy-subject要素のtoken [155]要素と同じです。ただし、このメソッドにのみ設定されます
supporting-token	この設定の内容はendpoint-policy-subject要素の'token' [155] element . , .
protection	この設定の内容はendpoint-policy-subject要素のprotection [156]要素と同じです。ただし、このメソッドにのみ設定されます

#### ● input(output)-message-policy-subject要素のセキュリティ設定

サービス実装クラスのメソッドの媒介変数および戻り値単位でセキュリティ設定をします。署名および暗号化と、これをサポートするセキュリティ・トークンを追加で設定できます。

input(output)-message-policy-subject要素のセキュリティ設定は、サービス実装クラスのメソッドの媒介変数および戻り値単位のセキュリティ設定で、以下の2つに分類されます。

**[例 17.12] << jeus-webservices-config.xsd >>**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0">
    .....
    <xs:complexType name="messageSecurityPolicyType">
        <xs:sequence>
            <xs:element name="supporting-token" type="supportingTokenType"
minOccurs="0" />
            <xs:element name="protection" type="protectionType" minOccurs="0" />
        </xs:sequence>
    </xs:complexType>
    .....
</xs:schema>
```

以下は、下位の要素についての説明です。

要素	説明
supporting-token	この設定の内容はendpoint-policy-subject要素のtoken [155]要素と同じです。ただし、このメソッドの媒介変数および戻り値にのみ設定されます
protection	この設定の内容はendpoint-policy-subject要素のprotection [156]要素と同じです。ただし、このメソッドの媒介変数および戻り値にのみ設定されます

## WSDLによるWebサービスの実装

WSDLで実装するには、WSDL文書に直接メッセージ・セキュリティ・ポリシーを設定し、**wsimport**ツールを使用してWebサービスを構成します。WSDLで実装するには、WS-Policyの下位スペックであるWS-SecurityPolicyの内容を正確に理解してWSDLに適用すると、簡単にWebサービスのメッセージ・セキュリティを実装できます。

### 参考

WS-SecurityPolicyの内容は<http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html>で確認してください。

以下は、メッセージ・セキュリティが設定されているWebサービスを実装するためのWSDLの例です。

キーストアの設定は、絶対パスまたは相対パスを含む実際のキーストア・ファイルの位置を指定します。相対パスに指定する場合は、サービス実装クラスが含まれているclassesディレクトリー下位のMETA-INFディレクトリーに位置する必要があります。

**[例 17.13] << jeus-webservices-config.xsd >>**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions
...
targetNamespace="http://tmax.com/" name="NewWebServiceService">
<ns1:Policy xmlns:ns1="http://schemas.xmlsoap.org/ws/2004/09/policy"
wsu:Id="NewWebServicePortBindingPolicy">
  <ns1:ExactlyOne>
    <ns1:All>

<ns7:SymmetricBindingxmlns:ns7="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">

  <ns1:Policy>
    <ns1:ExactlyOne>
      <ns1:All>
        <ns7:AlgorithmSuite>
          <ns1:Policy>
            <ns1:ExactlyOne>
              <ns1:All>
                <ns7:TripleDes />
              </ns1:All>
            </ns1:ExactlyOne>
          </ns1:Policy>
        </ns7:AlgorithmSuite>
        <ns7:IncludeTimestamp />
        <ns7:Layout>
          <ns1:Policy>
            <ns1:ExactlyOne>
              <ns1:All>
                <ns7:Strict />
              </ns1:All>
            </ns1:ExactlyOne>
          </ns1:Policy>
        </ns7:Layout>
        <ns7:OnlySignEntireHeadersAndBody />
        <ns7:ProtectionToken>
          <ns1:Policy>
            <ns1:ExactlyOne>
              <ns1:All>
                <ns7:X509Tokenns7:IncludeToken=
"http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/Never">

              <ns1:Policy>
                <ns1:ExactlyOne>
                  <ns1:All>
```

```

        <ns7:WssX509V3Token10/>
    </ns1:All>
</ns1:ExactlyOne>
</ns1:Policy>
</ns7:X509Token>
</ns1:All>
</ns1:ExactlyOne>
</ns1:Policy>
</ns7:ProtectionToken>
</ns1:All>
</ns1:ExactlyOne>
</ns1:Policy>
</ns7:SymmetricBinding>
<ns8:Wss11xmlns:ns8="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
    <ns1:Policy>
        <ns1:ExactlyOne>
            <ns1:All>
                <ns8:MustSupportRefEncryptedKey />
                <ns8:MustSupportRefIssuerSerial />
                <ns8:MustSupportRefThumbprint />
            </ns1:All>
        </ns1:ExactlyOne>
    </ns1:Policy>
</ns8:Wss11>
<sc:KeyStore wspp:visibility="private"
alias="xws-security-server"
storepass="changeit" type="JKS"
location="keystore.jks" />
<sc:TrustStore wspp:visibility="private"
peeralias="xws-security-client"
storepass="changeit" type="JKS"
location="cacerts.jks" />
<sc:ValidatorConfigurationxmlns:sc=
    "http://schemas.sun.com/2006/03/wss/server">
    <sc:Validator name="usernameValidator"
        classname="com.tmax.UsernamePasswordValidator" />
</sc:ValidatorConfiguration>
<ns9:UsingAddressingxmlns:ns9="http://www.w3.org/2006/05/addressing/wsdl" />
</ns1:All>
</ns1:ExactlyOne>
</ns1:Policy>
<ns10:Policy xmlns:ns10="http://schemas.xmlsoap.org/ws/2004/09/policy"
    wsu:Id="NewWebServicePortBinding_add_Input_Policy">
    <ns10:ExactlyOne>
        <ns10:All>
            <ns11:EncryptedPartxmlns:ns11=
                "http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
                <ns11:Body />
            </ns11:EncryptedPart>
            <ns12:SignedPartxmlns:ns12=

```

```

        "http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
    <ns12:Body />
    <ns12:Header Name="ReplyTo"
        Namespace="http://www.w3.org/2005/08/addressing" />
    <ns12:HeaderNamespace="http://www.w3.org/2005/08/addressing" Name="To"/>
    <ns12:Header Name="From"
        Namespace="http://www.w3.org/2005/08/addressing"/>
    <ns12:Header Name="MessageId"
        Namespace="http://www.w3.org/2005/08/addressing"/>
    <ns12:Header Name="FaultTo"
        Namespace="http://www.w3.org/2005/08/addressing"/>
    <ns12:Header Name="Action"
        Namespace="http://www.w3.org/2005/08/addressing"/>
    <ns12:Header Name="RelatesTo"
        Namespace="http://www.w3.org/2005/08/addressing"/>
</ns12:SignedParts>
<ns13:SignedSupportingTokensxmlns:ns13=
    "http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
    <ns10:Policy>
        <ns10:ExactlyOne>
            <ns10:All>
                <ns13:UsernameToken ns13:
                    IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/
                    IncludeToken/AlwaysToRecipient">
                    <ns10:Policy>
                        <ns10:ExactlyOne>
                            <ns10:All>
                                <ns13:WssUsernameToken10/>
                            </ns10:All>
                        </ns10:ExactlyOne>
                    </ns10:Policy>
                </ns13:UsernameToken>
            </ns10:All>
        </ns10:ExactlyOne>
    </ns10:Policy>
</ns13:SignedSupportingTokens>
</ns10:All>
</ns10:ExactlyOne>
</ns10:Policy>
<ns14:Policy xmlns:ns14="http://schemas.xmlsoap.org/ws/2004/09/policy"
    wsu:Id="NewWebServicePortBinding_add_Output_Policy">
    <ns14:ExactlyOne>
        <ns14:All>
            <ns15:EncryptedPartxmlns:ns15="http://schemas.xmlsoap.org/ws/
                2005/07/securitypolicy">
                <ns15:Body />
            </ns15:EncryptedParts>
            <ns16:SignedPartxmlns:ns16="http://schemas.xmlsoap.org/ws/
                2005/07/securitypolicy">
                <ns16:Body />
            </ns16:SignedParts>
        </ns14:All>
    </ns14:ExactlyOne>
</ns14:Policy>

```

```

        <ns16:Header Name="ReplyTo"
            Namespace="http://www.w3.org/2005/08/addressing" />
        <ns16:HeaderNamespace="http://www.w3.org/2005/08/addressing"
            Name="To" />
        <ns16:Header Name="From"
            Namespace="http://www.w3.org/2005/08/addressing" />
        <ns16:Header Name="MessageId"
            Namespace="http://www.w3.org/2005/08/addressing" />
        <ns16:Header Name="FaultTo"
            Namespace="http://www.w3.org/2005/08/addressing" />
        <ns16:Header Name="Action"
            Namespace="http://www.w3.org/2005/08/addressing" />
        <ns16:Header Name="RelatesTo"
            Namespace="http://www.w3.org/2005/08/addressing" />
    </ns16:SignedParts>
</ns14:All>
</ns14:ExactlyOne>
</ns14:Policy>
<types>
    ...
</types>
<message name="add">
    ...
</message>
<portType name="NewWebService">
    ...
</portType>
<binding name="NewWebServicePortBinding" type="tns:NewWebService">
    <ns17:PolicyReferencexmlns:ns17="http://schemas.xmlsoap.org/ws/2004/09/policy"
        URI="#NewWebServicePortBindingPolicy" />
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
    <operation name="add">
        <soap:operation soapAction="" />
        <input>
            <ns18:PolicyReferencexmlns:ns18="http://schemas.xmlsoap.org/ws/2004/09/policy"
                URI="#NewWebServicePortBinding_add_Input_Policy" />
            <soap:body use="literal" />
        </input>
        <output>
            <ns19:PolicyReferencexmlns:ns19="http://schemas.xmlsoap.org/ws/2004/09/policy"
                URI="#NewWebServicePortBinding_add_Output_Policy" />
            <soap:body use="literal" />
        </output>
    </operation>
</binding>
<service name="NewWebServiceService">
    <port name="NewWebServicePort" binding="tns:NewWebServicePortBinding">
        <soap:addresslocation="http://localhost:8088/usernameAuthentication_war/
            NewWebServiceService"/>
    </port>

```

```
</service>
</definitions>
```

### 17.4.1.2. クライアント設定

メッセージ・セキュリティが設定されているWebサービスのクライアントは、wsit-client.xmlファイルを利用し、追加でキーストアおよびコールバックのハンドラーなどを設定する必要があります。その他のメッセージ・セキュリティの設定は、サーバー側のWSDLに設定されているメッセージ・セキュリティ・ポリシーをランタイムに解析し、メッセージ・セキュリティ環境を自動構成します。

キーストアの設定では、絶対パスまたは相対パスを含む実際のキーストア・ファイルの位置を指定します。相対パスで指定する場合は、サービス実装クラスが含まれているclassesディレクトリー下位のMETA-INFディレクトリーにキーストア・ファイルが位置する必要があります。

#### [例 17.14] << jeus-webservices-config.xsd >>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
...
targetNamespace="http://server.fromjava/">
  <wsp:UsingPolicy />
  <wsp:Policy wsu:Id="TmaxBP0">
    <wsp:ExactlyOne>
      <wsp:All>
        <CallbackHandlerConfiguration
          xmlns="http://schemas.sun.com/2006/03/wss/client">
          <CallbackHandler default="user_jeus" name="usernameHandler" />
          <CallbackHandler default="password_jeus"
            name="passwordHandler" />
        </CallbackHandlerConfiguration>
        <TrustStore location="cacerts.jks" peeralias="xws-security-server"
          storepass="changeit" type="JKS"
          xmlns:ns0="http://java.sun.com/xml/ns/wsit/policy"
          ns0:visibility="private"
          xmlns="http://schemas.sun.com/2006/03/wss/client" />
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
  <types>
    ...
  </types>
  <message name="addNumbers">
    ...
  </message>
  <portType name="AddNumbersImpl">
    ...
  </portType>
  <binding name="AddNumbersImplPortBinding" type="tns:AddNumbersImpl">
    <wsp:PolicyReference URI="#TmaxBP0" />
  </binding>
</definitions>
```

```

<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http" />
<operation name="addNumbers">
  <soap:operation soapAction="" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
</binding>
<service name="AddNumbersImplService">
  <port binding="tns:AddNumbersImplPortBinding" name="AddNumbersImplPort">
    <soap:address
location="http://localhost:8088/DocLitEchoService/AddNumbersImplService" />
  </port>
</service>
</definitions>

```

## 17.4.2. ユーザー名の認証による対称バインディングの認証機能の強化

本節では、サーバーとクライアントにおける、ユーザー名の認証による対称バインディングの認証機能の強化方法について説明します。

### 17.4.2.1. サーバーの設定

サーバーのWebサービスの設定は、service-config.xmlファイルを作成し、ユーザー名のValidatorクラスを作成します。

#### Webサービス設定ファイルの作成

以下は、Webサービス設定ファイルの作成例です。

##### [例 17.15] <<service-config.xml>>

```

<?xml version="1.0" encoding="UTF-8"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <policy>
    <endpoint-policy-subject>
      <security-policy>
        <security-binding>
          <symmetric-binding>
            <protection-token>
              <x509-token>
                <include-token>true</include-token>
              </x509-token>
            </protection-token>
          </symmetric-binding>
        </security-binding>
      </security-policy>
    </endpoint-policy-subject>
  </policy>
</web-services-config>

```



```

        </symmetric-binding>
    </security-binding>
    <token>
        <signed-supporting-token>
            <username-token>
                <username-password-validator>
                    fromjava.server.UsernamePasswordValidator
                </username-password-validator>
                <include-token>true</include-token>
            </username-token>
        </signed-supporting-token>
    </token>
    <protection>
        <signed-part>...</signed-part>
        <encrypted-part>...</encrypted-part>
    </protection>
    <wss-version>11</wss-version>
    <keystore>...</keystore>
</security-policy>
</endpoint-policy-subject>
</policy>
</web-services-config>

```

## ユーザー名のValidatorクラスの作成

com.sun.xml.wss.impl.callback.PasswordValidationCallback.PasswordValidatorを実装し、ユーザー名トークンを使用してアクセスしたユーザーのユーザー名とパスワードを認証します。

### [例 17.16] << UsernamePasswordValidator.java >>

```

public class UsernamePasswordValidator implements
    PasswordValidationCallback.PasswordValidator {

    public boolean validate(PasswordValidationCallback.Request request)
        throws PasswordValidationCallback.PasswordValidationException {
        ...
        return false;
    }

    private boolean validateUserFromDB(String username, String password) {
        ...
        return false;
    }
}

```

## 17.4.2.2. クライアントの設定

クライアント側のWebサービスの設定は、追加でwsit-client.xmlにキーストア(Truststore)と共にユーザー名のハンドラーを設定する必要があります。

## キーストア(Truststore)の設定およびユーザー名のハンドラーの設定

wsit-client.xmlファイルを以下のように設定します。

### [例 17.17] <<wsit-client.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
...
targetNamespace="http://server.fromjava/">
  <wsp:UsingPolicy />
  <wsp:Policy wsu:Id="TmaxBP0">
    <wsp:ExactlyOne>
      <wsp:All>
        <CallbackHandlerConfiguration
          xmlns="http://schemas.sun.com/2006/03/wss/client">
          <CallbackHandler default="user_jeus" name="usernameHandler" />
          <CallbackHandler default="password_jeus"
            name="passwordHandler" />
        </CallbackHandlerConfiguration>
        <TrustStore location="cacerts.jks" peeralias="xws-security-server"
          storepass="changeit" type="JKS"
          xmlns:ns0="http://java.sun.com/xml/ns/wsit/policy"
          ns0:visibility="private"
          xmlns="http://schemas.sun.com/2006/03/wss/client" />
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
  ...
  <binding name="AddNumbersImplPortBinding" type="tns:AddNumbersImpl">
    <wsp:PolicyReference URI="#TmaxBP0" />
    ...
  </binding>
  ...
</definitions>
```

## ユーザー名のハンドラー・クラスの作成

javax.security.auth.callback.CallbackHandlerを実装し、各コールバックをCallbackHandlerに渡すことにより、多様な設定を行うことができます。渡されたコールバックの種類に合わせて設定します。

上記例では、NameCallbackとPasswordCallbackを受信し、コールバックにユーザー名とパスワードを設定するCallbackHandlerを構成します。

以下の例では、クラス・ファイルにユーザー名とパスワードを予め入力しておきましたが、場合によっては多様なreaderオブジェクトを使用してユーザーが直接入力することも可能です。

### [例 17.18] << UsernamePasswordCallbackHandler.java >>

```
public class UsernamePasswordCallbackHandler implements CallbackHandler {
```

```

    public void handle(Callback[] callbacks) throws IOException,
        UnsupportedCallbackException {
        for (int i = 0; i < callbacks.length; i++) {
            Callback callback = callbacks[i];
            if (callback instanceof NameCallback) {
                ((NameCallback) callback).setName("user_jeus");
            } else if (callback instanceof PasswordCallback) {
                ((PasswordCallback) callback).setPassword((
                    new String("password_jeus")).toCharArray());
            } else {
                throw new UnsupportedCallbackException(callback,
                    "Unknow callback for username or password");
            }
        }
    }
}

```

### 17.4.3. 相互認証セキュリティ(Mutual Certificates Security)

本節では、サーバーとクライアントで相互認証セキュリティを設定する方法について説明します。

#### 17.4.3.1. サーバーの設定

サーバーのWebサービスの設定は、service-config.xml設定ファイルで構成します。

**[例 17.19] << service-config.xml >>**

```

<?xml version="1.0" encoding="UTF-8"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <policy>
        <endpoint-policy-subject>
            <security-policy>
                <security-binding>
                    <asymmetric-binding>
                        <initiator-token>
                            <x509-token>
                                <include-token>true</include-token>
                            </x509-token>
                        </initiator-token>
                        <recipient-token>
                            <x509-token>
                                <include-token>false</include-token>
                            </x509-token>
                        </recipient-token>
                    </asymmetric-binding>
                </security-binding>
            </security-policy>
            <protection>
                <signed-part>...</signed-part>
            </protection>
        </endpoint-policy-subject>
    </policy>
</web-services-config>

```

```

        <encrypted-part>...</encrypted-part>
    </protection>
    <keystore>...</keystore>
    <truststore>...</truststore>
</security-policy>
</endpoint-policy-subject>
</policy>
</web-services-config>

```

### 17.4.3.2. クライアントの設定

クライアント側のWebサービスの設定は、追加でwsit-client.xmlにキーストア(Truststore)を設定します。

#### [例 17.20] << wsit-client.xml >>

```

<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
...
targetNamespace="http://server.fromjava/">
    <wsp:UsingPolicy/>
    <wsp:Policy wsu:Id="TmaxBP0">
        <wsp:ExactlyOne>
            <wsp:All>
                <KeyStore
                    alias="xws-security-client"
                    location="keystore.jks"
                    storepass="changeit" type="JKS"
                    xmlns:ns0="http://java.sun.com/xml/ns/wsdl/policy"
                    ns0:visibility="private"
                    xmlns="http://schemas.sun.com/2006/03/wss/client"/>
                <TrustStore
                    location="cacerts.jks"
                    peeralias="xws-security-server"
                    storepass="changeit"
                    type="JKS"
                    xmlns:ns0="http://java.sun.com/xml/ns/wsdl/policy"
                    ns0:visibility="private"
                    xmlns="http://schemas.sun.com/2006/03/wss/client"/>
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>
    ...
    <binding name="AddNumbersImplPortBinding" type="tns:AddNumbersImpl">
        <wsp:PolicyReference URI="#TmaxBP0"/>
        ...
    </binding>
    ...
</definitions>

```

## 17.4.4. SSLによるSAML認証

本節では、サーバーとクライアントでSSLによるSAML認証を設定する方法について説明します。

### 17.4.4.1. サーバーの設定

サーバーのWebサービスの設定は、service-config.xml設定ファイルで構成します。

**[例 17.21] << service-config.xml >>**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <policy>
    <endpoint-policy-subject>
      <security-policy>
        <security-binding>
          <transport-binding>
            <transport-token>
              <https-token/>
            </transport-token>
          </transport-binding>
        </security-binding>
      </security-policy>
    <operation-policy-subject>
      <operation-java-name>addNumbers</operation-java-name>
      <input-message-policy-subject>
        <security-policy>
          <supporting-token>
            <signed-supporting-token>
              <saml-token>
                </saml-token>
              </signed-supporting-token>
            </supporting-token>
          </security-policy>
        </input-message-policy-subject>
      </operation-policy-subject>
    </endpoint-policy-subject>
  </policy>
</web-services-config>
```

### 17.4.4.2. クライアントの設定

クライアント側のWebサービスの設定は、追加でwsit-client.xmlにSAMLメッセージ処理のためのコールバック・ハンドラー・クラスの設定を行う必要があります。

### SAMLメッセージ処理のためのコールバック・ハンドラー・クラスの設定

wsit-client.xmlファイルに、以下とSAMLメッセージ処理のためのコールバック・ハンドラー・クラスを設定します。

#### [例 17.22] << wsit-client.xml >>

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
...
targetNamespace="http://server.fromjava/">
  <ns5:Policy xmlns:ns5="http://schemas.xmlsoap.org/ws/2004/09/policy"
wsu:Id="TmaxBO1TmaxIn1">
    <ns5:ExactlyOne>
      <ns5:All>
        <CallbackHandlerConfiguration
xmlns="http://schemas.sun.com/2006/03/wss/client">
          <CallbackHandler classname="xwss.saml.SamlCallbackHandler"
name="samlHandler" />
        </CallbackHandlerConfiguration>
      </ns5:All>
    </ns5:ExactlyOne>
  </ns5:Policy>
  ...
  <binding name="AddNumbersImplPortBinding" type="tns:AddNumbersImpl">
    ...
    <operation name="addNumbers">
      <soap:operation soapAction="" />
      <input>
        <ns10:PolicyReference
xmlns:ns10="http://schemas.xmlsoap.org/ws/2004/09/policy"
URI="#TmaxBO1TmaxIn1" />
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
  </binding>
  ...
</definitions>
```

### SAMLメッセージ処理のためのコールバック・ハンドラー・クラスの作成

SAML標準に従ってSOAP要求メッセージに情報を格納するためには、例では `javax.security.auth.callback.CallbackHandler` を実装する `SamlCallbackHandler` を使用します。`SamlCallbackHandler` では、コールバックの種類によって必要なアサーションを保存します。

以下は、この例で使用される `SamlCallbackHandler` です。

#### [例 17.23] << SamlCallbackHandler.java >>

```
public class SamlCallbackHandler implements CallbackHandler {
  ...
  public void handle(Callback[] callbacks) throws IOException,
```

```

        UnsupportedCallbackException {
    for (int i = 0; i < callbacks.length; i++) {
        if (callbacks[i] instanceof SAMLCallback) {
            try {
                SAMLCallback samlCallback = (SAMLCallback) callbacks[i];
                if (samlCallback.getConfirmationMethod().equals(
                    samlCallback.SV_ASSERTION_TYPE)) {
                    samlCallback.setAssertionElement(createSVSAMLAssertion());
                    svAssertion = samlCallback.getAssertionElement();
                } else if (samlCallback.getConfirmationMethod().equals(
                    samlCallback.HOK_ASSERTION_TYPE)) {
                    samlCallback.setAssertionElement(createHOKSAMLAssertion());
                    hokAssertion = samlCallback.getAssertionElement();
                } else {
                    throw new Exception("SAML Assertion Type is not matched.");
                }
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        } else {
            throw unsupported;
        }
    }
}

private static Element createSVSAMLAssertion() {...}
private static Element createSVSAMLAssertion20() {...}
private Element createHOKSAMLAssertion() {...}
private Element createHOKSAMLAssertion20() {...}
...
}

```

## 17.4.5. セキュリティー対話(Secure Conversation)

本節では、サーバーとクライアントでセキュリティー対話を設定する方法について説明します。

### 17.4.5.1. サーバー設定

Webサービスのサーバー設定は、service-config.xml設定ファイルで構成します。

#### [例 17.24] <<service-config.xml>>

```

<?xml version="1.0" encoding="UTF-8"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <policy>
        <endpoint-policy-subject>
            <portcomponent-wsdl-name>AddNumbersPort</portcomponent-wsdl-name>
            <addressing-policy>

```

```

        <www-w3-org />
    </addressing-policy>
    <security-policy>
        <security-binding>
            <symmetric-binding>
                <protection-token>
                    <secure-conversation-token>
                        <asymmetric-binding-initiator-token>
                            <include-token>true</include-token>
                        </asymmetric-binding-initiator-token>
                        <asymmetric-binding-recipient-token>
                            </asymmetric-binding-recipient-token>
                        <include-token>true</include-token>
                    </secure-conversation-token>
                </protection-token>
                <encrypt-signature>true</encrypt-signature>
            </symmetric-binding>
        </security-binding>
        <trust>true</trust>
        <keystore>...</keystore>
        <truststore>...</truststore>
    </security-policy>
    <operation-policy-subject>
        <operation-wsdl-name>addNumbers</operation-wsdl-name>
        <input-message-policy-subject>
            <security-policy>
                <protection>
                    <signed-part>...</signed-part>
                    <encrypted-part>...</encrypted-part>
                </protection>
            </security-policy>
        </input-message-policy-subject>
    </operation-policy-subject>
</endpoint-policy-subject>
</policy>
</web-services-config>

```

### 17.4.5.2. クライアント設定

クライアント側のWebサービスの設定は、追加でwsit-client.xmlにキーストア(Truststore)を設定します。

#### [例 17.25] <<wsit-client.xml>>

```

<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
...
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/">
    <message name="add" />
    <message name="addResponse" />
    <portType name="NewWebService">

```



```

        <wsdl:operation name="add">
            <wsdl:input message="tns:add" />
            <wsdl:output message="tns:addResponse" />
        </wsdl:operation>
    </portType>
    <binding name="NewWebServicePortBinding" type="tns:NewWebService">
        <wsp:PolicyReference URI="#NewWebServicePortBindingPolicy" />
        <soap12:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <wsdl:operation name="add">
            <soap12:operation soapAction="http://xmlsoap.org/Ping"
                style="document" />
            <wsdl:input>
                <soap12:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
    </binding>
    <service name="NewWebServiceService">
        <wsdl:port name="NewWebServicePort" binding="tns:NewWebServicePortBinding">
            <soap12:address location="REPLACE_WITH_ACTUAL_ADDRESS" />
        </wsdl:port>
    </service>
    <wsp:Policy wsu:Id="NewWebServicePortBindingPolicy"
        xmlns:scc="http://schemas.sun.com/ws/2006/05/sc/client">
        <wsp:ExactlyOne>
            <wsp:All>
                <sc:KeyStore wspp:visibility="private"
                    location="./keystore.jks" type="JKS"
                    alias="xws-security-client" storepass="changeit">
                </sc:KeyStore>
                <sc:TrustStore wspp:visibility="private"
                    location="./cacerts.jks" type="JKS"
                    storepass="changeit" peeralias="xws-security-server"
                    stsalias="wssip">
                </sc:TrustStore>
                <scc:SCClientConfiguration wspp:visibility="private">
                    <scc:LifeTime>36000</scc:LifeTime>
                </scc:SCClientConfiguration>
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>
</definitions>

```

## 17.4.6. Webサービスの信頼(WS-Trust)

本節では、Webサービスの信頼のためのサーバーとSTS、クライアントの設定方法について説明します。

### 17.4.6.1. サーバー設定

Webサービスのサーバー設定は、service-config.xml設定ファイルで構成します。

#### [例 17.26] <<service-config.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <policy>
    <endpoint-policy-subject>
<!--
Webサービスの信頼は、WS-Addressingフレームワーク上で動作します。
-->

      <addressing-policy>
        <www-w3-org />
      </addressing-policy>
      <security-policy>
<!--
Webサービスは、x509トークンをセキュリティー・トークンとして使用することを明示します。
-->

        <security-binding>
          <symmetric-binding>
            <protection-token>
              <x509-token />
            </protection-token>
            <layout>Strict</layout>
          </symmetric-binding>
        </security-binding>
<!--
Webサービスは、追加で以下のアドレスのSTSがセキュリティー・トークンを発給することを明示します。
-->

          <token>
            <endorsing-supporting-token>
              <issued-token>
                <issuer-address>
http://localhost:8088/trust_sts/SecurityTokenService
                </issuer-address>
              </issued-token>
            </endorsing-supporting-token>
          </token>
          <wss-version>11</wss-version>
          <trust>true</trust>
          ...
        </security-policy>
        ...
      </endpoint-policy-subject>
```

```
</policy>
</web-services-config>
```

### 17.4.6.2. STS設定

STSのためのWS-SecurityPolicy設定を、WSDLのWS-Policyフレームワークで行います。

以下は、WS-SecurityPolicy設定の例です。

#### [例 17.27] WS-SecurityPolicy設定 : <<sts.wSDL>>

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions ...>
  <wsp:Policy wsu:Id="TmaxSTSServerPolicy">
    <wsp:ExactlyOne>
      <wsp:All>
        <sp:SymmetricBinding
          xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
            <sp:ProtectionToken>
              <wsp:Policy>
                <sp:X509Token
                  sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/Never">

                  <wsp:Policy>
                    <sp:RequireDerivedKeys />
                    <sp:RequireThumbprintReference />
                    <sp:WssX509V3Token10 />
                  </wsp:Policy>
                </sp:X509Token>
              </wsp:Policy>
            </sp:ProtectionToken>
            <sp:AlgorithmSuite>
              <wsp:Policy>
                <sp:Basic128 />
              </wsp:Policy>
            </sp:AlgorithmSuite>
            <sp:Layout>
              <wsp:Policy>
                <sp:Lax />
              </wsp:Policy>
            </sp:Layout>
            <sp:IncludeTimestamp />
            <sp:EncryptSignature />
            <sp:OnlySignEntireHeadersAndBody />
          </wsp:Policy>
        </sp:SymmetricBinding>
        <sp:SignedSupportingTokens
          xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
          <wsp:Policy>
```

```

        <sp:UsernameToken
sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRecipient">

            <wsp:Policy>
                <sp:WssUsernameToken10 />
            </wsp:Policy>
        </sp:UsernameToken>
    </wsp:Policy>
</sp:SignedSupportingTokens>
<sp:Wss11
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
    <wsp:Policy>
        <sp:MustSupportRefKeyIdentifier />
        <sp:MustSupportRefIssuerSerial />
        <sp:MustSupportRefThumbprint />
        <sp:MustSupportRefEncryptedKey />
    </wsp:Policy>
</sp:Wss11>
<sp:Trust10
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
    <wsp:Policy>
        <sp:MustSupportIssuedTokens />
        <sp:RequireClientEntropy />
        <sp:RequireServerEntropy />
    </wsp:Policy>
</sp:Trust10>
...
<sc:ValidatorConfiguration
    xmlns:sc="http://schemas.sun.com/2006/03/wss/server">
    <sc:Validator name="usernameValidator"
        classname="trust.sts.UsernamePasswordValidator" />
</sc:ValidatorConfiguration>
<tc:STSConfiguration
    xmlns:tc="http://schemas.sun.com/ws/2006/05/trust/server"
    encryptIssuedKey="true" encryptIssuedToken="false">
    <tc:LifeTime>36000</tc:LifeTime>
    <tc:Contract>
        com.sun.xml.ws.security.trust.impl.IssueSamlTokenContractImpl
    </tc:Contract>
    <tc:Issuer>TmaxsoftSTS</tc:Issuer>
    <tc:ServiceProviders>

<tc:ServiceProvider endPoint="http://localhost:8088/trust_server/FinancialService">
        <tc:CertAlias>bob</tc:CertAlias>
        <tc:TokenType>

http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1
        </tc:TokenType>
    </tc:ServiceProvider>
    <tc:ServiceProvider endPoint="default">

```

```

        <tc:CertAlias>bob</tc:CertAlias>
        <tc:TokenType>

http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1
        </tc:TokenType>
        </tc:ServiceProvider>
    </tc:ServiceProviders>
    </tc:STSConfiguration>
    <wsap10:UsingAddressing />
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
...
<wsdl:binding name="ISecurityTokenService_Binding"
    type="tns:ISecurityTokenService">
    <wsp:PolicyReference URI="#TmaxSTSServerPolicy" />
    ...
</wsdl:binding>
...
</wsdl:definitions>

```

### 17.4.6.3. クライアント設定

クライアント側のWebサービスの設定は、wsit-client.xmlにサーバー、STSのためのキーストア(Truststore)を設定します。

#### [例 17.28] <<wsit-client.xml>>

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://tempuri.org/" ...>
    <!-- FinancialService WSDL -->
    <wsp:Policy wsu:Id="TmaxFSClientPolicy" ...>
        <wsp:ExactlyOne>
            <wsp:All>
                <sc:KeyStore ... />
                <sc:TrustStore ... />
                <scc:SCClientConfiguration wspp:visibility="private">
                    <scc:LifeTime>36000</scc:LifeTime>
                </scc:SCClientConfiguration>
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>
    ...
    <wsdl:binding name="IFinancialService_Binding"
        type="tns:IFinancialService">
        <wsp:PolicyReference URI="#TmaxFSClientPolicy" />
        ...
    </wsdl:binding>
    <wsdl:service name="FinancialService">
        <wsdl:port name="IFinancialService_Port"

```

```

        binding="tns:IFinancialService_Binding">
        <soap:address
            location="http://localhost:8088/trust_server/FinancialService" />
    </wsdl:port>
</wsdl:service>

<!-- STSService WSDL -->
<wsp:Policy wsu:Id="TmaxSTSCClientPolicy" ...>
    <wsp:ExactlyOne>
        <wsp:All>
            <sc:KeyStore ... />
            <sc:TrustStore ... />
            <sc:CallbackHandlerConfiguration
                xmlns:sc="http://schemas.sun.com/2006/03/wss/client">
                <sc:CallbackHandler default="alice" name="usernameHandler" />
                <sc:CallbackHandler default="alice" name="passwordHandler" />
            </sc:CallbackHandlerConfiguration>
        </wsp:All>
    </wsp:ExactlyOne>
</wsp:Policy>
...
<wsdl:binding name="ISecurityTokenService_Binding"
    type="tns:ISecurityTokenService">
    <wsp:PolicyReference URI="#TmaxSTSCClientPolicy" />
    ...
</wsdl:binding>
<wsdl:service name="SecurityTokenService">
    <wsdl:port name="ISecurityTokenService_Port"
        binding="tns:ISecurityTokenService_Binding">
        <soap:address
            location="http://localhost:8088/trust_sts/STSImpService" />
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>

```

## 17.4.7. 実行

以下は実行画面です。

```

<<__Exception__>>
com.sun.xml.ws.server.UnsupportedMediaException: Unsupported Content-Type:
application/soap+xml Supported ones are: [text/xml]
    at com.sun.xml.ws.encoding.StreamSOAPCodec.decode(StreamSOAPCodec.java:295)
    at com.sun.xml.ws.encoding.StreamSOAPCodec.decode(StreamSOAPCodec.java:129)
    at com.sun.xml.ws.encoding.SOAPBindingCodec.decode(SOAPBindingCodec.java:287)
    at jeus.webservices.jaxws.transport.http.HttpAdapter.decodePacket
(HttpAdapter.java:322)
    at jeus.webservices.jaxws.transport.http.HttpAdapter.access$3

```

```

(HttpAdapter.java:287)
    at jeus.webservices.jaxws.transport.http.HttpAdapter$HttpToolkit.handle
(HttpAdapter.java:517)
    at jeus.webservices.jaxws.transport.http.HttpAdapter.handle
(HttpAdapter.java:272)
    at jeus.webservices.jaxws.transport.http.EndpointAdapter.handle
(EndpointAdapter.java:149)
    at jeus.webservices.jaxws.transport.http.servlet.WSServletDelegate.doGet
(WSServletDelegate.java:139)
    at jeus.webservices.jaxws.transport.http.servlet.WSServletDelegate.doPost
(WSServletDelegate.java:170)
    at jeus.webservices.jaxws.transport.http.servlet.WSServlet.doPost
(WSServlet.java:23)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:725)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:818)
    at jeus.servlet.engine.ServletWrapper.executeServlet(ServletWrapper.java:328)
    at jeus.servlet.engine.ServletWrapper.execute(ServletWrapper.java:222)
    at jeus.servlet.engine.HttpRequestProcessor.run(HttpRequestProcessor.java:278)
<<__!Exception__>>
[2999.01.01 00:00:00][0][bxxx] [TMAX-xx] Unsupported Content-Type:
application/soap+xml Supported ones are: [text/xml]
<<__Exception__>>
com.sun.xml.ws.server.UnsupportedMediaException: Unsupported Content-Type:
application/soap+xml Supported ones are: [text/xml]
    at com.sun.xml.ws.encoding.StreamSOAPCodec.decode(StreamSOAPCodec.java:295)
    at com.sun.xml.ws.encoding.StreamSOAPCodec.decode(StreamSOAPCodec.java:129)
    at com.sun.xml.ws.encoding.SOAPBindingCodec.decode(SOAPBindingCodec.java:287)
    at jeus.webservices.jaxws.transport.http.HttpAdapter.decodePacket
(HttpAdapter.java:322)
    at jeus.webservices.jaxws.transport.http.HttpAdapter.access$3
(HttpAdapter.java:287)
    at jeus.webservices.jaxws.transport.http.HttpAdapter$HttpToolkit.handle
(HttpAdapter.java:517)
    at jeus.webservices.jaxws.transport.http.HttpAdapter.handle(HttpAdapter.java:272)
    at jeus.webservices.jaxws.transport.http.EndpointAdapter.handle
(EndpointAdapter.java:149)
    at jeus.webservices.jaxws.transport.http.servlet.WSServletDelegate.doGet
(WSServletDelegate.java:139)
    at jeus.webservices.jaxws.transport.http.servlet.WSServletDelegate.doPost
(WSServletDelegate.java:170)
    at jeus.webservices.jaxws.transport.http.servlet.WSServlet.doPost
(WSServlet.java:23)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:725)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:818)
    at jeus.servlet.engine.ServletWrapper.executeServlet(ServletWrapper.java:328)
    at jeus.servlet.engine.ServletWrapper.execute(ServletWrapper.java:222)
    at jeus.servlet.engine.HttpRequestProcessor.run(HttpRequestProcessor.java:278)
<<__!Exception__>>
user alice is a validate user.

```

```
your company : Tmaxsoft
your department : Infra
```

クライアントを実行すると、クライアントがSTSのURL情報を取得するために、多様なURLサフィックスで接続を試行します。ログ・レベルが「FINE」以上である場合、上記のような例外が発生します。

## 17.5. JAX-RPC(JEUS 5) Webサービスのセキュリティからのマイグレーション方法

JEUS Webサービスのメッセージ・レベルのセキュリティは、Webサービスのセキュリティ・ポリシーにより動作します。すなわち、Webサービスのセキュリティ・ポリシーをWSDL文書内で直接設定するか、またはwsit-endpoint.xmlに設定します。

JEUSでは、追加的にservice-config.xmlによるWebサービスのポリシー設定をサポートします。これは、Webサービスのセキュリティ・ポリシーをサポートします。そのため、ユーザーがWebサービス・ポリシーについて知らなくても、サーバー側とクライアント側の間でservice-config.xmlファイルの共有が可能であれば、そのファイルを利用して両末端のメッセージ・レベルのセキュリティを簡単に処理できます。これは、JAX-RPC (JEUS 5) Webサービスのメッセージ・レベルのセキュリティ方式と類似しています。

### 17.5.1. 暗号化(Encryption)

Webサービスのメッセージ・レベルのセキュリティでの暗号化の設定は、以下のように区分されます。

- JAX-RPC Webサービスの設定
- JAX-WS Webサービスの設定

### JAX-RPC Webサービスの設定

JAX-RPC Webサービスのメッセージ・レベルのセキュリティでは、以下のようにjeus-webservices-dd.xml (サーバー)とjeus-web-dd.xml(クライアント)を通じて暗号化を設定します。

**[例 17.29] << jeus-webservices-dd.xml >>**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-webservices-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <service>
    <webservice-description-name>
      PingSecurityService
    </webservice-description-name>
    <port>
      <port-component-name>PingPort</port-component-name>
      <security>
        <request-receiver>
          <action-list>Encrypt</action-list>
        </request-receiver>
      </security>
    </port>
  </service>
</jeus-webservices-dd>
```



```

        <password-callback-class>
            ping.PingPWCallback
        </password-callback-class>
    </decryption>
    <keystore>
        <key-type>jks</key-type>
        <keystore-password>changeit</keystore-password>
        <keystore-filename>
            server-keystore.jks
        </keystore-filename>
    </keystore>
</decryption>
</request-receiver>
<response-sender>
    <action-list>Encrypt</action-list>
    <password-callback-class>
        ping.PingPWCallback
    </password-callback-class>
    <encryption-infos>
        <encryption-info>
            <encryptionUser>xws-security-client</encryptionUser>
            <keyIdentifier>DirectReference</keyIdentifier>
            <keystore>
                <key-type>jks</key-type>
                <keystore-password>changeit</keystore-password>
                <keystore-filename>
                    server-truststore.jks
                </keystore-filename>
            </keystore>
        </encryption-info>
    </encryption-infos>
</response-sender>
</security>
</port>
</service>
</jeus-webservices-dd>

```

## JAX-WS Webサービスの設定

JAX-WS Webサービスのメッセージ・レベルのセキュリティは、Webサービスを作成 (wsгенまたはwsimportを通じてWebサービスを作成) する際に、service-config.xmlファイルを-policy引数と共にコマンド文に設定します。

### [例 17.30] << service-config.xml >>

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <policy>
        <endpoint-policy-subject>

```

```

<portcomponent-wsdl-name>PingPort</portcomponent-wsdl-name>
<security-policy>
  <security-binding>
    <asymmetric-binding>
      <initiator-token>
        <x509-token/>
      </initiator-token>
      <recipient-token>
        <x509-token/>
      </recipient-token>
    </asymmetric-binding>
  </security-binding>
  <keystore>
    <keystore-file>
      <alias>xws-security-client</alias>
      <key-type>JKS</key-type>
      <keystore-password>changeit</keystore-password>
      <keystore-filename>
        client-keystore.jks
      </keystore-filename>
    </keystore-file>
  </keystore>
  <truststore>
    <keystore-file>
      <alias>xws-security-server</alias>
      <key-type>JKS</key-type>
      <keystore-password>changeit</keystore-password>
      <keystore-filename>
        client-truststore.jks
      </keystore-filename>
    </keystore-file>
  </truststore>
</security-policy>
<operation-policy-subject>
  <operation-wsdl-name>ping</operation-wsdl-name>
  <input-message-policy-subject>
    <security-policy>
      <protection>
        <encrypted-part>
          <body/>
        </encrypted-part>
      </protection>
    </security-policy>
  </input-message-policy-subject>
  <output-message-policy-subject>
    <security-policy>
      <protection>
        <encrypted-part>
          <body/>
        </encrypted-part>
      </protection>
    </security-policy>
  </output-message-policy-subject>

```

```

        </protection>
    </security-policy>
    </output-message-policy-subject>
    </operation-policy-subject>
    </endpoint-policy-subject>
</policy>
</web-services-config>

```

## 17.5.2. 署名(Signature)

Webサービスのメッセージ・レベルのセキュリティーでの署名の設定は、以下のように区分されます。

- JAX-RPC Webサービスの設定
- JAX-WS Webサービスの設定

### JAX-RPC Webサービスの設定

JAX-RPC Webサービスのメッセージ・レベルのセキュリティーでは、以下のようにjeus-webservices-dd.xml (サーバー)またはjeus-web-dd.xml(クライアント)で署名を設定します。

#### [例 17.31] << jeus-webservices-dd.xml >>

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-webservices-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <service>
    <webservice-description-name>
      PingSecurityService
    </webservice-description-name>
    <port>
      <port-component-name>PingPort</port-component-name>
      <security>
        <request-receiver>
          <action-list>Signature</action-list>
          <password-callback-class>
            ping.PingPWCallback
          </password-callback-class>
          <signature-verification>
            <keystore>
              <key-type>jks</key-type>
              <keystore-password>changeit</keystore-password>
              <keystore-filename>
                server-truststore.jks
              </keystore-filename>
            </keystore>
          </signature-verification>
        </request-receiver>
      </security>
    </port>
  </service>
</jeus-webservices-dd>

```

```

        <response-sender>
            <action-list>Signature</action-list>
            <password-callback-class>
                ping.PingPWCallback
            </password-callback-class>
            <user>xws-security-server</user>
            <signature-infos>
                <signature-info>
                    <keyIdentifier>DirectReference</keyIdentifier>
                    <keystore>
                        <key-type>jks</key-type>
                        <keystore-password>changeit</keystore-password>
                        <keystore-filename>
                            server-keystore.jks
                        </keystore-filename>
                    </keystore>
                </signature-info>
            </signature-infos>
        </response-sender>
    </security>
</port>
</service>
</jeus-webservices-dd>

```

## JAX-WS Webサービスの設定

JAX-WS(JEUS 8) Webサービスのメッセージ・レベルのセキュリティーは、Webサービスを作成(wsgenまたはwsimportを通じてWebサービスを作成)する際に、service-config.xmlファイルを-policy引数と共にコマンド文に設定します。

### [例 17.32] << service-config.xml >>

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <policy>
        <endpoint-policy-subject>
            <portcomponent-wsdl-name>PingPort</portcomponent-wsdl-name>
            <security-policy>
                <security-binding>
                    <asymmetric-binding>
                        <initiator-token>
                            <x509-token/>
                        </initiator-token>
                        <recipient-token>
                            <x509-token/>
                        </recipient-token>
                    </asymmetric-binding>
                </security-binding>
                <keystore>

```

```

        <keystore-file>
            <alias>xws-security-client</alias>
            <key-type>JKS</key-type>
            <keystore-password>changeit</keystore-password>
            <keystore-filename>
                client-keystore.jks
            </keystore-filename>
        </keystore-file>
    </keystore>
    <truststore>
        <keystore-file>
            <alias>xws-security-server</alias>
            <key-type>JKS</key-type>
            <keystore-password>changeit</keystore-password>
            <keystore-filename>
                client-truststore.jks
            </keystore-filename>
        </keystore-file>
    </truststore>
</security-policy>
<operation-policy-subject>
    <operation-wsdl-name>ping</operation-wsdl-name>
    <input-message-policy-subject>
        <security-policy>
            <protection>
                <signed-part>
                    <body/>
                </signed-part>
            </protection>
        </security-policy>
    </input-message-policy-subject>
    <output-message-policy-subject>
        <security-policy>
            <protection>
                <signed-part>
                    <body/>
                </signed-part>
            </protection>
        </security-policy>
    </output-message-policy-subject>
</operation-policy-subject>
</endpoint-policy-subject>
</policy>
</web-services-config>

```

### 17.5.3. タイムスタンプ

Webサービスのメッセージ・レベルのセキュリティーでのタイムスタンプの設定は、以下のように区分されます。

- JAX-RPC Webサービスの設定
- JAX-WS Webサービスの設定

### JAX-RPC Webサービスの設定

JAX-RPC Webサービスのメッセージ・レベルのセキュリティーでは、以下のようにjeus-webservices-dd.xml (サーバー)またはjeus-web-dd.xml(クライアント)でタイムスタンプを設定します。

#### [例 17.33] << jeus-webservices-dd.xml >>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-webservices-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <service>
    <webservice-description-name>
      PingSecurityService
    </webservice-description-name>
    <port>
      <port-component-name>PingPort</port-component-name>
      <security>
        <request-receiver>
          <action-list>Signature Timestamp</action-list>
          <password-callback-class>
            ping.PingPWCallback
          </password-callback-class>
          <signature-verification>
            <keystore>
              <key-type>jks</key-type>
              <keystore-password>changeit</keystore-password>
              <keystore-filename>
                server-truststore.jks
              </keystore-filename>
            </keystore>
          </signature-verification>
        </request-receiver>
        <response-sender>
          <action-list>Signature</action-list>
          <password-callback-class>
            ping.PingPWCallback
          </password-callback-class>
          <user>xws-security-server</user>
          <signature-infos>
            <signature-info>
              <keyIdentifier>DirectReference</keyIdentifier>
            </signature-info>
          </signature-infos>
        </response-sender>
      </security>
    </port>
  </service>
</jeus-webservices-dd>
```

```

        <keystore>
            <key-type>jks</key-type>
            <keystore-password>changeit</keystore-password>
            <keystore-filename>
                server-keystore.jks
            </keystore-filename>
        </keystore>
    </signature-info>
</signature-infos>
</response-sender>
</security>
</port>
</service>
</jeus-webservices-dd>

```

## JAX-WS(JEUS 8) Webサービスの設定

JAX-WS(JEUS 8) Webサービスのメッセージ・レベルのセキュリティは、Webサービスを作成(wsgenまたはwsimportでWebサービスを作成)する際に、service-config.xmlファイルを-policy引数と共にコマンド文に設定します。

### [例 17.34] << service-config.xml >>

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <policy>
        <endpoint-policy-subject>
            <portcomponent-wsdl-name>PingPort</portcomponent-wsdl-name>
        <security-policy>
            <security-binding>
                <asymmetric-binding>
                    <initiator-token>
                        <x509-token/>
                    </initiator-token>
                    <recipient-token>
                        <x509-token/>
                    </recipient-token>
                </asymmetric-binding>
                <timestamp>true</timestamp>
            </security-binding>
            <keystore>
                <keystore-file>
                    <alias>xws-security-client</alias>
                    <key-type>JKS</key-type>
                    <keystore-password>changeit</keystore-password>
                    <keystore-filename>
                        client-keystore.jks
                    </keystore-filename>
                </keystore-file>
            </keystore>
        </security-policy>
    </policy>
</web-services-config>

```

```

        </keystore>
        <truststore>
            <keystore-file>
                <alias>xws-security-server</alias>
                <key-type>JKS</key-type>
                <keystore-password>changeit</keystore-password>
                <keystore-filename>
                    client-truststore.jks
                </keystore-filename>
            </keystore-file>
        </truststore>
    </security-policy>
    <operation-policy-subject>
        <operation-wsdl-name>ping</operation-wsdl-name>
        <input-message-policy-subject>
            <security-policy>
                <protection>
                    <signed-part>
                        <body/>
                    </signed-part>
                </protection>
            </security-policy>
        </input-message-policy-subject>
        <output-message-policy-subject>
            <security-policy>
                <protection>
                    <signed-part>
                        <body/>
                    </signed-part>
                </protection>
            </security-policy>
        </output-message-policy-subject>
    </operation-policy-subject>
</endpoint-policy-subject>
</policy>
</web-services-config>

```

## 17.5.4. ユーザー名トークン

Webサービスのメッセージ・レベルのセキュリティーでのユーザー名トークンの設定は、以下のように区分されます。

- JAX-RPC Webサービスの設定
- JAX-WS Webサービスの設定



## JAX-RPC Webサービスの設定

JAX-RPC Webサービスのメッセージ・レベルのセキュリティでは、以下のようにjeus-webservices-dd.xml (サーバー)またはjeus-web-dd.xml(クライアント)でユーザー名トークンを設定します。

### [例 17.35] << jeus-webservices-dd.xml >>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-webservices-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <service>
    <webservice-description-name>
      PingSecurityService
    </webservice-description-name>
    <port>
      <port-component-name>PingPort</port-component-name>
      <security>
        <request-receiver>
          <action-list>Signature UsernameToken</action-list>
          <password-callback-class>
            ping.PingPWCallback
          </password-callback-class>
          <signature-verification>
            <keystore>
              <key-type>jks</key-type>
              <keystore-password>changeit</keystore-password>
              <keystore-filename>
                server-truststore.jks
              </keystore-filename>
            </keystore>
          </signature-verification>
        </request-receiver>
        <response-sender>
          <action-list>Signature UsernameToken</action-list>
          <user>xws-security-server</user>
          <password-callback-class>
            ping.PingPWCallback
          </password-callback-class>
          <signature-infos>
            <signature-info>
              <keyIdentifier>DirectReference</keyIdentifier>
              <keystore>
                <key-type>jks</key-type>
                <keystore-password>changeit</keystore-password>
                <keystore-filename>
                  server-keystore.jks
                </keystore-filename>
              </keystore>
            </signature-info>
          </signature-infos>
        </response-sender>
      </security>
    </port>
  </service>
</jeus-webservices-dd>
```

```

        </security>
    </port>
</service>
</jeus-webservices-dd>

```

## JAX-WS(JEUS 8) Webサービスの設定

JAX-WS(JEUS 8) Webサービスのメッセージ・レベルのセキュリティは、Webサービスを作成(wsgenまたはwsimportを通じてWebサービスを作成)する際に、service-config.xmlファイルを-policy引数と共に設定します。

### [例 17.36] << service-config.xml >>

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <policy>
    <endpoint-policy-subject>
      <security-policy>
        <security-binding>
          <symmetric-binding>
            <protection-token>
              <x509-token>
                <include-token>true</include-token>
              </x509-token>
            </protection-token>
          </symmetric-binding>
        </security-binding>
      <token>
        <signed-supporting-token>
          <username-token>
            <username-password-validator>
              fromjava.server.UsernamePasswordValidator
            </username-password-validator>
            <include-token>true</include-token>
          </username-token>
        </signed-supporting-token>
      </token>
    <keystore>
      <keystore-file>
        <alias>xws-security-server</alias>
        <key-type>JKS</key-type>
        <keystore-password>changeit</keystore-password>
        <keystore-filename>
          keystore.jks
        </keystore-filename>
      </keystore-file>
    </keystore>
    <wss-version>1.1</wss-version>
    <protection>

```

```

        <signed-part>
            <body/>
        </signed-part>
        <encrypted-part>
            <body/>
        </encrypted-part>
    </protection>
</security-policy>
</endpoint-policy-subject>
</policy>
</web-services-config>

```

## 17.6. アクセス制御が設定されているWebサービスの呼び出し

JEUS Webサービスは、アクセスが許容されているユーザーのみがWebサービスを呼び出せるようにアクセス制御を設定でき、Webサービスのバックエンドによってそれぞれ設定方法が異なります。JEUS Webサービスのアクセス制御の設定方法は「[27.4. アクセス制御の設定](#)」と同じで、クライアントの呼び出し方法のみ異なります。

アクセス制御(Basic Authentication)が設定されているWebサービスを呼び出すには、WSDLにアクセスしてその内容をベースにしてポータブル・アーティファクトを作成し、それを使用してWebサービスを呼び出します。この場合、ユーザー名とパスワードの入力を要求する場合に備えて、設定を行う必要があります。

### 17.6.1. ポータブル・アーティファクトの作成

アクセス制御が設定されているWebサービスのクライアントのためのポータブル・アーティファクトを作成します。

**wsimport**ツールを使用してポータブル・アーティファクトを作成するためには、コンソールに以下のように入力します。

```

$ wsimport -Xauthfile authorization.txt
http://localhost:8088/AddNumbers/AddNumbersImplService?wsdl

```

authorization.txtファイルは、WSDLにアクセスするための権限設定ファイルです。以下は、権限設定ファイルの例です。

**[例 17.37] << authorization.txt >>**

```

http://jeus:jeus@localhost:8088/AddNumbers/AddNumbersImplService?wsdl

```

上記例では、ユーザー名は「jeus」、パスワードは「jeus」です。

## 17.6.2. Webサービスのクライアントの作成

JAX-WS Webサービスのクライアントが自ら認証を行うには、以下の2つの設定がクライアント・プログラム内に挿入されている必要があります。

- javax.xml.ws.BindingProvider.USERNAME\_PROPERTY
- javax.xml.ws.BindingProvider.PASSWORD\_PROPERTY

以下の例は、実際にプログラムで上記設定を実装した例です。

```
Echo port = // ... SEI(Service Endpoint Interface)の取得
((javax.xml.ws.BindingProvider) port).getRequestContext().
    put(javax.xml.ws.BindingProvider.USERNAME_PROPERTY, "jeus");
((javax.xml.ws.BindingProvider) port).getRequestContext().
    put(javax.xml.ws.BindingProvider.PASSWORD_PROPERTY, "jeus");
String s = port.echoString("JEUS");
```

ランタイム時にJAX-WS Webサービスのクライアント・エンジンは動的スタブを作成するため、一度リモートWSDLにアクセスしますが、この際付加的な権限設定が必要です。その権限設定はjava.net.Authenticatorクラスを使用してクライアント・プログラムにインストールします。

以下は、java.net.Authenticatorクラスを使用したクライアント・プログラムの例です。

### [例 17.38] << AddNumberClient.java >>

```
public class AddNumbersClient {
    ...
    public void execute() {
        Authenticator.setDefault(new MyAuthenticator());

        AddNumbersImpl port = new AddNumbersImplService().getAddNumbersImplPort();
        ((javax.xml.ws.BindingProvider) port).getRequestContext().
            put(javax.xml.ws.BindingProvider.USERNAME_PROPERTY, "jeus");
        ((javax.xml.ws.BindingProvider) port).getRequestContext().
            put(javax.xml.ws.BindingProvider.PASSWORD_PROPERTY, "jeus");

        int number1 = 10;
        int number2 = 20;

        port.addNumbers(number1, number2);
        ...
    }

    class MyAuthenticator extends Authenticator {
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication("jeus", "jeus".toCharArray());
        }
    }
}
```

# 第18章 Server-Sent Event

本章では、JEUSでHTML5のServer-Sent Event (SSE)を使用する方法について説明します。

## 18.1. 概要

Server-Sent Eventsは、サーバーが一方方向のクライアント、サーバー接続により、標準HTTPまたはHTTPSを使用してWebページにデータをプッシュできるようにします。Server-Sent Events通信モデルで、クライアント(例: ブラウザー)は初期接続を維持し、サーバーはデータを提供してクライアントに送信します。

Server-Sent EventsはHTML5仕様の一部として提案された標準技術であり、JEUSではJAX-RS(Java API for RESTful Web Services) 2.0によりServer-Sent Eventをサポートします。

## 18.2. JAX-RSリソースのServer-Sent Events (SSE)のサポート

### 18.2.1. 簡単なSSEリソース・メソッド

SSEをサポートするために、SseFeatureリソースを追加します。

#### [例 18.1] 簡単なSSEリソース・メソッド

```
...
import jeus.webservices.jaxrs.media.sse.EventOutput;
import jeus.webservices.jaxrs.media.sse.OutboundEvent;
import jeus.webservices.jaxrs.media.sse.SseFeature;
...

@Path("events")
public class SseResource {

    @GET
    @Produces(SseFeature.SERVER_SENT_EVENTS)
    public EventOutput getServerSentEvents() {
        EventOutput eventOutput = new EventOutput();
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    for (int i = 0; i < 5; i++) {
                        // ... code that waits a few seconds
                        OutboundEvent.Builder eventBuilder = new OutboundEvent.Builder();
```

```

        eventBuilder.name("example");
        eventBuilder.data(String.class, "Hello world " + i + "!");
        OutboundEvent event = eventBuilder.build();
        eventOutput.write(event);
    }
} catch (IOException e) {
    throw new RuntimeException(e);
} finally {
    try {
        eventOutput.close();
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
}
}).start();
return eventOutput;
}
}

```

上記のコードは、URI「/events」にデプロイされたリソースを定義します。リソースはチャンク出力メッセージを処理するためのEventOutputエンティティを返す1つの@GETリソース・メソッドを保持しています。

EventOutputがメソッドから返された後、JAX-RSランタイムはEventOutputをChunkedOutputとして認知し、クライアント接続を直ちに切断せずに応答ストリームにHTTPヘッダーを書き込み、転送したチャンク(SSEイベント)を待ちます。この時点にクライアントはヘッダーを読み込んで個別イベントのリスンを始めます。

サンプル・コードでリソース・メソッドは一連の5つのイベント転送のためにスレッドを作成しています。連続するイベントの間には若干の遅延があります。それぞれのイベントはOutboundEventタイプで表現されます。OutboundEventはSSEメッセージの標準形式を反映し、name、comment、idを表すプロパティを含みます。data(Class)メソッドはイベント・データを直列化するために使用されます。

SSEサポート・リソースに接続したクライアントは、エンティティ・ストリームから次のデータを受信します。

```

event: example
data: Hello world 0!

event: example
data: Hello world 1!

event: example
data: Hello world 2!

event: example
data: Hello world 3!

event: example
data: Hello world 4!

```

## 18.3. JAX-RSクライアントでのSSEイベント処理

JEUS JAX-RSは2つのプログラミング・モデルを使用してSSEイベントの受信と処理をサポートするクライアントAPIを提供します。

- Pullモデル：EventInputからイベントを引き出すモデル
- Pushモデル：EventSourceの非同期通知をリスンするモデル

### 18.3.1. EventInputを使用してSSEイベントを読み込む

クライアントでイベントはEventInputから読み込むことができます。以下のコードを参照してください。

#### [例 18.2] EventInputを使用してSSEイベントを読み込む

```
Client client = ClientBuilder.newBuilder().build();
WebTarget target = client.target("http://localhost:8088/sse_example/events");
EventInput eventInput = target.request().get(EventInput.class);
while (!eventInput.isClosed()) {
    final InboundEvent inboundEvent = eventInput.read();
    if (inboundEvent == null) {
        // connection has been closed
        break;
    }
    System.out.println(inboundEvent.getName() + ": " +
inboundEvent.readData(String.class));
}
```

クライアントは「簡単なSSEリソース・メソッド」のサンプルのSseResourceがデプロイされたサーバーに接続します。まずJAX-RS clientオブジェクトを作成した後、clientオブジェクトからWebTargetオブジェクトを取得してHTTPリクエストの呼び出しに使用します。返された応答エンティティは、EventInputから直接読み込むことができます。サンプル・コードでは、eventInput応答ストリームからインバウンドSSEイベントを読み込むためにループを開始します。eventInputから読み込んだ個別のチャンクはInboundEventです。InboundEvent.readData(Class)メソッドはイベント・データの逆直列化のために使用するメソッドです。

クライアント・コードは、以下の結果を表示します。

```
example: Hello world 0!
example: Hello world 1!
example: Hello world 2!
example: Hello world 3!
example: Hello world 4!
```

## 18.3.2. EventSourceを使用した非同期SSEの処理

非同期でSSEイベントを読み込むために使用するクライアントAPIはEventSourceです。EventSourceの使用方法は、以下の例を参照してください。

### [例 18.3] EventSourceを使用してSSEイベントを読み込む

```
Client client = ClientBuilder.newBuilder().build();
WebTarget target = client.target("http://localhost:8088/sse_example/events");
EventSource eventSource = new EventSource(target);
EventListener listener = new EventListener() {
    @Override
    public void onEvent(InboundEvent inboundEvent) {
        System.out.println(inboundEvent.getName() + ": " +
inboundEvent.readData(String.class));
    }
};
eventSource.addEventListener("example", listener);
eventSource.open();
...
eventSource.close();
```

クライアントは「簡単なSSEリソース・メソッド」のサンプルのSseResourceがデプロイされたサーバーに接続します。まずJAX-RS clientオブジェクトを作成した後、clientオブジェクトからWebTargetオブジェクトを取得します。Webターゲットに対するHTTPリクエストの呼び出しは、WebTargetオブジェクトで直接作成しません。EventSourceオブジェクトをtargetオブジェクトに初期化して作成します。作成されたEventSourceオブジェクトは自動でターゲットに接続しないため、後でeventSource.open()メソッドを使って手動で接続を行います。

EventListener実装は、インバウンドSSEイベントをリスンして処理するために使用します。InboundEvent.readData(Class)メソッドがinboundEventオブジェクトからイベント・データを逆直列化するために使用します。

サンプルからリスナーは以下の結果を表示します。

```
example: Hello world 0!
example: Hello world 1!
example: Hello world 2!
example: Hello world 3!
example: Hello world 4!
```



# 第19章 UDDIの使用

本章では、UDDIについての基本概念と、JEUSでのUDDIの使用方法について説明します。また、UDDIレジストリーをプログラマ的にアクセスできるUDDIクライアント、およびUDDIレジストリーに簡単にアクセスするためのWebベースのユーザー・インターフェースについても説明します。

## 19.1. 概要

UDDI(Universal Description, Discovery and Integration)仕様は、Webサービスに関する情報の公開と検索の方法を定義します。UDDIは、XML(Extensible Markup Language)とSOAP(Simple Object Access Protocol)のような既存の標準に基づいています。

UDDIは、一般的なXML形式で実装されたビジネスと、そのサービス技術(description)の分散レジストリーに基づくアクセス方法をとります。UDDIプロジェクトのコア・コンポーネントは、UDDIビジネスの登録とビジネス・エンティティを記述したXMLファイルとそのWebサービスです。

UDDIは、プログラミング・モデルとスキーマを提供し、これはレジストリーを使用して規則を定義します。UDDI仕様のすべてのAPIはXMLで定義されており、SOAPエンベロープでラッピングされ、HTTPを通じて送信されます。

### 19.1.1. UDDIの使用

UDDIは、Webでビジネス・レジストリーの共有オペレーションを含んでいます。大半の場合、プログラムとプログラマーはサービス情報を位置させる際、特にプログラマーの場合は、知らされているWebサービスとの互換性があるシステムを準備したり、自身のWebサービスを他の人が呼び出すことができるようにしたり記述するのにUDDIビジネス・レジストリーを使用します。これは、UDDIビジネス・レジストリーに指定された情報を使用します。

UDDIビジネス・レジストリーはビジネス・レベルで使用され、与えられたパートナーが特定のWebサービス・インターフェースを持っているかを確認するか、該当サービスで該当事業にある会社を探すか、サービスに関する技術情報を取得するためにパートナーがどのようにWebサービスをエクスポートさせたのかについての情報を位置させます。

## UDDIデータの構造

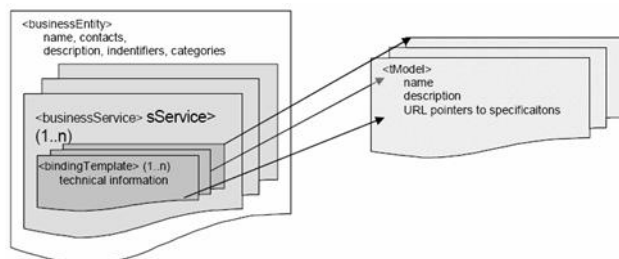
UDDIレジストリーで使用するコア情報モデルはXMLスキーマと定義されます。

XMLスキーマは4つのコア・タイプの情報で構成されており、技術者がパートナーのWebサービスを使用するために知っておくべき情報を提供します。この4つは、サービスの仕様に関するビジネス情報

(<businessEntity>)、サービス情報(<businessService>)、バインディング情報(<bindingTemplate>)、技術モデル情報(<tModel>)です。

以下の図は、Webサービスに関する情報を記述あるいは検索する際に使用される情報の階層構造とXMLタグです。

**[図 19.1] 情報階層と主要XMLタグの名前**



- ビジネス・エンティティ(businessEntity)

Webサービスを提供するビジネスあるいは機関を記述します。このエンティティはビジネスの基本情報（名前、実行しているビジネスの説明、連絡先）を示します。Dun & Bradstreet D-U-N-S® Numberのようなビジネス識別子も含まれています。

- ビジネス・サービス(businessService)

ビジネス・エンティティによって記述された機関が提供する関連Webサービスのグループを記述します。これは1つのビジネス・エンティティの論理子です。

- バインディング・テンプレート(bindingTemplate)

特定のWebサービスを利用するのに必要な技術モデルを記述します。このエンティティは、サービスに関連するバインディング情報と、そういったサービスが実装した技術仕様に関するリファレンスを提供し、多様なファイルとURLベースの発見メカニズムに対するポインターも提供します。

- 技術モデル(tModel)

Webサービスのタイプ、Webサービスで使用するプロトコル、分類システムといった、再使用可能な概念を表す技術モデルを記述します。この技術モデルは、ビジネス間で送受信する標準、Webサービス・インターフェースの定義のようなすべての種類のものを含みます。したがって、新しいWebサービスを構成してWSDL文書を記述した場合、このWSDLの定義は技術モデルに保存されます。

## 19.2. JEUSにおけるUDDIサーバーの運用

JEUSのUDDIは、UDDIレジストリー・サーバーとUDDIクライアント・ライブラリー、Webベースのユーザー・インターフェースから構成されています。JEUSのUDDIは、UDDIバージョン2.0と3.0のデータ構造とプログラミングAPIをサポートします。

本節では、JEUSにUDDIモジュールをデプロイする方法とJEUS UDDIサーバーの起動について説明します。

- JEUS UDDIデータストアを作成する方法
- JEUS UDDIサーバーをデプロイする方法
- JEUS UDDIサーバーを構成し、設定する方法

JEUS UDDIパッケージのコア・ライブラリーは、JEUS\_HOME/lib/systemディレクトリーに位置し、Webインターフェースを含むWebコンテキストに関連するモジュールはEARフォーマットで提供されます。

### 19.2.1. UDDIデータストアの作成

JEUSのUDDIサーバーは、データの永続性のためにリレーショナル・データベースが必要です。JEUS UDDIサーバーはANSI標準SQLをサポートする、いかなるリレーショナル・データベースであっても使用できます。

JEUS UDDIデータストアはJDBCを使用して設定します。

まず、JEUS UDDIサーバー・パッケージと組み合わせて提供されるデータベース・スクリプトを使用して、新しいUDDIデータベースを作成します（データベース・スクリプトは、JEUS\_HOME/lib/systemapps/uddi/dbscriptsディレクトリーに位置します）。データストアを構成するには、JEUSで「uddiDB」というデータソースを設定する必要があります。

JEUS\_HOME/lib/systemapps/uddi/confディレクトリーのサンプル・ファイルを参照してください。

### 19.2.2. UDDIサーバーのデプロイ

UDDIデータベースの構成設定が完了すれば、JEUS UDDIサーバーをデプロイします。

JEUS UDDIサーバーのEARパッケージは、JEUSがインストールされる際にJEUS\_HOME/lib/systemapps/uddiディレクトリーに作成されます。（ファイル名はjeusuddi\_v2c.earとjeusuddi\_v3c.earです）

JEUS UDDIサーバーのデプロイは、他のEARアプリケーションをデプロイする方法と同じです。例では、名前がuddiのコンテキストにUDDIがデプロイされると仮定しています。

#### [例 19.1] <<jeus-web-dd.xml>>

```
<jeus-web-dd>
  <res-ref>
    <jndi-info>
      <ref-name>jdbc/uddiDB</ref-name>
      <export-name>UDDIDB</export-name>
    </jndi-info>
  </res-ref>
</jeus-web-dd>
```

```

    </res-ref>
</jeus-web-dd>

```

#### [例 19.2] <<web.xml>>

```

<web-app>
    . . .
    <resource-ref>
        <res-ref-name>jdbc/uddiDB</res-ref-name>
        . . .
    </resource-ref>
</web-app>

```

JNDIデータソース名の「jdbc/uddiDB」は、デフォルト値として使用されるJNDIデータソース名です。サーバーに設定されている実際のデータソースと接続する<export-name>は「UDDIDB」です。他のJNDIデータソース名を使用する場合、この構成設定ファイルを修正します。

### 19.2.3. UDDIサーバーの構成設定

JEUS UDDIサーバーを構成するには、設定ファイルの**uddi.properties**を修正します。ファイルはJEUS\_HOME/lib/applicationディレクトリに位置します。この設定ファイルが存在しない場合、デプロイ時に基本設定ファイルを自動作成します。

以下は、uddi.propertiesファイルの基本設定に関する説明です。

[表 19.1] uddi.properties

UDDIプロパティ・キー	説明
uddi.operatorName	UDDIオペレータ名です
uddi.operatorURL	UDDIオペレータ・サイトのURLです
uddi.auth	現在使用中のUDDI認証モジュールです (デフォルト値: 'jeus.webservices.uddi.auth.DefaultAuthenticator')
uddi.dataSource	現在使用中のUDDIデータストア・モジュールです。 (デフォルト値: 'java:comp/env/jdbc/uddiDB')

### 19.2.4. 新しいユーザーの追加

UDDIレジストリーにUDDIデータを公開するには、UDDIレジストリーの認証を取得する必要があります。

JEUS UDDIレジストリーのPublisherの認証は2段階に分かれます。

#### 1. ユーザーID/パスワードを使用してユーザーを認証

JEUS UDDIレジストリーは、基本認証モジュールの「jeus.webservices.uddi.auth.DefaultAuthenticator」を持っています。基本認証モジュールは、認証に対して簡単に承認します。これは、UDDIユーザーのuseridを認証するだけです。

JEUS UDDIレジストリーは、追加認証モジュールの「jeus.webservices.uddi.auth.XMLDocAuthenticator」を提供します。これはXML文書に基づいて、「JEUS\_HOME/lib/application」にある「uddi-user.xml」を追加あるいは修正する必要があります。

以下は、XML文書の例です。

```
<?xml version="1.0" encoding="UTF-8"?>
<uddi-users>
  <user userid="administrator" password="jeusadmin" />
  <user userid="tmaxsoft" password="password" />
  <user userid="jeus" password="password" />
  <user userid="webservices" password="password" />
</uddi-users>
```

開発者は、Custom認証モジュールを開発できます。これは典型的にUDDIレジストリーは外部認証メカニズムを使用できることを意味します。追加的なJEUS UDDI認証モジュールは、JEUS UDDIユーザーが特定環境でどのように認証されるのか決定するのに従い、JEUS UDDIユーザーによって開発されます。

以下のコードは、Custom JEUS UDDI認証モジュールを開発する方法についての例です。

```
import jeus.webservices.uddi.auth.BaseAuthenticator;
import org.apache.juddi.error.RegistryException;

public class SampleAuthenticator implements BaseAuthenticator {

    public SampleAuthenticator() {
    }

    public String authenticate(String userID, String credential)
        throws RegistryException {
        // TODO Part implemented by developer.
        return userID;
    }
}
```

## 2. 認証されたユーザーをUDDIレジストリー・ユーザーとして登録

登録方法には、UDDI Webユーザー・インターフェースを使用する方法と、データストアに直接登録する方法があります。本節では、データストアに直接登録する方法のみを説明し、UDDI Webインターフェースを使用する方法「[19.3. JEUSサーバーでのUDDIエクスプローラの使用](#)」を参照してください。

UDDIレジストリーのユーザーのPublisherがUDDIデータストアに定義されているかを確認します。Publisherは、UDDI DBのPUBLISHER表にPublisherを識別する行がある場合に定義されます。SQLを使用してユーザーを定義できます。

以下は、「jeus」というPublisherを定義する例です。

```
INSERT INTO PUBLISHER (PUBLISHER_ID,PUBLISHER_NAME,ADMIN)
VALUES ('jeus','JEUS','false');
```

**[表 19.2] PUBLISHER表の列**

列名	説明
PUBLISHER_ID	PublisherのユーザーIDは認証時に使用されます。外部の認証サービスを通じて認証する際に使用される値と同じである必要があります
PUBLISHER_NAME	Publisherの名前(またはUDDI内で検証された名前)です
ADMIN	Publisherが管理権限を持っているか否かを示します。この列の値はtrueまたはfalseです

## 19.2.5. UDDIサーバーの実行

JEUSサーバーを起動します。UDDIサーバーのURLは以下のとおりです。

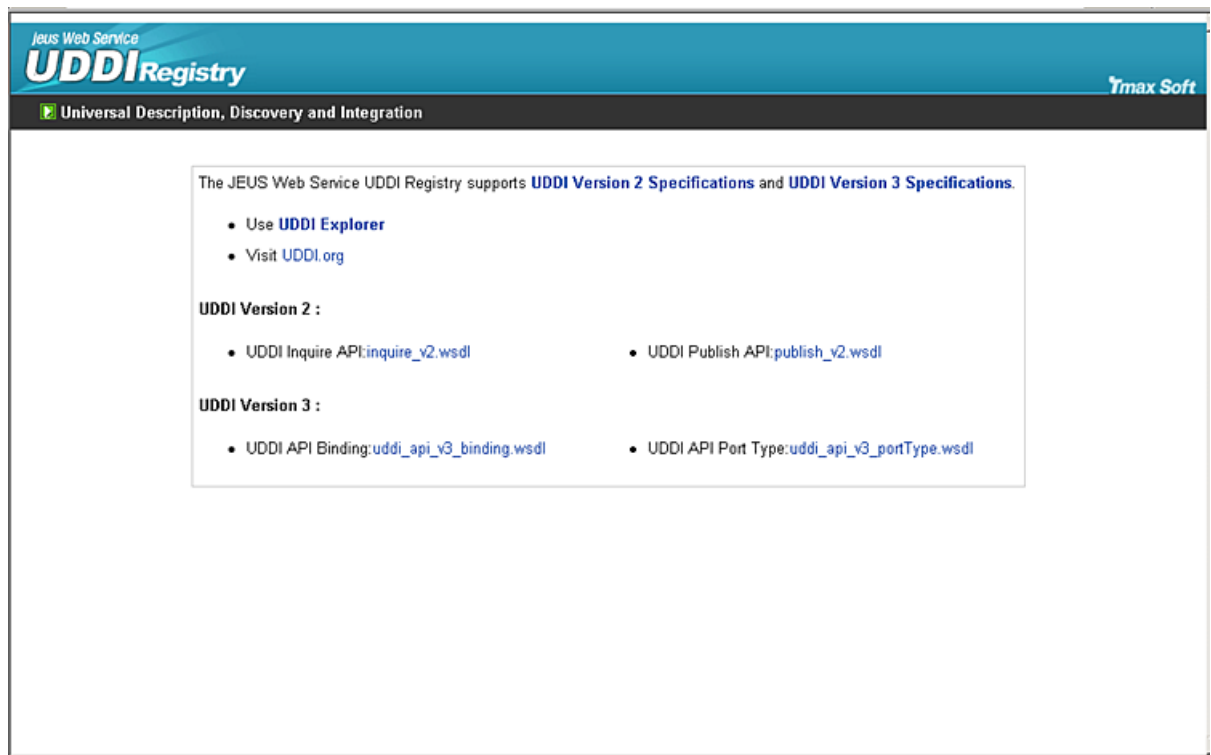
```
http://hostname:port/context/
```

基本的に、JEUS UDDIサーバーのコンテキスト・パスは「/uddi」です。JEUS HTTP Listenerのデフォルトのポート番号の8088を使用すると仮定した場合、URLは以下のとおりです。

```
http://localhost:8088/uddi/
```

WebブラウザでこのURLを呼び出すと、以下のページに移動します。

[図 19.2] JEUS UDDIレジストリーの初期画面



## 19.3. JEUSサーバーでのUDDIエクスプローラの使用

JEUS UDDIエクスプローラは、UDDIレジストリーのUDDIデータの登録、修正、クエリ、削除などを行うためのWebアプリケーションです。

JEUS UDDIエクスプローラは、以下の機能を提供します。

- Private UDDIレジストリーのクエリ
- Private UDDIレジストリーの公開(Publish)
- JEUS UDDIエクスプローラの設定

本節では、JEUS UDDIエクスプローラのWebユーザー・インターフェースの使用方法について説明します。

## 19.3.1. UDDIレジストリーのクエリ

JEUS UDDIエクスプローラの「Search」を利用すると、ビジネス、サービス、技術モデルを検索できます。

「**UDDI Registry**」を使用してUDDIレジストリー・オペレーターを選択でき、「**Search For a**」でEntityタイプを選択できます。UDDI RegistryとEntityタイプ(Business、Service、Technical Model)を選択後、検索する名前の一部あるいは全体を以下のテキスト・フィールドに入力します。ワイルドカードとして「%」を使用できます。また、「**exact name**」あるいは「**case sensitive**」を使用して結果をフィルタリングできます。

[図 19.3] UDDI検索画面

The screenshot shows the JEUS UDDI Search interface. The header includes the JEUS logo and 'Webservice UDDI' text, along with a user login status. The navigation bar has 'Home', 'Search', and 'Publish' options. The search area is split into two panels. The left panel allows selecting the UDDI Registry from a list of operator sites, currently showing 'Private'. The right panel is for searching by Business, Service, or TModel. The 'TModel' tab is active, with 'uddi' entered in the search field. There are checkboxes for 'exact name' and 'case sensitive' search, and a 'submit' button to execute the search.

検索する名前を入力後、**[submit]**ボタンを押すと検索が開始されます。選択されたUDDIレジストリーは、ユーザーが入力した名前で始まるビジネス、サービス、技術モデルの一覧を表示します。



JEUS UDDIエクスプローラは、検索したエントリーの詳細を以下のように出力します。

[図 19.4] UDDIレジストリーで検索した結果画面

JEUS / Webservice UDDI

Username: guest [login](#)

HomeSearchPublish

Search Result for a

TModel:

[uddi-org:UTS-10](#) - [http://uddi.org/pubs/uddi\\_v3.htm#UCASort](#)  
[description: UDDI Unicode Technical Standard #10 sort collation sequence find qualifier]

[uddi-org:andAllKeys](#) - [http://uddi.org/pubs/uddi\\_v3.htm#orAll](#)  
[description: UDDI find qualifier used to request that a logical OR be performed on bag contents prior to a search]

[uddi-org:andAllKeys](#) - [http://uddi.org/pubs/uddi\\_v3.htm#andAll](#)  
[description: UDDI find qualifier used to request that a logical AND be performed on bag contents prior to a search]

[uddi-org:approximateMatch:SQL99](#) - [http://uddi.org/pubs/uddi\\_v3.htm#wildcard](#)  
[description: UDDI approximate matching find qualifier]

[uddi-org:binarySort](#) - [http://uddi.org/pubs/uddi\\_v3.htm#sortOrd](#)  
[description: UDDI binary sort sortOrder qualifier]

[uddi-org:bindingSubset](#) - [http://uddi.org/pubs/uddi\\_v3.htm#bindSubset](#)  
[description: UDDI find qualifier for specifying use of categoryBags of bindingTemplate elements to satisfy the find\_business or find\_service inquiries.]

[uddi-org:caseInsensitiveMatch](#) - [http://uddi.org/pubs/uddi\\_v3.htm#caseinsens](#)  
[description: UDDI case insensitive matching find qualifier]

[uddi-org:caseInsensitiveSort](#) - [http://uddi.org/pubs/uddi\\_v3.htm#caseInsensSort](#)  
[description: UDDI sort qualifier used to sort results without regard to case]

[uddi-org:caseSensitiveMatch](#) -

Search for a

Business/Service/TModel:

BusinessServiceTModel

TModel Name:

uddi

exact name ☐

case sensitive ☐

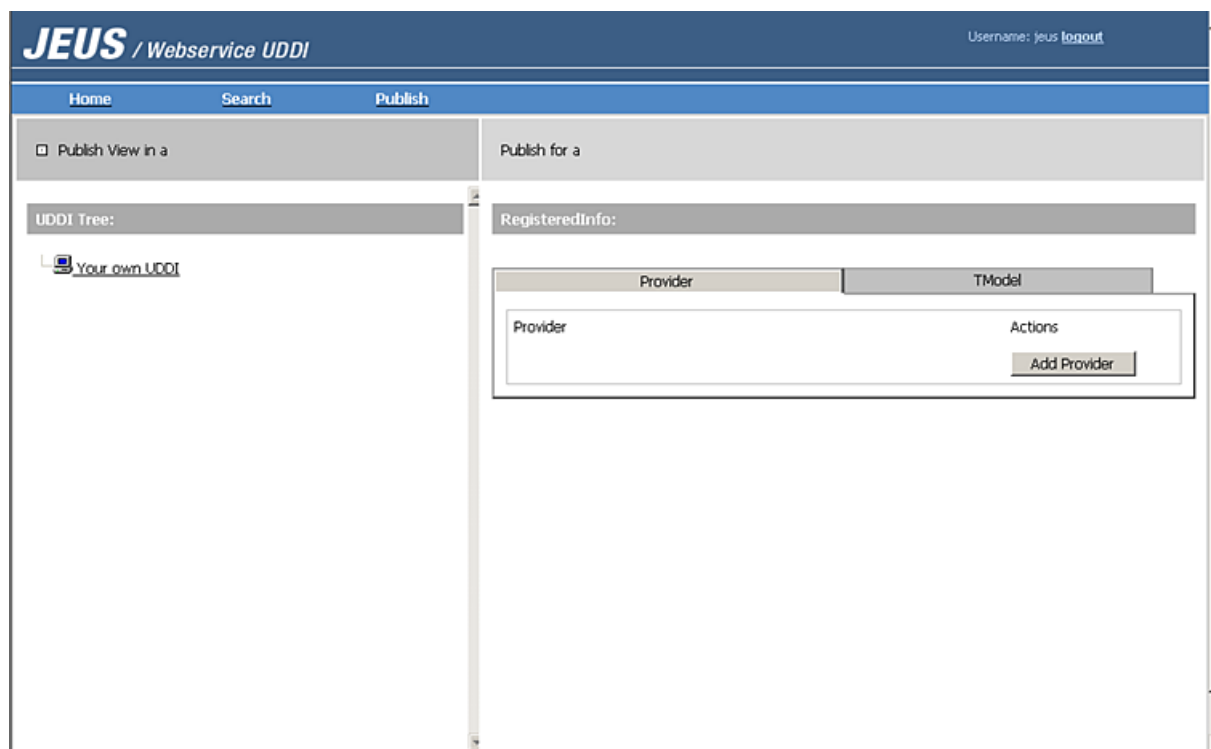
submit

## 19.3.2. UDDIレジストリーの公開

UDDIレジストリーに公開するためには、ユーザーIDとパスワードを入力してUDDIレジストリーにログインします。ユーザーが接続後、JEUS UDDIエクスプローラはユーザーが登録したビジネス一覧と技術モデルが表示されます。登録したビジネスあるいは技術モデルは、画面左側のツリー部分で[+]をクリックすると、登録されているサービスおよびその他の情報を確認できます。また、右側の画面では、これらについての詳細情報の閲覧および修正が可能です。

[Add Provider]、[Add TModel]ボタンを使用して、基本的なビジネスや技術モデルを追加できます。登録されたレジストリーを削除あるいは修正する場合は、[Delete]、[Edit]ボタンを使用します。

[図 19.5] 登録したエン트리画面



## ビジネスの追加

以下は、ビジネスを追加する手順についての説明です。

1. **RegisteredInfo**画面で[**Add Provider**]ボタンをクリックすると、**Provider**画面に移動して、デフォルト値の「New Provider」という名前を持つビジネスが追加されたことが確認できます。ユーザーは、この画面の[**Edit**]ボタンなどを使用して、各種情報を修正できます。「**Name**」項目に「JEUS Web Services」と入力し、[**Update**]ボタンをクリックします。
2. **Provider**画面の[**Detail**]、[**Contact**]、[**Identifier**]、[**Category**]、[**Discovery URL**]タブでは、ビジネスの説明やユーザーの連絡先、Discovery URL、カテゴリ、そして識別子などの付加情報を追加できます。「**Descriptions**」項目には「eBiz using Webservices」と入力します。

[図 19.6] プロバイダーの修正画面

JEUS / Webservice UDDI

Username: jeus [logout](#)

Home Search Publish

☐ Publish View in a

UDDI Tree:

[Your own UDDI \(1\)](#)

Publish for a

Provider:

Detail Service Contact Identifier Category Discovery URL Relationship

Provider Key:  
uddi:57E6F930-A1E1-11DB-B930-EA48E268BB77

Name  
(en) New Provider

Actions  
[Edit](#) [Delete](#)

Language:  
English

[Update](#) [Cancel](#)

Name:  
JEUS Web Services

[Add Name](#)

Descriptions

Actions  
[Add Desc](#)

Operational Information  
Owner: jeus  
Created: Fri Jan 12 11:05:15 GMT+09:00 2007  
Modified: Fri Jan 12 11:05:15 GMT+09:00 2007

## サービスの追加

以下は、サービスを追加する手順についての説明です。

1. **Provider**画面の[**Service**]タブで[**Add Service**]ボタンをクリックします。サービスは常にビジネスに関連しているため、クリックするとこのビジネスに関連するサービスが新しく作成され、**Service**画面に移動します。
2. **Service**画面でも、**Provider**画面と同様に、各タブを通じて各種情報を修正または表示できます。

[図 19.7] サービスの追加画面

The screenshot displays the JEUS / Webservice UDDI web application. The top navigation bar includes 'Home', 'Search', and 'Publish' tabs. The 'Publish' tab is active. On the left, the 'UDDI Tree' shows a hierarchy: 'Your own UDDI (1)' > 'JEUS Web Services (1)' > 'JEUS'. The main content area is divided into two sections. The top section has a 'Publish View in a' dropdown and a 'Publish for a' dropdown. Below these is a 'Provider:' label. The bottom section features a tabbed interface with tabs for 'Detail', 'Service', 'Contact', 'Identifier', 'Category', 'Discovery URL', and 'Relationship'. The 'Service' tab is selected, showing a 'Service' input field and an 'Add Service' button. The 'Actions' label is also visible.

## tModelの追加

以下は、tModelを追加する手順についての説明です。

1. **RegisteredInfo**画面に戻り、**[Add TModel]**ボタンをクリックすると、デフォルト値の「New TModel」という名前のユーザーの、新しく追加された技術モデルと共に、**TModel**画面に移動します。
2. **TModel**画面では、付加的な情報である技術モデルについての説明、Overview URL、カテゴリー、そして識別子を追加できます。「**Name**」項目に「tModel for eBiz」と入力します。
3. **TModel**画面で**[Overview Document]**タブに移動し、**[Add OverviewDoc]**ボタンをクリックすると、ユーザーの技術モデルのOverview Doc情報の画面に移動します。  
「**OverviewURL**」のサンプル・アドレスに「http://localhost:8088/ws/tmodel.wsdl」と入力します。
4. **[Add]**ボタンをクリックすると、技術モデルに保存します。

[図 19.8] tModelの保存結果画面

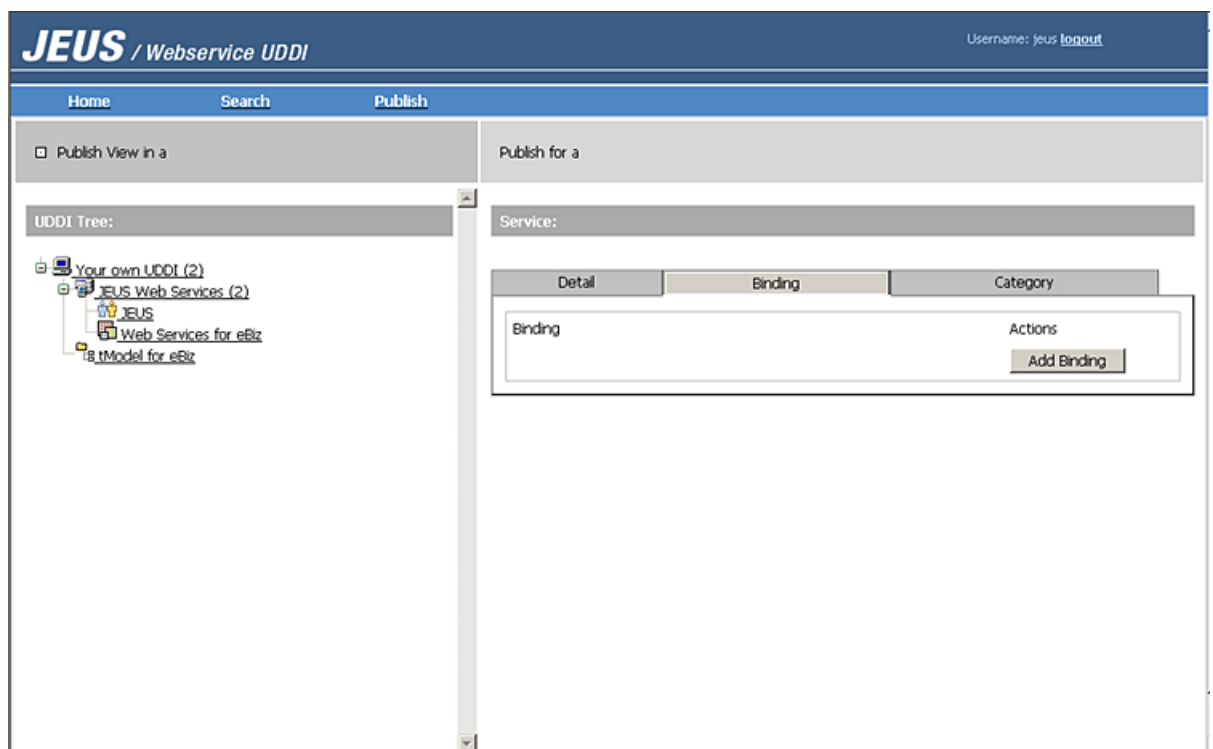
The screenshot shows the JEUS / Webservice UDDI interface. The top navigation bar includes 'Home', 'Search', and 'Publish' tabs. The 'Publish' tab is active. On the left, the 'UDDI Tree' shows a hierarchy of 'Your own UDDI (2)' containing 'JEUS Web Services (2)', which includes 'JEUS', 'Web Services for eBiz', and 'tModel for eBiz'. The main area on the right is titled 'TModel:' and contains a form with four tabs: 'Detail', 'Identifier', 'Category', and 'Overview Document'. The 'Overview Document' tab is selected. It features a table with two columns: 'Overview Documents' and 'Actions'. The first row in the table has the URL 'http://localhost:8088/' in the 'Overview Documents' column and an 'Add' button in the 'Actions' column. Below the table, there is a 'text' dropdown menu and a 'Cancel' button. At the bottom right of the form, there is an 'Add OverviewDoc' button.

## バインディング・テンプレートの追加

以下は、バインディング・テンプレートを追加する手順についての説明です。

1. 画面左側のツリー部分でビジネスに関連する[+]をクリックします。ビジネスに関連するサービスをクリックすると、画面右側にサービスについての詳細が表示されます。
2. バインディング・テンプレートを追加するには、[Binding]タブで[Add Binding]ボタンをクリックすると、Binding画面に移動します。

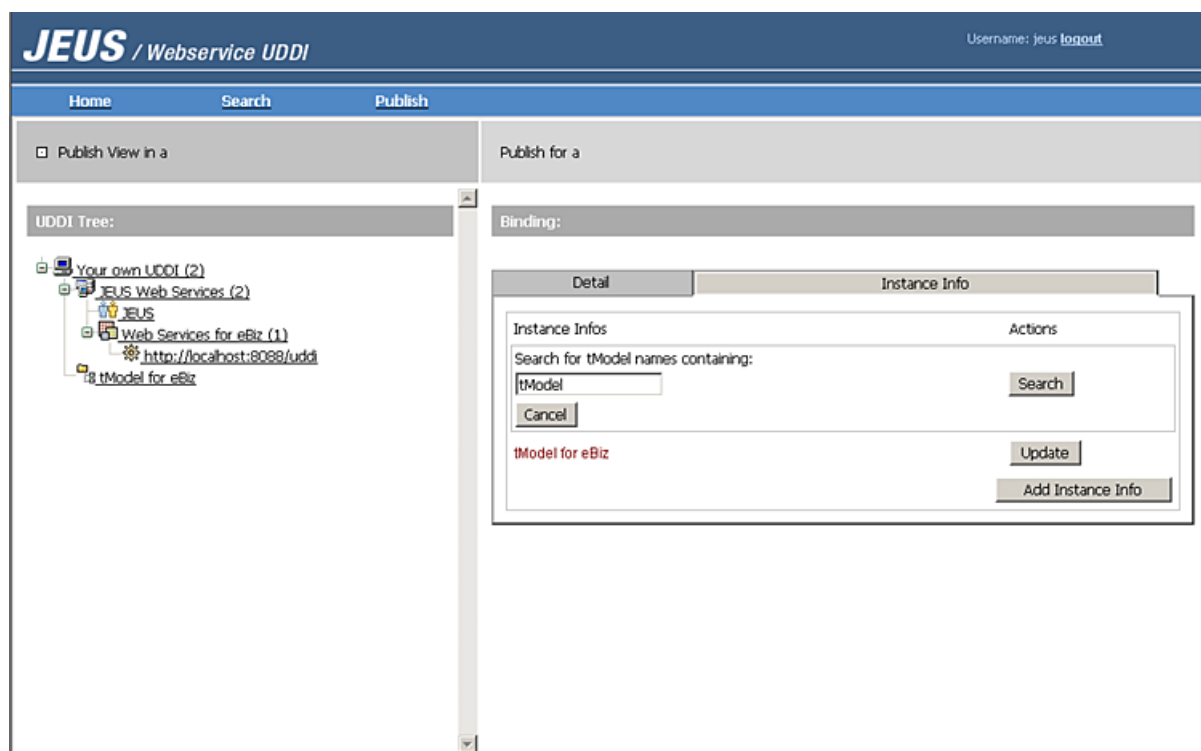
[図 19.9] バインディング・テンプレートの追加画面



3. Binding画面の[Detail]タブでは、Access Pointのアドレスを修正できます。「Address」項目に「http://localhost:8088/uddi」と入力します。
4. [Instance Info]タブに移動し、[Add Instance Info]ボタンをクリックします。[Search]ボタンをクリックすると、ユーザーが公開されたtModelを検索できる画面を表示します。検索するtModelの名前全体あるいは最初の数文字を「Search for tModel names containing」に入力して検索できます。該当項目に「tModel」と入力します。

5. 画面左側のツリービューで「tModel for eBiz」を選択し、[Add Instance Info]ボタンをクリックすると、技術モデルをバインディングしたサービスが保存されます。

[図 19.10] バインディング・テンプレートの保存結果画面



### 19.3.3. JEUS UDDIエクスプローラの設定

JEUS UDDIエクスプローラでJEUS UDDIレジストリーを使用するためにユーザーを登録できます。

まず、管理者権限を持つユーザーでログインする必要があります。ログイン後、メニューで**User**リンクを選択すると、以下の画面が表示されます。ユーザーはこの画面で新規ユーザーの追加、ユーザー情報の修正や削除が可能です。

[Add User]ボタンをクリックすると、以下のような新規ユーザー登録画面が表示されます。

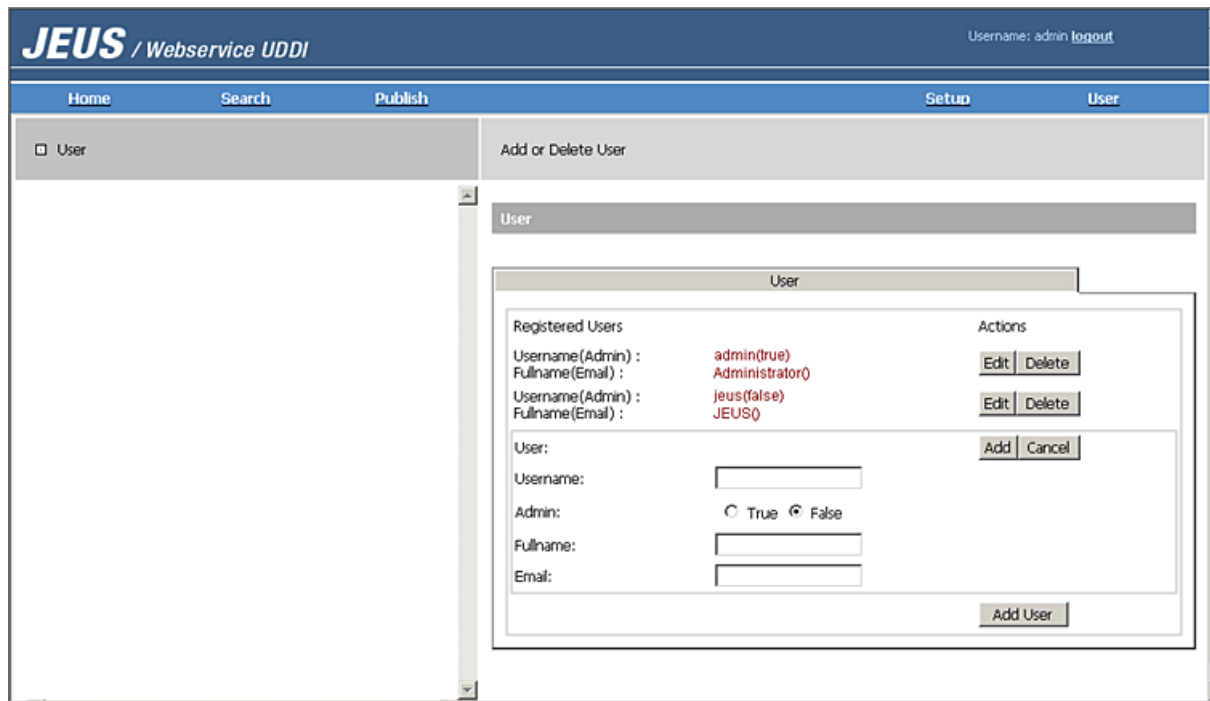
[図 19.11] JEUS UDDIエクスプローラでのユーザー管理画面

Registered Users		Actions
Username(Admin) :	admin(true)	Edit Delete
Fullname(Email) :	Administrator()	
Username(Admin) :	jeus(false)	Edit Delete
Fullname(Email) :	JEUS()	
		Add User

「Username」と「Fullname」は必須入力事項です。「Username」は、JEUS UDDIエクスプローラにログインしたり、UDDIデータを公開したりするためのユーザーIDです。値を入力後、[Add]をクリックすると、ユーザーが登録されます。



[図 19.12] JEUS UDDIエクスプローラでのユーザー登録画面



## 19.4. UDDIクライアントの作成

JEUS UDDIクライアント・ライブラリーは、UDDIレジストリーのJavaインターフェースを提供します。これを通じて構築したWebサービスを公開でき、必要なWebサービスを検索できます。JEUS UDDIクライアント・ライブラリーは一種のJavaクラス・ライブラリーで、UDDIレジストリーと相互作用するAPIを提供します。

本節では、UDDIクライアント・ライブラリーを使用して、UDDIクライアントの作成、コンパイル、実行方法について説明します。ここで提示する例はUDDI v2.0ベースのUDDIクライアントであり、UDDI v3.0ベースのUDDIクライアントも類似した方法で使用できます。

### 19.4.1. UDDIクライアントの作成

以下は、UDDIクライアントを作成する手順についての説明です。

- UDDIクライアント・クラス

UDDIクライアント・アプリケーションのコア・クラスです。これは、UDDIレジストリーに接続し、クエリを実行し、結果を処理するすべてのメソッドを持っています。

以下のサンプル・コードは、どのようにオブジェクトを作成し、UDDIレジストリーを参照するかについて示しています。ここで示す例のアプリケーションで、「UDDIクライアント」はUDDIレジストリーと相互作用するために使用します。

```
UDDIClient client = new UDDIClient();
client.setInquiryURL("http://localhost:8088/uddi/inquiry");
client.setPublishURL("http://localhost:8088/uddi/publish");
```

## ● UDDIレジストリーでのビジネス・クエリ

```
Vector inNames = new Vector();
inNames.add(new Name("test_Biz"));

BusinessList list = client.find_business(null, inNames, null, null, null, null, 0);
```

find\_businessメソッドは、多数の媒介変数を持っています。2番目の媒介変数は検索語で、7番目の媒介変数は条件に合う戻り値の最大数です。(0は条件に一致するすべての値を返すことを意味します)

より詳しく検索するためには、識別子(identifiers)、カテゴリー(categories)、URL、技術モデル(tModel)などを検索条件に入れることができます。上の方法で、条件に一致するビジネス・リストを表示できます。

以下は、リストに含まれているビジネスの名前を出力するサンプル・コードです。

```
BusinessInfos infos = list.getBusinessInfos();
Vector businesses = infos.getBusinessInfoVector();

if (businesses != null) {
    for (int i = 0; i < businesses.size(); i++) {
        BusinessInfo info = (BusinessInfo)businesses.elementAt(i);
        Vector outNames = info.getNameVector();
        for (int j = 0; j < outNames.size(); j++) {
            Name name = (Name)outNames.elementAt(j);
            System.out.println(name.getValue());
        }
    }
}
```

## ● UDDIレジストリーでのビジネスの公開

新しいUDDIデータをUDDIレジストリーで公開するには、UDDIレジストリーから認証を取得する必要があります。以下は、UDDIレジストリーから認証を取得するコードです。

```
AuthToken authToken = client.get_authToken("userID", "password");
```

次の段階では、新しいビジネス・エンティティを作成し、認証を取得したUDDIレジストリーに公開します。例ではビジネス名のみを定義しています。

```
BusinessEntity businessEntity = new BusinessEntity();
businessEntity.setBusinessKey("");
businessEntity.addName(new Name("TmaxSoft", "en"));

Vector businessVector = new Vector();
businessVector.add(businessEntity);
```

```
BusinessDetail detail = client.save_business(  
    authToken.getAuthInfoString(), businessVector);
```

#### ● UDDIレジストリーからのビジネスの削除

```
Vector inNames = new Vector();  
inNames.add(new Name("TmaxSoft"));  
  
BusinessList list = client.find_business(null, inNames, null, null, null, null, 0);
```

上記方法で、公開されたビジネス・リストを取得しました。次に、取得した各ビジネスを削除します。

以下は、そのサンプル・コードです。

```
Vector businessInfoVector =  
    list.getBusinessInfos().getBusinessInfoVector();  
for (int i = 0; i < businessInfoVector.size(); i++) {  
    BusinessInfo info =  
        (BusinessInfo)businessInfoVector.elementAt(i);  
    DispositionReport dispositionReport = client.delete_business(  
        authToken.getAuthInfoString(), info.getBusinessKey());  
}
```

## 19.4.2. UDDIクライアントのコンパイル

UDDIクライアント・コードをコンパイルするには、クライアント・ソースコードが位置するディレクトリーに移動します。この例では、JEUS\_HOME/samples/webservice/uddi/clientディレクトリーで以下のコマンドを実行します。

```
JEUS_HOME/samples/webservice/uddi/client$ ant build
```

## 19.4.3. UDDIクライアントの実行

UDDIクライアント・コードを実行するには、クライアント・ソースコードが位置するディレクトリーに移動します。この例では、JEUS\_HOME/samples/webservice/uddi/clientディレクトリーで以下のコマンドを実行します。

```
JEUS_HOME/samples/webservice/uddi/client$ ant
```

以下は、実行結果の一例です。

```
##### Running FindBusinessSample #####  
Found Business name: test_Biz  
##### Done #####
```

## 19.5. XMLデジタル署名の使用方法

本節では、XMLデジタル署名の概念と作成、および検証方法について説明します。

### 19.5.1. デジタル署名

UDDI v3.0の仕様において重要な変化の1つが、デジタル署名のサポートです。UDDIエンティティがデジタル署名されると、データの整合性(integrity)と確実性(authenticity)がUDDIによって渡されます。UDDIレジストリーの要求者(inquirer)は、署名されたデータのみを要求するようクエリを調整でき、要求者がUDDIレジストリーから持って来たデータを検証する際に、データが公開者(publisher)が意図したことなのかを確信できるようになります。

UDDIレジストリーの公開者は、UDDIエンティティを所有したと主張する者によって、渡し間違えていないことを確信できるようになります。また、公開者が署名されたデータを持つと、公開者はそのデータの整合性について確信できるようになります。

デジタル署名は、UDDIのデータ品質を向上させ、高い信頼性を必要とするWebサービス環境で要求するデータ保護と否認防止を可能にします。

UDDI v3.0の仕様では、XML署名の構文と処理方法(<http://www.w3.org/TR/xmlsig-core/>)を使用して、以下の上位レベルUDDIエンティティのXMLデジタル署名をサポートします。

- ビジネス・エンティティ(businessEntity)
- ビジネス・サービス(businessService)
- バインディング・テンプレート(bindingTemplate)
- 技術モデル(tModel)
- publisherAssertion

UDDI v3.0の仕様は、XML署名と検証をクライアント側の役割と定義しています。これに従って、JEUS環境でもクライアント側のXML署名と検証をサポートしており、本書で記述しているUDDIクライアントはJEUS UDDIクライアントを使用していると仮定します。

---

#### 参考

XMLデジタル署名のサポートについての情報は、UDDI v3.0の仕様の「Appendix I, Support For Digital Signatures」を参照してください。

---

## 19.5.2. UDDIクライアントにおけるXML署名の作成方法

XML署名は、公開(publication)APIの呼び出しの結果として、UDDIレジストリーに保存される上位レベルの要素に対して計算されます。

以下は、各APIの呼び出し別に実行される要素です。

API	要素
save_business	businessEntity
save_service	businessService
save_binding	bindingTemplate
save_tModel	tModel
set_publisherAssertions	publisherAssertion
add_publisherAssertions	

JEUS UDDIクライアントは、「java.security.\*」のjava.security.PrivateKeyオブジェクトとjava.security.cert.X509Certificateオブジェクトを使用して、上位レベルのUDDIエンティティーを署名します。

以下は、PrivateKeyオブジェクトとX509Certificateオブジェクトを取得する例です。

```
import java.security.KeyStore;
import java.security.PrivateKey;
import java.security.cert.X509Certificate;
...

String keystoreType = "JKS";
String privateKeyAlias = "...";
String privateKeyPasswd = "...";
String certificateAlias = "...";
...
KeyStore keyStore = KeyStore.getInstance(keystoreType);
keyStore.load(...);

PrivateKey privateKey = (PrivateKey)keyStore.getKey(
    privateKeyAlias, privateKeyPasswd.toCharArray());
X509Certificate certificate =
    (X509Certificate)keyStore.getCertificate(certificateAlias);
```

取得したPrivateKeyオブジェクトとX509Certificateオブジェクトを使用してXML署名をサポートするJEUS UDDIクライアント・プログラミング方法は、既存のJEUS UDDIクライアント・プログラミング方法と基本的に同じです。プログラミング方法の詳細については「[19.4. UDDIクライアントの作成](#)」を参照してください。

公開されるUDDIエンティティーは、jeus.uddi.v3.client.UDDIClientクラスで提供する以下のメソッドを使用して署名されます。

```

public BindingDetail save_binding(
    String authInfo, bindingTemplate bindingTemplate,
    PrivateKey privateKey, X509Certificate certificate)
public BusinessDetail save_business(
    String authInfo, BusinessEntity businessEntity,
    PrivateKey privateKey, X509Certificate certificate)
public ServiceDetail save_service(
    String authInfo, BusinessService businessService,
    PrivateKey privateKey, X509Certificate certificate)
public TModelDetail save_tModel(
    String authInfo, TModel tModel,
    PrivateKey privateKey, X509Certificate certificate)
public PublisherAssertions set_publisherAssertions(
    String authInfo, PublisherAssertion publisherAssertion,
    PrivateKey privateKey, X509Certificate certificate)

```

署名するUDDIエンティティはエンティティ・キーを含む必要があります。エンティティ・キーのないUDDIエンティティを公開する場合、UDDIレジストリーはそのUDDIエンティティに対して新しいエンティティ・キーを割り当て、UDDIエンティティに対する署名を無効にします。そのため、公開者はエンティティ・キーが含まれているUDDIエンティティを公開する役割があります。

たとえば、ビジネス・サービスはサービス・キーとビジネス・キーを含む必要があります、ビジネス・サービスがバインディング・テンプレートを含んでいる場合、バインディング・テンプレートはバインディング・キーとサービス・キーを含む必要があります。

### 19.5.3. UDDIクライアントにおけるXML署名の検証

XML署名の検証は、要求(inquiry)API呼び出しの結果として応答する上位レベルの要素に対して実行されます。

以下は、各APIの呼び出し別に実行される要素です。

API	element
get_businessDetail	businessEntity
get_serviceDetail	businessService
get_bindingDetail	bindingTemplate
get_tModelDetail	tModel
find-relatedBusinesses API	publisherAssertion

典型的なUDDIレジストリーは、クライアント側の署名検証を行いません。

JEUS UDDIクライアントは署名検証の方法を提供しません。該当要求API呼び出しに対する応答に署名要素が含まれている場合、内部で署名検証を自動的に実行します。署名が有効でない場合、`java.security.SignatureException`を返します。

## 19.6. UDDIサブスクリプションの使用方法

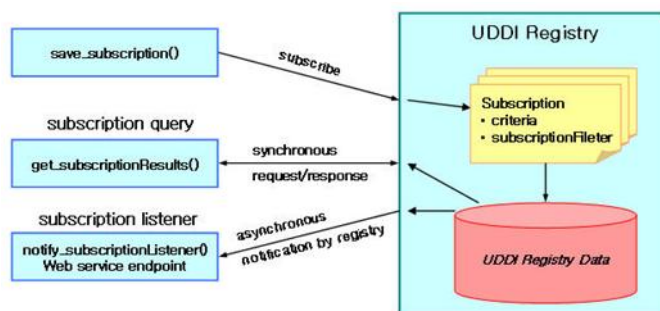
本節では、UDDIサブスクリプションの概念と作成方法、クライアント・プログラミングおよびEメールを受信するためのサーバー設定方法について説明します。

### 19.6.1. 基本概念

UDDI v3.0の仕様(specification)の新しいサブスクリプションAPIは、同期型(synchronous)、非同期型(asynchronous)の方法で通知(notification)機能をサポートします。

サブスクリプションAPIは、UDDIレジストリーを追跡できる方法を提供します。これを使用して加入者(subscriber)は特別なクエリまたは加入者が関心を持つエンティティーをベースにサブスクリプションを作成できます。このサブスクリプションをベースにして、UDDI情報に対する同期型要求またはレジストリーによって加入者に行われる非同期型通知を通じ、UDDIレジストリーの内容変更を追跡できます。

【図 19.13】 UDDIサブスクリプションの流れ



UDDIサブスクリプションは以下の場合に有効です。

- UDDIレジストリーに登録されている新規ビジネスまたはサービスに関する通知
- 既存ビジネスまたはサービスの変化を検知

### 19.6.2. UDDIサブスクリプションの作成方法

サブスクリプションの作成とその結果を検知する手順は次のとおりです。

1. UDDIレジストリーが提供する非同期型通知を受けるためには、サービスを作成する必要があります（加入者が同期型要求のみを使用する場合は、この段階は省略できます）。加入者は加入者が実装したHTTP/SOAP WebサービスまたはEメールを通じてサブスクリプション結果を受け取るよう選択できます。
2. サブスクリプションを使用するためのフィルタ条件を選択します。この際、応答結果が分析しやすくなるよう、検索条件を詳しく設定します。一般的に加入者は、使用したサブスクリプション・フィルタ[subscriptionFilter]条件は可能な限り制限されていることを保証する必要があります。

3. save\_subscription APIを使用してサブスクリプション要求を保存します。
4. 必要に応じて、HTTP/SOAPまたはEメールによる通知を処理します。

### 19.6.3. UDDIサブスクリプションの例

本節では、非同期型通知と同期型要求の例について説明します。

#### 非同期型通知

以下は、非同期型通知形式のサブスクリプションの作動例です。

1. ビジネス変化を追跡するサブスクリプションを作成します。

このために、加入者はバインディング・テンプレートを持つサービスを、通知を送信するUDDIレジストリーに登録する必要があり、このバインディング・テンプレートには「notify\_subscriptionListener」サービスまたはEメールを通じて受信できるようにエンドポイントを表示します。

```
<save_binding xmlns="urn:uddi-org:api_v3">
  <authInfo>myAuthCode</authInfo>
  <bindingTemplate bindingKey=" " serviceKey="uddi:myservicekey">
    <description>notify_subscriptionListener binding for
      my subscription.
    </description>
    <accessPoint URLType="https">
      https://www.myCompany.com/services/notify_subscriptionListener
    </accessPoint>
    <tModelInstanceDetails>
      <tModelInstanceInfo
        tModelKey="uddi:uddi.org:v3_subscriptionlistener"/>
      </tModelInstanceInfo>
    </tModelInstanceDetails>
  </bindingTemplate>
</save_binding>

<save_binding xmlns="urn:uddi-org:api_v3">
  <authInfo>myAuthCode</authInfo>
  <bindingTemplate bindingKey=" " serviceKey="uddi:myservicekey">
    <description>
      E-mail binding for my subscription.
    </description>
    <accessPoint URLType="email">
      mailto:mySubscriberEmail@xyz.com
    </accessPoint>
  </bindingTemplate>
</save_binding>
```

(出典: <http://uddi.org/pubs/uddi-v3.0.2-20041019.htm>)



2. 周期的(例では5日ごと)に通知するようにサブスクリプションを作成し、保存します。例では、brief属性は通知結果にエンティティー・キーのみを使用するよう制限するために使用されています。

```
<save_subscription xmlns="urn:uddi-org:sub_v3">
  <authInfo>myAuthCode</authInfo>
  <subscriptions>
    <subscription brief="true">
      <subscriptionFilter>
        <find_service xmlns="urn:uddi-org:api_v3" >
          <findQualifiers>
            <findQualifier>
              uddi:uddi.org:findqualifier:sql99:like
            </findQualifier>
          </findQualifiers>
          <categoryBag>
            <keyedReference
              tModeKey="uddi:uddi.org:ubr:taxonomy:naics"
              keyName="Motor Vehicle Parts"
              keyValue="42112_"/>
          </categoryBag>
        </find_service>
      </subscriptionFilter>
      <bindingKey>
        bindingKeyOfTheClientsNotifySubscriptionListenerService
      </bindingKey>
      <notificationInterval>P5D</notificationInterval>
      <maxEntities>1000</maxEntities>
    </subscription>
  </subscriptions>
</save_subscription>
```

(出典: <http://uddi.org/pubs/uddi-v3.0.2-20041019.htm>)

以下は、notify\_subscriptionListener APIを実装したクライアントの呼び出しで通知される結果です。

```
<notify_subscriptionListener>
  <subscriptionResultsList>
    <coveragePeriod>
      <startPoint>20020101T00:00:00</startPoint>
      <endPoint>20020131T00:00:00</endPoint>
    </coveragePeriod>
    <subscription brief="true">
      <subscriptionFilter>
        <find_service xmlns="urn:uddi-org:api_v3" >
          <findQualifiers>
            <findQualifier>
              uddi:uddi.org:findqualifier:sql99:like
            </findQualifier>
          </findQualifiers>
          <categoryBag>
```

```

        <keyedReference
            tModeKey="uddi:uddi.org:ubr:taxonomy:naics"
            keyName="Motor Vehicle Parts"
            keyValue="42112_" />
    </categoryBag>
</find_service>
</subscriptionFilter>
<bindingKey>
    bindingKeyOfTheClientsNotifySubscriptionListenerService
</bindingKey>
<notificationInterval>P5D</notificationInterval>
<maxEntities>1000</maxEntities>
<expiresAfter>20030101T00:00:00</expiresAfter>
</subscription>
<keyBag>
    <deleted>>false</deleted>
    <serviceKey>matchingKey1</serviceKey>
    <serviceKey>matchingKey2</serviceKey>
    <serviceKey>matchingKey3</serviceKey>
    <serviceKey>matchingKey4</serviceKey>
</keyBag>
<keyBag>
    <deleted>>true</deleted>
    <serviceKey>matchingKey5</serviceKey>
    <serviceKey>matchingKey6</serviceKey>
</keyBag>
</subscriptionResultsList>
</notify_subscriptionListener>

```

(出典: <http://uddi.org/pubs/uddi-v3.0.2-20041019.htm>)

brief属性を「true」に設定したサブスクリプションは、エンティティー・キーのみで結果を通知します。エンティティー・キーは<keyBag>に含まれて通知され、既に削除されたエンティティーのキーを含んだ<keyBag>は、<deleted>が「true」に設定されて通知されます。

## 同期型要求

APIを使用して同期的にサブスクリプション結果を取得できます。同期型要求のためのサブスクリプションの保存方法は、サブスクリプション・リスナーのための<bindingKey>を必要としないことを除いては、非同期型通知で説明したサブスクリプションの保存方法と同じです。

以下は、典型的なget\_subscriptionResults APIを使用する例です。

```

<get_subscriptionResults>
    <authInfo>myAuthCode</authInfo>
    <subscriptionKey>mySubscriptionKey</subscriptionKey>
    <coveragePeriod>
        <startPoint>20020101T00:00:00</startPoint>
    </coveragePeriod>
</get_subscriptionResults>

```

(出典: <http://uddi.org/pubs/uddi-v3.0.2-20041019.htm>)

上の要求は、同期的に<subscriptionResultList>形式で応答します。

```
<subscriptionResultsList>
  <chunkToken>"nodeGeneratedToken"</chunkToken>
  <coveragePeriod>
    <startPoint>20020101T00:00:00</startPoint>
  </coveragePeriod>
  <subscription brief="true">
    <subscriptionFilter>
      <find_service xmlns="urn:uddi-org:api_v3">
        <findQualifiers>
          <findQualifier>
            uddi:uddi.org:findqualifier:sql99:like
          </findQualifier>
        </findQualifiers>
        <categoryBag>
          <keyedReference
            tModeKey="uddi:uddi.org:ubr:taxonomy:naics"
            keyName="Motor Vehicle Parts"
            keyValue="42112_" />
        </categoryBag>
      </find_service>
    </subscriptionFilter>
    <bindingKey>
      bindingKeyOfTheClientsNotifySubscriptionListenerService
    </bindingKey>
    <notificationInterval>P5D</notificationInterval>
    <maxEntities>1000</maxEntities>
    <expiresAfter>20030101T00:00:00</expiresAfter>
  </subscription>
  <keyBag>
    <deleted>>false</deleted>
    <serviceKey>matchingKey1</serviceKey>
    <serviceKey>matchingKey2</serviceKey>
    <serviceKey>matchingKey3</serviceKey>
    <serviceKey>matchingKey4</serviceKey>
  </keyBag>
</subscriptionResultsList>
```

(出典: <http://uddi.org/pubs/uddi-v3.0.2-20041019.htm>)

## 19.6.4. UDDIサブスクリプション・クライアント・プログラミング

JEUS UDDIクライアントは、サブスクリプションAPIをサポートするために、jeus.uddi.v3.client.UDDIClient クラスに以下のようなメソッドを提供します。

```

public subscriptions save_subscription(
    String authInfo, Subscription subscription)
public void delete_subscription(
    String authInfo, String subscriptionKey)
public Subscriptions get_subscriptions(String authInfo)
public SubscriptionResultsList get_subscriptionResults(
    String authInfo, String subscriptionKey,
    CoveragePeriod coveragePeriod, String chunkToken)

```

サブスクリプション・クライアント・プログラミングは、基本的に既存のJEUS UDDIクライアント・プログラミングと同じです。プログラミングする際は、サブスクリプション・フィルタ条件をできるだけ明確に設定することに注意してください。

以下は、簡単なJEUS UDDIサブスクリプション・クライアント・プログラミングの例です。

```

UDDIClient client;
. . .

AuthToken authToken = client.get_authToken(user, password);

// Make a subscription
Subscription subscription = new Subscription();

// Make a subscriptionFilter
FindBusiness findBusiness = new FindBusiness();
findBusiness.addName(new Name("biz"));
subscription.setSubscriptionFilter(
    new SubscriptionFilter(findBusiness));

// execute save_subscription request
Subscriptions subscriptions = client.save_subscription(
    authToken.getAuthInfoString(), subscription);

Subscription savedSubscription = subscriptions.getSubscription(0);
String subscriptionKey =
    savedSubscription.getSubscriptionKeyString();
. . .

```

登録されているサブスクリプションから取得したサブスクリプション・キーは、delete\_subscription APIを使用してサブスクリプションを消したり、get\_subscriptionResults APIを使用して同期型要求で結果を取得する際に使用されます。get\_subscriptions APIは、加入者が登録したすべてのサブスクリプションを照会する際に使用されます。

加入者がサブスクリプションを登録する際、登録されているサブスクリプションの結果がエンティティー・キーのみを使用するよう制限されることを望む場合、以下のように、サブスクリプションのbrief属性を「true」に設定します。

```

. . .
// Make a Subscription

```

```
Subscription subscription = new Subscription();
Subscription.setBrief(true);
. . .
```

加入者が非同期的通知を受けることを望む場合、サブスクリプション・リスナーのための<bindingKey>をサブスクリプションに設定し、通知間隔を設定します。通知間隔は、XMLスキーマに定義されているxsd:durationタイプで、「PnYnMnDTnHnMnS」を持っています。

```
. . .
// Make a Subscription
Subscription subscription = new Subscription();
subscription.setBindingKey(
    bindingKeyOfTheClientsNotifySubscriptionListenerService);
subscription.setNotificationInterval("P5D");
. . .
```

## 19.6.5. Eメール通知を受けるためのUDDIサーバーの設定

JEUS UDDIサーバーが加入者にEメールを通じてサブスクリプション結果を通知するためには、メールサーバーを設定する必要があります。この設定は、JEUS UDDIサーバーの設定ファイルである「uddi.properties」で行います。

```
. . .
uddi.subscription.mail.smtp.host=...
uddi.subscription.mail.smtp.port=...
uddi.subscription.mail.smtp.from=...
. . .
```

## 19.7. UDDI WSDLの公開の使用方法

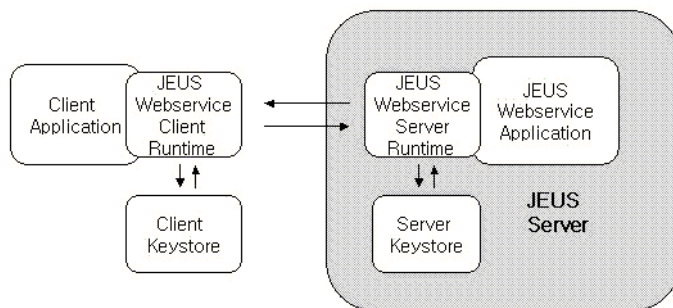
本節では、UDDI WSDLの公開の使用方法について説明します。

### 19.7.1. UDDI WSDLの公開

e-business標準のための機構であるOASISでは、UDDI v2.0、v3.0の仕様(specification)以外に、これに関連するTechnical Noteについて作業しています。そのうちの1つがUDDI WSDLの公開(Using WSDL in a UDDI Registry、Version 2.0.2)です。

WSDLは、抽象インターフェースと任意のネットワーク・サービスのプロトコル・バインディングを提供することによって、UDDI標準を補完できます。OASISの「Using WSDL in a UDDI Registry」は、このようなWSDLのディスクリプションとUDDIのデータ構造とのマッピングについて記述しています。詳細については、OASIS TC Using WSDL in a UDDI Registry、Version 2.0.2を参照してください。

**[図 19.14] WSDL descriptionとUDDI Data Structureのマッピング**



## 19.7.2. wsdl2uddiの使用

本節では、JEUSで提供するwsdl2uddiツールについて説明し、wsdl2uddiを使用してwsdlをUDDIに公開する方法を説明します。

### wsdl2uddi

JEUS UDDI WSDLの公開は、このようなマッピングを自動化するツールであるwsdl2uddiを提供します。基本的な動作方法は、コンソールで以下のように入力し、コマンドを実行します。

```
$ wsdl2uddi
```

使用方法は以下のとおりです。

```
Usage: wsdl2uddi UDDIVersion WSDLURI [wsdl-options] -uddiInquiry
UDDIInquiryURI -uddiPublish UDDIPublish -uddiUsername UDDIUsername
-uddiPassword UDDIPassword [options]

* wsdl-options
  -wsdlUsername          username to access the WSDL-URI
  -wsdlPassword          password to access the WSDL-URI

* options
  -level LEVEL           specify log level.
                        LEVEL : SEVERE, WARNING, INFO, FINE, FINER, FINEST
  -verbose               same as -level FINE
```

以下は、各パラメータについての説明です。

パラメータ	説明
UDDIVersion(必須値)	v2またはv3です
WSDLURI(必須値)	WSDLが存在するURI値です

パラメータ	説明
wsdl-options	セキュリティが設定されているWSDLにアクセスする場合、ユーザー名とパスワードを設定する場合に使用します
UDDIInquiryURI(必須値)	UDDIのInquiry URI値を指定します
UDDIPublishURI(必須値)	UDDIのpublish URI値を指定します
UDDIUsername(必須値)	UDDIにアクセスする際に必要なユーザー名を設定します
UDDIPassword(必須値)	UDDIにアクセスする際に必要なパスワードを設定します
options	ログレベルを指定する場合に使用します

## 任意のWSDLをUDDIに公開

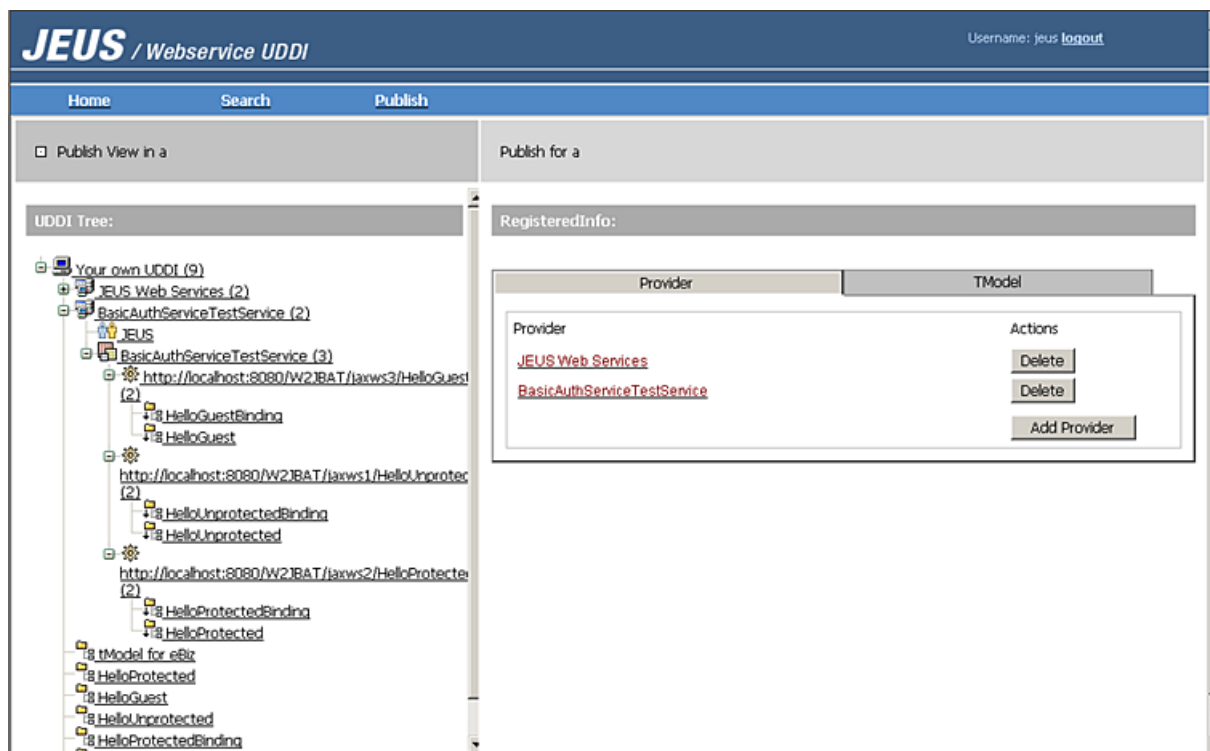
wsdl2uddiツールを利用して、1つのWSDLをUDDIに公開します。

```
$ wsdl2uddi v2 BasicAuthServiceTestService.wsdl -uddiInquiry
http://localhost:8088/uddi/inquiry -uddiPublish
http://localhost:8088/uddi/publish -uddiUsername jeus -uddiPassword jeus
[2006.12.12 17:22:47][2][ ] [client-10] [WSVC-5240] Analyzing WSDL :
BasicAuthServiceTestService.wsdl
```

上のように入力すると、wsdl2uddiツールは自動的にwsdlを解析して、バインディング・ルールに従ってUDDIに公開します。

以下は、公開されたwsdlをUDDIで確認した画面です。

[図 19.15] UDDIに公開されたwsdl







# 第20章 JEUS WebサービスのXML

本章では、JEUS 8 WebサービスがサポートするXMLに関連する様々な技術について説明します。

## 20.1. 概要

XML文書をスキーマからコンパイルされたJavaクラスでJavaオブジェクト化(バインディング)し、XML文書の情報をプログラミングするように操作できるよう(programmatic)にするJAXB(Java Architecture for XML Binding)について説明します。その後、JAXP(Java API for XML Processing)に新しくAPIを導入し始めたStAX(Streaming APIs For XML)についても記述します。

- JAXB(Java Architecture for XML Binding)
- JAXP(Java API for XML Processing)
- SJSXP(Sun Java Streaming XML Parser)

各章を説明する前に、XML文書や文書内の内容とJavaオブジェクトの変換で一般的に使用される用語について説明します。

区分	意味
アンマーシャリング (Unmarshalling)	XML文書あるいはXMLコンテンツをJavaクラスでオブジェクト化(Java Content Tree)するプロセスです
マーシャリング(Marshalling)	Javaオブジェクト(Java Content Tree)をXML文書あるいはXMLコンテンツに変換するプロセスです

## 20.2. JAXB(XMLバインディングのためのJavaアーキテクチャー)

或るスキーマを遵守するXML文書に関連したアプリケーションは、その文書を作成、修正、読み取りが可能です。このようなアプリケーションで使用されるXMLに関連するデータ・バインディングは、SAXやDOM APIを利用することもできますが、保守(スキーマ変化に従う)面において容易ではありません。したがって、JAXBはXML文書とJavaオブジェクト間のマッピングを自動化させるAPIおよびツールを提供します。

JAXBの機能は以下のとおりです。

- XMLコンテンツをJavaオブジェクトに変換するアンマーシャリング機能
- Javaオブジェクトに表現されたオブジェクトへのアクセスや修正

- Javaで表現されたオブジェクトを再びXMLコンテンツに変換するマーシャリング機能

このように、JAXBがXMLとJavaコード間で標準化され、効率的なマッピングを提供することで、XMLとWebサービス技術を使用するアプリケーションをより容易に開発可能にします。次の章からは、JAXBで提供するツールとサンプル・プログラムを用いて、JAXBの詳細について説明します。

## 20.2.1. バインディング・コンパイラ(XJC)関連のプログラミング

XMLコンテンツをJavaオブジェクトに変換するには、変換前にJavaクラスを持っている必要があります。このようなJavaクラスはスキーマから生成されますが、これをバインディング・コンパイルといいます。

本節では、JEUS 8 Webサービスが基本的に提供するバインディング・コンパイラ・ツールであるXJCについて説明します。

### XJC

基本的な動作方法は、コマンド・ライン(Command Line)で以下のように入力し、コマンドを実行します。

```
JEUS_HOME/bin$ xjc.cmd -help
```

使用方法是以下のとおりです。

```
Usage: xjc [-options ...] <schema file/URL/dir/jar> ...
        [-b <bindinfo>] ...
If dir is specified, all schema files in it will be compiled.
If jar is specified, /META-INF/sun-jaxb.episode binding file will be
compiled.
Options:
  -nv                : do not perform strict validation of the input
                      schema(s)
  -extension         : allow vendor extensions - do not strictly
                      follow the Compatibility Rules and App E.2
                      from the JAXB Spec
  -b <file/dir>      : specify external bindings files (each <file>
                      must have its own -b)
                      If a directory is given, **/*.xjb is searched
  -d <dir>           : generated files will go into this directory
  -p <pkg>           : specifies the target package
  -httpproxy <proxy> : set HTTP/HTTPS proxy. Format is
                      [user[:password]@]proxyHost:proxyPort
  -httpproxyfile <f> : Works like -httpproxy but takes the argument
                      in a file to protect password
  -classpath <arg>   : specify where to find user class files
  -catalog <file>    : specify catalog files to resolve external
                      entity references support TR9401, XCatalog,
                      and OASIS XML Catalog format.
  -readOnly          : generated files will be in read-only mode
```

```

-npa                : suppress generation of package level
                    : annotations(**/package-info.java)
-no-header          : suppress generation of a file header with
                    : timestamp
-target 2.0         : behave like XJC 2.0 and generate code that
                    : doesnt use any 2.1 features.
-xmlschema          : treat input as W3C XML Schema (default)
-relaxng            : treat input as RELAX NG (experimental,
                    : unsupported)
-relaxng-compact    : treat input as RELAX NG compact syntax
                    : (experimental, unsupported)
-dtd                : treat input as XML DTD (experimental,
                    : unsupported)
-wsdl               : treat input as WSDL and compile schemas inside
                    : it(experimental,unsupported)
-verbose            : be extra verbose
-quiet              : suppress compiler output
-help               : display this help message
-version            : display version information

Extensions:
-Xlocator           : enable source location support for generated
                    : code
-Xsync-methods      : generate accessor methods with the
                    : 'synchronized' keyword
-mark-generated     : mark the generated code as @javax.annotation.
                    : Generated
-episode <FILE>     : generate the episode file for separate
                    : compilation

```

---

## 参考

JEUS 8 WebサービスはXJCのAntタスクもサポートしますが、詳細については『*JEUS リファレンスガイド*』の「4.12. xjc」および『*JEUS リファレンスガイド*』の「5.5.5. xjc」を参照してください。

---

## XJC Antタスクを利用したプログラミング

以下は、JAXBを使用して、1つのスキーマとXML文書をメモリー上のJavaオブジェクトに変更してXML文書のコンテンツをプログラミング的に扱う例です。

全体的な流れは以下のとおりです。

1. 1つのXML文書をアンマーシャリングしてJavaオブジェクトに変換します。
2. 変換されたJavaオブジェクトに対してプログラミング的に加工します。
3. Javaオブジェクトを再びXML文書にマーシャリング(この例では出力)します。

以下は、この例のbuild.xmlの一部です。

**[例 20.1] << build.xml >>**

```
...
<project basedir="." default="run">
...
    <taskdef name="xjc" classname="com.sun.tools.xjc.XJCTask">
        <classpath refid="classpath" />
    </taskdef>
    <target name="compile" description="Compile all Java source
        files">
        <echo message="Compiling the schema..." />
        <mkdir dir="gen-src" />
        <xjc extension="true" schema="po.xsd" package="primer.myPo"
            destdir="gen-src">
            <produces dir="gen-src/primer.myPo" includes="**/*.java" />
        </xjc>
        <echo message="Compiling the java source files..." />
        <mkdir dir="classes" />
        <javac destdir="classes" debug="on">
            <src path="src" />
            <src path="gen-src" />
            <classpath refid="classpath" />
        </javac>
    </target>
    <target name="run" depends="compile" description="Run the sample
        app">
        <echo message="Running the sample application..." />
        <java classname="Main" fork="true">
            <classpath refid="classpath" />
        </java>
    </target>
...
</project>
```

以下は、この例のMain.javaについての説明です。

**[例 20.2] << Main.java >>**

```
Schema schema = SchemaFactory.newInstance(W3C_XML_SCHEMA_NS_URI).
    newSchema(new File("src/conf/ts.xsd"));

JAXBContext jc = JAXBContext.newInstance("com.tmaxsoft");

//
Unmarshaller unmarshaller = jc.createUnmarshaller();
unmarshaller.setSchema(schema);

...
```

```
//
Marshaller marshaller = jc.createMarshaller();
marshaller.setSchema(schema);
marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

...
```

build.xmlのXJC Antタスクを使用して作成したJavaクラスのパッケージ名を使用してJAXB Contextオブジェクトを作成し、これを利用してUnmarshallerとMarshallerを作成します。その後、スキーマに合わせて実装されたXML文書であるか否かを判断するためのスキーマ・オブジェクトを作成してUnmarshallerとMarshallerに登録します。

#### [例 20.3] << Main.java(続き)>>

```
Object ts = unmarshaller.unmarshal(new File("src/conf/tsInput.xml"));
TmaxSoftType tst = (TmaxSoftType) ((JAXBElement) ts).getValue();
Address address = tst.getAddress1();
address.setName("John Bob");
address.setStreet("242 Main Street");
address.setCity("Beverly Hills");
```

polInput.xmlというXML文書をアンマーシャリングします。アンマーシャリングされたJavaオブジェクトを修正します。

#### [例 20.4] << Main.java(続き)>>

```
marshaller.marshal(ts, System.out);
```

画面に出力(マーシャリング)します。

## 20.2.2. Schemagen関連プログラミング

JEUS 8 Webサービスは、スキーマからJavaクラスを作成するスキーマ・コンパイラであるXJCと一緒に、ユーザーが予め作成したJavaクラスから特定のXMLスキーマを作成できるツールを提供します。このようなツールをGeneratorといいます。本節では、JEUS 8 Webサービスが基本的に提供するXMLスキーマを作成するSchemagenについて説明します。

### Schemagen

基本的な動作方法は、コマンドラインで以下のように入力し、コマンドを実行します。

```
JEUS_HOME/bin$ schemagen.cmd -help
```

使用方法は以下のとおりです。

```
Usage: schemagen [-options ...] <java files>
Options:
    -d <path>          : specify where to place processor and javac
```

```
generated class files
-cp <path>          : specify where to find user specified files
-classpath <path>   : specify where to find user specified files
-episode <file>     : generate episode file for separate
                     compilation
-version            : display version information
-help              : display this usage message
```

---

## 参考

JEUS 8 WebサービスはSchemagenのAntタスクもサポートしますが、詳細については『*JEUS リファレンスガイド*』の「4.13. schemagen」および『*JEUS リファレンスガイド*』の「5.5.6. schemagen」を参照してください。

---

## Schemagen Antタスクを使用したプログラミング

全体的な流れは以下のとおりです。

1. Javaソースを使用し、ソースレベルでSchemagenツールを使用してスキーマを作成します。
2. Javaソースをコンパイルします。
3. コンパイルされたJavaソースに対し、プログラミング的にデータを入力してJavaオブジェクトを作成します。
4. 前でJavaオブジェクトを取得したスキーマを使用し、マーシャリング(この例では出力)します。

以下は、この例のbuild.xmlの一部です。

### [例 20.5] << build.xml >>

```
...
<project basedir="." default="run">
...
  <taskdef name="schemagen"
    classname="com.sun.tools.jxc.SchemaGenTask">
    <classpath refid="classpath" />
  </taskdef>
  <target name="compile"
    description="Compile all Java source files">
    <echo message="Generating schemas..." />
    <mkdir dir="schemas" />
    <schemagen destdir="schemas">
      <src path="src" />
      <classpath refid="classpath" />
    </schemagen>
    <echo message="Compiling the java source files..." />
    <mkdir dir="classes" />
```

```

        <javac destdir="classes" debug="on">
            <src path="src" />
            <classpath refid="classpath" />
        </javac>
    </target>
    <target name="run" depends="compile"
        description="Run the sample app">
        <echo message="Running the sample application..." />
        <java classname="Main" fork="true">
            <classpath refid="classpath" />
        </java>
    </target>
    ...
</project>

```

以下は、この例で1つのスキーマを示すJavaクラスです。

- BusinessCard.java
- Address.java
- jaxb.index
- package-info.java
- Main.java

上記の項目のうちMain.javaについて簡単に説明します。

#### [例 20.6] << Main.java >>

```

BusinessCard card =
    new BusinessCard("John Doe", "Sr. Widget Designer", "Acme, Inc.",
        new Address(null, "123 Widget Way", "Anytown", "MA", (short) 12345),
        "123.456.7890", null, "123.456.7891", "John.Doe@Acme.ORG");

JAXBContext context = JAXBContext.newInstance(BusinessCard.class);
Marshaller m = context.createMarshaller();
m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
m.marshal(card, new FileOutputStream(new File("src/conf/bcard.xml")));

...

```

コンパイルされたJavaソースを使用してJavaオブジェクトを1つ作成し、Marshallerを1つ登録し、bcard.xmlというファイルにマーシャリングします。

**[例 20.7] << Main.java (続き) >>**

```
Unmarshaller um = context.createUnmarshaller();
Schema schema = SchemaFactory.newInstance(W3C_XML_SCHEMA_NS_URI)
    .newSchema(Main.class.getResource("schema1.xsd"));
um.setSchema(schema);
Object bce = um.unmarshal(new File("src/conf/bcard.xml"));
m.marshal(bce, System.out);
```

作成されたスキーマ・ファイルを使用してUnmarshallerを1つ作成し、前でマーシャリングしたbcard.xmlファイルをアンマーシャリングします。これを再び画面に出力(マーシャリング)します。

## 20.3. JAXP(XMLを扱うためのJava標準API)

XMLとJavaは、プラットフォームに独立的ということから、様々な方面で一緒に使用されてきました。JEUS 8 Webサービスは、XML文書を扱うためのJava標準のAPIをサポートします。この標準は、Javaの標準団体であるJCPが明示したXML文書を扱う方式に関する標準(<http://jcp.org/en/jsr/detail?id=206>)に従っています。JAXPの主要APIは以下のとおりです。

- SAX
- DOM
- TrAX
- DOM
- StAX

### 20.3.1. StAX(JavaストリーミングのXMLパーサ)

JEUS 8 Webサービスは、XMLをパースする際に使用するJavaストリーミング方式もサポートします。この標準は、Javaの標準団体であるJCPが明示したJavaストリーミングのXMLパーサのためのAPIに関する標準(<http://www.jcp.org/en/jsr/detail?id=173>)に従っています。



## Part II. JAX-RPCサービスのサポート

JEUS 8のWebサービスは、JAX-WS方式のWebサービスを公式サポートすると共に、JEUS 5 WebサービスでサポートしていたJAX-RPC Webサービスも下位互換性のために公式サポートします。今までJAX-WS Webサービスを構成してサービスし、クライアント・プログラムを作成してこれ呼び出すなど、様々なJAX-WSの機能について説明してきました。本パートからは、下位互換性のためのJAX-RPC Webサービスに関するJEUS 8 Webサービスのサポートについて説明します。

JAX-WS方式のWebサービスとJAX-RPC方式のWebサービスの違いについて簡単に説明すると、次のとおりです。まず、JAX-WS方式のWebサービスはJava EE 7 Webサービスの標準で、JAX-RPC方式のWebサービスはJ2EE 1.4 Webサービスの標準です。JEUS 8 WebサービスはJava EE 7 Webサービスの標準であるJAX-WS方式のWebサービスを公式サポートし、同時にJ2EE 1.4 Webサービス標準のJAX-RPC方式のWebサービスも下位互換性のためにサポートします。

JAX-RPC方式のWebサービスを実装するには、JAX-WS方式のWebサービスと比べて、デプロイメント記述子およびJAX-RPCマッピングファイルなどを追加で実装する必要があります。この後からの章では、このようなJAX-RPC WebサービスをJEUS 8 Webサービスがサポートする方法について詳しく説明します。



# 第21章 JAX-RPC Webサービスの実装

本章では、JAX-RPC Webサービスを実装する様々な方式について説明します。

## 21.1. 概要

JEUS JAX-RPC Webサービスの実装とは、Webサービスを構成するバックエンド・コンポーネントのJavaコードを作成およびコンパイルすることを意味します。基本的にJEUS JAX-RPC Webサービスは、Javaクラス、EJBという2つのタイプのWebサービスのバックエンド・コンポーネントをサポートします。

JEUS JAX-RPC Webサービスを実装する主な手順は以下のとおりです。

1. Webサービスを構成するバックエンド・コンポーネントのJavaコードを記述します。
2. SOAPメッセージの内容を内容を直接処理したり、SOAPメッセージの添付ファイルに直接アクセスしたりする必要がある場合は、SOAPメッセージ・ハンドラーおよびハンドラー・チェーンを直接作成します。
3. Javaコードをコンパイルします。

## 21.2. JavaクラスによるWebサービスの実装

JavaクラスWebサービスは、JAX-RPC Webサービスの実装においてWebサービスを作成する最も簡単な方法です。まず簡単な例を通じて実装方法について説明します。

### 21.2.1. 実装例

JavaクラスWebサービスを実装するには、サービス・エンドポイント・インターフェース(SEI)と実装クラスを定義する必要があります。

サービス・エンドポイント・インターフェースは、JavaクラスWebサービス・エンドポイントがJavaメソッドの形式でサポートするWebサービス・オペレーションを定義しています。実装クラスは、このようなサービス・エンドポイント・インターフェースを実装します。

以下のエンドポイント・インターフェースは、echoStringとechoString\_doubleというWebサービス・オペレーションを定義しています。

```
package jeustest.webservices.java2wsdl.doclit;  
  
public interface Echo extends java.rmi.Remote {  
    public String echoString(String arg11)
```

```

        throws java.rmi.RemoteException;
    public String echoString_double(String arg1, String arg2)
        throws java.rmi.RemoteException;
}

```

エンドポイント・インターフェースは、外部に公開されてアクセス可能なWebサービス・オペレーションを定義します。すなわち、SOAPというプロトコルを使用してアクセスできるオペレーションを定義します。エンドポイント・インターフェースは、java.rmi.Remoteインターフェースを直接的または間接的に拡張する必要があります。そして、定義されたメソッドはjava.rmi.RemoteExceptionタイプを返す必要があります。サービス・オペレーションを定義した後には、これを実装するロジックが必要となりますが、その役割を担うのが実装クラスです。

以下は、実装クラスの例です。

```

package jeustest.webservices.java2wsdl.doclit;

public class EchoImpl implements Echo {
    public String echoString(String input0)
        throws java.rmi.RemoteException {
        return input0;
    }

    public String echoString_double(String input0, String input1)
        throws java.rmi.RemoteException {
        return input0+input1;
    }
}

```

実装クラスはエンドポイント・インターフェースを実装しています。これはJava EEサーバー内でインスタンス化されて実行され、ランタイム時にはWebサービスで動作するはずです。

## 21.2.2. Javaクラスの作成規則

Javaクラスのバックエンドとして、Webサービスを実装するときに以下の規則を考慮する必要があります。

- サービス・エンドポイント・インターフェースを定義します。
- publicクラスを定義します。
- パラメータのないデフォルト生成子を定義します。
- Webサービスで、public、non-staticでエクスポートされるJavaクラスのメソッドを定義します。
- スレッドに安全な(Thread-safety)Javaコードを作成します。
- 相互運用性のためにオーバー・ローディングされたメソッドを使用してはいけません。

上記の条件を満たす場合、以下の手順に従ってWebサービスを実装します。

- Webサービス・メソッドを含んでいるJavaクラスを定義します。
- 明示的に公開するメソッドのためのインターフェースを定義します。

## 21.3. EJB Webサービスの実装

EJBは、Java EE Webサービスの開発において、JavaクラスのWebサービス・プログラミング・モデルより多少複雑ですが、多様な機能を提供します。このような複雑性は、トランザクションを自動で管理するようになったことで付随的に発生するものといえます。

EJB Webサービスを実装するには、EJBに関する理解がある程度必要です。Webサービスの実装においてEJBを使用しなくてもすむ場合、本章は飛ばしても構いません。

### 21.3.1. 実装例

ステートレス・セッションEJBは、Webサービスでエクスポートされる場合があります。この場合にもサービス・エンドポイント・インターフェースが必要です。

以下のエンドポイント・インターフェース(HelloIF.java)は、sayHello(String)Webサービス・オペレーションを定義しています。

**[例 21.1] << HelloIF.java >>**

```
package helloejb;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface HelloIF extends Remote {
    public String sayHello(String s) throws RemoteException;
}
```

エンドポイント・インターフェースは、外部に公開されてアクセス可能なWebサービス・オペレーションを定義します。すなわち、SOAPというプロトコルを使用してアクセスできるオペレーションを定義します。エンドポイント・インターフェースは、java.rmi.Remoteインターフェースを直接的または間接的に拡張する必要があります。そして、定義されたメソッドは、java.rmi.RemoteExceptionタイプを返す必要があります。サービス・オペレーションを定義した後は、これを実装するロジックが必要となりますが、ここではEJBがその役割を果たします。

以下はリモート・インターフェースの例です。

### [例 21.2] << Hello.java >>

```
package helloejb;

import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Hello extends EJBObject
{
    String sayHello(String s) throws RemoteException;
}
```

以下はホーム・インターフェースの例です。

### [例 21.3] << HelloHome.java >>

```
package helloejb;

import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface HelloHome extends EJBHome
{
    Hello create() throws RemoteException, CreateException;
}
```

以下は、セッションBeanを実装したEJBエンドポイントBeanクラスです。

### [例 21.4] << HelloEJB >>

```
package helloejb;

import java.rmi.RemoteException;

import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import java.lang.*;

public class HelloEJB implements SessionBean {

    public HelloEJB() {}
    public String sayHello(String s) throws RemoteException {
        try {
            Thread.currentThread().sleep(500);
        } catch (Exception ex) {
            throw new RemoteException("" + ex);
        }
        return "Hello World!" + s;
    }
}
```

```

public void ejbCreate() {}
public void ejbRemove() throws RemoteException {}
public void setSessionContext(SessionContext sc) {}
public void ejbActivate() {}
public void ejbPassivate() {}
}

```

### 21.3.2. EJB Webサービスの作成規則

Webサービス・オペレーションがone-wayに設定されている場合、EJBメソッドのJavaコードはvoid型を返す必要があります。この条件を除くと、ステートレス・セッションEJB Webサービスのためのコーディングは一般EJBコーディングと同じです。

上記の条件を満たす場合、以下の手順に従ってWebサービスを実装します。

1. Webサービス・メソッドを含むEJBを実装します。
2. サービスで明示的に公開するメソッドのためのインターフェースを定義します。

## 21.4. WSDLによるWebサービスの実装

本書の大部分は、Webサービスを構成するバックエンド・コンポーネントのJavaコードの作成から始めることを仮定しています。

開発環境によって、WSDLからサービス・インターフェースを作成できます。この場合、ユーザーはwsdl2java Ant Taskを実行し、サービス・インターフェースを取得できます。wsdl2java Ant Taskは、ユーザーが作成したWSDLファイル、あるいは持っているWSDLファイルを入力します。

以下は、wsdl2java Antを実行するためのbuildファイルの例です。

#### [例 21.5] << build.xml >>

```

<target name="do-package-war">

    ...

    <antcall target="wsdl2java">
        <param name="wsdl2java.option"
            value="-import:server -d ${build.war.dir}/WEB-INF/classes
                -package sample.datahandleronly.service
                -outputmapping ${build.war.dir}/WEB-INF/SubmitBookService-mapping.xml
                -compile ${src.web}/WEB-INF/wsdl/SubmitBookService.wsdl" />
    </antcall>

    ...

```

```
</target>
```

以下のコマンドを使用してWebサービス・インターフェース・ソースコードを作成できます。

```
$ ant do-package-war
```

ユーザーは、作成されたサービス・インターフェースのサービス実装を「[21.2. JavaクラスによるWebサービスの実装](#)」と「[21.3. EJB Webサービスの実装](#)」の実装方式に従って作成します。

---

#### 参考

wsdl2java Taskの詳細については『*JEUS リファレンスガイド*』の「4.10. wsdl2java」または『*JEUS リファレンスガイド*』の「5.5.2. wsdl2java」を参照してください。

---

## 21.5. SAAJの使用

通常のSOAPメッセージはSOAPのボディー内に含まれますが、特定のJavaタイプをWebサービス・オペレーションを実装するメソッドのパラメータや戻り値として使用する場合はSOAP添付ファイル形式で送信されます。

JEUS Webサービスは、SAAJ(SOAP with Attachments API for Java)を使用するためにJavaデータ型からMIME型への型転換を以下のように定義しています。

**[表 21.1] Required Mappings : Java to MIME**

Java型	MIME型
java.awt.Image	image/gif or image/jpeg
java.lang.String	text/plain
javax.mail.internet.MimeMultipart	multipart/*
javax.xml.transform.Source	text/xml or application/xml

上記の各MIME型について、JAX-RPC Endpoint Stubが特定のJavaデータ型を適切にエンコードされたデータのストリームに変換または逆変換作業を行います。詳細については「[第24章 JAX-RPC WebサービスのSOAPメッセージ・ハンドラーの作成](#)」を参照してください。



# 第22章 JAX-RPC Webサービスの作成とデプロイ

本章では、JavaクラスとEJBをエンドポイントとして持っているJAX-RPC Webサービスの作成とデプロイ方法について説明します。

## 22.1. JavaクラスWebサービスの作成とデプロイ

JEUS JAX-RPC Webサービスの作成は、開発の便宜を図るためにコマンドライン・ツールとApache Antツールを使用します。JAX-RPC Webサービスを作成およびデプロイするには、以下の手順に従います。

1. サービス設定ファイル(service-config.xml)の作成
2. Java EE WebサービスのためのWSDLとJAX-RPCマッピング・ファイルの作成
3. WebサービスのDDファイル(webservices.xml、jeus-webservices-dd.xml)の作成
4. 作成されたWebサービス・モジュールのパッケージングとデプロイ

---

### 参考

詳細については『*JEUS リファレンスガイド*』の「4.9. java2wsdl」、『*JEUS リファレンスガイド*』の「4.10. wsdl2java」、『*JEUS リファレンスガイド*』の「5.5.1. java2wsdl」、『*JEUS リファレンスガイド*』の「5.5.2. wsdl2java」を参照してください。

---

### 22.1.1. サービス設定ファイルの作成

JEUS JAX-RPC Webサービスは、service-config.xmlのように、Webサービスを作成するための設定をXMLファイルに保存します。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <service>
    <service-name>DocLitEchoService</service-name>
    <target-namespace>urn:DocLitService</target-namespace>
    <output-wsdl-file>DocLitEchoService.wsdl</output-wsdl-file>
    <output-jaxrpc-mapping-file>
```

```

        DocLitEchoService-mapping.xml
    </output-jaxrpc-mapping-file>
    <style>wrapped</style>
    <interface>
        <endpoint-interface-class>
            jeustest.webservices.java2wsdl.doclit.Echo
        </endpoint-interface-class>
    </interface>
</service>
</web-services-config>

```

上記例では、<service-name>に設定されている「DocLitEchoService」という名前のWebサービスが作成されます。作成されるWSDLとマッピング・ファイルは、それぞれDocLitEchoService.wsdlとDocLitEchoService-mapping.xmlになります。

サービス・スタイルは文書方式のうちWRAPPED方式で、Webサービス・エンドポイントはjeustest.webservices.java2wsdl.doclit.EchoというJavaクラス・ファイルで設定されています。

---

#### 参考

service-config.xmlファイルの詳細については『JEUS  
jeus-webservices-config.xml設定』を参照してください。

XMLリファレンスガイド』の「25.

---

## 22.1.2. WSDLファイルとJAX-RPCマッピング・ファイルの作成

JEUS JAX-RPC Webサービスは、Webサービスを作成するために、コマンドライン・ツール方式とAntタスク方式を提供します。どちらの方式を選んでも構わず、ユーザーの目的と様々な状況によって選択します。

### コマンドライン・ツールの使用

JAX-RPC Webサービスは、Webサービスを作成するためのコマンドライン・ツールを提供します。

```

Usage: java2wsdl [options] configuration_file
where [options] include:
-classpath <path>          specify where to find input class files
-cp <path>                 same as -classpath <path>
-d <directory>            specify where to place generated output files
-verbose                   [optional] turn verbose mode on

```

「22.1.1. サービス設定ファイルの作成」で作成したservice-config.xmlファイルを使用してWebサービスを作成するには、コマンドラインで以下のように入力してコマンドを実行します。

以下は、JEUS\_HOME/sample/classesの下にコンパイルされたJavaクラスのバックエンド・ファイルがある場合にコマンドを実行する例です。

```

JEUS_HOME/sample$java2wsdl -cp ./classes service-config.xml

```

上のようにコマンドを実行すると、JAX-RPCマッピング・ファイルのDocLitEchoService-mapping.xmlとWSDLファイルのDocLitEchoService.wsdlが作成されます。

## Antツールの使用

JEUS JAX-RPC Webサービスは、Webサービスを作成するためのAntタスクである`java2wsdl`を提供します。`java2wsdl`は、サービス設定ファイルの位置の入力により、WSDLファイルとJAX-RPCマッピング・ファイルを作成します。

以下のように実行し、Javaクラス・ファイルをコンパイルします。

```
$ ant compile
```

上のコマンドを実行すると、WSDLファイルとJAX-RPCマッピング・ファイルも「./build」ディレクトリーに自動作成されます。

コンパイルAntタスクは、内部的に-pre-compile > do-compile > -post-compileという順で行われ、WSDLファイルとJAX-RPCマッピング・ファイルを作成する-post-compileは以下のように構成されています。

```
<target name="-post-compile">
  <mkdir dir="${build.war.dir}/WEB-INF/wsdl" />
  <antcall target="java2wsdl">
    <param name="java2wsdl.option"
      value="-classpath ${build.classes.dir}
        -d ${build.war.dir}/WEB-INF ${src.conf}/service-config.xml" />
  </antcall>
</target>
```

---

### 参考

`java2wsdl` Antタスクの詳細については『*JEUS リファレンスガイド*』の「5.5.1. `java2wsdl`」を参照してください。

---

## 22.1.3. WebサービスのDDファイルの作成

WebサービスのDDファイルは、Webサービスのデプロイに関して記述したXMLファイルです。Webサービスのデプロイに関する情報と、Webサービス・バックエンドを発見する方法についての情報をWebサービス・エンジンに提供します。

WebサービスのDDファイルには、Java EE Webサービス仕様に規定されているDDファイルである`webservices.xml`とJEUS WebサービスのためのDDファイルである`jeus-webservices-dd.xml`があります。

## Java EE WebサービスのDDファイルの作成

Java EE WebサービスのDDファイル名は必ず`webservices.xml`に指定します。

### [例 22.1] << webservices.xml >>

```
<?xml version="1.0"?>
<webservices version="1.1" xmlns="http://java.sun.com/xml/ns/j2ee">
  <webservice-description>
    <webservice-description-name>DocLitEchoService
    </webservice-description-name>
    <wsdl-file>WEB-INF/wsdl/DocLitEchoService.wsdl</wsdl-file>
    <jaxrpc-mapping-file>
      WEB-INF/DocLitEchoService-mapping.xml
    </jaxrpc-mapping-file>
    <port-component>
      <port-component-name>EchoPort</port-component-name>
      <wsdl-port xmlns:ns2="urn:DocLitService">
        ns2:EchoPort
      </wsdl-port>
      <service-endpoint-interface>
        jeustest.webservices.java2wsdl.doclit.Echo
      </service-endpoint-interface>
      <service-impl-bean>
        <servlet-link>EchoServlet</servlet-link>
      </service-impl-bean>
    </port-component>
  </webservice-description>
</webservices>
```

DocLitEchoServiceというサービス名を持ち、WSDLファイルはWEB-INF/wsdlディレクトリーに存在し、DocLitEchoService.wsdlという名前を持っています。JAX-RPCマッピング・ファイルはWEB-INFディレクトリーに存在し、DocLitEchoService-mapping.xmlという名前を持っています。このサービスにアクセスするには、WSDLに表記されているポートのうちEchoPortという名前のポートを使用し、このポートに対するSEIとサーブレットが定義されます。詳細については「[第25章 JAX-RPC Webサービス設定ファイルの作成](#)」を参照してください。

## JEUS WebサービスのDDファイルの作成

JEUS WebサービスのDDファイル名はjeus-webservices-dd.xmlに指定します。

### [例 22.2] << jeus-webservices-dd.xml >>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-webservices-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <service>
    <webservice-description-name>
      DocLitEchoService
    </webservice-description-name>
    <port>
      <port-component-name>EchoPort</port-component-name>
    </port>
```

```
</service>
</jeus-webservices-dd>
```

---

#### 参考

DDファイルについての詳細は『JEUS XMLリファレンスガイド』の「23. jeus-webservices-dd.xml」を参照してください。

---

## 22.1.4. パッケージングとデプロイ

Java クラスWebサービスは、JavaクラスとWebサービスのDDファイルをWebモジュールのようにパッケージングします。Webサービスは、WebアプリケーションWARファイルとEJB JARファイルを含む、スタンダード・エンタープライズ・アプリケーション(Enterprise Application)EARファイルでパッケージングされます。

---

#### 参考

JEUS WebコンテナのコンテキストとWebアプリケーションの詳細については『JEUS Webエンジンガイド』の「第3章 Webコンテキスト」を参照してください。

---

### 22.1.4.1. サブレットのDDファイルの作成

サブレットのDDファイル(web.xml)の一般的な役割は、サブレットとJSPコンポーネントのランタイム属性を記述することです。JavaクラスWebサービスは、ランタイム時にサブレットに組み込み形式で連動するため、web.xmlファイルはJavaクラスWebサービスをデプロイする際に必要です。

以下はweb.xmlファイルの作成例です。

#### [例 22.3] << web.xml >>

```
<?xml version="1.0"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
  <servlet>
    <servlet-name>EchoServlet</servlet-name>
    <servlet-class>
      jeustest.webservices.java2wsdl.doclit.EchoImpl
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>EchoServlet</servlet-name>
    <url-pattern>/DocLitEchoService</url-pattern>
  </servlet-mapping>
</web-app>
```

<servlet-class>は実際にサービスを実装したロジックが入っているJavaクラスの名前をテキスト・ノードの値で設定し、<servlet-mapping>の配下に<url-pattern>は実際にサービスにアクセスするためのURL値を設定します。

### 22.1.4.2. JEUS WebモジュールのDDファイルの作成

JEUS WebモジュールのDDファイルとは、デプロイ対象のWebモジュールのコンテキストを定義したファイルです。名前はjeus-web-dd.xmlです。

以下は、DocLitEchoServiceをコンテキストとして持つサービスのJEUS WebモジュールのDDファイルです。

#### [例 22.4] << jeus-web-dd.xml >>

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <context-path>/DocLitEchoService</context-path>
  <enable-jsp>true</enable-jsp>
  <auto-reload>
    <enable-reload>false</enable-reload>
    <check-on-demand>false</check-on-demand>
  </auto-reload>
  <max-instance-pool-size>-1</max-instance-pool-size>
  <session-config>
    <tracking-mode>
      <url>false</url>
    </tracking-mode>
  </session-config>
</jeus-web-dd>
```

### 22.1.4.3. WARパッケージング

WARファイルは拡張子が「.war」であるJAR圧縮ファイルで、圧縮と圧縮解除アルゴリズムのzlibアルゴリズム標準に従って圧縮されるファイルです。WARファイルは、サーブレット、JSPのようなWebコンポーネントをパッケージングする用途でのみ使用されます。以下のように特定のディレクトリ構造を持つ必要があります。

```
WAR
|
+-- WEB-INF
    |-- web.xml (サーブレットのDD)
    |-- webservices.xml (標準WebサービスのDD)
    |-- jeus-web-services-dd.xml (JEUS WebサービスのDD)
    |-- jeus-web-dd.xml (JEUS WebモジュールのDD)
    |-- Jax-rpc mappingファイル
+-- wsdl
    |
    +-- wsdlファイル
|
+-- classes
```

```
|
+--- Javaクラス・コンポーネント、ハンドラー実装など
```

前章で作成したWSDLファイル、JAX-RPCマッピング・ファイル、標準WebサービスのDD(webserivices.xml)、JEUS WebサービスのDD(jeus-webservices-dd.xml)、サーブレットのDD(web.xml)、JEUS WebモジュールのDDをWARファイル形式でパッケージングします。

JAX-RPCマッピング・ファイルとWSDLファイルがwebservices.xmlファイルの<wsdl-file>と<jax-rpc-mappng-file>で記述した位置に存在する場合、必ずしも上の構造に従わなくても構いません。

#### 22.1.4.4. EARパッケージングとデプロイ

本節では、EARパッケージングとデプロイについて説明します。

##### EARパッケージング

Java EEアプリケーション・プログラムは、WebコンポーネントやEJB、Java EEコネクターを使用できる独立的なビジネス・ソリューションで、EAR(Enterprise ARchive)ファイルでパッケージングできます。EARファイルはアプリケーション・プログラムのXML DDを持っており、Java EEコンポーネントとコネクターなどを、EJB JARやWAR、あるいはRARファイル形式でパッケージングします。

JavaクラスWebサービスは、サーブレット・プログラミング・モデルの上で作成されたため、WARファイル形式でパッケージングされます。また、EJBエンドポイントはSOAPメッセージを扱う場所がEJBであるため、EJB JARファイル形式でパッケージングされます。

```
EAR
|
+--- WARファイル
|
+--- JARファイル
|
|   +--- EJB Webサービス・コンポーネント
|
+--- META-INF
|
|   +--- 標準application.xmlファイル
```

META-INFディレクトリー内の標準application.xmlファイルでは、EARファイルにパッケージングしたJava EEコンポーネントを記述します。

以下は、WebモジュールをJava EEコンポーネントとして持っているEARアプリケーション・プログラムのapplication.xmlファイルの作成例です。

##### [例 22.5] << application.xml >>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application version="6"
  xmlns="http://java.sun.com/xml/ns/javaee">
```

```
<module>
  <web>
    <web-uri>DocLitEchoService.war</web-uri>
    <context-root>DocLitEchoService</context-root>
  </web>
</module>
</application>
```

<web>では、Webモジュール・パッケージングの名前と、モジュールのコンテキストについての情報を設定します。

## Webサービスのデプロイ(EARアプリケーション・プログラムのデプロイ)

Webサービスのデプロイは、一般的なJava EEアプリケーション・プログラムのデプロイと同じです。EARファイルをデプロイすることによって、Webサービスはインターネットでアクセス可能なサービスとして公開されます。

実際にアクセス可能な、デプロイされたサービスのURLアドレスは以下のとおりです。

```
http://host:port/DocLitEchoService/DocLitEchoService
```

## 22.2. EJB Webサービスの作成とデプロイ

EJBを利用したWebサービスを実装したら、実際にEJBのビジネス・ロジックをWebサービスに転換し、デプロイする必要があります。EJB Webサービスを作成し、デプロイする方法は、JavaクラスWebサービスを作成し、デプロイする方法と同じです。一方、開発の便宜を図るために、JEUS JAX-RPC Webサービスの作成はコマンドライン・ツールとApache Antツールを使用します。

全体的な流れは以下のとおりです。

1. サービス設定ファイルの作成(service-config.xml)
2. Java EE WebサービスのためのWSDLとJAX-RPCマッピング・ファイルの作成
3. WebサービスのDDファイルの作成(web services.xml、jeus-web services-dd.xml、ejb-jar.xml)
4. 作成されたWebサービス・モジュールのパッケージングとデプロイ

---

### 参考

詳細については、『JEUS リファレンスガイド』の「4.9. java2wsdl」、『JEUS リファレンスガイド』の「4.10. wsdl2java」、『JEUS リファレンスガイド』の「5.5.1. java2wsdl」、『JEUS リファレンスガイド』の「5.5.2. wsdl2java」を参照してください。

---



## 22.2.1. サービス設定ファイルの作成

以下は、サービス設定ファイルのservice-config.xmlの作成例です。

### [例 22.6] << service-config.xml >>

```
<?xml version="1.0"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <service>
    <service-name>AddressBookService</service-name>
    <target-namespace>urn:AddressBookService</target-namespace>
    <style>wrapped</style>
    <use>literal</use>
    <interface>
      <endpoint-interface-class>
        address.AddressBookIF
      </endpoint-interface-class>
    </interface>
  </service>
</web-services-config>
```

上記例では、<service-name>に設定されているAddressBookServiceという名前のWebサービスが作成されます。作成されるWSDLとマッピング・ファイルは、それぞれAddressBookService.wsdlとAddressBookService-mapping.xmlになります。

サービス・スタイルは文書方式のうちWRAPPED方式で、Webサービス・エンドポイントはaddress.AddressBookIFというJavaクラス・ファイルで設定されています。

---

#### 参考

service-config.xmlファイルの詳細については『JEUS jeus-webservices-config.xml設定』を参照してください。

XMLリファレンスガイド』の「25.

## 22.2.2. WSDLファイルとJAX-RPCマッピング・ファイルの作成

JEUS JAX-RPC Webサービスは、Webサービスを作成するためのコマンドライン・ツール方式とAntタスク方式を提供します。どちらの方式を選んでも構わず、ユーザーの目的と様々な状況によって選択します。

### コマンドライン・ツールの使用

JEUS JAX-RPC Webサービスは、Webサービスを作成するためのコマンドライン・ツールを提供します。

```
Usage: java2wsdl [options] configuration_file

where [options] include:
-classpath <path>    specify where to find input class files
-cp <path>           same as -classpath <path>
```

```
-d <directory>      specify where to place generated output files
-level <log-level>   specify a log level
-verbose            [optional] turn verbose mode on
```

前章で作成したservice-config.xmlファイルを使用してWebサービスを作成するには、コマンドラインで以下のように入力してコマンドを実行します。

以下の例は、コンパイルされたJavaクラス・バックエンド・ファイルがJEUS\_HOME/sample/classesに存在する場合です。

```
JEUS_HOME/sample$java2wsdl -cp ./classes ejb-service-config.xml
```

上のようコマンドを実行すると、JAX-RPCマッピング・ファイルのAddressBookService-mapping.xmlとWSDLファイルのAddressBookService.wsdlが作成されます。

## Antツールの使用

JEUS JAX-RPC Webサービスは、Webサービスを作成するためのAntタスクであるjava2wsdlを提供します。java2wsdlは、サービス設定ファイルの位置の入力により、WSDLファイルとJAX-RPCマッピング・ファイルを作成します。

以下のコマンドを実行して、Javaクラス・ファイルをコンパイルします。

```
$ ant compile
```

上のコマンドを実行すると、WSDLファイルとJAX-RPCマッピング・ファイルも「./build」ディレクトリーに自動作成されます。

コンパイルAntタスクは、内部的に-pre-compile > do-compile > -post-compileという順で行われ、WSDLファイルとJAX-RPCマッピング・ファイルを作成する-post-compileは以下のように構成されています。

```
<target name="-post-compile">
  <mkdir dir="${build.classes.dir}/META-INF/wsdl" />
  <antcall target="java2wsdl">
    <param name="java2wsdl.option"
      value="-classpath ${build.classes.dir}
      -d ${build.classes.dir}/META-INF ${src.conf}/ejb-service-config.xml" />
  </antcall>
</target>
```

サービス設定ファイルのパスを<java2wsdl>の「configfilepath」属性に入力し、コンパイルされたJavaクラス・ファイルのパスを<classpath>の「refid」属性に入力します。上記のコマンドを実行すると、WSDLファイルとJAX-RPCマッピング・ファイルが作成されます。

---

### 参考

java2wsdl Taskの詳細については『JEUS リファレンスガイド』の「5.5.1. java2wsdl」を参照してください。

---

### 22.2.3. WebサービスのDDファイルの作成

WebサービスのDDファイルは、Webサービスのデプロイに関して記述したXMLファイルです。Webサービスのデプロイに関する情報と、Webサービス・バックエンドを発見する方法についての情報をWebサービス・エンジンに提供します。

WebサービスのDDファイルには、Java EE Webサービス仕様に規定されているDDファイルである `webservices.xml` と JEUS WebサービスのためのDDファイルである `jeus-webservices-dd.xml` があります

### Java EE WebサービスのDDファイルの作成

Java EE WebサービスのDDファイル名は必ず `webservices.xml` にします。

#### [例 22.7] << webservices.xml >>

```
<?xml version="1.0"?>
<webservices version="1.1" xmlns="http://java.sun.com/xml/ns/j2ee">
  <webservice-description>
    <webservice-description-name>
      AddressBookService
    </webservice-description-name>
    <wsdl-file>
      META-INF/wsdl/AddressBookService.wsdl
    </wsdl-file>
    <jaxrpc-mapping-file>
      META-INF/AddressBookService-mapping.xml
    </jaxrpc-mapping-file>
    <port-component>
      <port-component-name>
        AddressBookIFPort
      </port-component-name>
      <wsdl-port xmlns:ns2="urn:AddressBookService">
        ns2:AddressBookIFPort
      </wsdl-port>
      <service-endpoint-interface>
        address.AddressBookIF
      </service-endpoint-interface>
      <service-impl-bean>
        <ejb-link>AddressEJB</ejb-link>
      </service-impl-bean>
    </port-component>
  </webservice-description>
</webservices>
```

AddressBookServiceというサービス名を持ち、WSDLファイルはJARファイルのMETA-INF/wsdlディレクトリに存在し、AddressBookService.wsdlという名前を持っています。JAX-RPCマッピング・ファイルはMETA-INFディレクトリに存在し、AddressBookService-mapping.xmlという名前を持っています。

このサービスにアクセスするには、WSDLに表記されているポートのうちAddressBookIFPortという名前のポートを使用し、このポートに対するSEIとEJBが定義されます。

## JEUS WebサービスのDDファイルの作成

JEUS WebサービスのDDファイル名はjeus-webservices-dd.xmlに指定します。

### [例 22.8] << jeus-webservices-dd.xml >>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-webservices-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <ejb-context-path>webservice</ejb-context-path>
  <service>
    <webservice-description-name>
      AddressBookService
    </webservice-description-name>
    <port>
      <port-component-name>
        AddressBookIFPort
      </port-component-name>
      <ejb-endpoint-url>
        /AddressBookService
      </ejb-endpoint-url>
    </port>
  </service>
</jeus-webservices-dd>
```

JEUS WebサービスのDDファイルには、EJB WebサービスにアクセスするためのURLパスを記述します。URLパスは、コンテキスト・パスとURLパターンを含みます。コンテキスト・パスは<ejb-context-path>に指定します。URLパターンは<port>配下の<ejb-endpoint-url>に指定します。

上記の場合、WebサービスにアクセスするためのURLアドレスは以下のとおりです。

```
http://host:port/webservice/AddressBookService
```

## 22.2.4. パッケージングとデプロイ

EJB Webサービスは、EJB JARパッケージ内にWebサービスのDDファイルとWSDLをパッケージングします。Webサービスは、WebアプリケーションWARファイルとEJB JARファイルを含む、スタンダード・エンタープライズ・アプリケーション(Enterprise Application)EARファイルでパッケージングされます。

### 22.2.4.1. EJB DDファイルの作成

以下は、ejb-jar.xmlファイルの例です。

### [例 22.9] << ejb-jar.xml >>

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd"
  version="2.1">
<display-name>AddressEJB</display-name>
  <enterprise-beans>
    <session>
      <display-name>AddressEJB</display-name>
      <ejb-name>AddressEJB</ejb-name>
      <service-endpoint>
        address.AddressBookIF
      </service-endpoint>
      <ejb-class>address.AddressBookEJB</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

<ejb-class>は実際にサービスを実装したロジックが入っているBeanクラスの名前をテキスト・ノードの値で設定します。<service-endpoint>はEJB WebサービスのSEIクラスの名前をテキスト・ノードの値で設定します。

以下は、JEUSに必要な別のEJB DDです。ファイル名はjeus-ejb-dd.xmlです。

### [例 22.10] << jeus-ejb-dd.xml >>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <beanlist>
    <jeus-bean>
      <ejb-name>AddressEJB</ejb-name>
      <export-name>AddressEJB</export-name>
    </jeus-bean>
  </beanlist>
</jeus-ejb-dd>
```

## 22.2.4.2. JARパッケージング

これまでに作成したEJB実装クラスとSEI、WebサービスのDD、EJB DDを1つのJARファイルにパッケージングします。

JARパッケージングした構造は以下のとおりです。

```
JAR
|
```

```

+-- META-INF
    |-- ejb-jar.xml (標準EJB DD)
    |-- webservices.xml (標準WebサービスのDD)
    |-- jeus-webservices-dd.xml (JEUS WebサービスのDD)
    |-- jeus-ejb-dd.xml (JEUS EJB DD)
+-- Jax-rpc mappingファイル
+-- wsdl
    |
    +-- wsdlファイル
    |
+-- EJBクラス・コンポーネント, J2EE Portable Artifact

```

JAX-RPCマッピング・ファイルとWSDLファイルがwebervices.xmlファイルの<wsdl-file>と<jax-rpc-mapping-file>で記述した位置に存在する場合、必ずしも上の構造に従わなくても構いません。

### 22.2.4.3. EARパッケージングとデプロイ

本節では、EARパッケージングとデプロイについて説明します。

#### EARパッケージング

Java EEアプリケーション・プログラムは、WebコンポーネントやEJB、Java EEコネクタを使用できる独立したビジネス・ソリューションで、EAR(Enterprise ARchive)ファイルでパッケージングできます。EARファイルはアプリケーション・プログラムのXML DDを持っており、Java EEコンポーネントとコネクタなどを、EJB JARやWAR、あるいはRARファイル形式でパッケージングします。

JSE(JAX-RPC Service Endpoint)は、サーブレット・プログラミング・モデルの上で作成されたため、WARファイル形式でパッケージングされます。また、EJBエンドポイントはSOAPメッセージを扱う場所がEJBであるため、EJB JARファイル形式でパッケージングされます。

```

EAR
|
+-- WARファイル
    |
    +-- JSE(JAX-RPC Servlet Endpoint)Webサービス・コンポーネント
    |
+-- JARファイル
    |
    +-- EJB Webサービス・コンポーネント
    |
+-- META-INF
    |
    +-- 標準application.xmlファイル

```

META-INFディレクトリー内の標準application.xmlファイルでは、EARファイルにパッケージングしたJava EEコンポーネントを記述します。

以下は、WebモジュールをJava EEコンポーネントとして持っているEARアプリケーション・プログラムのapplication.xmlファイルの例です。

**[例 22.11] << application.xml >>**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application version="6"
  xmlns:ns1="http://java.sun.com/xml/ns/javaee">
  <module>
    <ejb>AddressBook.jar</ejb>
  </module>
</application>
```

<ejb>では、EJB WebサービスのJARパッケージングの名前の情報を持っています。

## Webサービスのデプロイ(EARアプリケーション・プログラムのデプロイ)

Webサービスのデプロイは、一般的なJava EEアプリケーション・プログラムのデプロイと同じです。EARファイルをデプロイすることによって、Webサービスはインターネットでアクセス可能なサービスとして公開されます。





# 第23章 JAX-RPC Webサービスの呼び出し

本章では、JAX-RPC Webサービスを呼び出すための様々なクライアント作成と呼び出し方法について説明します。

## 23.1. JAX-RPC Webサービスの呼び出し(Java SEクライアント)

Webサービスの呼び出しは、Webサービスを使用するためのクライアント・アプリケーション・プログラムが実行する動作を意味します。クライアント・アプリケーション・プログラムは、JavaまたはNETといった多様な技術を利用して、JEUSにデプロイされたWebサービスを呼び出すことができます。

Webサービス・クライアントは、Webサービスにサービスを要求するプログラムです。JEUS JAX-RPC Webサービスを使用して、2種類のWebサービス・クライアントを作成できます。

- **スタブ・クライアント**

このタイプのWebサービス・クライアントは、WebサービスのWSDLから作成されたスタブを使用します。

- **DII(Dynamic Invocation Interface)クライアント**

このタイプのWebサービス・クライアントは、JAX-RPCクライアントAPIを使用します。

### 23.1.1. スタブ・クライアント

スタブ・クライアントは特定のWebサービスのWSDLから生成されたローカル・スタブ上のメソッドを呼び出します。スタブ・オブジェクトはリモートWebサービスと相互適用を担当します。

本節では、「[第22章 JAX-RPC Webサービスの作成とデプロイ](#)」で作成したDocLitEchoService Webサービスのメソッドを呼び出すWebサービス・クライアントのプログラムを作成する方法について説明します。

#### 23.1.1.1. WSDLからWebサービスのスタブの作成

Webサービスのスタブ・ソースコードを作成するために、wsdl2java Antタスクやwsdl2javaコマンドを使用します。

Antタスクを使用する場合、build.xmlのコードは以下のとおりです。

**[例 23.1] << build.xml >>**

```
...  
<target name="-pre-compile">
```

```

<antcall target="wsdl2java">
  <param name="wsdl2java.option"
    value="-gen:client -d ${build.classes.dir}
      -package echo -compile -verbose ${src.conf}/DocLitEchoService.wsdl" />
</antcall>
</target>
...

```

## 参考

wsdl2java Taskの詳細については『*JEUS リファレンスガイド*』の「4.10. wsdl2java」を参照してください。

Webサービスのスタブ・ソースコードを生成するには、以下のコマンドを実行します。

```
wsclient$ ant -pre-compile
```

コマンドの実行に成功すると、Webサービスのスタブ・ソースコードが作成されます。スタブ・ソースコードは、wsdl2javaタスクのdestDir属性で指定したディレクトリーの下にコンパイルされます。

```

wsclient/build/classes/echo/
|
+-- DocLitEchoService.java
|
+-- DocLitEchoService_Impl.java
|
+-- Echo.java
|
+-- ... </screen>

```

以下は、作成されたクラスについての説明です。

クラス	説明
Echo.java	portTypeインターフェース・クラス
Echo_Stub.java	portTypeインターフェース・クラスのスタブ・クラス
DocLitEchoService.java	サービス・インターフェース・クラス
DocLitEchoService_Impl.java	サービス実装クラス
*.java	その他の作成クラス

作成されたJavaファイルまたはJavaファイル内のメソッドの名前はWebサービスのWSDLにマッピングされます。WSDLに対応するJavaクラスのマッピングは以下のとおりです。

WSDL要素	マッピング
<service>	サービス・インターフェースと実装Javaクラスにマッピングされます。

WSDL要素	マッピング
	<service>のname属性値がインターフェースの名前です。実装ファイルは<service>の名前の後ろに「_Impl」が付きます
<port>	<service>のサービス・インターフェースと実装クラス内のメソッドにマッピングされます。メソッドの名前は「get + <port>のname属性値」という構成です
<portType>	Webサービス・オペレーションのためのJavaインターフェースと実装Javaクラスにマッピングされます。Javaファイル名は<portType>のname属性を使用して作成されます。実装クラスのJavaファイル名は<portType>のname属性値に「_Stub」が付きます

### 23.1.1.2. Webサービス・クライアントの作成

本節では、Webサービス・クライアントを作成するために、上の4つのクラスをどのように使用するかについて説明します。

#### 1. Webサービス・オブジェクトの作成

リモートWebサービスのためにサービス実装オブジェクトを作成します。例では、サービス実装クラスはDocLitEchoService\_Implです。

以下は、Webサービス・オブジェクトを作成するためのJavaコードです。

```
DocLitEchoService service = new DocLitEchoService_Impl();
```

#### 2. スタブ・オブジェクトからのポート・オブジェクトの取得

サービス実装オブジェクトが作成されると、次にサービス実装オブジェクトからポート・オブジェクトを取得します。

サービス・インターフェース・クラスは、ポート・オブジェクトを取得するためのメソッドを提供します。メソッド名は「get + WSDLの<port>のname属性値」で構成され、WSDLの<portType>のname属性を名前を持つタイプが返されます。WSDLポート名はWebサービスのWSDL文書の<port>内に記述されています。<port>は<service>の下位要素です。

この例におけるWSDL文書は以下のとおりです。

```
<wsdl:portType name="Echo">
    . . .
</wsdl:portType>
. . .
<wsdl:service name="DocLitEchoService">
    <wsdl:port binding="impl:EchoSoapBinding" name="EchoPort">
        <soap:address location="http://localhost:8088/DocLitEchoService/DocLitEchoService"/>
    </wsdl:port>
</wsdl:service>
```

```
</wsdl:port>
</wsdl:service>
```

以下は、サービス・インターフェース・クラスのDocLitEchoService.javaの一部です。

```
<package echo;

public interface DocLitEchoService extends javax.xml.rpc.Service {
    public java.lang.String getEchoPortAddress();

    public echo.Echo getEchoPort()
        throws javax.xml.rpc.ServiceException;
}
```

サービス・オブジェクトからポート・オブジェクトを取得するための以下のソースコードを実装します。

```
DocLitEchoService service = new DocLitEchoService_Impl();
Echo port = service.getEchoPort();
```

### 3. ポート・オブジェクト上のオペレーションの実行

リモートWebサービスで提供するオペレーションのためのポート・オブジェクトを作成した場合、ポート・オブジェクトのメソッドを呼び出すことで、Webサービスのオペレーションを実行できます。例にて、Webサービスのポートは「echoString」というオペレーションを提供します。

以下は、オペレーションを実行するためのコードです。

```
DocLitEchoService service = new DocLitEchoService_Impl();
Echo port = service.getEchoPort();
String s = port.echoString("JEUS");
```

または、以下のコードで実装します。

```
Echo port = new DocLitEchoService_Impl().getEchoPort();
String s = port.echoString("JEUS");
```

以下はWebサービス・クライアントを実装した例で、wsclient/src/j2seディレクトリーにあると仮定します。

#### [例 23.2] <<ProxyClient.java>>

```
package j2se;

import echo.DocLitEchoService_Impl;
import echo.Echo;

public class ProxyClient {
    public static void main(String args[]) {
        ProxyClient client = new ProxyClient();
        client.run();
    }
}
```

```

public void run() {
    try {
        Echo port = new DocLitEchoService_Impl().getEchoPort();
        String s = port.echoString("JEUS");
        System.out.println("response = " + s);
    } catch (Exception e) {
        e.printStackTrace();
        System.exit(1);
    }
}
}

```

### 23.1.1.3. Webサービス・クライアントのコンパイル

ProxyClientコードをコンパイルするために、wsclientディレクトリーで以下のコマンドをコンソールに入力します。

```
JEUS_HOME/samples/webservice/jaxrpc/from_java/doclit/doclit-client$ ant build
```

以下は、コンパイルが成功した場合のコンパイルされたクラスです。

```
JEUS_HOME/samples/webservice/jaxrpc/from_java/doclit/doclit-client/build/classes/j2se/ProxyClient.class
```

### 23.1.1.4. Webサービス・クライアントの実行

Webサービス・クライアントが位置するディレクトリーに移動し、以下のコマンドを実行します。

```
JEUS_HOME/samples/webservice/jaxrpc/from_java/doclit/doclit-client$ ant run
```

実行結果は以下のとおりです。

```
Response = JEUS
```

## 23.1.2. DIIクライアント

DIIを使用することによって、リモート・オペレーションの署名やWebサービスの名前を知らなくても、リモート・オペレーションを呼び出すことができます。

本節では、DIIクライアントを作成する方法について説明します。

### 23.1.2.1. DIIクライアントの作成

DIIクライアントを作成する際はJAX-RPC 1.1 APIを使用します。JEUS JAX-RPC WebサービスはJAX-RPC 1.1 APIをサポートします。本節では、JAX-RPC APIの簡単な使用方法について説明します。

---

#### 参考

JAX-RPCの詳細については、JAX-RPCの仕様(<http://www.jcp.org/en/jsr/detail?id=101>)とOracleのJAX-RPCホームページ(<http://www.oracle.com/technetwork/java/docs-142876.html>)を参照してください。

---

DIIの呼び出しは、RPC方式のWebサービスを呼び出す場合にのみ使用できます。例では、RPC方式のRpcEncEchoServiceのechoStringオペレーションを呼び出すDIIクライアントを作成します。

DIIクライアントの作成方法は以下のとおりです。

1. 以下の構文をDIIクライアント・コードに追加します。

```
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.ParameterMode;
import javax.xml.namespace.QName;
```

2. ServiceFactoryとServiceオブジェクトを作成します。

```
String targetNamespace = "urn:RpcEncService";
String serviceName = "RpcEncService";
ServiceFactory factory = ServiceFactory.newInstance();
Service service = factory.createService(new QName(targetNamespace, serviceName));
```

3. Callオブジェクトを作成し、設定します。

```
String operationName = "echoString";
QName QNAME_XSD_STRING = new QName("http://www.w3.org/2001/XMLSchema", "string");
Call call = (Call) service.createCall();
call.setTargetEndpointAddress(endpoint);
call.setOperationName(new QName(targetNamespace, operationName));
call.addParameter("String_1", QNAME_XSD_STRING, ParameterMode.IN);
call.setProperty(Call.OPERATION_STYLE_PROPERTY, "rpc");
call.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY,
    "http://schemas.xmlsoap.org/soap/encoding/");
call.setReturnType(QNAME_XSD_STRING);
```

4. 最終的に、Callオブジェクト上でWebサービス・オペレーションを実行します。

```
String ret = (String)call.invoke(new Object[] { "JEUS" } );
```

以下は、DIIClientを実装した例で、wsclient/srcディレクトリーに位置すると仮定します。

**[例 23.3] << DIIClient.java >>**

```
package j2se;

import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.ParameterMode;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;

public class DIIClient {
    private static final String NS_XSD =
        "http://www.w3.org/2001/XMLSchema";
    private static final QName QNAME_XSD_STRING = new QName(NS_XSD, "string");

    public void run() {
        try {
            ServiceFactory factory = ServiceFactory.newInstance();
            String endpoint = "http://localhost:8088/" +
                "RpcEncEchoService/RpcEncEchoService";
            String targetNamespace = "urn:RpcEncService";
            Service service = factory.createService(
                new QName(targetNamespace, "RpcEncService"));
            Call call = service.createCall();

            call.setTargetEndpointAddress(endpoint);
            call.setOperationName(
                new QName(targetNamespace, "echoString"));
            call.addParameter("String_1", QNAME_XSD_STRING, ParameterMode.IN);
            call.setProperty(Call.OPERATION_STYLE_PROPERTY, "rpc");
            call.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY,
                "http://schemas.xmlsoap.org/soap/encoding/");
            call.setReturnType(QNAME_XSD_STRING);

            String ret = (String)call.invoke(new Object[] { "JEUS" });
            System.out.println("response = " + ret);
        } catch (Exception e) {
            System.err.println(e.toString());
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        DIIClient client = new DIIClient();
        try {
```

```

        client.run();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

### 23.1.2.2. DIIクライアントのコンパイル

DIIクライアント・コードをコンパイルするために、wsclientディレクトリーに移動し、以下のコマンドを実行します。

```
JEUS_HOME/samples/webservice/jaxrpc/from_java/rpcenc/rpcenc-client$ ant build
```

以下は、コンパイルが成功した場合のコンパイルされたクラスです。

```
JEUS_HOME/samples/webservice/jaxrpc/from_java/rpcenc/rpcenc-client/build/classes/j2se/DIIClient.class
```

### 23.1.2.3. DIIクライアントの実行

DIIクライアントが位置するディレクトリーに移動し、以下のコマンドを実行します。

```
JEUS_HOME/samples/webservice/jaxrpc/from_java/rpcenc/rpcenc-client$ ant run
```

以下は実行結果です。

```
Response = JEUS
```

## 23.2. JAX-RPC Webサービスの呼び出し(Java EEクライアント)

EJB、サーブレットのように、JEUSサーバーにデプロイされたコンポーネントからWebサービスを呼び出すのは、独立的なクライアントから呼び出す原理と本質的に同じです。しかし、現在のJava EE Webサービス仕様ではJava EE Webサービス・クライアントの移植性(Portability)のためにプログラミング・モデルを別途定義しており、このようなモデルに従ってクライアントを作成することをお勧めします。

### 23.2.1. Java EEクライアント・プログラミング・モデル

Webサービスは、クライアントの代わりにビジネス・ロジックを実行するメソッドの集合と言えます。クライアントはメソッドがローカルで実行されるのか、リモートで実行されるのかを区別できません。クライアントは、JAX-RPC仕様に定義されたSEIを使用してWebサービスにアクセスします。

Java EEクライアントは、JAX-RPCに定義されたサービス・インターフェースを実装したサービス・オブジェクトにアクセスするためにJNDIを使用します。サービス・オブジェクトは、クライアントがSEIを実装したスタブや



プロキシを取得するために使用するファクトリーです。サービス・インターフェースはJAX-RPCに定義されているjavax.xml.rpc.Serviceインターフェースであるか、これを継承して作成されたサービス・インターフェースです。

クライアント開発者は、SEIとサービス・インターフェースを取得することから始まり、これはJAX-RPCに定義されたWSDL→Javaマッピングの法則によって作成されます。クライアント開発者は、スタブの開発時点では作成しないことを推奨し、開発時点ではスタブではないインターフェースを使用できます。スタブは、クライアント・モジュールをデプロイする時点で、クライアントが駆動する環境に合わせて自動的に作成されます。WebサービスのJNDIルックアップは論理的な名前によって構成され、この名前はクライアントDDに定義されます。

## 23.2.2. Java EEクライアント・プログラミングの手順

Java EEクライアント・プログラミングの基本的な手順は以下のとおりです。

### 1. JNDIルックアップを使用したサービス(インターフェース)を取得します。

クライアント開発者はサービス・リファレンスとして扱われる論理的なサービスのJNDI名を、クライアントのDDに定義する必要があります。これは必須ではありませんが、すべてのサービスの論理的なリファレンス名をJNDI名前空間のserviceというサブ・コンテキストの下に構成することをお勧めします。

コンテナは、サービス・インターフェースの実装をクライアント環境コンテキスト(java:comp/env)の下にサービス・リファレンスの論理的な名前バイディングする必要があります。

```
InitialContext jndiContext = new InitialContext();
Service service =
    (Service) jndiContext.lookup( " java:comp/env/service/DocLitEchoService" );
```

上記例では、サービス・インターフェースjavax.xml.rpc.Serviceがweb.xmlの<service-ref>の下位要素である<service-interface>に設定されており、JNDI名は<service-ref-name>に設定されて互いにバイディングされます。

---

#### 参考

JAX-RPCでは、ServiceFactoryというクラスでサービスを作成できます。しかし、Java EEアプリケーション・プログラムでは、ServiceFactoryの使用を推奨しません。Java EEクライアントは、常にJNDIサービスのルックアップを使用してサービスを取得する必要があります。

---

### 2. サービス・インターフェースのAPIを利用したスタブまたはCallオブジェクトを作成します。

サービス・インターフェースは、クライアントがサービス・ポートにアクセスするために、スタブと動的プロキシ、あるいはDII Callオブジェクトを作成する場合に使用します。クライアント開発者は、アプリケーション・プログラムが使用するサービス・インターフェースのタイプを明示的にクライアントDD(web.xml)に宣言する必要があります。

以下は、サービス・インターフェースのタイプとJNDI名が明示されたweb.xmlの一例です。

### [例 23.4] << web.xml >>

```
<?xml version="1.0"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
  <servlet>
    <servlet-name>jsp_helloClient</servlet-name>
    <jsp-file>/helloClient.jsp</jsp-file>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <service-ref>
    <service-ref-name>
      service/DocLitEchoService
    </service-ref-name>
    <service-interface>
      javax.xml.rpc.Service
    </service-interface>
    <wsdl-file>
      WEB-INF/wsdl/DocLitEchoService.wsdl
    </wsdl-file>
    <jaxrpc-mapping-file>
      WEB-INF/DocLitEchoService-mapping.xml
    </jaxrpc-mapping-file>
    <port-component-ref>
      <service-endpoint-interface>
        echo.Echo
      </service-endpoint-interface>
    </port-component-ref>
  </service-ref>
</web-app>
```

### 3. スタブやCallオブジェクトを利用したWebサービスを呼び出します。

- スタブ・オブジェクトを利用した呼び出し

クライアントは、サービスのルックアップを利用して取得したサービス・インターフェースの、以下のような関数を利用して、スタブと動的なプロキシを作成できます。

```
java.rmi.Remote getPort(QName portName, Class serviceEndpointInterface)
    throws ServiceException;
java.rmi.Remote getPort(java.lang.Class serviceEndpointInterface)
    throws ServiceException;
```

- Callオブジェクトを利用した呼び出し

クライアントは、JNDIルックアップを利用して取得したサービス・インターフェースのDIIメソッドを使用して、Callオブジェクトを取得できます。

以下は、動的ポートにアクセスするAPIです。

```

Call createCall() throws ServiceException;
Call createCall()(QName portName) throws ServiceException;
Call createCall(QName portName, String operationName)
    throws ServiceException;
Call createCall(QName portName, QName operationName)
    throws ServiceException;
Call[] getCalls(QName portName) throws ServiceException;

```

## 23.2.3. Java EEクライアントの作成と例

Java EEクライアントは、WSDLを使用した場合と使用していない場合のどちらでも作成は可能です。またクライアントは、JSP、サーブレット、EJBなど複数の形式が可能で、前述のプログラミング・モデルに一致さえすれば問題ありません。

本節では、JSP形式のJava EEクライアントを作成する例を紹介し、作成方法について説明します。

### 23.2.3.1. WSDLを使用したサービスの呼び出し

WSDLを使用してサービスを呼び出すには、以下の手順に従います。

1. JEUSは、wsdl2java Ant Taskとwsdl2java Command Lineツールを使用してJava EEクライアント・ポータブル・アーティファクトとJAX-RPCマッピング・ファイルを作成します。wsdl2javaタスクの詳細については『JEUS リファレンスガイド』の「5.5.2. wsdl2java」を参照してください。

以下は、Antタスクを使用するbuild.xmlの例です。

#### [例 23.5] << build.xml >>

```

<target name="do-package-war">

    ...

    <antcall target="wsdl2java">
        <param name="wsdl2java.option"
            value="-import:client
        -d ${build.war.dir}/WEB-INF/classes -package echo
        -outputmapping ${build.war.dir}/WEB-INF/DocLitEchoService-mapping.xml
        -compile ${src.web}/WEB-INF/wsdl/DocLitEchoService.wsdl" />
    </antcall>

    ...

</target>

```

ant buildコマンドを実行すると、Java EEクライアント・ポータブル・アーティファクトとJAX-RPCマッピング・ファイルが作成されます。

## 2. クライアント・プログラムを作成します。

WSDLを使用してクライアントを開発する場合、以下のようなjava.xml.rpc.Serviceインターフェース・メソッドを利用してスタブやCallオブジェクトを取得できます。

```
java.rmi.Remote getPort(QName portName, Class serviceEndpointInterface)
    throws ServiceException;
java.rmi.Remote getPort(java.lang.Class serviceEndpointInterface)
    throws ServiceException;
Call createCall() throws ServiceException;
Call createCall()(QName portName) throws ServiceException;
Call createCall(QName portName, String operationName)
    throws ServiceException;
Call createCall(QName portName, QName operationName)
    throws ServiceException;
Call[] getCalls(QName portName) throws ServiceException;
```

以下は、上のメソッドのうち動的なプロキシ(Dynamic Proxy)を作成し、Webサービスを呼び出すようにJSPで作成されたJava EEクライアントの例です。

### [例 23.6] << helloClient.jsp >>

```
<%@ page language="java" %>
<%@ page import="javax.naming.*" %>
<%@ page import="javax.rmi.*" %>
<%@ page import="java.rmi.RemoteException" %>
<%@ page import="java.util.*" %>

<%@ page import="javax.naming.InitialContext" %>
<%@ page import="javax.xml.rpc.Service" %>

<%@ page import="echo.*" %>
<%@ page errorPage="/error.html" %>

<%! String msgToSend = "msg_sent_by_jspClient";
    String ret=null;
    String exceptionString="";
%>
<%
    try {
        InitialContext jndiContext = new InitialContext();
        Service service = (Service)jndiContext.lookup(
            "java:comp/env/service/DocLitEchoService");
        java.rmi.Remote port = service.getPort(Echo.class);
        Echo echoPort = (Echo)port;
        ret = echoPort.echoString(msgToSend);
    } catch (Exception e) {
        exceptionString = e.toString();
        e.printStackTrace();
    }
%>
```

```

    }
%>
<%= "Result is "+ret+"....."
%>

```

3. 前述の例の標準DD(web.xml)ファイルを作成します。

**[例 23.7] << web.xml >>**

```

<?xml version="1.0"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
  <servlet>
    <servlet-name>jsp_helloClient</servlet-name>
    <jsp-file>/helloClient.jsp</jsp-file>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <service-ref>
    <service-ref-name>
      service/DocLitEchoService
    </service-ref-name>
    <service-interface>
      javax.xml.rpc.Service
    </service-interface>
    <wsdl-file>
      WEB-INF/wsdl/DocLitEchoService.wsdl
    </wsdl-file>
    <jaxrpc-mapping-file>
      WEB-INF/DocLitEchoService-mapping.xml
    </jaxrpc-mapping-file>
    <port-component-ref>
      <service-endpoint-interface>
        echo.Echo
      </service-endpoint-interface>
    </port-component-ref>
  </service-ref>
</web-app>

```

web.xmlの<service-ref-name>にJSPクライアントでJNDIルックアップで取得するサービス名を設定します。

<service-interface>にサービス・インターフェースがjavax.xml.rpc.Serviceインターフェースであることが設定されています。このインターフェースの実装は、<wsdl-file>に設定されているWSDLファイルの情報でクライアントがJEUSにデプロイされる際に作成されます。この際に必要なJavaパッケージとタイプに関する情報は、<jaxrpc-mapping-file>に設定されているJAX-RPCマッピング・ファイルから取得します。

以下は、前述のJEUS Web DD(jeus-web-dd.xml)です。

**[例 23.8] << jeus-web-dd.xml >>**

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <service-ref>
    <service-client>
      <service-ref-name>
        service/DocLitEchoService
      </service-ref-name>
      <port-info>
        <wsdl-port xmlns:ns1="urn:DocLitService">
          ns1:Echo
        </wsdl-port>
        <stub-property>
          <name>
            javax.xml.rpc.service.endpoint.address
          </name>
          <value>
            http://localhost:8088/DocLitEchoService/DocLitEchoService
          </value>
        </stub-property>
      </port-info>
    </service-client>
  </service-ref>
</jeus-web-dd>
```

### 23.2.3.2. WSDLを使用せずにサービスを呼び出す場合

WS-Iという標準機構は、Webサービス間の相互運用性を実現するためのBasic Profileを定義し、これを遵守することを要求しています。現在のBasic Profile 1.0は、WSDLの公開を要求しています。

しかし、Basic Profileを遵守していないサービスに接続する場合、WSDLが常に公開されているとはいえません。この場合、WSDLにかかわらず、DII方式を使用することによってWebサービスを呼び出すことができます。しかし、結局この方法も、WSDLに入っているオペレーション・スタイルやオペレーション名のような詳細をCallオブジェクトに提供することを要します。

WSDLがクライアントDDに定義されていない場合、クライアント開発者は以下のようなサービス・インターフェースのメソッドを使用してCallオブジェクトを作成します。

```
Call createCall() throws ServiceException;
```

以下は、Callオブジェクトを使用してWebサービスを呼び出すJSP Java EEクライアントの例です。

**[例 23.9] << helloClient.jsp >>**

```
<%@ page language="java" %>
<%@ page import="javax.naming.*" %>
<%@ page import="javax.rmi.*" %>
```

```

<%@ page import="java.rmi.RemoteException" %>
<%@ page import="java.util.*" %>

<%@ page import="javax.naming.InitialContext" %>
<%@ page import="javax.xml.rpc.Service" %>
<%@ page import="javax.xml.rpc.Call" %>
<%@ page import="javax.xml.namespace.QName" %>
<%@ page import="javax.xml.rpc.ParameterMode" %>

<%@ page errorPage="/error.html" %>

<%! String msgToSend = "msg_sent_by_jspClient";
    String ret=null;
    String exceptionString="";
%>

<%
    try {
        InitialContext jndiContext = new InitialContext();
        Service service = (Service)jndiContext.lookup(
            "java:comp/env/service/RpcEncEchoService");

        String targetNamespace = "urn:RpcEncService";
        QName operationName = new QName(targetNamespace,"echoString");
        Call call = service.createCall();
        call.setOperationName(operationName);
        call.addParameter("String_1",
            new QName("http://www.w3.org/2001/XMLSchema", "string"),ParameterMode.IN);

        call.setProperty(Call.OPERATION_STYLE_PROPERTY, "rpc");
        call.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY,
            "http://schemas.xmlsoap.org/soap/encoding/");

        call.setReturnType(new QName("http://www.w3.org/2001/XMLSchema","string"));
        call.setTargetEndpointAddress("http://localhost:8088/" +
            "RpcEncEchoService/RpcEncEchoService");

        ret = (String)call.invoke(new Object[]{msgToSend});
    } catch (Exception e) {
        exceptionString = e.toString();
        e.printStackTrace();
    }
%>
<%= "Result is "+ret+"....."
%>

```

Callオブジェクトを作成した後に、オペレーション名とオペレーションが持つパラメータを追加します。このとき、オペレーションのスタイルとエンコーディング方式を設定することもできます。

設定方法は以下のとおりです。

```
call.setProperty(Call.OPERATION_STYLE_PROPERTY, "rpc");
call.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY,
"http://schemas.xmlsoap.org/soap/encoding/");
```

前で示した例の標準DD(web.xml)は以下のように構成されています。

#### [例 23.10] << web.xml >>

```
<?xml version="1.0"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
  <servlet>
    <servlet-name>jsp_helloClient</servlet-name>
    <jsp-file>/helloClient.jsp</jsp-file>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <service-ref>
    <service-ref-name>
      service/DocLitEchoService2
    </service-ref-name>
    <service-interface>
      javax.xml.rpc.Service
    </service-interface>
  </service-ref>
</web-app>
```

<service-ref-name>にJSPクライアントでJNDIルックアップで取得するサービス名を設定します。  
<service-interface>にサービス・インターフェースがjavax.xml.rpc.Serviceインターフェースであることが設定されています。WSDLを使用せずにWebサービスを呼び出す場合は、<jaxrpc-mapping-file>を使用したJAX-RPCマッピング・ファイルの設定を使用できません。

### 23.2.3.3. Webサービス・クライアントのパッケージング

Java EE Webサービス・クライアントは単一のJava EEコンポーネントであるため、コンポーネント固有のパッケージング方式に従います。

以下は、JSPクライアントのパッケージングについての説明です。

```
WAR
|
+-- JSP Client
|
+-- WEB-INF
    |
    +-- web.xml (サーブレットのDD)
    |
    +-- jeus-web-dd.xml (JEUS Specific DD)
    |
    +-- Jax-rpcマッピング・ファイル
```



```
|
+-- wsdl
    |
    +-- wsdlファイル
|
+-- classes
    |
    +-- Javaクラス・コンポーネント(SEI, サービス・インターフェース..)
```

上記のようにパッケージングしたJava EEクライアントをデプロイしてJSPを呼び出すと、Webサービスを呼び出せるようになります。

JAX-RPCマッピング・ファイルとWSDLファイルが、webservices.xmlファイルに<wsdl-file>と<jax-rpc-mapping-file>で記述した位置にある場合、上の構造に従わなくても構いません。



# 第24章 JAX-RPC WebサービスのSOAPメッセージ・ハンドラーの作成

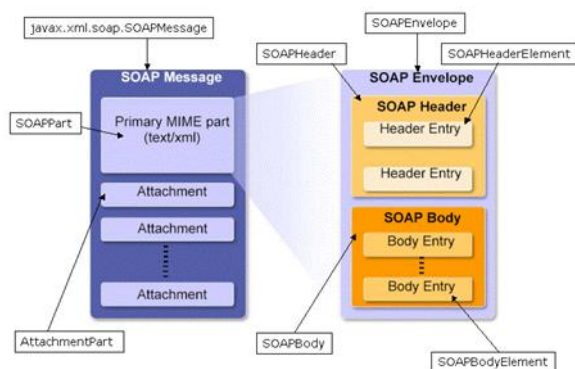
本章では、SAAJプログラミング・モデルについて紹介し、JAX-RPCと連動して使用するメッセージ・ハンドラーについて説明します。

## 24.1. SAAJの使用

SAAJ(SOAP with Attachments API for Java)を使用すると、SOAPメッセージの作成や読み込み作業を行うことができます。本節では、SAAJのプログラミング・モデルについて簡単に説明します。APIに関する詳細については記述していないため、必要な場合はSAAJの仕様を参照してください。

SAAJは、添付のないXMLのような単純なSOAPメッセージから、MIME添付を持つ、複雑なSOAPメッセージを処理する作業まで可能です。SAAJはAbstract Factoryパターンを基本とし、MessageFactoryでメッセージ(SOAPメッセージ)を作成します。SOAPメッセージは、SOAP文書表現するSOAPパートとMIME添付を表現する0個以上のAttachmentPartオブジェクトを含みます。

[図 24.1] 添付を持つSOAPの構造



(出典:<http://www.oracle.com/technetwork/java/index.html>)

### 24.1.1. SOAPメッセージの作成

空のSOAPメッセージを作成するには、MessageFactoryオブジェクトからSOAPメッセージ・オブジェクトを新しく作成します。

以下は、SOAPメッセージ・オブジェクトを作成するソースの例です。

```
MessageFactory msgFactory = MessageFactory.newInstance();
SOAPMessage message = msgFactory.createMessage();
```

## 24.1.2. SAAJ文書の作成と使用

SAAJは、SOAP文書を作成して処理するのに必要な様々なインターフェースを提供します。

SOAP文書は、要素と属性から構成されたXMLインスタンスです。便宜上、SOAP文書の重要な部分は、SAAJに対応する型を持ちます。エンベロープはSOAPEnvelope、ハンドラーはSOAPHeader、ボディーはSOAPBodyに対応します。SOAPElement型は、SOAPの名前空間に属していないアプリケーション・プログラムに従属的な(application-specific)要素に対応します。

### ● SOAPPart SOAPEnvelope型

SwA(SOAP with Attachment)メッセージで作業を行うと、SOAPPartとSOAPEnvelope型を頻繁に使用することになります。SOAPPartはSwAメッセージのMIME部分の最上位で、SOAPMessage.getSOAPPart()メソッドを呼び出してアクセス可能です。

SOAPEnvelopeへのアクセスは、SOAPPartのgetEnvelope()メソッドを呼び出すことによって可能です。SOAPEnvelopeはXML SOAP文書の最上位で、SOAPHeaderとSOAPBodyへのアクセスのためのメソッドを含んでいます。

### ● SOAPElement型

SOAPElement型は、SOAP 1.1あるいは1.2 XMLの名前空間で表現できないアプリケーション・プログラムに従属的な(application-specific)要素を直接的に表現する場合に使用します。このタイプはXML要素を表現できます。

XML要素が他のXML要素を含むことができるように、SOAP要素は他のSOAPElementオブジェクトを含むことができます。このタイプは、他のSOAP型(SOAPEnvelope、SOAPBody、SOAPBodyElement、SOAPHeader、SOAPHeaderElement、fault element)の上位型(super-type)です。

### ● SOAPHeader型

SOAPメッセージのHeader elementは、0またはそれ以上のヘッダー・ブロックを持つことができます。

SOAPHeader型はHeader elementを示し、SOAPHeaderElement型はSOAPメッセージにおいて、それぞれのヘッダー・ブロックを示します。SOAPHeader型は、SOAPHeaderElementオブジェクトを追加、削除、チェックするメソッドを提供します。

### ● SOAPHeaderElement型

SOAPHeaderElementは、特定のヘッダー・ブロックの属性と子要素を変更、チェックできるようにヘッダー・ブロックを抽象化したモデルです。

SOAPHeaderElement型は、actorとmustUnderstandのような属性だけでなく、1つあるいはそれ以上のSOAPElementオブジェクトを含むことができます。

- **SOAPBody型とSOAPBodyElement型**

SOAPBody型は、SOAPメッセージのボディー要素を示します。SOAPBodyElementはSOAPElementを拡張し、継承するメソッド以外の追加的なメソッドはありません。

以下は、SOAPBody型とSOAPBodyElement型の活用例です。

```
Name echo_Name = soapFactory.createName("echo", "tmax",
    "http://www.tmax.com/saaaj/test");
SOAPBody body = message.getSOAPBody();
SOAPBodyElement echo_Element = body.addBodyElement(echo_Name);
```

### 24.1.3. SAAJを使用したSOAPメッセージの送信

SAAJを使用したSOAPメッセージの作成を終えると、送信を実行できます。

SAAJは、簡単に構成された独自のメッセージ送信システムを持っており、これは基本APIの一部で構成されます。SAAJを使用することによって、要求または応答スタイルのSOAPメッセージをWebサービスと交換することができます。

以下は、SOAPConnectionを作成してメッセージを送信するように実装した例です。

```
//Build a SOAPMessage from a file
MessageFactory msgFactory = MessageFactory.newInstance();
MimeHeaders mimeHeaders = new MimeHeaders();
MimeHeaders.addHeader("Content-Type", "text/xml; charset=UTF-8");
FileInputStream file = new FileInputStream("soap.xml");
SOAPMessage requestMsg = msgFactory.createMessage(mimeHeaders,file);
file.close();

String address = "...";

//Send the SOAP message to the BookQuote Web Service
SOAPConnectionFactory conFactory = SOAPConnectionFactory.newInstance();
URL url = new URL(address);
SOAPMessage replyMsg = connection.call(requestMsg, url);
```

## 24.2. SOAPメッセージ・ハンドラーの作成

メッセージ・ハンドラーは、JAX-RPCクライアントとWebサービス・エンドポイントによって送受信されるSOAPメッセージを直接処理し、静的に作成されたスタブ、動的に作成されたプロキシ、DII、Javaクラス・エンドポイント、EJBエンドポイントと一緒に使用できます。

メッセージ・ハンドラーの最も重要な目的は、JAX-RPCクライアントとWebサービス・エンドポイントが送受信するSOAPメッセージのヘッダー・ブロックを加え、読み取り、処理するメカニズムを提供することです。

メッセージ・ハンドラーはJava EEコンテナによって処理されます。JAX-RPCクライアントAPIを使用してSOAPメッセージを送信する場合、JAX-RPCランタイムがWebサービスで接続されたネットワークに送信する前に、メッセージ・ハンドラー・チェーンを通じてSOAPメッセージを加工します。同様に、SOAP応答メッセージがJAX-RPCクライアントによって受信される場合、クライアント・アプリケーション・プログラムに結果が返される前に、すでに駆動しているメッセージ・ハンドラー・チェーンによって加工されます。

このようなハンドラーは様々な使い道がありますが、そのうちセキュリティに関する、つまりクライアントで暗号化されたSOAPメッセージを送信したい場合は、暗号化のためのハンドラーを使用してメッセージを加工し、送信できます。Webサービスは、その要求メッセージを暗号解読のためのハンドラーを利用して解読した後、そのデータをWebサービスを実現したバックエンドに送ります。これに対するSOAPメッセージ応答は、この過程を逆に実行します。

また、もう1つの例としては、SOAPメッセージのヘッダー部分に保存されている情報にアクセスする場合です。SOAPヘッダーにWebサービスの特定の情報を保存でき、ハンドラーにその情報を処理できるようにします。

## 24.2.1. メッセージ・ハンドラーの作成

SOAPメッセージ・ハンドラーは、Webサービスの要求と応答ですべてのSOAPメッセージを盗んで加工できます。また、Webサービス・エンドポイントとWebサービスを呼び出すクライアントの両方でハンドラーを作成できます。

以下は、`javax.xml.rpc.handler` APIの主なクラスとインターフェースについて説明します。

区分	説明
<code>javax.xml.rpc.handler. Handler</code>	SOAP要求メッセージと応答メッセージ、faultメッセージを扱うメソッドを含む重要インターフェースです
<code>javax.xml.rpc.handler. HandlerChain</code>	ハンドラー・リストを示すインターフェースです。リストが持つ要素は、 <code>java.xml.rpc.handler.Handler</code> タイプです
<code>javax.xml.rpc.handler. HandlerRegistry</code>	プログラミング・レベルでハンドラーの設定を調整できるようにサポートするインターフェースです
<code>javax.xml.rpc.handler. HandlerInfo</code>	<code>webservices.xml</code> に定義されるハンドラーの初期パラメータと同じハンドラー情報を含むクラスです
<code>javax.xml.rpc.handler. MessageContext</code>	ハンドラーによって処理されるメッセージの内容を抽象化したインターフェースです。ハンドラー・チェーン内でハンドラーはメッセージ処理に関係する状態情報を共有できます
<code>javax.xml.rpc.handler.soap. SOAPMessageContext</code>	SOAP要求と応答メッセージにアクセスできるメソッドを提供する <code>MessageContext</code> の下位インターフェースです
<code>javax.xml.rpc.handler. GenericHandler</code>	ハンドラー・インターフェースを実装した抽象クラスで、SOAPメッセージ・ハンドラー開発者はこのクラスを継承して実装します

区分	説明
javax.xml.soap.SOAPMessage	SOAPメッセージの要求と応答を含むクラスです

以下は、メッセージ・ハンドラーとハンドラー・チェーンを追加することで、Webサービスを更新する手順です。

1. メッセージ・ハンドラーとハンドラー・チェーンの設計
2. javax.xml.rpc.handler.Handlerインターフェースを実装するJavaクラスの作成およびハンドラー・チェーンの実装
3. Javaコードのコンパイル
4. WebサービスのDDファイル(webservices.xml)の作成
5. Webサービスのパッケージングとデプロイ

Webサービス・クライアントでもSOAPメッセージ・ハンドラーを使用できます。これについては、本章の後半部分で記述します。

## 24.2.2. メッセージ・ハンドラーとハンドラー・チェーンの設計

SOAPメッセージ・ハンドラーを設計する前に、次の項目を確認してください。

- 追加するメッセージ・ハンドラーの数
- 追加するハンドラーの実行手順
- メッセージ・ハンドラーだけでWebサービスを構成するのか？（バックエンド構成要素を呼び出すかどうか）

ハンドラーはwebservices.xml内に設定する必要があり、実行手順が決まっているハンドラーの集合をハンドラー・チェーンといいます。ハンドラー・チェーンで構成された各ハンドラーは、要求SOAPメッセージを処理するためのメソッドと応答SOAPメッセージを処理するためのメソッドを持っています。

Webサービスを呼び出すと、JEUS Webサービスはハンドラーを次のようなメカニズムで実行します。

1. webservices.xmlに設定されている順にハンドラー・チェーン内の各ハンドラーのhandleRequest()メソッドが実行されます。
2. ハンドラー・チェーンの最後のハンドラーのhandleRequest()が実行された後に、JEUS WebサービスはWebサービス・バックエンドを呼び出します（バックエンドが存在する場合）。
3. バックエンドの実行が終了したら、ハンドラー・チェーンに属したハンドラーがwebservices.xmlに定義された順の逆の順番で呼ばれ、各ハンドラーのhandleResponse()メソッドが呼び出されます。

4. webservices.xmlに定義された一番目のハンドラーのhandleResponse()が実行されたら、Webサービスを呼び出したクライアント側にSOAPメッセージを送ります。

### 24.2.3. ハンドラー・インターフェースの実装

SOAPメッセージ・ハンドラー・クラスは、javax.xml.rpc.handler.Handlerインターフェースを直接実装するか、これを実装したjavax.xml.rpc.handler.GenericHandlerを継承して実装します。

メッセージ・ハンドラー・クラスは、ハンドラー・インターフェースの以下のような関数を実装します。

区分	説明
init()	<p>HandlerInfoオブジェクトは、webservices.xmlに定義された初期化情報のようなSOAPメッセージ・ハンドラーに関する情報を含んでいます。HandlerInfo.getHandlerConfig()メソッドを実行すると、name-value形式のマップを返します。</p> <p>ハンドラーを初期化するには、init()メソッドを実装します</p>
destroy()	<p>Handler.destroy()メソッドは、ハンドラー・オブジェクトのインスタンスを削除する場合に呼ばれ、ハンドラーの作成周期期間に取得したリソースを解放するにはこの関数を実装します</p>
getHeaders()	<p>ハンドラー・インスタンスによって処理されるヘッダー・ブロックを取得する場合に使用します</p>
handleRequest()	<p>Handler.handleRequest()メソッドは、バックエンドにSOAPメッセージを送信する前にそのメッセージを盗む場合に使用します。MessageContextオブジェクトはSOAPメッセージ・ハンドラーによって処理されるメッセージを持っており、MessageContextのサブ・インターフェースであるSOAPMessageContextを使用するとSOAP要求メッセージの内容を取得あるいは変更できます。</p> <p>メッセージを処理するには、前述のSAAJを使用します。SOAPMessageContext.getMessage()を実行すると、SAAJ APIに属するSOAPメッセージを取得でき、SOAPメッセージはSOAPPartオブジェクトと添付で構成されます。SOAPメッセージの操作方法はSAAJプログラムと同一です</p>
handleResponse()	<p>Handler.handleResponse()メソッドは、SOAPメッセージのバックエンド処理後にWebサービスを呼び出したクライアントに返答する前にSOAPメッセージを盗む場合に使用します。MessageContextオブジェクトはSOAPメッセージ・ハンドラーによって処理されるメッセージを持っており、MessageContextのサブ・インターフェースであるSOAPMessageContextを使用すると、SOAP応答メッセージの内容を取得あるいは変更できます。</p> <p>メッセージを処理するには、前述のSAAJを使用します。SOAPMessageContext.getMessage()を実行すると、SAAJ APIに属するSOAP</p>



区分	説明
	<p>メッセージを取得でき、SOAPメッセージはSOAPPartオブジェクトと添付で構成されます。</p> <p>SOAPメッセージの操作方法はSAAJプログラムと同一です</p>
handleFault()	<p>Handler.handleFault()メソッドは、SOAPメッセージ処理モデルをベースにSOAP faultを処理するメソッドです。handleRequest()メソッドとhandleResponse()メソッドによって発生したSOAP faultだけでなく、バックエンドで発生したエラーを処理するためにこのメソッドは実装します。</p> <p>MessageContextオブジェクトはSOAPメッセージ・ハンドラーによって処理されるメッセージを持っており、MessageContextのサブ・インターフェースであるSOAPMessageContextを使用すると、SOAP応答メッセージの内容を取得あるいは変更できます。</p> <p>メッセージを処理するには、前述のSAAJを使用します</p>

## 24.2.4. Java EE WebサービスのDDファイルの作成

Java EE WebサービスのDDファイルはwebservices.xmlで、webservices.xmlファイルにはSOAPメッセージ・ハンドラーについての情報、ハンドラーの処理順序を定義できます。

以下は、メッセージ・ハンドラーの情報を設定する例です。

### [例 24.1] メッセージ・ハンドラーを定義する<port-component>の例

```
<port-component>
  <port-component-name>FileAttPort</port-component-name>
  . . .
  <handler>
    <handler-name>ServerAttachmentHandler</handler-name>
    <handler-class>
      filetransfer.ServerAttachmentHandler
    </handler-class>
    <init-param>
      <param-name>directory</param-name>
      <param-value>/temp</param-value>
    </init-param>
  </handler>
</port-component>
```

## 24.2.5. クライアントにおけるSOAPメッセージ・ハンドラーの使用

前節までは、JEUSサーバーで動作し、Webサービスで実行するハンドラーを作成することについて説明しましたが、クライアント・アプリケーション・プログラムでもハンドラーを作成することが可能です。

クライアントでハンドラーを作成する方法は、サーバー側でハンドラーを作成する方法と同じです。javax.xml.rpc.handler.Handlerインターフェースを実装するJavaクラスを作成します。

しかし、クライアント側でハンドラーを作成後の作業は、サーバー側でハンドラーを作成する方法と異なります。クライアントではハンドラーをDDに記述せずに、javax.xml.rpc.handler.HandlerInfoとjavax.xml.rpc.handler.HandlerRegistryクラスを使用してプログラムで直接ハンドラーを登録する必要があります。次節の例を参照してください。

## 24.2.6. ファイルを送受信するWebサービスとクライアントの例

本節では、クライアントからFile\_Send.txtというファイルをサーバーに送信し、サーバーでこのファイルを指定されたフォルダに保存後、ファイルの受信を知らせるメッセージを送信する、Webサービス・ファイルの送受信例について説明します。

クライアントは以下の順で作成します。

1. ファイルを受信するメッセージ・ハンドラーの実装
2. ファイルを受信した後、ファイルの受信を知らせるメッセージを送信するサービス・バックエンドの実装
3. Webサービス・デプロイメント記述子の作成およびWebサービスの作成とデプロイ
4. Webサービス・クライアント・ハンドラーの作成
5. Webサービス・クライアントの作成
6. 実行

### 24.2.6.1. ファイル受信メッセージ・ハンドラーの実装

以下は、Webサービスのメッセージ・ハンドラーの実装を示しています。バックエンドにSOAP要求が送信される前に、handleRequest()が先に実行されます。handleRequest()関数内では、SAAJ APIを使用して、SOAPメッセージに添付されているファイルを処理します。

**[例 24.2] << ServerAttachmentHandler.java >>**

```
package filetransfer;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
```

```

import java.io.OutputStream;
import java.util.Iterator;
import java.util.Map;

import javax.xml.namespace.QName;
import javax.xml.rpc.JAXRPCException;
import javax.xml.rpc.handler.HandlerInfo;
import javax.xml.rpc.handler.MessageContext;
import javax.xml.rpc.handler.soap.SOAPMessageContext;
import javax.xml.soap.AttachmentPart;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEXception;
import javax.xml.soap.SOAPMessage;

import javax.xml.rpc.handler.GenericHandler;

public final class ServerAttachmentHandler
    extends GenericHandler
{
    private File dir;

    private final String DIR_PROP="directory";

    public void init(HandlerInfo info) {
        super.init(info);
        Map m = info.getHandlerConfig();

        String dirName = (String) m.get(DIR_PROP);

        if (dirName == null) {
            throw new JAXRPCException("Property named: "
                + DIR_PROP + " was not found");
        }
        dir = new File(dirName);

        if (! dir.exists()) {
            if (! dir.mkdirs()) {
                throw new JAXRPCException("Unable to create directory: " + dirName);
            }
        }
        if (! dir.canWrite()) {
            throw new JAXRPCException(
                "Don't have write permission for " + dirName);
        }
    }

    private String getFileName(SOAPMessage request)
        throws SOAPEXception
    {

```

```

SOAPBody body =
    request.getSOAPPart().getEnvelope().getBody();
Object obj = body.getChildElements().next();
SOAPElement opElem = (SOAPElement) obj;
SOAPElement paramElem = (SOAPElement)opElem.getChildElements().next();
return paramElem.getValue();
}

private void copyFile(InputStream is, OutputStream os)
    throws IOException
{
    byte [] b = new byte[8192];
    int nr;
    while ((nr = is.read(b)) != -1) {
        os.write(b, 0, nr);
    }
}

public boolean handleRequest(MessageContext mc) {
    SOAPMessageContext ctx = (SOAPMessageContext) mc;
    SOAPMessage request = ctx.getMessage();
    if (request.countAttachments() == 0) {
        throw new JAXRPCException("*** Expected attachments");
    }
    try {
        Iterator it = request.getAttachments();

        while(it.hasNext()) {
            AttachmentPart part = (AttachmentPart) it.next();
            String fileName = getFileName(request);
            System.out.println("Received file named: " + fileName);

            File outFile = new File(dir, fileName);
            OutputStream os = null;
            InputStream is = null;

            try {
                os = new FileOutputStream(outFile);
                is = part.getDataHandler().getInputStream();

                copyFile(is, os);

            } catch (IOException ioe) {
                ioe.printStackTrace();
                throw new JAXRPCException("Exception writing file " + fileName,
                    ioe);
            } finally {
                try {
                    if (is != null) is.close();
                } catch (IOException ignore) {}
            }
        }
    }
}

```

```

        try {
            if (os != null) os.close();
        } catch (IOException ignore) {}
    }
}
} catch (SOAPException e) {
    e.printStackTrace();
    throw new JAXRPCException(e);
}
return true;
}
public QName[] getHeaders() {
    // TODO Auto-generated method stub
    return null;
}
}

```

### 24.2.6.2. Webサービス・バックエンドの実装

以下は、Webサービス・バックエンドの実装を示します。これは、メッセージ・ハンドラーからファイルを受信後に実行され、クライアントにファイルを受信したと返答するクラスです。

#### [例 24.3] << FileTransfer.java >>

```

package filetransfer;

public class FileTransfer implements FileTransferIF{

    public String receiveFile(String s) {
        return "Received file named: " + s;
    }
}

```

以下は、SEIを実装した例です。

#### [例 24.4] << FileTransferIF.java >>

```

package filetransfer;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface FileTransferIF extends Remote {
    public String receiveFile(String s) throws RemoteException;
}

```

### 24.2.6.3. WebサービスのDDファイルの作成およびWebサービスの作成とデプロイ

以下は、WebサービスのDDファイル(webservices.xml)です。

#### [例 24.5] << webservices.xml >>

```
<?xml version="1.0"?>
<webservices version="1.1" xmlns="http://java.sun.com/xml/ns/j2ee">
  <webservice-description>
    <webservice-description-name>
      FileAttachmentService
    </webservice-description-name>
    <wsdl-file>
      WEB-INF/wsdl/FileAttachmentService.wsdl
    </wsdl-file>
    <jaxrpc-mapping-file>
      WEB-INF/FileAttachmentService-mapping.xml
    </jaxrpc-mapping-file>
    <port-component>
      <port-component-name>FileAttPort</port-component-name>
      <wsdl-port xmlns:ns2="urn:FileAttachmentService">
        ns2:FileTransferIFPort
      </wsdl-port>
      <service-endpoint-interface>
        filetransfer.FileTransferIF
      </service-endpoint-interface>
      <service-impl-bean>
        <servlet-link>FileAttachmentServlet</servlet-link>
      </service-impl-bean>
      <handler>
        <handler-name>ServerAttachmentHandler</handler-name>
        <handler-class>
          filetransfer.ServerAttachmentHandler
        </handler-class>
        <init-param>
          <param-name>directory</param-name>
          <param-value>/temp</param-value>
        </init-param>
      </handler>
    </port-component>
  </webservice-description>
</webservices>
```

次に、web.xmlとjeus-webservices-dd.xmlを以下のように作成します。

#### [例 24.6] << web.xml >>

```
<?xml version="1.0"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
```

```

<servlet>
    <servlet-name>FileAttachmentServlet</servlet-name>
    <servlet-class>filetransfer.FileTransfer</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>FileAttachmentServlet</servlet-name>
    <url-pattern>/FileAttachmentService</url-pattern>
</servlet-mapping>
</web-app>

```

#### [例 24.7] << jeus-webservices-dd.xml >>

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-webservices-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <service>
        <web-service-description-name>
            FileAttachmentService
        </web-service-description-name>
        <port>
            <port-component-name>FileAttPort</port-component-name>
        </port>
    </service>
</jeus-webservices-dd>

```

上のようなDDファイルを作成したら、EARでパッケージングしてJEUSにデプロイします。

このサービスにアクセスできるアドレスは以下のとおりです。

```
http://localhost:8088/FileAttachmentService/FileAttachmentService
```

### 24.2.6.4. Webサービス・クライアント・ハンドラーの作成

クライアント・ハンドラーでは、handleRequest()でメッセージ・コンテキストを直接処理し、SAAJを介してSOAPメッセージに添付形式でファイルを添付します。

#### [例 24.8] << ClientAttachmentHandler.java >>

```

package filetransfer;

import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import javax.xml.namespace.QName;
import javax.xml.rpc.JAXRPCException;
import javax.xml.rpc.handler.MessageContext;
import javax.xml.rpc.handler.soap.SOAPMessageContext;
import javax.xml.soap.AttachmentPart;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEException;

```

```

import javax.xml.soap.SOAPMessage;

import javax.xml.rpc.handler.GenericHandler;

public final class ClientAttachmentHandler
    extends GenericHandler
{
    private String getFileName(SOAPMessage request)
        throws SOAPException
    {
        SOAPBody body = request.getSOAPPart().getEnvelope().getBody();

        SOAPElement opElem = (SOAPElement)body.getChildElements().next();

        SOAPElement paramElem = (SOAPElement)opElem.getChildElements().next();
        return paramElem.getValue();
    }

    public boolean handleRequest(MessageContext mc) {

        SOAPMessageContext ctx = (SOAPMessageContext) mc;
        SOAPMessage request = ctx.getMessage();

        try {
            String fileName = getFileName(request);
            AttachmentPart part = request.createAttachmentPart();
            part.setContentType("application/x-zip-compressed");
            FileDataSource fds = new FileDataSource(fileName);
            part.setDataHandler(new DataHandler(fds));
            request.addAttachmentPart(part);
        } catch (SOAPException e) {
            e.printStackTrace();
            throw new JAXRPCException(e);
        }
        return true;
    }

    public QName[] getHeaders() {
        // TODO Auto-generated method stub
        return null;
    }
}

```

#### 24.2.6.5. Webサービス・クライアントの作成

Webサービス・クライアントでは、Webサービス・クライアント・ハンドラーをハンドラー・レジストリーに登録します。ファイル送信はクライアント・ハンドラーによって行われます。



### [例 24.9] << Client.java >>

```
package filetransfer;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

import javax.xml.namespace.QName;
import javax.xml.rpc.Service;
import javax.xml.rpc.Stub;
import javax.xml.rpc.handler.HandlerInfo;
import javax.xml.rpc.handler.HandlerRegistry;

import FileAttachmentService_pkg.*;

public final class Client {

    public static void main(String[] args) throws Exception {

        Service svc = new FileAttachmentService_Impl();
        HandlerRegistry registry = svc.getHandlerRegistry();
        List list = new ArrayList();
        list.add(new HandlerInfo(ClientAttachmentHandler.class, null, null));

        QName portName = new QName("urn:FileAttachmentService", "FileTransferIFPort");
        registry.setHandlerChain(portName, list);

        FileTransferIF port =
            ((FileAttachmentService_Impl)svc).getFileTransferIFPort();
        String result = port.receiveFile("File_send.txt");
        System.out.println("** File transfer result: "+result);
    }
}
```

## 24.2.6.6. 実行

上記のように実装したWebサービスをクライアントで実行するには、クライアントのスタブを作成します。

```
JEUS_HOME/samples/webservice/jaxrpc_saaj_msg_handler/fileAttachment/fileAttachment-c
lient$ ant build
```

スタブを作成後、以下のように実行します。

```
JEUS_HOME/samples/webservice/jaxrpc_saaj_msg_handler/fileAttachment/fileAttachment-c
lient$ ant run
```

以下のような結果が出力されます。

```
** File transfer result: Received file named: File_send.txt
```

クライアントからFile\_send.txtというファイルがサーバーに送信され、webservices.xmlに設定した通りに「/temp」フォルダにFile\_send.txtファイルが保存されていることを確認できます。

# 第25章 JAX-RPC Webサービス設定ファイルの作成

本章では、JAX-RPC Webサービスの標準WebサービスのDDファイルの作成、JAX-RPCマッピング・ファイルの作成方法について説明します。

## 25.1. JAX-RPC WebサービスのDDファイルの作成

Java EE JAX-RPC Webサービスは、webservices.xmlという名前を持つWebサービス記述子をJavaクラスのエンドポイントやEJBエンドポイントを含む圧縮ファイル(WARやJAR)に含むことを要求します。

webservices.xmlをEJBエンドポイントの場合はEJB JARファイルのMETA-INFディレクトリーに、サーブレット・エンドポイントの場合はWARファイルのWEB-INFディレクトリーに置きます。

以下は、FileAttachmentServiceというWebサービスのwebservices.xmlファイルです。

### [例 25.1] << webservices.xml >>

```
<?xml version="1.0"?>
<webservices version="1.1" xmlns="http://java.sun.com/xml/ns/j2ee">
  <web service-description>
    <web service-description-name>
      FileAttachmentService
    </web service-description-name>
    <wsdl-file>
      WEB-INF/wsdl/FileAttachmentService.wsdl
    </wsdl-file>
    <jaxrpc-mapping-file>
      WEB-INF/FileAttachmentService-mapping.xml
    </jaxrpc-mapping-file>
    <port-component>
      <port-component-name>FileAttPort</port-component-name>
      <wsdl-port xmlns:ns2="urn:FileAttachmentService">
        ns2:FileTransferIFPort
      </wsdl-port>
      <service-endpoint-interface>
        filetransfer.FileTransferIF
      </service-endpoint-interface>
      <service-impl-bean>
        <servlet-link>FileAttachmentServlet</servlet-link>
      </service-impl-bean>
      <handler>
```

```

        <handler-name>ServerAttachmentHandler</handler-name>
        <handler-class>
            filetransfer.ServerAttachmentHandler
        </handler-class>
        <init-param>
            <param-name>directory</param-name>
            <param-value>/temp</param-value>
        </init-param>
    </handler>
</port-component>
</webservice-description>
</webservices>

```

webservices.xmlファイルのルート要素は<webservices>であり、<webservice-description>を1つあるいはそれ以上を必須要素として持ちます。

<webservice-description>は、同じWSDLを使用するJavaクラスやEJBエンドポイントの集合を記述します。すなわち、パッケージ内の他のWSDLファイルについて、<webservice-description>が1つずつ存在する必要があります。たとえば、2つの異なるJavaクラス・エンドポイントがWSDLをそれぞれ別に持っていて、1つのWARファイル内に存在する場合、各JSEを記述するためには<webservice-description>は2つ存在する必要があります。

<webservice-description>は、Java EEエンドポイントをWSDLポートに定義、DDの実装、JAX-RPCマッピング・ファイル、そしてエンドポイント・インターフェースにバインディングさせる役割をし、本章ではこれらの関係と必要性について説明します。

- **<wsdl-file>**

<wsdl-file>はWSDL文書のWARやEJB JARファイル内での相対位置を示し、webservices.xmlファイルはWSDLファイルと同じWARやJARファイル内に位置する必要があります。

- **<jaxrpc-mapping-file>**

<jaxrpc-mapping-file>はJAX-RPCマッピング・ファイルの位置を指定します。JAX-RPCマッピング・ファイルはWSDLファイルとJava EEエンドポイント間のマッピングを定義します。JAX-RPCマッピング・ファイルに関する詳細は「[25.2. Webサービスのマッピング・ファイルの作成](#)」を参照してください。

- **<port-component>**

<port-component>は特定のJavaクラスやEJBエンドポイントをWSDL文書内の特定のポート要素にマッピングする役割をします。Javaクラスの場合、WSDLポート定義とバインディング定義は、Webサービスをホストするために必要なサーブレットを作成するのに使用されます。

EJBエンドポイントの場合、ポートとバインディング定義は、EJBコンテナがSOAPメッセージを無状態のBeanオブジェクトに送るためにマーシャリングするのに使用されます。

- **<port-component-name>**

<port-component-name>は、特定のJavaクラスやEJBエンドポイントを指定するための名前を提供します。この名前はwebservices.xmlファイル内で一意である必要があります。

#### – <service-endpoint-interface>

Javaクラスをバックエンドに持つWebサービスの場合、SEIの名前を指定する要素です。

EJBエンドポイントWebサービスの場合も同様にこの要素で定義され、追加でejb-jar.xmlの<service-endpoint>でも同じ名前で定義する必要があります。

#### – <service-impl-bean>

<service-impl-bean>は、サービスをデプロイする際に、どのサービスのロジック実装の定義がJava EE エンドポイントになるかを定義します。Javaクラスの場合はweb.xmlのサーブレット定義を指し、EJBエンドポイントの場合はejb-jar.xmlのセッション定義を指します。

JSEの場合は<servlet-link>に定義されます。以下はその例です。

#### [例 25.2] DocLitEchoService : <<webservices.xml >>

```
<?xml version="1.0"?>
<webservices ...>
  <webservice-description>
    <webservice-description-name>
      DocLitEchoService
    </webservice-description-name>
    <wsdl-file>WEB-INF/wsdl/DocLitEchoService.wsdl</wsdl-file>
    <jaxrpc-mapping-file>
      WEB-INF/DocLitEchoService-mapping.xml
    </jaxrpc-mapping-file>
    <port-component>
      <port-component-name>EchoPort</port-component-name>
      <wsdl-port xmlns:ns2="urn:DocLitService">
        ns2:EchoPort
      </wsdl-port>
      <service-endpoint-interface>
        jeustest.webservices.java2wsdl.doclit.Echo
      </service-endpoint-interface>
      <service-impl-bean>
        <servlet-link>EchoServlet</servlet-link>
      </service-impl-bean>
    </port-component>
  </webservice-description>
</webservices>
```

<servlet-link>の値は、web.xml内の<servlet-name>の値と一致する必要があります。

**[例 25.3] DocLitEchoService : << web.xml >>**

```
<?xml version="1.0"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
  <servlet>
    <servlet-name>EchoServlet</servlet-name>
    <servlet-class>
      jeustest.webservices.java2wsdl.doclit.EchoImpl
    </servlet-class>
    . . .
  </servlet>
</web-app>
```

上と同様に、EJBエンドポイントWebサービスの場合、<ejb-link>を使用して定義します。

**[例 25.4] AddressBookService : << webservices.xml >>**

```
<?xml version="1.0"?>
<webservices ...>
  <web-service-description>
    <web-service-description-name>
      AddressBookService
    </web-service-description-name>
    <wsdl-file>META-INF/wsdl/AddressBookService.wsdl</wsdl-file>
    <jaxrpc-mapping-file>
      META-INF/AddressBookService-mapping.xml
    </jaxrpc-mapping-file>
    <port-component>
      <port-component-name>
        AddressBookIFPort
      </port-component-name>
      <wsdl-port xmlns:ns2="urn:AddressBookService">
        ns2:AddressBookIFPort
      </wsdl-port>
      <service-endpoint-interface>
        address.AddressBookIF
      </service-endpoint-interface>
      <service-impl-bean>
        <ejb-link>AddressEJB</ejb-link>
      </service-impl-bean>
    </port-component>
  </web-service-description>
</webservices>
```

<ejb-link>の値は、ejb-jar.xmlファイル内の<ejb-name>の値と一致する必要があります。

**[例 25.5] AddressBookService : << web.xml >>**

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar ...>
  <display-name>AddressEJB</display-name>
```

```

<enterprise-beans>
  <session>
    <display-name>AddressEJB</display-name>
    <ejb-name>AddressEJB</ejb-name>
    <service-endpoint>
      address.AddressBookIF</service-endpoint>
    <ejb-class>address.AddressBookEJB</ejb-class>
    . . .
  </session>
</enterprise-beans>
</ejb-jar>

```

<servlet-link>や<ejb-link>を通じて接続される対象は、webservicess.xmlを含む圧縮ファイルに存在する必要があります。

#### – <handler>

メッセージ・ハンドラーは、Java EEエンドポイントが送受信するメッセージをフィルタリングします。

webservicess.xmlファイル内でこのメッセージ・ハンドラーを設定し、処理手順を指定できます。メッセージ・ハンドラーについての詳細は「[第24章 JAX-RPC WebサービスのSOAPメッセージ・ハンドラーの作成](#)」を参照してください。

以下は、ハンドラーを設定したwebservicess.xmlの作成例です。

#### [例 25.6] FileAttachmentService : <<webservicess.xml >>

```

<?xml version="1.0"?>
<webservicess version="1.1" xmlns="http://java.sun.com/xml/ns/j2ee">
  <webservice-description>
    . . .
    <port-component>
      <port-component-name>FileAttPort</port-component-name>
      . . .
      <handler>
        <handler-name>ServerAttachmentHandler</handler-name>
        <handler-class>
          filetransfer.ServerAttachmentHandler
        </handler-class>
        <init-param>
          <param-name>directory</param-name>
          <param-value>/temp</param-value>
        </init-param>
      </handler>
    </port-component>
  </webservice-description>
</webservicess>

```

以下は、<handler>要素の下位項目についての説明です。

項目	説明
<handler-name>	定義される値はwebservices.xmlファイル内で一意の値である必要があり、設定したハンドラーに対する識別子の役割をします
<handler-class>	<p>ハンドラーを実装したクラスを定義します。</p> <p>定義されるハンドラー・クラスは、javax.xml.rpc.handler.Handlerインターフェースを直接あるいは間接的に実装する必要があります</p>
<init-param>	必須項目ではありません。メッセージ・ハンドラーの作成周期の初期段階でハンドラーを初期化するために渡されるパラメータを定義します。ランタイム時にHandlerInfo.getHandlerConfig()メソッドを実行することで取得されるjava.util.Mapオブジェクトからパラメータを取得できます
<soap-header>	ハンドラーが処理するSOAPヘッダー・ブロックのQ Name(Qualified Name)を定義し、<handler>は1つ以上の<soap-header>を含むことができます
<soap-role>	<p>ヘッダー・ブロックを処理する際にハンドラーが決めるロールを定義します。</p> <p>&lt;soap-role&gt;値が&lt;soap-header&gt;とペアを組んで定義する場合のみ、いかなるヘッダー・ブロックをハンドラーが処理するのかを正確に知ることができます。</p> <p>SOAPメッセージは、各ヘッダー・ブロックを処理するための役割を明示的に指定できます。actor属性が定義されている場合は指定された役割を実行するノードのみがヘッダー・ブロックを処理し、actor属性が定義されていない場合は最終的なメッセージ受信者がヘッダー・ブロックを処理します</p>

## 25.2. Webサービスのマッピング・ファイルの作成

JAX-RPCマッピング・ファイルは、JEUS Webサービスに組み込まれているJAX-RPCコンパイラがWSDL文書とWebサービス・エンドポイントを示すJavaインターフェースの関係を定義します。大抵の場合はWSDLとJavaのマッピングはマッピング・ファイルがなくても問題はありませんが、明示的な定義が必要な場合もあります。そういった必要に応じてJAX-RPCマッピング・ファイルが導入されました。

JAX-RPCマッピング・ファイルはJava EE Webサービス・エンドポイントやJava EE Webサービス・クライアントを使用する度に必要で、WSDL文書とJAX-RPCマッピング・ファイルは1対1で対応します。つまり、各WSDLファイルにはJAX-RPCマッピング・ファイルが1つずつ存在します。

### 25.2.1. JAX-RPCマッピング・ファイルの内容

JAX-RPCマッピング・ファイルは以下のような関係がある要素のマッピングを定義します。

- XML複合型(Complex Type)とJava Bean



- Faultメッセージと例外クラス
- WSDLのportType定義とSEI(Service Endpoint Interface)
- WSDLのサービス定義とサービス・インターフェース

JAX-RPCマッピング・ファイルに定義されていない要素は、WSDLとXMLのJavaオブジェクトへの標準マッピング方式に基づいてマッピングが行われます。また、マッピング・ファイルで定義されている要素は常に標準マッピング方式より優先されます。

## 25.2.2. JAX-RPCマッピング・ファイルの作成

マッピング・ファイルは内容が複雑でサイズも大きい方です。

全体的な設定ファイルの構成は以下のとおりです。

### [例 25.7] JAX-RPCマッピング・ファイルの構造

```
<java-wsdl-mapping>
  <package-mapping/>
  <java-xml-type-mapping/>
  <exception-mapping/>
  <service-interface-mapping/>
  <service-endpoint-interface-mapping>
    <service-endpoint-method-mapping/>
    <service-endpoint-method-mapping/>
    . . .
  </service-endpoint-interface-mapping>
  <service-interface-mapping/>
  <service-endpoint-interface-mapping>
    <service-endpoint-method-mapping/>
    <service-endpoint-method-mapping/>
    . . .
  </service-endpoint-interface-mapping>
</java-wsdl-mapping>
```

- **<java-wsdl-mapping>**

JAX-RPCマッピング・ファイルの最上位要素であり、他のマッピング要素を含んでいます。

- **<package-mapping>**

JAX-RPCコンパイラがWSDLに定義されている様々なタイプに対してJavaクラスとインターフェースの定義を作成する際に使用されます。指定必須項目です。

<package-type>の値はJavaパッケージの名前で、<namespaceURI>の値は指定されたJavaパッケージにマッピングされるべきXMLの名前空間です。

```

<java-wsdl-mapping version="1.1"
  xmlns="http://java.sun.com/xml/ns/j2ee">
  <package-mapping>
    <package-type>address</package-type>
    <namespaceURI>urn:AddressBookService</namespaceURI>
  </package-mapping>
  . . .
</java-wsdl-mapping>

```

- **<java-xml-type-mapping>**

XMLスキーマ複合型(Complex Type)や単純型(Simple Type)を使用する際に必要で、XMLスキーマやJavaタイプ間の関係を定義します。XMLスキーマに定義されているbuilt-inタイプがJavaに標準でマッピングされる場合、この要素を使用する必要はありません。

```

<java-wsdl-mapping version="1.1"
  xmlns="http://java.sun.com/xml/ns/j2ee">
  . . .
  <java-xml-type-mapping>
    <java-type>address.Address</java-type>
    <ns1:root-type-qname
      xmlns:ns1="http://java.sun.com/xml/ns/j2ee"
      xmlns="urn:AddressBookService">
      Address
    </ns1:root-type-qname>
    <qname-scope>complexType</qname-scope>
    <variable-mapping>
      <java-variable-name>addr</java-variable-name>
      <xml-element-name>addr</xml-element-name>
    </variable-mapping>
    <variable-mapping>
      <java-variable-name>street</java-variable-name>
      <xml-element-name>street</xml-element-name>
    </variable-mapping>
    <variable-mapping>
      <java-variable-name>zipcode</java-variable-name>
      <xml-element-name>zipcode</xml-element-name>
    </variable-mapping>
  </java-xml-type-mapping>
  . . .
</java-wsdl-mapping>

```

- **<exception-mapping>**

WSDL faultメッセージをJava例外クラスにマッピングします。

```

<java-xml-mapping>
  . . .
  <exception-mapping>

```

```

        <exception-type>CLASS_NAME</exception-type>
        <wsdl-message>WSDL_MESSAGE_NAME</wsdl-message>
    </exception-mapping>
    . . .
</java-xml-mapping>

```

## ● <service-interface-mapping>

WSDLのサービス定義をJAX-RPCサービス・インターフェース・タイプにマッピングします。

<service-interface>はWSDLのサービス定義を示すJavaインターフェースのクラス名を定義し、サービス・インターフェースのパッケージ名は<package-mapping>に定義されているパッケージ名と一致する必要があります。

<port-mapping>は、サービス・インターフェースのgetPortName()メソッドのポート名をWSDLの<port>と対応するように定義します。

```

<java-xml-mapping...>
    . . .
    <service-interface-mapping>
        <service-interface>
            address.AddressBookService
        </service-interface>
        <ns3:wsdl-service-name
            xmlns:ns3="http://java.sun.com/xml/ns/j2ee"
            xmlns="urn:AddressBookService">
            AddressBookService
        </ns3:wsdl-service-name>
        <port-mapping>
            <port-name>AddressBookIFPort</port-name>
            <java-port-name>AddressBookIFPort</java-port-name>
        </port-mapping>
    </service-interface-mapping>
    . . .
</java-xml-mapping>

```

上のように定義された場合、作成されるJavaサービス・インターフェースの定義は以下のとおりです。

### [例 25.8] << AddressBookService.java >>

```

package address;

public interface AddressBookService extends javax.xml.rpc.Service {
    public java.lang.String getAddressBookIFPortAddress();
    public address.AddressBookIF getAddressBookIFPort()
        throws javax.xml.rpc.ServiceException;
    public address.AddressBookIF getAddressBookIFPort(java.net.URL portAddress)
        throws javax.xml.rpc.ServiceException;
}

```

- **<service-endpoint-interface-mapping>**

SEIをWSDLのportTypeとバインディング定義にマッピングします。この要素はJAX-RPCコンパイラが適切なエンドポイント・スタブとエンドポイント・インターフェースを作成するのに必要な情報を提供します。また、WSDLのオペレーションとメッセージ・パートの定義がJavaエンドポイント・メソッドの定義にどのようにマッピングされるかについての情報を提供します。

以下は、AddressBookServiceのマッピング・ファイルに定義された例です。

```
<java-wsdl-mapping...>
. . .
<service-endpoint-interface-mapping>
  <service-endpoint-interface>
    address.AddressBookIF
  </service-endpoint-interface>
  <ns4:wsdl-port-type
    xmlns:ns4="http://java.sun.com/xml/ns/j2ee"
    xmlns="urn:AddressBookService">
    AddressBookIF
  </ns4:wsdl-port-type>
  <ns5:wsdl-binding
    xmlns:ns5="http://java.sun.com/xml/ns/j2ee"
    xmlns="urn:AddressBookService">
    AddressBookIFSoapBinding
  </ns5:wsdl-binding>
  <service-endpoint-method-mapping>
    <java-method-name>add</java-method-name>
    <wsdl-operation>add</wsdl-operation>
    <wrapped-element/>
    <method-param-parts-mapping>
      <param-position>0</param-position>
      <param-type>address.PersonInfo</param-type>
      <wsdl-message-mapping>
        <ns6:wsdl-message
          xmlns:ns6="http://java.sun.com/xml/ns/j2ee"
          xmlns="urn:AddressBookService">
          addRequest
        </ns6:wsdl-message>
        <wsdl-message-part-name>
          parameters
        </wsdl-message-part-name>
        <parameter-mode>IN</parameter-mode>
      </wsdl-message-mapping>
    </method-param-parts-mapping>
    <wsdl-return-value-mapping>
      <method-return-value>
        address.PersonInfo[]
      </method-return-value>
      <ns7:wsdl-message
        xmlns:ns7="http://java.sun.com/xml/ns/j2ee"
```

```
        xmlns="urn:AddressBookService">
        addResponse
    </ns7:wsdl-message>
    <wsdl-message-part-name>
        parameters
    </wsdl-message-part-name>
    </wsdl-return-value-mapping>
</service-endpoint-method-mapping>
    . . .
</service-endpoint-interface-mapping>
    . . .
</java-wsdl-mapping>
```



# 第26章 JAX-RPC Webサービスのデータ型

本章では、JAX-RPC Webサービスに関する様々なデータ型の問題について説明します。

まず、標準Java/XMLデータ型マッピング(type mapping)とユーザー定義クラスでWebサービス・パラメータで使用するJAX-RPCのvalue型について説明します。次に、出力または入出力パラメータのためのJAX-RPCのHolderクラスについて説明します。最後に、Webサービス・クライアントを使用してエラー情報を送信する方法について説明します。

## 26.1. 概要

本節では、WebサービスとWebサービス・クライアントで使用するデータ型に関する事項について説明します。

以下は、Webサービスのデータ型の特徴です。

- **JavaとXML型のマッピング**

XMLインスタンスをJavaオブジェクトに直列化または逆直列化するためには、XML型とJavaクラスのタイプとの型マッピングが必要です。

JEUS Webサービスは、JAX-RPCの仕様で説明されているJavaと標準XML型マッピングに従っています。

- **JAX-RPC Value型の使用**

JAX-RPC Value型とは、Webサービスの要求や応答に対して戻り値として渡すことができるタイプのことをいいます。

JAX-RPC Value型は一般的にユーザー定義のJavaBeansコンポーネントとして表現されます。JEUS Webサービスは、ユーザーが作成したJavaBeans型をWebサービスのパラメータと戻り値として使用できるようにします。

- **入出力パラメータとしてのHolderの使用**

入出力パラメータは、Webサービスの実行時に入出力で使用されるパラメータです。Webサービスが複数の出力値を返す場合に入出力パラメータを使用できます。

JEUS Webサービスは、標準JAX-RPC Holderクラスを使用して入出力パラメータをサポートしています。

- **SOAPフォルトとしての例外の使用**

Webサービス実行中にエラーが発生すると、Webサービス・クライアントにエラーの内容を送信しなければなりません。

このため、SOAP標準ではSOAPフォルトを定義しています。JEUS Webサービスは標準SOAPフォルトをサポートします。

## 26.2. JavaとXML型のマッピング

これまでは、1つの型のみ(Stringのみ)をパラメータと変換値として使用するWebサービスの例を紹介してきました。JEUS Webサービスは、Java型をXMLまたはWSDLにマッピングします。たとえば、JEUS Webサービスはjava.lang.StringクラスをXML xsd:stringデータ型にマッピングします。

本節では、JEUS Webサービスでサポートするデータ型の種類と、JEUS Webサービスで使用するための制限事項について説明します。

---

### 参考

アプリケーション開発者は、JavaとXML型のマッピングについて詳しく理解する必要はありません。しかし、すべてのJavaクラスがパラメータと戻り値の型として使用できるわけではないことに注意してください。

---

### 26.2.1. 組み込み型マッピング

以下は、JavaとXMLのデータ型とJEUS Webサービスの組み込み型マッピング(Built-in Type Mapping)を整理した表です。

**[表 26.1] 組み込み型のXML/Java型のマッピング**

XMLデータ型	Javaデータ型
xsd:string	java.lang.String
xsd:boolean	boolean, java.lang.Boolean *
xsd:double	double, java.lang.Double *
xsd:float	float, java.lang.Float *
xsd:int	int, java.lang.Integer
xsd:integer	java.math.BigInteger
xsd:long	long, java.lang.Long *
xsd:short	short, java.lang.Short *
xsd:byte	byte, java.lang.Byte
xsd:Decimal	java.math.BigDecimal
xsd:base64Binary	byte[ ]



XMLデータ型	Javaデータ型
xsd:hexBinary	byte[ ]
xsd:QName	javax.xml.rpc.namespace.QName
xsd:dateTime	java.util.Calendar
xsd:gYearMonth	java.util.Calendar
xsd:gYear	java.util.Calendar
xsd:gMonthDay	java.util.Calendar
xsd:anyURI	java.net.URI (JDK 1.4 or over) / java.lang.String (JDK 1.4)
xsd:duration	java.lang.String
xsd:name	java.lang.String
xsd:NCName	java.lang.String
xsd:NMTOKEN	java.lang.String
xsd:nomalizedString	java.lang.String
xsd:time	java.util.Calendar
xsd:token	java.lang.String
xsd:unsignedByte	short
xsd:unsignedLong	java.math.BigInteger
xsd:unsignedInt	long
xsd:unsignedShort	int
SOAP-ENC:base64	byte[ ]
SOAP-ENC:string	java.lang.String
SOAP-ENC:boolean	boolean, java.lang.Boolean *
SOAP-ENC:double	double, java.lang.Double *
SOAP-ENC:float	float, java.lang.Float *
SOAP-ENC:int	int, java.lang.Integer *
SOAP-ENC:long	long, java.lang.Long *
SOAP-ENC:short	short, java.lang.Short *
SOAP-ENC:byte	byte, java.lang.Byte *
SOAP-ENC:interger	java.math.BigInteger
SOAP-ENC:decimal	java.math.BigDecimal
SOAP-ENC:Array	java.math.BigDecimal

#### 参考

1. 「xsd」接頭辞は、XML namespace URI(<http://www.w3.org/2001/XMLSchema>)を示します。

2. 「SOAP-ENC」接頭辞は、XML namespace URI(<http://schemas.xmlsoap.org/soap/encoding>)を示します。

3. WSDLで、或るオブジェクトがnullと定義されている場合、サービス呼び出し者はxsd:nilをデータとして送受信する際に使用できます。そして、Java primitive型はWrapperクラスに置き換えられます。上の表では「\*」で示しています。

---

DIIクライアントでは、タイプを指定する際にXMLのQNameを使用できます。

```
String NS_XSD = "http://www.w3.org/2001/XMLSchema";
String XSD_DATETIME = new QName(NS_XSD, "dateTime");
call.addParameter("arg1", XSD_DATETIME, PARAM_MODE_IN);
```

## 26.2.2. 配列

JEUS Webサービスは、JAX-RPC型で定義した配列をサポートします。たとえば、int[] とString[] や、多次元配列のjava.math.BigDecimal[] もサポートします。

## 26.2.3. ユーザー定義型 : JAX-RPC Value型

JEUS Webサービスは、アプリケーションのために作成したすべてのユーザー定義型をサポートします。JAX-RPC仕様ではこのようなクラスをValue型といいます。

JEUS Webサービスでこれをサポートするために、ユーザー定義クラスは以下の事項に従う必要があります。

- パラメータのないpublic default構築子を持つ必要があります。
- 直接または間接的にjava.rmi.Remoteを実装してはいけません。
- メンバー・フィールドのタイプはJEUS Webサービスがサポートするタイプである必要があります。

クラスは、public、private、またはprotectedフィールドを含むことができます。Webサービスの呼び出し中に渡される値のために、フィールドは以下の条件を満たす必要があります。

- publicフィールドはfinalやtransientになれません。
- publicフィールドではないのは、関連するgetterとsetterメソッドを持つ必要があります。

上記規則に従うJavaBeansコンポーネントをサポートします。

## 26.3. JAX-RPC Value型の使用

「26.2.3. ユーザー定義型 : JAX-RPC Value型」で言及した規則に従うユーザー定義ハンドルは、Webサービスのパラメータ型として使用できます。このような種類のユーザー定義型をJAX-RPC Value型といいます。

本節では、JAX-RPC Value型を使用する例について説明します。CalcServiceは2つの数字と1つの演算子を受けて、結果を数字で返す例です。また、CalcServiceのためのWebサービス・クライアントを作成します。

### 26.3.1. JAX-RPC Value型を使用するWebサービスの作成

以下は、CalcServiceソースコードであるCalculator.javaコードです。

#### [例 26.1] <<Calculator.java>>

```
package calc;

public class Calculator implements CalculatorIF {
    public Calculator() { }
    public double calc(CalcData data) {
        String op = data.getOp();
        double num1 = data.getNum1();
        double num2 = data.getNum2();
        double ret = -9999.0;

        if (op.equals("plus")) {
            ret = num1 + num2;
        } else if (op.equals("minus")) {
            ret = num1 - num2;
        } else if (op.equals("mult")) {
            ret = num1 * num2;
        } else if (op.equals("div")) {
            if (num2 != 0)
                ret = num1 / num2;
        }
        return ret;
    }
}
```

calc()メソッドは、CalcData型を引数として受け付け、エラーが発生すると-9999.0を返します。後では、より拡張したエラー処理方法について説明します。

以下はCalcData.javaのソースコードです。

#### [例 26.2] <<CalcData.java>>

```
package calc;

public class CalcData {
```

```

private double num1;
private double num2;
private String op;

public CalcData() { }
public double getNum1() { return num1; }
public double getNum2() { return num2; }
public String getOp() { return op; }

public void setNum1(double n) { num1 = n; }
public void setNum2(double n) { num2 = n; }
public void setOp(String s) { op = s; }
}

```

CalcDataクラスはJAX-RPC Value型の要求事項に従います。これは、CalcDataのインスタンスは値として渡すことができることを意味します。

以下は、サービス・エンドポイント・インターフェース・ファイルのCalculatorIF.javaコードです。

#### **[例 26.3] <<CalculatorIF.java>>**

```

package calc;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface CalculatorIF extends Remote {
    public double calc(CalcData data) throws RemoteException;
}

```

このファイルをコンパイルするには、以下のコマンドを実行します。

```
$ ant compile
```

このコマンドは、結果クラス・ファイルをbuildディレクトリーの下に移します。

デプロイ可能なEARファイルを作成するには、以下のコマンドを実行します。

```
ant wsear
```

Webサービス・モジュールをデプロイすると、以下のアドレスでこのサービスにアクセスできます。

```
http://localhost:8088/Calculator1Service/Calculator1Service?wsdl
```

## **26.3.2. JAX-RPC Value型を使用するWebサービス・クライアントの作成**

以下のコマンドを実行すると、CalcService Webサービスのためのプロキシ・クライアントを作成します。

```
ant wsdl2java
```

作成されたStubコードのパッケージ名はcom.test.calcと仮定します。

クライアント・プログラムのソースコードは以下のとおりです。

**[例 26.4] <<CalcClient.java>>**

```
import com.test.calc.*;
import javax.xml.rpc.soap.SOAPFaultException;

public class CalcClient {
    public static void main(String[] args) {
        CalcClient calc = new CalcClient();

        if (args.length != 3) {
            System.out.println("usage: java CalcClient num1 op num2");
            System.out.println(" where op is one of " + "'plus', 'minus', 'mult', 'div'");
            System.exit(1);
        }
        try {
            calc.run(args);
        } catch (SOAPFaultException e) {
            System.err.println("faultcode = " + e.getFaultCode());
            System.err.println("faultString = " + e.getFaultString());
        } catch (Exception e) {
            System.err.println(e.toString());
            e.printStackTrace();
        }
    }

    public void run(String[] args) throws Exception {
        CalculatorIF port = new
            Calculator1Service_Impl().getCalculatorIFPort();
        CalcData data = new CalcData();
        data.setNum1((new Double(args[0])).doubleValue());
        data.setNum2((new Double(args[2])).doubleValue());
        data.setOp(args[1]);

        double ret = port.calc(data);
        System.out.println(ret);
    }
}
```

クライアント・コードの実装が完了すると、以下のコマンドを実行してクライアント・コードとStubコードをコンパイルします。

```
$ ant build
```

クライアントを実行するには、以下のようにAntタスクを実行します。

```
$ ant runclient
```

実行されると、以下のような結果が出力されます。

```
2.0
```

## 26.4. Holderクラス

Webサービスが複数の値を返すためには、JAX-RPC Value型を定義するか、出力または入出力パラメータを1つ以上指定しなければなりません。Holderクラスは、出力または入出力パラメータとして使用されるHolderクラスです。

### 26.4.1. 組み込み型Holderクラス

JEUS Webサービスは、単純データ型のJAX-RPC Holderクラスを提供します。JAX-RPCがサポートする標準Holderクラスは以下のとおりです。

**[表 26.2] 組み込み型Holderクラス**

Holder Class	Java Data Type
javax.xml.rpc.holders.BooleanHolder	boolean
javax.xml.rpc.holders.ByteHolder	byte
javax.xml.rpc.holders.ShortHolder	short
javax.xml.rpc.holders.IntHolder	int
javax.xml.rpc.holders.LongHolder	long
javax.xml.rpc.holders.FloatHolder	float
javax.xml.rpc.holders.DoubleHolder	double
javax.xml.rpc.holders.BigDecimalHolder	java.math.BigDecimal
javax.xml.rpc.holders.BigIntegerHolder	java.math.BigInteger
javax.xml.rpc.holders.ByteArrayHolder	byte[ ]
javax.xml.rpc.holders.CalendarHolder	java.util.Calendar
javax.xml.rpc.holders.QNameHolder	javax.xml.namespace.QName
javax.xml.rpc.holders.StringHolder	java.lang.String

各Holderクラスが持つ値にアクセスするためにvalueフィールドを使用します。

以下はCaculator.javaを修正したソースです。

**[例 26.5] << Calculator.java >>**

```
package calc;
```

```

import javax.xml.rpc.holders.DoubleHolder;

public class Calculator implements CalculatorIF {
    public Calculator() { }

    public void calc(CalcData data, DoubleHolder result){
        String op = data.getOp();
        double num1 = data.getNum1();
        double num2 = data.getNum2();
        double ret = -9999.0;

        if (op.equals("plus")) {
            ret = num1 + num2;
        } else if (op.equals("minus")) {
            ret = num1 - num2;
        } else if (op.equals("mult")) {
            ret = num1 * num2;
        } else if (op.equals("div")) {
            if (num2 != 0)
                ret = num1 / num2;
        }

        result.value = ret;
    }
}

```

ソースコードは、組み込み型Holderクラスのjavax.xml.rpc.holders.DoubleHolderをインポートしています。

```

import javax.xml.rpc.holders.DoubleHolder;

```

calc()メソッドの署名も修正されています。リターン型はvoidで、2番目のパラメータでHolderオブジェクトを受け取ります。

```

public void calc(CalcData data, DoubleHolder result){
    . . .
}

```

Holderオブジェクトが持つオブジェクト値にアクセスするためには、Holderクラスのvalueフィールドを使用します。

```

result.value = ret;

```

WebサービスのためのWebサービス・クライアントを作成する前に、このWebサービスをパッケージ化し、デプロイする必要があります。また、クライアント・プログラムのCalcClient.javaも修正される必要があります。

以下は、クライアント・ソース・コードのCalcClient.javaのrun()メソッドを示しています。

#### [例 26.6] << CalcClient.javaのrun()メソッド >>

```

public void run(String[] args) throws Exception {
    CalculatorIF port = new CalcService_Impl().getCalculatorIFPort();
}

```

```

CalcData data = new CalcData();
data.setNum1((new Double(args[0])).doubleValue());
data.setNum2((new Double(args[2])).doubleValue());
data.setOp(args[1]);

DoubleHolder ret = new DoubleHolder();
port.calc(data, ret);
System.out.println(ret.value);
}

```

---

#### 注

CalcClient.javaをコンパイルする前にプロキシ・ソースコードを再作成する必要があります。

---

## 26.4.2. ユーザー定義型のためのHolderクラスの作成

以下は、ユーザーが作成したクラスのHolderクラスを作成する手順です。

1. javax.xml.rpc.holders.Holderインターフェースを実装します。

2. ユーザが作成したクラスをTypeHolderと命名します。

TypeはHolderオブジェクトが持つクラスの名前です。たとえば、CalcDataクラスのためのHolderを作成するには、Holderクラスの名前はCalcDataHolderになります。

3. 作成したHolderクラスをpublicで宣言します。

4. valueという名前を持つpublicフィールドを作成します。このデータ型はHolderクラスのタイプと同じです。

5. valueフィールドをデフォルト値に初期化する、パラメータのないデフォルト構築子を作成します。

6. パラメータ値にvalueフィールドを設定する構築子を作成します。

JAX-RPC Value型のHolderクラスを使用する方法を説明するために、CalcServiceの例を修正します。以下の例にクラスのCalcDataは計算結果値をメンバーに含みます。private変数のresultとgetter/setterメソッドをメンバーに追加しています。

#### [例 26.7] << CalcData.java >>

```

package calc;

public class CalcData {
    private double num1;
    private double num2;

```



```

private String op;
private double result;

public CalcData() { }
public double getNum1() { return num1; }
public double getNum2() { return num2; }
public String getOp() { return op; }
public double getResult() { return result; }

public void setNum1(double n) { num1 = n; }
public void setNum2(double n) { num2 = n; }
public void setOp(String s) { op = s; }
public void setResult(double n) { result = n; }
}

```

CalcDataのHolderクラスはCalcDataHolderです。CalcDataHolderクラスは、データ型がCalcDataであるpublicのvalueフィールドとvalueフィールドを初期化する2つの構築子を持っています。

**[例 26.8] << CalcDataHolder.java >>**

```

package calc;
import javax.xml.rpc.holders.Holder;

public class CalcDataHolder implements Holder {
    public CalcData value;

    public CalcDataHolder() {
    }

    public CalcDataHolder(CalcData value) {
        this.value = value;
    }
}

```

Holderクラスを使用するCalculator.javaソースコードは以下のとおりです。calc()メソッドは、データ型がCalcDataHolderのパラメータを指定しています。Holderオブジェクトが持つオブジェクトにアクセスするためにpublicのvalueフィールドを使用しています。

```

package calc;

public class Calculator implements CalculatorIF {
    public Calculator() { }

    public void calc(CalcDataHolder calcData) {
        CalcData data = calcData.value;
        String op = data.getOp();
        double num1 = data.getNum1();
        double num2 = data.getNum2();
        double ret = -9999.0;
    }
}

```

```

        if (op.equals("plus")) {
            ret = num1 + num2;
        } else if (op.equals("minus")) {
            ret = num1 - num2;
        } else if (op.equals("mult")) {
            ret = num1 * num2;
        } else if (op.equals("div")) {
            if (num2 != 0)
                ret = num1 / num2;
        }
        data.setResult(ret);

        // The following line is not necessary in this case.
        // But we will use it to show the usage of holder class.
        calcData.value = data;
    }
}

```

以下は、クライアント・ソース・コードの内容です。Ant `wsdl2java` コマンド、またはWSDL文書からクライアントのためのHolderクラスを作成します。Ant `wsdl2java` コマンドで作成されたHolderクラスは、ユーザーが作成したものとは異なることに留意してください。

```

import com.test.calc.*; // generated by ant process-wsdl
import com.test.calc.holders.*; // generated by ant process-wsdl

public class CalcClient {
    public static void main(String[] args) {
        CalcClient calc = new CalcClient();

        if (args.length != 3) {
            System.out.println("usage: java CalcClient num1 op num2");
            System.out.println(" where op is one of " + "'plus', 'minus', 'mult', 'div'");
            System.exit(1);
        }

        try {
            calc.run(args);
        } catch (Exception e) {
            System.err.println(e.toString());
            e.printStackTrace();
        }
    }

    public void run(String[] args) throws Exception {
        CalculatorIF port = new Calculator3Service_Impl().getCalculatorIFPort();
        CalcData data = new CalcData();
        CalcDataHolder dataHolder = new CalcDataHolder(data);
    }
}

```

```

        data.setNum1((new Double(args[0])).doubleValue());
        data.setNum2((new Double(args[2])).doubleValue());
        data.setOp(args[1]);

        port.calc(dataHolder);
        System.out.println(dataHolder.value.getResult());
    }
}

```

## 26.5. ExceptionとSOAPフォルト

JEUS Webサービスで`java.rmi.RemoteException`あるいは`java.lang.Exception`クラスを継承したクラスをSOAPフォルトで使用できます。SOAPフォルトは、SOAPメッセージを使用してエラーや状態情報を送信するために使用されます。詳細については、SOAP 1.1仕様の「4.4 SOAPフォルト」を参照してください。

例外状態をWebサービス・アプリケーションのクライアントに送信するには、Webサービスで`java.rmi.RemoteException`やユーザー定義例外を返すように指定します。発生した例外は、自動的にSOAPフォルトにラップされ、SOAPメッセージのボディー部分に含まれてWebサービス・クライアントに送信されます。

CalcServiceの例では、エラーが発生した場合、戻り値として-9999.0を使用していたソースコードを以下のように変更できます。

### [例 26.9] << Calculator.java >>

```

package calc;

import java.rmi.RemoteException;

public class Calculator {
    public Calculator() { }
    public double calc(CalcData data)
        throws RemoteException, DevideByZeroException {
        String op = data.getOp();
        double num1 = data.getNum1();
        double num2 = data.getNum2();
        double ret = 0;

        if (op.equals("plus")) {
            ret = num1 + num2;
        } else if (op.equals("minus")) {
            ret = num1 - num2;
        } else if (op.equals("mult")) {
            ret = num1 * num2;
        } else if (op.equals("div")) {
            if (num2 != 0)
                ret = num1 / num2;
        }
    }
}

```

```

        else
            throw new DivideByZeroException("divide by zero");
    } else {
        throw new RemoteException(
            "invalid operation : " + op);
    }

    return ret;
}
}

```

calc()メソッドは、知らない演算子が指定された場合や、割った値が0の場合にjava.rmi.RemoteExceptionを発生させます。Webサービス・クライアント・プログラムでjava.rmi.RemoteExceptionまたはjava.lang.Exceptionをキャッチする場合に使用できます。

## 26.6. MIME型のDataHandler型へのマッピング

Webサービス・クライアントは、SOAPメッセージに添付を追加して送信できます。JAX-RPC仕様は、添付のMIME型に使用するJava型マッピングを定義していますが、場合によってはWebサービス・クライアントがMIME型に関係なく、常にjavax.activation.DataHandler型にマッピングして使用することもできます。

本節では、WSDL-to-Javaマッピング・ツールを使用してMIME型を常にDataHandler型にマッピングする方法について説明します。

### 26.6.1. Wsdl2javaでのdataHandlerOnlyオプションの使用

以下はMIMEパートを持つWebサービスのWSDLです。

```

<message name="submission">
  <part name="title" type="xsd:string" />
  <part name="price" type="xsd:float" />
  <part name="attachment" type="xsd:hexBinary" />
</message>
. . .
<operation name="submit">
  . . .
  <input>
    . . .
    <mime:part>
      <mime:content part="attachment" type="application/xml" />
    </mime:part>
    . . .
  </input>
  . . .
</operation>

```

基本的に、このようなWSDLでwsdl2javaツールを使用してSEIを作成すると、以下のようにMIME型の「application/xml」はjavax.xml.transform.Source型にマッピングされます。

```
public interface SubmitBook extends java.rmi.Remote {
    public String submit(String title, float price,
        javax.xml.transform.Source attachment)
        throws java.rmi.RemoteException;
}
```

Antタスク、wsdl2javaでattribute、dataHandlerOnly="true"と設定した場合や、コマンドライン・ツールで-datahandleronlyオプションを使用した場合、以下のようにMIME型に関係なくパートは常にjavax.activation.DataHandler型にマッピングされます。

```
public interface SubmitBook extends java.rmi.Remote {
    public String submit(String title, float price,
        javax.activation.DataHandler attachment)
        throws java.rmi.RemoteException;
}
```

したがって、Webサービス・クライアント開発者はDataHandler型で添付を送信する必要があります。

以下は、DataHandler型を使用したWebサービス・クライアントの例です。

```
// Creates a FileInputStream from the specified path name
FileInputStream inputStream =
    new FileInputStream(new File("attachment/book.xml"));
DataHandler dataHandler = new DataHandler(inputStream, "application/xml");

// Get a Service port
SubmitBook port = new SubmitBookService_Impl().getSubmitBookPort();
String result = port.submit("Sample for a option: datahandleronly",
    12.34f, dataHandler);
System.out.println("response = " + result);
```

## 26.7. Doc/Literalでのデータ・バインディングの不使用

JAX-RPC仕様では、XML型に関するJava型マッピングを定義しています。しかし、WSDLの型によってマッピングされたJava型を使用するより、SOAP要素を直接構成してメッセージを送信の方が利便的な場合があります。

本節では、WSDL-to-Javaマッピング・ツールを使用して、XML型に関係なくjavax.xml.soap.SOAP要素を使用する方法について説明します。

### 26.7.1. Wsdl2javaでのnoDataBindingオプションの使用

以下はDocument/Literalに記述されたWSDLです。

```

<definitions name="BookQuoteService" ... >
  <types>
    <xsd:schema targetNamespace="...">
      <xsd:complexType name="Book">
        <xsd:sequence>
          <xsd:element name="title" type="xsd:string" />
          <xsd:element name="isbn" type="xsd:string" />
          <xsd:element name="authors" type="xsd:string" />
        </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="Book" type="mh:Book" />
      <xsd:element name="Result" type="xsd:float" />
    </xsd:schema>
  </types>

  <message name="getBookPriceRequest">
    <part name="book" element="mh:Book" />
  </message>
  <message name="getBookPriceResponse">
    <part name="result" element="mh:Result" />
  </message>
  . . .

  <binding name="BookServiceSoapBinding" type="mh:BookQuote">
    <soap:binding style="document" ... />
    <operation name="getBookPrice">
      <input>
        <soap:body use="literal" ... />
      </input>
      <output>
        <soap:body use="literal" ... />
      </output>
    </operation>
  </binding>
  . . .
</definitions>

```

このWSDLからwsdl2javaに作成したSEIは、以下のようにオブジェクト型のInputパラメータを持ちます。

```

public interface BookQuote extends java.rmi.Remote {
    public float getBookPrice(sample.nodatabinding.stub.Book book)
        throws java.rmi.RemoteException;
}

```

wsdl2javaで作成したSEIを作成する場合、Antタスク、wsdl2javaでattribute、noDataBinding="true"と設定した場合や、コマンドライン・ツールで-nodatabindingオプションを使用した場合、以下のようにXML型に関係なくInputパラメータおよびReturn value型はjavax.xml.soap.SOAP要素になります。

```

public interface BookQuote extends java.rmi.Remote {
    public javax.xml.soap.SOAPElement
        getBookPrice(javax.xml.soap.SOAPElement book)
}

```

```
throws java.rmi.RemoteException;  
}
```

wsdl2javaのnodatabindingオプションは、Document/LiteralのWSDLでのみ有効です。この場合、Webサービス・クライアントの開発者はSOAP要素型でメッセージを直接構成する必要があります。

以下は、SOAP要素型を使用したWebサービス・クライアントの例です。

```
// Creates a FileDataSource from the specified path name  
SOAPFactory factroy = SOAPFactory.newInstance();  
  
// Create a SOAPElement object  
SOAPElement book = factroy.createElement( "Book", "mh",  
                                             "http://www.tmaxsoft.com/j2eews/BookQuote");  
  
SOAPElement title = factroy.createElement("title");  
title.addTextNode("Sample for a option: nodatabinding");  
book.addChildElement(title);  
  
SOAPElement isbn = factroy.createElement("isbn");  
isbn.addTextNode("123-456-789");  
book.addChildElement(isbn);  
  
SOAPElement authors = factroy.createElement("authors");  
authors.addTextNode("TmaxSoft Co., Ltd.");  
book.addChildElement(authors);  
  
// Get a Service port  
BookQuote port = new BookQuoteService_Impl().getBookQuotePort();  
SOAPElement price = port.getBookPrice(book);  
System.out.println("price = " + price.getValue());
```





# 第27章 JAX-RPC Webサービスのセキュリティ

本章では、JEUSでJAX-RPC Webサービスにセキュリティを適用する方法について説明します。

## 27.1. 概要

Webサービスにセキュリティを適用するには、次の2つの方法があります。

- トランスポート・レベルのセキュリティ(Transport-level Security)

SSLを使用してクライアントとWebサービス間の接続においてセキュリティを保証します。

トランスポート・レベルのセキュリティを適用すると、SSLを利用して、クライアントとJEUSサーバー間の接続のセキュリティを確保できます。しかし、このような方式は接続自体を安全にするだけで、クライアントとJEUSサーバー間にルーターやメッセージ・キューのような媒介(Intermediary)がある場合は、媒介はSOAPメッセージを暗号化できない可能性があります。また、一部のメッセージにはトランスポートレベルのセキュリティを適用できない場合があります。

- メッセージ・レベルのセキュリティ

SOAPメッセージを電子署名あるいは暗号化します。

メッセージ・レベルのセキュリティは、SSLセキュリティの長所を生かしつつ、更に柔軟性をプラスします。メッセージ・レベルのセキュリティは、メッセージの送信時に媒介が1つ以上存在する場合でも、セキュリティが維持できるEnd-to-Endセキュリティです。これは、接続におけるセキュリティが維持されるというよりも、SOAPメッセージそのものの署名と暗号化を意味します。また、部分的な署名と暗号化が可能になります。

## 27.2. トランスポート・レベルのセキュリティ

Webサービスのトランスポート・レベルのセキュリティは、Webサービス・クライアントのアプリケーション・プログラムとWebサービス間の接続の安全性を確保することを意味します。

手順は以下のとおりです。

1. JEUSサーバーのSSL設定です。

Webサービスの開発部分は追加作業が必要ありません。JEUS サーバーでのSSL設定についての詳細は『JEUS Webエンジンガイド』を参照してください。

2. アプリケーション・プログラムのSSLの設定は、以下の手順で行います。

- a. 証明書をインポートします。(インターネット・エクスプローラを介して、証明書をローカルSS・ディレクトリーに保存します)
- b. インポートした証明書をキーストアに保存します。
- c. wsdl2javaを使用してWSDLからスタブを作成したり、クライアントからWebサービスを呼び出したりするために、クライアントを実行する場合にシステム・プロパティの値を以下のように設定します。

```
-Djavax.net.ssl.trustStore=keystore_name
```

```
-Djavax.net.ssl.trustStorePassword=keystore_password
```

- d. Antではなく、wsdl2javaコンソール・ツールを使用する場合、以下のような環境変数の設定が必要です。

```
set WSDL2JAVA_OPTS=-Djavax.net.ssl.trustStore=keystore_name  
-Djavax.net.ssl.trustStorePassword=keystore_password
```

## 27.3. メッセージ・レベルのセキュリティ

メッセージ・レベルのセキュリティは、WebサービスとWebサービスを呼び出すクライアント間のSOAPメッセージが署名や暗号化されることを意味します。

JEUS Webサービスは、以下のようなOASIS Webサービスのセキュリティ標準1.0をサポートします。

- OASIS Standard 1.0
  - SOAP Message Security V1.0
  - Username Token Profile V1.0
  - X.509 Token Profile V1.0

上の標準仕様は、認証、権限付与、データの整合性および機密性の保証に関する要求条件を明示し、様々な企業のアプリケーションに存在するセキュリティの盲点をカバーします。

### 27.3.1. Webサービスのセキュリティの適用

JEUS Webサービスでは、以下のようにセキュリティの適用が可能です。

- クライアントは、X.509証明書をを用いてSOAPメッセージを署名・暗号化して、セキュリティが適用されたWebサービスを呼び出すことができます。また、呼び出されたWebサービスは再びX.509証明書を使用してSOAPメッセージを署名・暗号化して送信することができます。

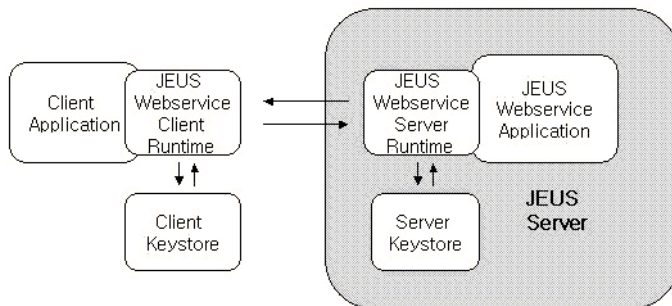
SOAPメッセージ内にはセキュリティーに関連したすべての情報が含まれています。そのため、クライアントとWebサービス間の媒体は、SOAPメッセージから必要な部分だけを選んで処理できます。

- 基本的に、JEUS Webサービスは暗号化する部分を別途指定していない場合、SOAPメッセージのボディー部分が暗号化されます。これは署名の場合も同様で、この2つを混用することもできます。

## 27.3.2. Webサービスのセキュリティー・アーキテクチャー

以下は、JEUS Webサービスのセキュリティー・アーキテクチャーを示した図です。

[図 27.1] JEUS Webサービスのセキュリティー・アーキテクチャー



サーバーのキーストアの設定はjeus-webservices-dd.xmlに行い、クライアントのキーストアの設定はjeus-web-dd.xmlに行います。

## JEUS Java EE WebサービスのクライアントとWebサービス間のセキュリティー・メッセージの送受信

Webサービスにセキュリティーを適用する場合、jeus-webservices-dd.xmlに<security>を設定し、キーストアを作成します。そして、キーストアにアクセスするためのパスワードやUsernameTokenに使用するパスワードを設定するために、コールバック・クラスを1つ作成します。このアプリケーションをJEUSサーバーに再配置すると、セキュリティー設定が適用されます。

Java EEクライアントはjeus-web-dd.xmlに<security>を設定し、クライアントのキーを格納するキーストアを作成します。UsernameTokenに使用するパスワードとキーストアの秘密鍵を取り出すためのパスワードを設定するには、コールバック・クラスを1つ作成します。

Java EEクライアント・アプリケーションが実行されると、Webサービス・クライアントのランタイムはjeus-web-dd.xmlで<security>に設定された値に従ってセキュリティー・ロジックを動作させ、クライアント・キーストアから秘密鍵を取り出して署名するか、サーバーの公開鍵を使用してメッセージを暗号化して、JEUSサーバーのWebサービスのサーバー・ランタイムにメッセージを送信します。

サーバー・ランタイムは、jeus-webservices-dd.xmlで<security>に設定されている値を基にセキュリティ・ロジックを実行させます。

- メッセージが署名されている場合、メッセージに含まれている署名に使用されているキー情報を基にサーバーのキーストアからクライアントの公開鍵を取り出し、署名を検証します。
- メッセージが暗号化されている場合、メッセージに含まれている暗号化に使用されているキー情報を基にサーバーのキーストアからサーバーの秘密鍵を取り出し、暗号を解読します。

サーバーからのWebサービスの呼び出しが終わり、応答をする場合、セキュリティを適用する際の手順は上記と似ています。

- Webサービスは、jeus-webservices-dd.xmlの<security>に設定されている値を基にキーストアからサーバーの秘密鍵やクライアントの公開鍵を取り出し、応答メッセージの一部または全体を署名あるいは暗号化します。
- クライアントのランタイムは、jeus-web-dd.xmlの<security>に設定されている値を基にクライアントのキーストアからサーバーの公開鍵やクライアントの秘密鍵を取り出し、これを使用して応答メッセージを検証および解読します。

## JEUSと他のベンダー間のセキュリティ・メッセージの相互互換性

前述のとおり、Webサービスのセキュリティはメッセージ・レベルのセキュリティを適用して使用するようになっています。セキュリティが適用されたSOAPメッセージ内部にはセキュリティに関連したすべての情報が含まれており、このような情報がOASISのWebサービスのセキュリティ標準に準拠する場合、複数のベンダー間の相互互換性が保証されます。

JEUSはOASISのセキュリティ標準の最新バージョンであるWebサービスのセキュリティ標準1.0に準拠しています。したがって、この標準仕様に準拠するベンダー間でメッセージの送受信および処理が可能となります。

### 27.3.3. JEUS Webサービスのセキュリティ設定

JEUS Webサービスとクライアントでセキュリティ・メッセージを送受信するためには、JEUS Webサービスはjeus-webservices-dd.xmlに<security>を追加設定し、クライアントはjeus-web-dd.xmlに<security>を追加設定します。

#### サーバーのセキュリティ設定

jeus-webservices-dd.xmlに追加される<security>は、Webサービスの1つのポートあたり1つの設定が要求されます。詳細については『JEUS XMLリファレンスガイド』の「23. jeus-webservices-dd.xml」を参照してください。

以下は、サーバー設定に関する例です。

**[例 27.1] <<jeus-webservices-dd.xml>>**

```
<jeus-webservices-dd>
  <service>
    <webservice-description-name/>
    <port>
      <port-component-name/>
      <security>
        <request-receiver/>
        <response-sender/>
      </security>
    </port>
  </service>
</jeus-webservices-dd>
```

<security>の下位要素である<request-receiver>に、サーバーがクライアントから受信するメッセージのセキュリティ処理を行うために必要な情報を設定します。<response-sender>には、サーバーがクライアントに送信するメッセージのセキュリティ処理を行うために必要な情報を設定します。

## クライアントのセキュリティ設定

jeus-web-dd.xmlに追加される<security>は、<port-info>ごとに1つの設定が必要です。詳細については『JEUS XMLリファレンスガイド』の「19. jeus-web-dd.xml」を参照してください。

以下は、クライアント設定を実装した例です。

**[例 27.2] <<jeus-web-dd.xml>>**

```
<jeus-web-dd>
  <service-ref>
    <service-client>
      <service-ref-name/>
      <port-info>
        <wsdl-port/>
        <security>
          <request-sender/>
          <response-receiver/>
        </security>
      </port-info>
      . . .
    </service-client>
    <service-client>
      . . .
    </service-client>
    . . .
  </service-ref>
</jeus-webservices-client-dd>
```

<security>の下位要素である<request-sender>には、クライアントがサーバーに送信するメッセージのセキュリティ処理を行うために必要な情報を設定します。<response-receiver>には、クライアントがサーバーから受信するメッセージのセキュリティ処理を行うために必要な情報を設定します。

## 27.3.4. パスワード・コールバック・クラスの作成

JEUS Webサービスのセキュリティを適用するには、パスワード・コールバック・クラスを作成する必要があります。これは、キーストアに保存されている秘密鍵の暗号やUsernameTokenに使用されるパスワードを設定するために必要です。コールバック・クラスはjavax.security.auth.callback.CallbackHandlerを実装したクラスである必要があります。

以下はパスワード・コールバック・クラスを実装した例です。

### [例 27.3] << PWCallback.java >>

```
package jeustest.webservices.wssec.doall;

import com.tmax.ws.security.WSPasswordCallback;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;
import java.io.IOException;

public class PWCallback implements CallbackHandler {
    public void handle(Callback[] callbacks)
        throws IOException, UnsupportedCallbackException {
        for (int i = 0; i < callbacks.length; i++) {
            if (callbacks[i] instanceof WSPasswordCallback) {
                WSPasswordCallback pc = (WSPasswordCallback) callbacks[i];
                checkPassword(pc);
            } else {
                throw new UnsupportedCallbackException(
                    callbacks[i], "Unrecognized Callback");
            }
        }
    }

    public void checkPassword(WSPasswordCallback pc) {
        String userId = pc.getIdentifier();
        if (pc.getUsage() == WSPasswordCallback.USERNAME_TOKEN) {
            if (userId.equals("16c73ab6-b892-458f-abf5-2f875f74882e")) {
                pc.setPassword("security2");
            }
            if (userId.equals("key_tmax1")) {
                pc.setPassword("keypass_tmax1");
            }
        } else if (pc.getUsage() == WSPasswordCallback.DECRYPT) {
            if (userId.equals("16c73ab6-b892-458f-abf5-2f875f74882e")) {
```

```

        pc.setPassword("security");
    }
    if (userId.equals("key_tmax1")) {
        pc.setPassword("keypass_tmax1");
    }
} else if(pc.getUsage() == WSPasswordCallback.SIGNATURE) {
    if (userId.equals("16c73ab6-b892-458f-abf5-2f875f74882e")) {
        pc.setPassword("security");
    }
    if (userId.equals("key_tmax1")) {
        pc.setPassword("keypass_tmax1");
    }
}
}
}
}

```

WSPasswordCallback.getUsage()は、引数として渡されるコールバック・パラメータがどの場合に使用されるのかについての値を返します。UsernameTokenのパスワードを設定する場合はUSERNAME\_TOKENという値を持っており、暗号化されたメッセージを解読するために使用する秘密鍵の暗号を設定する場合はDECRYPT値を返します。それ以外にも、署名をするために必要な秘密鍵をキーストアから取り出すための暗号を設定する場合やセッション・キーを設定する場合にもこのような実装が必要です。

このように作成されたコールバック・クラスは、jeus-webservices-dd.xmlやjeus-web-dd.xmlの<password-callback-class>にクラス名を設定することにより、セキュリティに適用可能となります。

## 27.3.5. JEUS Webサービス・サーバーのセキュリティ適用例

ストリングを送受信するWebサービスのセキュリティ適用は、以下の手順で実装されます。

1. SOAPメッセージを暗号化および署名するのに必要なキーストアを作成します。
2. Javaクラスを作成します。
3. WebサービスのDDを作成します。
4. パッケージングとデプロイを行います。

本節では、各手順について説明します。

### 27.3.5.1. キーストアの作成

SOAPメッセージの暗号化や署名にはキーを作成する必要があり、Javaで提供するkeytoolを利用して簡単に実装できます。keytoolの他にも様々な方法が存在しますが、ここではkeytoolを使用します。詳細についてはkeytoolのヘルプを参照してください。

サンプルが存在するフォルダでkeygenを実行すると、以下の作業が行われます。

1. server-keystore.jksというキーストア・ファイルを作成し、キーストアのパスワードはkeystore\_passwordにします。キーストアにはJEUS\_SERVERというエイリアスを持つ秘密鍵をRSAアルゴリズムで作成し、秘密鍵のパスワードはkey\_passwordにします。この際、JEUS\_SERVERという秘密鍵の公開鍵もペアで作成されます。

```
$ keytool -genkey -alias JEUS_SERVER -keyalg rsa -keypass  
key_password -keystore server-keystore.jks -storepass  
keystore_password
```

2. client-keystore.jksというキーストア・ファイルを作成し、キーストアのパスワードはkeystore\_passwordにします。キーストアにはJEUS\_CLIENTというエイリアスを持つ秘密鍵をRSAアルゴリズムで作成し、秘密鍵のパスワードはkey\_passwordにします。この際、JEUS\_CLIENTという秘密鍵の公開鍵もペアで作成されます。

```
$ keytool -genkey -alias JEUS_CLIENT -keyalg rsa -keypass  
key_password -keystore client-keystore.jks -storepass  
keystore_password
```

3. サーバーのキーストアとして使用するserver-keystore.jksというキーストアからJEUS\_SERVERの公開鍵を取り出します。

```
$ keytool -export -alias JEUS_SERVER -storepass keystore_password  
-keystore server-keystore.jks -file jeus_server.cert
```

4. クライアントのキーストアであるclient-keystore.jksにJEUS\_SERVERというエイリアスを保存します。

```
$ keytool -import -file jeus_server.cert -keystore  
client-keystore.jks -storepass keystore_password -alias JEUS_SERVER
```

5. クライアントのキーストアとして使用するclient-keystore.jksというキーストアからJEUS\_CLIENTの公開鍵を取り出します。

```
$ keytool -export -alias JEUS_CLIENT -storepass keystore_password  
-keystore client-keystore.jks -file jeus_client.cert
```

6. サーバのキーストアであるserver-keystore.jksにJEUS\_CLIENTというエイリアスで保存します。

```
$ keytool -import -file jeus_client.cert -keystore  
server-keystore.jks -storepass keystore_password -alias JEUS_CLIENT
```

## 27.3.5.2. Javaクラスの作成

基本的なWebサービスの実装は、セキュリティが適用されていない場合とほぼ同じです。



**[例 27.4] << Ping.java >>**

```
package ping;

public interface Ping extends java.rmi.Remote {
    public String ping(String arg) throws
        java.rmi.RemoteException;
}
```

**[例 27.5] << PingImpl.java >>**

```
package ping;

public class PingImpl implements Ping {
    public String ping(String arg) throws
        java.rmi.RemoteException {
        return arg;
    }
}
```

ここで、秘密鍵を取り出したり、UsernameTokenのパスワードを設定したりするためのコールバック・クラスを作成します。

**[例 27.6] << PingPWCallback.java >>**

```
package ping;

import com.tmax.ws.security.WSPasswordCallback;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;
import java.io.IOException;

public class PingPWCallback implements CallbackHandler {
    public void handle(Callback[] callbacks) throws
        IOException, UnsupportedCallbackException {
        for ( int i = 0 ; i<callbacks.length; i++) {
            if(callbacks[i] instanceof WSPasswordCallback) {
                WSPasswordCallback pc = (WSPasswordCallback) callbacks[i];

                if(pc.getUsage()== WSPasswordCallback.USERNAME_TOKEN) {
                    pc.setPassword("usertoken_password");
                }

                if(pc.getUsage()== WSPasswordCallback.DECRYPT){
                    pc.setPassword("key_password");
                }

                if(pc.getUsage()== WSPasswordCallback.SIGNATURE) {
                    pc.setPassword("key_password");
                }
            }
        }
    }
}
```

```

    }
  }
}
}

```

### 27.3.5.3. DDファイルの作成

キーストアを作成し、Javaクラスの作成が完了すると、WebサービスのDDを作成します。JEUSの特定設定を担当するjeus-webservices-dd.xmlで追加作業を行うことを除いては、一般的なWebサービスのDDの作成と同じです。

以下は、サーバー側でクライアントのメッセージを処理する部分の設定です。ファイル全体の内容はサンプルを参照してください。

#### [例 27.7] << jeus-webservices-dd.xml >>

```

. . .
<security>
  <request-receiver>
    <action-list>
      UsernameToken Signature Encrypt
    </action-list>
    <password-callback-class>
      ping.PingPWCallback
    </password-callback-class>
    <decryption>
      <keystore>
        <key-type>jks</key-type>
        <keystore-password>
          keystore_password
        </keystore-password>
        <keystore-filename>
          server-keystore.jks
        </keystore-filename>
      </keystore>
    </decryption>
    <signature-verification>
      <keystore>
        <key-type>jks</key-type>
        <keystore-password>
          keystore_password
        </keystore-password>
        <keystore-filename>
          server-keystore.jks
        </keystore-filename>
      </keystore>
    </signature-verification>
  </request-receiver>
. . .

```

```
</security>
. . .
```

#### 27.3.5.4. パッケージングとデプロイ

Webサービスのセキュリティ・モジュールのパッケージングとデプロイ方式は、追加でキーストア(server-keystore.jks)をコンテキストのクラス・パス上に置かなければならないことを除いては、既存のWebサービス・モジュールの方式とほぼ同じです。あるいは、jeus-webservices-dd.xmlの<keystore-filename>に絶対パスを含んだキーストアのファイル名を入力する方法もあります。

サンプル・ホームであるpingSecurityServiceディレクトリーで以下のように入力すると、パッケージングとパッケージングされたモジュールに対してJEUSサーバーのデプロイが完了します。

```
$ ant
```

これでセキュリティが適用されたWebサービスを開始できます。

#### 27.3.6. JEUS Webサービス・クライアントのセキュリティ適用例

JEUSのWebサービス Java EEクライアントの場合もサーバー側の設定と類似しており、一般的なWebサービス・クライアントの作成作業にコールバック・クラスと設定ファイル作業、キーストアのパッケージング作業が追加されます。

JEUS Webサービス・クライアントのセキュリティの適用は以下の手順で実装されます。

1. SOAPメッセージを暗号化し、署名するのに必要なキーストアを作成します。
2. Javaクラスを作成します。
3. WebサービスのDDを作成します。
4. パッケージングとデプロイを行います。

本節では、それぞれの手順について説明します。

##### 27.3.6.1. キーストアの作成

キーストアは、「[27.3.5.1. キーストアの作成](#)」で作成したキーストアのうちclient-keystore.jksを使用します。

##### 27.3.6.2. Javaクラスの作成

以下は、クライアントであるpingClient.jspファイルの一部です。WebサービスのセキュリティはWebサービスのポート単位で適用されるため、呼び出し対象のポート名を明示する必要があります。

#### [例 27.8] << pingClient.jsp >>

```
. . .
InitialContext jndiContext = new InitialContext();

Service service = (Service)jndiContext.lookup(
    "java:comp/env/service/PingSecurityService");

QName portName = new QName("urn:PingSecurityService","PingPort");
java.rmi.Remote port = service.getPort(portName, Ping.class);

Ping pingPort = (Ping)port;
ret = pingPort.ping(msgToSend);
. . .
```

クライアントもサーバー側と同様にコールバック・クラスを作成します。ここでは、サーバー側で作成したコールバッククラスを再使用します。

### 27.3.6.3. DDファイルの作成

他の設定ファイル作業は、JEUSの特定設定を担当するjeus-web-dd.xmlで追加作業を行うことを除いては、一般的なWebサービス・クライアントのDDの作成と同じです。

以下は、クライアントからサーバーに送信するメッセージを処理する部分の設定です。ファイル全体の内容はサンプルを参照してください。

#### [例 27.9] << jeus-web-dd.xml >>

```
. . .
<security>
    <request-sender>
        <action-list>
            UsernameToken Signature Encrypt
        </action-list>
        <password-callback-class>
            ping.PingPWCallback
        </password-callback-class>
        <user>JEUS_CLIENT</user>
        <signature-infos>
            <signature-info>
                <keyIdentifier>DirectReference</keyIdentifier>
                <keystore>
                    <key-type>jks</key-type>
                    <keystore-password>
                        keystore_password
                    </keystore-password>
                    <keystore-filename>
                        client-keystore.jks
                    </keystore-filename>
                </keystore>
            </signature-info>
        </signature-infos>
    </request-sender>
</security>
```

```

        </keystore>
    </signature-info>
</signature-infos>
<encryption-infos>
    <encryption-info>
        <encryptionUser>JEUS_SERVER</encryptionUser>
        <keyIdentifier>DirectReference</keyIdentifier>
        <keystore>
            <key-type>jks</key-type>
            <keystore-password>
                keystore_password
            </keystore-password>
            <keystore-filename>
                client-keystore.jks
            </keystore-filename>
        </keystore>
    </encryption-info>
</encryption-infos>
</request-sender>
. . .
</security>
. . .

```

#### 27.3.6.4. パッケージングとデプロイ

WebサービスJava EEクライアントのセキュリティ・モジュールのパッケージングとデプロイ方式は、追加でキーストア(client-keystore.jks)をコンテキストのクラスパス上に置かなければならないことを除いては、既存のWebサービスJava EEクライアントモジュールの方式とほぼ同じです。あるいは、jeus-web-dd.xmlの<keystore-filename>に絶対パスを含んだキーストアのファイル名を入力する方法もあります。

サンプル・ホームであるpingSecurityServiceJavaeeClientディレクトリーで以下のように入力すると、パッケージングとJEUSサーバーのデプロイが完了します。

```
$ ant
```

パッケージングとデプロイが完了すると、セキュリティが適用されたWebサービス・クライアントを開始できます。

#### 27.3.7. セキュリティAPIを利用したJEUS Webサービス・クライアントの作成

JEUS Webサービスは、Java EEとJavaSEの両方の環境で適用可能なWebサービス・クライアントのセキュリティAPIを提供し、Webサービス・クライアントの作成を容易にします。

以下は、APIを使用したWebサービス・セキュリティ・クライアントです。

**[例 27.10] << pingClient.jsp >>**

```
<%@ page language="java" %>
<%@ page import="javax.naming.*" %>
<%@ page import="javax.rmi.*" %>
<%@ page import="java.rmi.RemoteException" %>
<%@ page import="java.util.*" %>

<%@ page import="javax.naming.InitialContext" %>
<%@ page import="javax.xml.rpc.Service" %>
<%@ page import="javax.xml.namespace.QName" %>

<%@ page import="jeus.webservices.wssecurity.SecurityConfiguration"%>
<%@ page import="jeus.webservices.wssecurity.Keystore" %>

<%@ page import="ping.*" %>
<%@ page errorPage="/error.html" %>

<%! String msgToSend = "msg_sent_by_jspClient";
    String ret=null;
    String exceptionString="";
%>

<%
    try {

(1)        Keystore keystore = new Keystore ("JKS", "keystore_password",
            "client-keystore.jks");
            Keystore truststore = new Keystore ("JKS","keystore_password",
            "client-keystore.jks");

            InitialContext jndiContext = new InitialContext();

            Service service =
                (Service)jndiContext.lookup("java:comp/env/service/PingSecurityService");
            QName portName = new QName("urn:PingSecurityService", "PingPort");

(2)        SecurityConfiguration sconfig =
                new SecurityConfiguration(service, portName);
(3)        sconfig.setUsername("JEUS_CLIENT");
(4)        sconfig.setRequestPasswordCallbackClass("ping.PingPWCallback");

(5)        sconfig.addUTRequest(true, true);
(6)        sconfig.addSignRequest(null, "DirectReference", keystore);
(7)        sconfig.addEncryptRequest(null, "DirectReference",
            truststore, "JEUS_SERVER", null);

(4)        sconfig.setResponsePasswordCallbackClass("ping.PingPWCallback");
            sconfig.addUTResponse();
            sconfig.addSignResponse(truststore);
```

```

sconfig.addDecryptResponse(keystore);

java.rmi.Remote port = service.getPort(portName, Ping.class);
Ping pingPort = (Ping)port;

((javax.xml.rpc.Stub)pingPort)._setProperty(
    javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,
    "http://localhost:8088/PingSecurity" + "/PingSecurityService");

System.out.println("Send : " + msgToSend);
ret = pingPort.ping(msgToSend);
System.out.println("You received : " + ret);
} catch (Exception e) {
    exceptionString = e.toString();
    e.printStackTrace();
}
}
%>
<%= "Result is " + ret + "....."
%>
<%= exceptionString
%>

```

上記例のうち、Webサービス・セキュリティ設定に関する部分は太字で表示しています。

セキュリティ設定のために`jeus.webservices.wssecurity.SecurityConfiguration`オブジェクトを作成し、セキュリティの要求メッセージを作成するには、基本的にユーザー名とパスワード・コールバック・クラスを指定する必要があります。コールバック・クラスの作成については「[27.3.4. パスワード・コールバック・クラスの作成](#)」を参照してください。

その他に、ユーザー名トークンや署名、暗号化に関する設定を追加する場合、各APIを使用して追加できます。この際、キーストア設定が必要な場合は`jeus.webservices.wssecurity.Keystore`クラスを作成して設定が可能です。

#### ● (1) Keystoreオブジェクトの作成

Webサービス・セキュリティAPIを適用するには、署名や暗号化のためにKeystoreオブジェクトを作成する必要があります。

```
public Keystore(String keyType, String keystorePassword, String keystoreFilename)
```

パラメータ	説明
keystoreFilename	キーストア・ファイルの名前を入力します
keystorePassword	キーストア・ファイルの暗号を入力します
keyType	キーストアが保存しているキータイプを入力します。サポートされるキー・タイプは「JKS」あるいは「PKCS12」のうちの1つです

#### ● (2) SecurityConfigurationオブジェクトの作成

Webサービス・セキュリティAPIを適用するために最初に作成すべきオブジェクトです。

```
public SecurityConfiguration(javax.xml.rpc.Service service, QName portName)
    throws ConfigurationException
```

パラメータ	説明
service	Java EEクライアントで作成する場合は、JNDIを介して取得するServiceオブジェクトを入力し、Java SEクライアントで作成する場合はService_Impl()オブジェクトを介して作成したオブジェクトをパラメータに入力します
portName	WSDLに公開されたポートのQNameを入力します

### • (3) ユーザー名の設定

Webサービス・セキュリティの要求メッセージを作成するには、ユーザー名トークンに指定する名前や署名に使用される秘密鍵の名前が必要であり、この関数を使用して設定できます。

```
public void setUsername(String username)
```

### • (4) Callbackクラス名の設定

Webサービス・セキュリティの要求や応答メッセージを送受信する際、ユーザー名トークンで使用されるパスワードや秘密鍵の暗号を設定する場合にパスワード・コールバック・クラスを作成する必要があります。この関数を使用して、作成されたコールバック・クラスの名前を指定します。

```
public void setRequestPasswordCallbackClass(String classname)
public void setResponsePasswordCallbackClass(String classname)
```

### • (5) ユーザー名トークンの処理

以下の関数は、Webサービス・セキュリティの要求メッセージにUsernameToken要素を追加する場合に使用します。

```
public void addUTRequest(boolean addNonceCreated, boolean passwordDigest)
```

パスワードは平文またはダイジェスト形式で作成できます。平文形式で作成する場合は、NonceとCreated項目を追加するかどうかを選択できます。

以下の関数は、Webサービス・セキュリティの応答メッセージにUsernameToken要素があると判断する場合に使用します。

```
public void addUTResponse()
```

### • (6) 署名の処理

以下の関数は、Webサービス・セキュリティの要求メッセージの特定部分に署名する場合に使用します。



```
public void addSignRequest(QName signPart, String keyIdentifier,
                           Keystore keystore)
    throws SecurityConfigurationException
```

パラメータ	説明
signPart	nullと設定すると、基本的にSOAPメッセージのボディー部分に署名します
keyIdentifier	以下のうち1つを入力します <ul style="list-style-type: none"> <li>– IssuerSerial</li> <li>– DirectReference</li> <li>– SKIKeyIdentifier</li> <li>– X509KeyIdentifier</li> </ul>
keystore	署名に必要な秘密鍵を保存しているKeystoreオブジェクトを入力します

#### • (7) 暗号化の処理

以下の関数は、Webサービス・セキュリティの要求メッセージの特定部分を暗号化する場合に使用します。

```
public void addEncryptRequest(QName encPart, String keyIdentifier,
                              Keystore keystore, String encryptUser, String algorithm)
    throws SecurityConfigurationException
```

パラメータ	説明
encPart	暗号化するSOAPメッセージの部分で、QName形式で入力します。 nullと設定した場合、基本的にSOAPメッセージのボディー部分を暗号化します
keyIdentifier	以下のうち1つを入力します <ul style="list-style-type: none"> <li>– IssuerSerial</li> <li>– DirectReference</li> <li>– SKIKeyIdentifier</li> <li>– X509KeyIdentifier</li> <li>– EmbeddedKeyName</li> </ul>
keystore	暗号化に必要な公開鍵を格納しているKeystoreオブジェクトを入力します
encryptUser	暗号化に使用する公開鍵のエイリアスを入力します
algorithm	暗号化に使用されるアルゴリズムです。以下のうち1つを入力します <ul style="list-style-type: none"> <li>– AES_128</li> </ul>

パラメータ	説明
	<ul style="list-style-type: none"> <li>- AES_256</li> <li>- TRIPLE_DES</li> <li>- AES_192</li> </ul>

以下の関数は、Webサービス・セキュリティの要求メッセージの特定部分を暗号化する場合に使用します。特に、特定コールバック・クラスにバイトの配列でキーを設定し、そのキーを呼び出す場合にこの関数を使用します。

```
public void addEncryptRequest(QName encPart,
    String keyIdentifier, String embeddedKeyCallbackClass,
    String keyName, String encryptUser, String algorithm)
```

以下の関数は、Webサービス・セキュリティの応答メッセージに暗号化された部分があると判断する場合に使用します。keystoreパラメータには、暗号を解読するための秘密鍵を持っているKeystoreオブジェクトを入力します。

```
public void addDecryptResponse(Keystore keystore)
```

## [参考]

以下は、本例では適用されていませんが、追加で利用できる機能についての説明です。

### ● タイム・スタンプの処理

以下の関数は、Webサービス・セキュリティの要求メッセージにTimeStamp要素を追加する場合に使用します。

```
public void addTimeStampRequest()
```

以下の関数は、Webサービス・セキュリティの要求メッセージに有効期間を別途設定する場合に使用します。秒単位となり、デフォルト設定値は300秒です。

```
public void addTimeStampRequest(int timeToLive)
```

以下の関数は、Webサービス・セキュリティの応答メッセージにTimeStamp要素があると判断する場合に使用します。

```
public void addTimeStampResponse()
```

以下の関数は、Webサービス・セキュリティの応答メッセージのTimeStamp要素に設定された有効期間は、パラメータに入力した時間内でなければなりません。

```
public void addTimeStampResponse(int timeToLive)
```

## 27.4. アクセス制御の設定

JEUS Webサービスは、アクセスが許可されたユーザーのみがWebサービスを呼び出せるようにアクセス制御を設定することができます。設定方法は、Webサービス・バックエンドによって異なります。本節では、Webサービス・アクセス制御の設定方法とアクセス制御が設定されたWebサービスを呼び出す方法について説明します。

### 27.4.1. JavaクラスWebサービスのアクセス制御の設定

特定のURLへのアクセスを制限することによって、Webサービスへのアクセスを制限できます。このためには、web.xmlとjeus-web-dd.xmlのようなWebアプリケーション・プログラムのDDにセキュリティ情報を追加する必要があります。また、セキュリティ・ドメインの設定が必要です。

### セキュリティ・ドメインの設定

アクセス制御の設定を行うには、まずユーザーを登録する必要があります。

以下のパスのaccounts.xmlにユーザーを登録します。

```
JEUS_HOME/config/{NODE_NAME}/security/{SECURITY_DOMAIN_NAME}/accounts.xml
```

以下はユーザー登録の例で、ユーザーの名前は「jeus」、パスワードはbase64円コーディングを使用して登録します。

#### [例 27.11] << accounts.xml >>

```
<accounts xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <users>
    <user>
      <name>jeus</name>
      <password>{SHA}McbQlyhI3yiOGlHGTg8DQVWkyhg=</password>
    </user>
  </users>
</accounts>
```

### DDファイルの作成

以下のように、jeus-web-dd.xmlに<role-mapping>を追加します。

#### [例 27.12] << jeus-web-dd.xml >>

```
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <docbase>BA_DocLitEchoService</docbase>
  <role-mapping>
    <role-permission>
```

```

        <principal>jeus</principal>
        <role>Administrator</role>
    </role-permission>
</role-mapping>
</jeus-web-dd>

```

そして、web.xmlに以下のようにセキュリティー関連の設定を追加します。

#### [例 27.13] << web.xml >>

```

<web-app...>
    . . .
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>MySecureBit</web-resource-name>
            <url-pattern>/BA_DocLitEchoService</url-pattern>
            <http-method>POST</http-method>
            <http-method>GET</http-method>
        </web-resource-collection>
        <auth-constraint>
            <role-name>Administrator</role-name>
        </auth-constraint>
        <user-data-constraint>
            <transport-guarantee>NONE</transport-guarantee>
        </user-data-constraint>
    </security-constraint>

    <login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>default</realm-name>
    </login-config>
</web-app>

```

上記のように設定すると、URLが/BA\_DocLitEchoServiceのすべての要求に対し、ユーザー名とパスワードが「jeus」である場合にのみアクセスが許可されます。

## 27.4.2. EJB Webサービスのアクセス制御の設定

EJB Webサービスのアクセス制御もJavaクラスWebサービスのアクセス制御と同様に、特定のURLへのアクセスを制御することによってWebサービスへのアクセスを制限できます。このためには、ejb-jar.xmlとjeus-ejb-dd.xmlのようなEJB DDとjeus-webservices-dd.xmlのようなWebサービスのDDにセキュリティー情報を追加する必要があります。また、セキュリティー・ドメインの設定も必要になります。

### セキュリティー・ドメインの設定

アクセス制御を設定するには、まずユーザーを登録する必要があります。

以下のパスのaccounts.xmlにユーザーを登録します。

```
JEUS_HOME/config/{NODE_NAME}/security/{SECURITY_DOMAIN_NAME}/accounts.xml
```

以下はユーザー登録の例で、ユーザー名とパスワードは「jeus」で、パスワードはbase64エンコーディングを使用して登録します。

**[例 27.14] << accounts.xml >>**

```
<accounts xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <users>
    <user>
      <name>jeus</name>
      <password>{SHA}McbQlyhI3yiOG1HGTg8DQVWkyhg=</password>
    </user>
  </users>
</accounts>
```

## DDファイルの作成

以下のようにjeus-ejb-dd.xmlに<role-permission>を追加します。

**[例 27.15] << jeus-ejb-dd.xml >>**

```
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <module-info>
    <role-permission>
      <principal>jeus</principal>
      <role>Administrator</role>
    </role-permission>
  </module-info>
  <beanlist>
    ...
  </beanlist>
</jeus-ejb-dd>
```

そして、以下のようにejb-jar.xmlにセキュリティー関連設定を追加します。

**[例 27.16] << ejb-jar.xml >>**

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar ...>
  <display-name>AddressEjb</display-name>
  <enterprise-beans>
    ...
  </enterprise-beans>
  <assembly-descriptor>
    <security-role>
      <role-name>Administrator</role-name>
    </security-role>
    <method-permission>
      <role-name>Administrator</role-name>
```

```

        <method>
            <ejb-name>AddressEJB</ejb-name>
            <method-intf>ServiceEndpoint</method-intf>
            <method-name>listAll</method-name>
        </method>
    </method-permission>
</assembly-descriptor>
</ejb-jar>

```

WebサービスのDDに以下のような設定を書き加えます。指定をしていない場合はデフォルト値が設定されます。SSLを使用して送受信するデータの整合性を保証するには、<ejb-transport-guarantee>にINTEGRALを設定します。データの機密性と整合性の両方を保証するには、「CONFIDENTIAL」を設定します。もちろん、この場合にはHTTPS通信を許可するためにHTTPリスナーを設定する必要があります。

#### [例 27.17] << jeus-webservices-dd.xml >>

```

<jeus-webservices-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <ejb-context-path>webservice</ejb-context-path>
    <ejb-login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>default</realm-name>
    </ejb-login-config>
    <service>
        <webservice-description-name>...</webservice-description-name>
        <port>
            <port-component-name>...</port-component-name>
            <ejb-endpoint-url>... </ejb-endpoint-url>
            <ejb-transport-guarantee>
                CONFIDENTIAL
            </ejb-transport-guarantee>
        </port>
    </service>
</jeus-webservices-dd>

```

### 27.4.3. アクセス制御が設定されているWebサービスの呼び出し

アクセス制御(Basic Authentication)が設定されているWebサービスを呼び出すためには、WSDLにアクセスして、その内容に基づいてスタブを作成します。Webサービスを呼び出す際にユーザー名とパスワードを要求する場合は、その設定が必要です。

### WSDLからのスタブの作成

コンソール・ツールを利用する場合、以下のように設定します。

```

wsdl2java -gen:client -username jeus -password jeus
http://localhost:8088/BA_DocLitEchoService/BA_DocLitEchoService?wsdl

```

## アクセス制御が設定されたWebサービスのクライアントの作成

JAX-RPC Webサービス・クライアントが自ら認証するためには、以下の2つの設定をクライアント・プログラム内に挿入します。

- javax.xml.rpc.Stub.USERNAME\_PROPERTY
- javax.xml.rpc.Stub.PASSWORD\_PROPERTY

以下は、プログラム内に上記設定が実際に挿入された例です。

```
Echo port = // ... サービス・エンドポイント・インターフェースの取得
((javax.xml.rpc.Stub) port)._setProperty(javax.xml.rpc.Stub.USERNAME_PROPERTY,
    "jeus");
((javax.xml.rpc.Stub) port)._setProperty(javax.xml.rpc.Stub.PASSWORD_PROPERTY,
    "jeus");
String s = port.echoString("JEUS");
```





# 用語集

## DIIクライアント

Dynamic Invocation Interface(動的起動インターフェース)クライアントの略語で、JAX-RPCのクライアントAPIに従って作成されたWebサービス・クライアントです。

## EJB

Enterprise Java Beansの略語です。

## In/Out Parameter(入出力パラメータ)

入出力に使用されるWebサービス操作のパラメータです。

## JAX-RPC

Java API for XML-Based RPCの略語で、JCP(Java Community Process)で定義したJava Webサービスのための標準APIです。

## JAX-RPCのホルダー・クラス

入出力パラメータをサポートするJavaクラスです。

## JAX-RPCの値タイプ

Webサービスの要求と応答値によってネットワークに送信される内容のデータ型です。

## JEUS

Java Enterprise User Solutionの略語です。本ガイドで紹介するJEUS 8はJava EE 7互換のWebアプリケーション・サーバー(WAS)です。

## Proxy Client(プロキシ・クライアント)

wsdl2java AntタスクによってWSDLテキストから生成されたスタブ・コードを使用して作成されたWebサービス・クライアントです。

## SAAJ

SOAP message with Attachments API for Javaの略語です。JCP(Java Community Process)によって定義された、SOAPメッセージ処理のための標準 Java APIです。

## SOAP

Simple Object Access Protocolの略語です。SOAPは、プログラミング言語やOSに関係なく、ネットワーク上でXMLデータを転送するために、W3Cによって定義された標準プロトコルです。

## WAS

Web Application Serverの略語です。複雑なWebアプリケーションを実行および管理するミドルウェアです。

## WSDL

Web Service Description Language (Webサービス記述言語)の略語で、Webサービスのインターフェースと実装を記述するために W3Cによって定義された標準XMLインスタンスです。

# 索引

## A

Action MAP, 123  
AddNumbersResponseクラス, 29  
AddNumbersクラス, 29  
All Operator, 116

## C

com.sun.xml.ws.simpl.callback.PasswordValidatorCallbackPasswordValidator,  
165

## D

DII, 261, 265  
DIIクライアント, 261

## E

EWS, 2  
ExactlyOne Operator, 116

## F

Fast Infoset標準, 105

## H

Holder, 307  
Holderクラスの作成, 316

## J

Java API for XML Processing, 229  
Java Architecture for XML Binding, 229  
Java Transaction API(JTA), 140  
java.net.Authenticator, 192  
javax.activation.DataSource, 75  
javax.security.auth.callback.CallbackHandler, 166, 170  
javax.xml.rpc.handler API  
    javax.xml.rpc.handler.GenericHandler, 282  
    javax.xml.rpc.handler.Handler, 282

    javax.xml.rpc.handler.HandlerChain, 282  
    javax.xml.rpc.handler.HandlerInfo, 282  
    javax.xml.rpc.handler.HandlerRegistry, 282  
    javax.xml.rpc.handler.MessageContext, 282  
    javax.xml.rpc.handler.soap.SOAPMessageContext,  
        282  
    javax.xml.soap.SOAPMessage, 283  
javax.xml.rpc.Stub.PASSWORD\_PROPERTY, 347  
javax.xml.rpc.Stub.USERNAME\_PROPERTY, 347  
javax.xml.transform.Source, 75  
javax.xml.ws.BindingProvider.PASSWORD\_PROPERTY,  
192  
javax.xml.ws.BindingProvider.USERNAME\_PROPERTY,  
192  
javax.xml.ws.handler.LogicalHandler, 65  
javax.xml.ws.handler.SOAPHandler, 65  
JavaクラスのWebサービス・バックエンド, 8  
JavaストリーミングのXMLパーサ, 236  
JAX-RPC, 2, 5  
JAX-RPC Value型, 307, 310, 311  
JAX-RPC Webサービスのセキュリティー方式  
    トランスポート・レベルのセキュリティー, 325  
    メッセージ・レベルのセキュリティー, 325  
JAX-RPC Webサービスの呼び出し, 261  
JAX-RPCマッピング・ファイル, 295, 300  
    <exception-mapping>, 302  
    <java-wsdl-mapping>, 301  
    <java-xml-type-mapping>, 302  
    <package-mapping>, 301  
    <service-endpoint-interface-mapping>, 304  
    <service-interface-mapping>, 303  
JAX-RPC方式, 15  
JAX-WS, 2, 4, 5  
JAX-WS JMS転送, 111  
JAX-WS方式, 15  
JAXB, 2, 4, 229  
JAXB機能, 229  
JAXP, 229, 236  
JAXPの主要API  
    DOM, 236  
    SAX, 236  
    StAX, 236  
    TrAX, 236

JEUS UDDIエクスプローラ, 203  
JEUS UDDIサーバーの実行, 202  
JEUS Webサービス, 7  
JEUS Webサービスの構造, 7  
jeus-web-dd.xmlの<security>, 329  
javax.xml.soap.SOAPMessage, 75

## K

Keystore  
    keystoreFilename, 339  
    keystorePassword, 339  
    keyType, 339

## M

MAP(Message Addressing Properties), 121  
MIME添付, 93  
MTOM, 3, 93

## N

Negotiation Pessimistic方式, 107

## O

Optional Operator, 116

## R

RPC方式, 9

## S

SAAJ, 3, 5, 244, 279  
Schemagen, 233  
SecurityConfiguration, 340  
    portName, 340  
    service, 340  
SEI, 85  
Service.Mode.Message, 72  
Service.Mode.PAYLOAD, 72  
ServiceFactory, 269  
SJSXP, 229  
SOA, 1  
SOAP, 2, 3  
SOAPBodyElement型, 281

SOAPBody型, 281  
SOAPElement型, 280  
SOAPEnvelope型, 280  
SOAPHeaderElement型, 280  
SOAPHeader型, 280  
SOAPPart, 280  
SOAPの特徴, 4  
SOAPハンドラー(SOAP Handler), 61  
SOAPフォルト, 307, 319  
SOAPメッセージのエンコーディング  
    RPC方式, 5  
    ドキュメント方式, 5  
SOAPメッセージ・ハンドラーの作成方式  
    JAX-RPC方式, 11  
    JAX-WS方式, 11  
StAX, 4, 236  
Streaming API for XML, 3  
STS(Security Token Service), 147  
Sun Java Streaming XML Parser, 229

## U

UDDI, 2, 3, 197  
UDDI Property Key  
    uddi.auth, 200  
    uddi.dataSource, 200  
    uddi.operatorName, 200  
    uddi.operatorURL, 200  
UDDI WSDLの公開, 225  
uddi.propertiesファイルの設定, 200  
UDDIのクエリ, 204  
UDDIのデプロイ, 199  
UDDIエクスプローラ, 212  
UDDIクライアント, 213  
UDDIクライアントにおけるXML署名の作成, 217  
    add\_publisherAssertions, 217  
    save\_binding, 217  
    save\_business, 217  
    save\_service, 217  
    save\_tModel, 217  
    set\_publisherAssertions, 217  
UDDIクライアントにおけるXML署名の検証, 218  
    find-relatedBusinesses API, 218

- get\_bindingDetail, 218
- get\_businessDetail, 218
- get\_serviceDetail, 218
- get\_tModelDetail, 218
- UDDIサブスクリプション, 219
- UDDIサブスクリプションAPI, 219
- UDDIサーバー, 198
- UDDIサーバーの構成設定, 200
- UDDIデータの構造, 197
  - 技術モデル(tModel), 198
  - バインディング・テンプレート, 198
  - ビジネス・エンティティー, 198
  - ビジネス・サービス, 198
- UDDIデータストア, 199
- UDDIユーザー, 200
- UDDIレジストリー, 206
- UDDI標準, 4

## W

- Web Service Metadata for the Java Platform, 3
- web.xmlの<async-supported>, 91
- webservices.xml
  - <handler-class>, 300
  - <handler-name>, 300
  - <init-param>, 300
  - <jaxrpc-mapping-file>, 296
  - <port-component>, 296
  - <port-component><handler>, 299
  - <port-component><port-component-name>, 296
  - <port-component><service-endpoint-interface>, 297
  - <port-component><service-impl-bean>, 297
  - <soap-header>, 300
  - <soap-role>, 300
  - <wsdl-file>, 296
- WebサービスのDDファイル, 247
  - jeus-webservices-dd.xml, 247
  - webservices.xml, 247
- Webサービスのセキュリティ, 141, 143
- Webサービスのセキュリティ・ポリシー, 142
- Webサービスのセキュリティ対話, 146
- Webサービスのセキュリティ方式
  - トランスポート・レベルのセキュリティ, 141
  - メッセージ・レベルのセキュリティ, 141
- Webサービスのトランスポート・レベルのセキュリティ, 141, 325
- Webサービスのポリシー, 115
- Webサービスのポリシーの特徴, 115
- Webサービスの信頼, 147
- Webサービスの実装方式
  - WSDLからの開始, 10
  - サービス・エンドポイントから開始, 10
- Webサービス・アドレッシング, 121
- Webサービス・トランザクション, 137
- Webサービス・トランザクションの属性
  - 一貫性, 137
  - 永続性, 137
  - 原子性, 137
  - 独立性, 137
- Webサービス標準, 2
  - EWS, 2
  - JAX-RPC, 2
  - JAX-WS, 2
  - JAXB, 2
  - MTOM, 3
  - SAAJ, 3
  - SOAP, 3
  - Streaming API for XML, 3
  - UDDI, 3
  - Web Service Metadata for the Java Platform, 3
  - WS-Addressing, 3
  - WS-AtomicTransaction, 3
  - WS-Coordination, 3
  - WS-MetadataExchange, 3
  - WS-Policy, 3
  - WS-ReliableMessaging, 3
  - WS-SecureConversation, 3
  - WS-Security, 3
  - WS-Security Policy, 3
  - WSDL, 3
  - XML, 3
  - XML Infoset, 3
  - XML Namespace, 3
  - XML Schema, 3
- WS-Addressing, 3

- WS-AtomicTransaction, 3
- WS-Coordination, 3
- WS-MetadataExchange, 3
- WS-Policy, 3
- WS-Reliable Messaging, 131
- WS-ReliableMessaging, 3
- WS-SecureConversation, 3
- WS-Security, 3
- WS-Security Policy, 3
- WS-SecurityPolicy設定, 175
- WSDL, 2, 3, 4
- wsdl2uddi, 226
- WSDLからWebサービスを実装, 10
- ws:swaRefスキーマ, 99

## X

- XJC, 230
- xmime:expectedContentType, 94
- XML, 3
- XML Infoset, 3
- XML Namespace, 3
- XML Schema, 3
- XMLデジタル署名, 216
- XOP, 93

## あ

- 一貫性, 137
- 永続性, 137
- アンマーシャリング(Unmarshalling), 229

## か

- 組み込み型マッピング, 308
- 原子性, 137
- 高信頼性メッセージング技術(WS-Reliable Messaging), 131
- 高信頼性メッセージングのスペック
  - WS-AtomicTransactions, 131
  - WS-Coordination, 131
  - WS-Reliable Messaging, 131
- キーストア, 339
- コンテンツ・ネゴシエーション, 107
- コーディネーター・サービス, 139

## さ

- サービス・インターフェース, 268, 269
- サービス・エンドポイントからWebサービスの実装, 10
- サービス・エンドポイント・インターフェース, 85
- スタブ・クライアント, 261
- ステートレス・セッションBeanのWebサービス・バックエンド, 8

## た

- 添付, 279
- 動的プロキシ方式, 39
- 独立性, 137
- ディスパッチ・インターフェース, 71
- ディスパッチ方式, 39
- データ型, 307
- トランザクションのスペック
  - WS-AtomicTransaction, 137
  - WS-Coordination, 137
- トランスポート・レベルのセキュリティー, 141, 325

## は

- 標準Holderクラス
  - javax.xml.rpc.holders.BigDecimalHolder, 314
  - javax.xml.rpc.holders.BigIntegerHolder, 314
  - javax.xml.rpc.holders.BooleanHolder, 314
  - javax.xml.rpc.holders.ByteArrayHolder, 314
  - javax.xml.rpc.holders.ByteHolder, 314
  - javax.xml.rpc.holders.CalendarHolder, 314
  - javax.xml.rpc.holders.DoubleHolder, 314
  - javax.xml.rpc.holders.FloatHolder, 314
  - javax.xml.rpc.holders.IntHolder, 314
  - javax.xml.rpc.holders.LongHolder, 314
  - javax.xml.rpc.holders.QNameHolder, 314
  - javax.xml.rpc.holders.ShortHolder, 314
  - javax.xml.rpc.holders.StringHolder, 314
- ハンドラー・インターフェース関数
  - destroy(), 284
  - getHeaders(), 284
  - handleFault(), 285
  - handleRequest(), 284
  - handleResponse(), 284
  - init(), 284

バインディング・コンパイラ, 230  
バックエンド・コンポーネント, 239  
プロバイダー・インターフェース, 71  
ポリシー・コンテナ(Policy Container), 116  
ポリシー演算子(Operator), 116

## ま

マーシャリング(Marshalling), 229  
メッセージ, 2  
メッセージ・ハンドラー, 281  
メッセージ・レベルのセキュリティ, 141, 325, 326

## ら

論理ハンドラー(Logical Handler), 61

