Oracle® Application Server

Web Services 開発者ガイド

10*g*（10.1.3.1.0）

部品番号 : B31868-01

2006 年 12 月

ORACLE®

Oracle Application Server Web Services 開発者ガイド , 10*g*（10.1.3.1.0）

部品番号 : B31868-01

原本名 : Oracle Application Server Web Services Developer's Guide, 10*g* (10.1.3.1.0)

原本部品番号 : B28974-01

原本著者 : Thomas Pfaeffle

原本協力者 : Simeon M. Greene, Sumit Gupta, Bill Jones, Tim Julien, Gigi Lee, Mike Lehmann, Jon Maron, Kevin Minder, Bob Naugle, Eric Rajkovic, Ekkehard Rohwedder, Shih-Chang Chen, Quan Wang, Ellen Siegal（エディタ）

Copyright © 2006 Oracle. All rights reserved.

制限付権利の説明

このプログラム（ソフトウェアおよびドキュメントを含む）には、オラクル社およびその関連会社に所有権のある情報が含まれています。このプログラムの使用または開示は、オラクル社およびその関連会社との契約に記された制約条件に従うものとします。著作権、特許権およびその他の知的財産権と工業所有権に関する法律により保護されています。

独立して作成された他のソフトウェアとの互換性を得るために必要な場合、もしくは法律によって規定される場合を除き、このプログラムのリバース・エンジニアリング、逆アセンブル、逆コンパイル等は禁止されています。

このドキュメントの情報は、予告なしに変更される場合があります。オラクル社およびその関連会社は、このドキュメントに誤りが無いことの保証は致し兼ねます。これらのプログラムのライセンス契約で許諾されている場合を除き、プログラムを形式、手段（電子的または機械的）、目的に関係なく、複製または転用することはできません。

このプログラムが米国政府機関、もしくは米国政府機関に代わってこのプログラムをライセンスまたは使用する者に提供される場合は、次の注意が適用されます。

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

このプログラムは、核、航空産業、大量輸送、医療あるいはその他の危険が伴うアプリケーションへの用途を目的としておりません。このプログラムをかかる目的で使用する際、上述のアプリケーションを安全に使用するために、適切な安全装置、バックアップ、冗長性（redundancy）、その他の対策を講じることは使用者の責任となります。万一かかるプログラムの使用に起因して損害が発生いたしましても、オラクル社およびその関連会社は一切責任を負いかねます。

Oracle、JD Edwards、PeopleSoft、Siebel は米国 Oracle Corporation およびその子会社、関連会社の登録商標です。その他の名称は、他社の商標の可能性があります。

このプログラムは、第三者の Web サイトへリンクし、第三者のコンテンツ、製品、サービスへアクセスすることがあります。オラクル社およびその関連会社は第三者の Web サイトで提供されるコンテンツについては、一切の責任を負いかねます。当該コンテンツの利用は、お客様の責任になります。第三者の製品またはサービスを購入する場合は、第三者と直接の取引となります。オラクル社およびその関連会社は、第三者の製品およびサービスの品質、契約の履行（製品またはサービスの提供、保証義務を含む）に関しては責任を負いかねます。また、第三者との取引により損失や損害が発生いたしましても、オラクル社およびその関連会社は一切の責任を負いかねます。

目次

6 WSDL からの Web サービスのアセンブル

7 Java クラスを使用した Web サービスのアセンブル

8 EJB を使用した Web サービスのアセンブル

9 JMS 宛先を使用した Web サービスのアセンブル

10 データベース Web サービスのアセンブル

11 注釈を使用した Web サービスのアセンブル

12 REST Web サービスのアセンブル

13 Web サービス・デプロイのテスト

14 J2EE Web サービス・クライアントのアセンブル

15 J2SE Web サービス・クライアントのアセンブル

19 Web サービスのパッケージ化およびデプロイ

A Web サービス・クライアントの API および JAR

B WSDL 1.1 API の Oracle 実装

C oracle-webservices.xml デプロイメント・ディスクリプタ・スキーマ

D service-ref-mapping スキーマ

E エラー・メッセージ接頭辞

F トラブルシューティング

G サード・パーティ・ライセンス

索引

はじめに

このマニュアルでは、Oracle Application Server Web Services および Oracle WebServicesAssembler ツールを使用して、Java クラス、EJB、データベース・リソース、JMS 宛先および J2SE 5.0 注釈といった様々なリソースから、Web サービスをアセンブルする方法について説明します。REST スタイルの Web サービスをアセンブルすることもできます。J2SE および J2EE クライアントをアセンブルしてこのサービスにアクセスする方法については、開発者ガイドも参照してください。このマニュアルでは、OracleAS Web Services でサポートされているメッセージ書式とデータ型について説明しています。

この章の項目は、次のとおりです。

対象読者

『Oracle Application Server Web Services 開発者ガイド』は、次の作業を実行するアプリケーション・プログラマ、システム管理者およびその他のユーザーを対象としています。

- Java クラス、EJB、データベース・リソースおよび JMS キューからの Web サービスのアセンブル
- J2SE および J2EE Web サービス・クライアントのアセンブル
- rpc-literal、rpc-encoded および document-literal 書式の SOAP メッセージの処理

ドキュメントのアクセシビリティについて

オラクル社は、障害のあるお客様にもオラクル社の製品、サービスおよびサポート・ドキュメントを簡単にご利用いただけることを目標としています。オラクル社のドキュメントには、ユーザーが障害支援技術を使用して情報を利用できる機能が組み込まれています。HTML 形式のドキュメントで用意されており、障害のあるお客様が簡単にアクセスできるようにマークアップされています。標準規格は改善されつつあります。オラクル社はドキュメントをすべてのお客様がご利用できるように、市場をリードする他の技術ベンダーと積極的に連携して技術的な問題に対応しています。オラクル社のアクセシビリティについての詳細情報は、Oracle Accessibility Program の Web サイト `http://www.oracle.com/accessibility/` を参照してください。

ドキュメント内のサンプル・コードのアクセシビリティについて

スクリーン・リーダーは、ドキュメント内のサンプル・コードを正確に読めない場合があります。コード表記規則では閉じ括弧だけを行に記述する必要があります。しかし JAWS は括弧だけの行を読まない場合があります。

外部 Web サイトのドキュメントのアクセシビリティについて

このドキュメントにはオラクル社およびその関連会社が所有または管理しない Web サイトへのリンクが含まれている場合があります。オラクル社およびその関連会社は、それらの Web サイトのアクセシビリティに関しての評価や言及は行っておりません。

Oracle サポート・サービスへの TTY アクセス

アメリカ国内では、Oracle サポート・サービスへ 24 時間年中無休でテキスト電話（TTY）アクセスが提供されています。TTY サポートについては、(800)446-2398 にお電話ください。

関連ドキュメント

詳細は、次の Oracle ドキュメントを参照してください。

- 『Oracle Application Server Web Services アドバンスト開発者ガイド』

  このマニュアルは、基本的な Web サービスのアセンブリの枠外となるトピックについて説明しています。たとえば、一般的な相互運用性の問題の診断方法、Web サービス管理機能（信頼性、監査、ロギングなど）を有効にする方法、Java 値タイプのカスタム・シリアライズの使用方法などの説明があります。

  また、Web サービス起動フレームワーク（WSIF）、Web サービス・プロバイダ API、メッセージ添付ファイルおよび管理機能（信頼性、ロギングおよび監査）の使用方法の説明もあります。さらに、トランスポート・メカニズムとしての JMS の使用方法など Web サービスの代替方針についても説明しています。

  参考までに xx ページの「『Oracle Application Server Web Services アドバンスト開発者ガイド』の内容」に、『Oracle Application Server Web Services アドバンスト開発者ガイド』の内容をリストします。

- 『Oracle Application Server Web Services Java API Reference』

  リファレンスでは、OracleAS Web Services Java API に対する Javadoc ツールの出力を提供します。

- 『Oracle Application Server Web Services セキュリティ・ガイド』

  このマニュアルでは、Oracle Application Server Web Services の Web サービスに適用できる、様々なセキュリティ計画について説明しています。適用できる計画には、ユーザー名トークン、X.509 トークン、SAML トークン、XML 暗号化および XML 署名があります。このマニュアルでは、クライアントおよびサービスでインバウンド・メッセージおよびアウトバウンド・メッセージ用に使用できる構成オプションについて説明します。また、様々な状況にあわせてこれらのオプションを構成する方法についても説明します。

- 『Oracle Containers for J2EE セキュリティ・ガイド』

  このマニュアルでは、OC4J に固有のセキュリティ機能および実装について説明しています。その他の Java セキュリティ・テクノロジはもちろん、Java Authentication and Authorization Service（JAAS）の使用方法に関する説明もあります。

- 『Oracle Containers for J2EE サービス・ガイド』

  このマニュアルでは、OC4J で提供される標準ベースの Java サービス（JTA、JNDI、JMS、JAAS など）および Oracle Application Server Java Object Cache について説明しています。

- 『Oracle Containers for J2EE 構成および管理ガイド』

  このマニュアルでは、OC4J のアプリケーションを構成および管理する方法について説明しています。これには、Oracle Enterprise Manager 10*g* Application Server Control コンソールの使用方法、OC4J で提供される標準準拠の MBean の使用方法、および該当する場合は OC4J 固有の XML 構成ファイルを直接使用する方法が含まれます。

- 『Oracle Containers for J2EE デプロイメント・ガイド』

  このマニュアルでは、OC4J 環境にアプリケーションをデプロイするための情報および手順について説明しています。Oracle Enterprise Manager 10*g* に付属するデプロイ・プラン・エディタの説明もあります。

- 『Oracle Containers for J2EE 開発者ガイド』

  このマニュアルでは、OC4J 上で実行するアプリケーションを作成する開発者が一般に関心を持つ項目について説明しています。つまり、サーブレット、EJB、または JSP コンテナなど、特定のコンテナに固有でない問題です。（たとえば、クラスのロードなど。）

Oracle Application Server の主要なドキュメント・グループからのリスト

- 『Oracle Application Server セキュリティ・ガイド』
- 『Oracle Application Server 管理者ガイド』
- 『Oracle Application Server Certificate Authority 管理者ガイド』
- 『Oracle Application Server Single Sign-On 管理者ガイド』
- 『Oracle Application Server エンタープライズ・デプロイメント・ガイド』

Oracle Web Services Manager は、サービス指向アーキテクチャを管理するための包括的なソリューションです。IT 管理者は、アクセス・ポリシー、ロギング・ポリシー、コンテンツ検証などの Web サービス操作を管理するポリシーを集中的に定義し、これらのポリシーでサービスを包み込むことができます。既存のサービスを変更する必要はありません。

- 『Oracle Web Services Manager クイック・スタート・ガイド』
- 『Oracle Web Services Manager インストレーション・ガイド』
- 『Oracle Web Services Manager 管理者ガイド』
- 『Oracle Web Services Manager デプロイメント・ガイド』
- 『Oracle Web Services Manager 拡張ガイド』

『Oracle Application Server Web Services アドバンスト開発者ガイド』の内容

このマニュアルは、『Oracle Application Server Web Services アドバンスト開発者ガイド』と併用されることを想定しています。このマニュアルでは、基本的な Web サービスのアセンブリの枠外となるトピックについて説明しています。

参考までに『Oracle Application Server Web Services アドバンスト開発者ガイド』の内容をここにリストします。

- 第 1 章「相互運用可能な Web サービスの実現」
- 第 2 章「メッセージ添付ファイルの処理」
- 第 3 章「Web サービスの管理」
- 第 4 章「Web サービスのセキュリティの確保」
- 第 5 章「Web サービスの信頼性の確保」
- 第 6 章「メッセージの監査およトレース」
- 第 7 章「Java 値タイプのカスタム・シリアライズの実装」
- 第 8 章「Web サービス・トランスポートとしての JMS の使用方法」
- 第 9 章「Web サービス起動フレームワークの使用方法」
- 第 10 章「Web サービス・プロバイダの使用方法」
- 付録 A「Web サービス管理スキーマの概要」
- 付録 B「Oracle Web Services のクライアント・スキーマ」
- 付録 C「OracleAS Web Services の信頼性スキーマ」
- 付録 D「OracleAS Web Services の監査スキーマ」
- 付録 E「OracleAS Web Services のトレース・スキーマ」
- 付録 F「JAX-RPC マッピング・ファイル記述子」
- 付録 G「Web サービス Mbean」
- 付録 H「Java 型の XML および WSDL 型へのマッピング」
- 付録 I「トラブルシューティング」
- 付録 J「サード・パーティ・ライセンス」

関連仕様へのリンク

次の各項に、このマニュアルの本文に出現するドキュメントへの参照をリストします。

- Java テクノロジ関連ドキュメント
- OC4J 関連ドキュメント
- SOAP 関連ドキュメント
- WSDL 関連ドキュメント
- UDDI 関連ドキュメント
- 暗号化関連ドキュメント

Java テクノロジ関連ドキュメント

- Java 2 Platform Enterprise Edition（J2EE）、バージョン 5（1.5）API 仕様

  `http://java.sun.com/javaee/5/docs/api`

- Java 2 Platform Enterprise Edition（J2EE）、バージョン 1.4 API 仕様

  `http://java.sun.com/j2ee/1.4/docs/api`

- J2EE デプロイメント・ディスクリプタ用 XML スキーマでは、Java 2 Platform, Enterprise Edition（J2EE）1.4 以降の仕様で記述された J2EE デプロイメント・ディスクリプタで使用されるドキュメン書式がリストされています。

  `http://java.sun.com/xml/ns/j2ee/`

- J2EE クライアント・スキーマでは、J2EE Web サービス・クライアント用の XSD が提供されています。

  `http://java.sun.com/xml/ns/j2ee/j2ee_web_services_client_1_1.xsd`

- Web アプリケーションおよび Web サービスを構築するための Java API for XML-based RPC（JAX-RPC）。この機能には、SOAP 1.1 仕様に基づいた XML ベースの RPC 機能が組み込まれています。

  `http://java.sun.com/webservices/jaxrpc/index.jsp`

- Java サーブレット 2.4 仕様

  `http://www.jcp.org/aboutJava/communityprocess/final/jsr154/index.html`

OC4J 関連ドキュメント

- 独自のデプロイメント・ディスクリプタを含む OC4J スキーマのリストは、次のとおりです。

  `http://www.oracle.com/technology/oracleas/schema/index.html`

- Oracle UDDI v2.0 サーバー実装

  `http://www.oracle.com/technology/tech/webservices/htdocs/uddi/index.html`

- 『Oracle Database JPublisher ユーザーズ・ガイド』

SOAP 関連ドキュメント

- SOAP 1.1 および 1.2 仕様（メイン・ページ）

  `http://www.w3.org/TR/SOAP`

- SOAP 1.1 仕様
  - 仕様

    `http://www.w3.org/TR/2000/NOTE-SOAP-20000508/`
  - SOAP 1.1 メッセージ・エンコーディング

    `http://schemas.xmlsoap.org/soap/encoding/`
  - SOAP 1.1 バインディング・スキーマ

    `http://schemas.xmlsoap.org/wsdl/soap/2003-02-11.xsd`

    SOAP 1.2 バインディング・スキーマは、ターゲットの名前空間が次のサイトの記載内容となる点を除いて SOAP 1.1 バインディング・スキーマと同じです。

    `http://schemas.xmlsoap.org/wsdl/soap12/`
- SOAP 1.2 仕様
  - SOAP 1.2 Part 1: Primer

    `http://www.w3.org/TR/soap12-part0/`
  - SOAP 1.2 Part 1: Messaging Format

    `http://www.w3.org/TR/soap12-part1/`
  - SOAP 1.2 Part 2 Recommendation（Adjuncts）

    `http://www.w3.org/TR/soap12-part2/`
  - HTTP トランスポート（SOAP 1.2 対応）

    `http://www.w3.org/2003/05/soap/bindings/HTTP`
  - SOAP バインディング・スキーマ

    `http://schemas.xmlsoap.org/wsdl/soap/2003-02-11.xsd`
  - SOAP スキーマの「fault code」要素の定義

    `http://schemas.xmlsoap.org/soap/envelope/2004-01-21.xsd`

WSDL 関連ドキュメント

Web Services Description Language（WSDL）仕様

`http://www.w3.org/TR/wsdl`

UDDI 関連ドキュメント

Universal Description, Discovery, and Integration 仕様

`http://www.uddi.org/`

暗号化関連ドキュメント

- 主要トランスポート・アルゴリズム
  - RSA-1_5

    `http://www.w3.org/2001/04/xmlenc#rsa-1_5`
  - RSA-OAEP-MGF1P

    `http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p`
- 署名キー
  - RSA-SHA1

    `http://www.w3.org/2000/09/xmldsig#rsa-sha1`
  - RSA-MD5

    `http://www.w3.org/2001/04/xmldsig-more#rsa-md5`
  - HMAC-SHA1

    `http://www.w3.org/2000/09/xmldsig#hmac-sha1`
  - DSA-SHA1

    `http://www.w3.org/2000/09/xmldsig#dsa-sha1`

表記規則

このマニュアルでは、次の表記規則を使用します。

規則	意味
太字	太字は、操作に関連する Graphical User Interface 要素、または本文中で定義されている用語および用語集に記載されている用語を示します。
イタリック	イタリックは、ユーザーが特定の値を指定するプレースホルダ変数を示します。
固定幅フォント	固定幅フォントは、段落内のコマンド、URL、サンプル内のコード、画面に表示されるテキスト、または入力するテキストを示します。

サポートおよびサービス

次の各項に、各サービスに接続するための URL を記載します。

Oracle サポート・サービス

オラクル製品サポートの購入方法、および Oracle サポート・サービスへの連絡方法の詳細は、次の URL を参照してください。

```
http://www.oracle.co.jp/support/
```

製品マニュアル

製品のマニュアルは、次の URL にあります。

```
http://otn.oracle.co.jp/document/
```

研修およびトレーニング

研修に関する情報とスケジュールは、次の URL で入手できます。

```
http://www.oracle.co.jp/education/
```

その他の情報

オラクル製品やサービスに関するその他の情報については、次の URL から参照してください。

```
http://www.oracle.co.jp
http://otn.oracle.co.jp
```

注意： ドキュメント内に記載されている URL や参照ドキュメントには、Oracle Corporation が提供する英語の情報も含まれています。日本語版の情報については、前述の URL を参照してください。

1

Web サービスの概要

この章では、Oracle Application Server Web Services リリース 10.1.3.1 の概要について説明します。第 3 章「OracleAS Web Services アーキテクチャとライフ・サイクル」に、Oracle Application Server Web Services のアーキテクチャを示します。

- Web サービスの理解
- Web サービスの標準
- 新機能および拡張機能
- 旧リリースの OracleAS Web Services との互換性
- クラスタリング環境と高可用性
- スタンドアロンでの OC4J と Oracle Application Server 環境での OC4J

Web サービスの理解

Web サービスはメッセージ・プロトコルのセット、プログラミング標準、ネットワーク登録 / 検出機能で構成されています。これらの機能を組み合せて使用することで、Web に接続している任意のデバイスから、認可されたユーザーにインターネット経由でビジネス機能を公開できます。

Web サービスは、Universal Resource Identifier (URI) によって識別されるソフトウェア・アプリケーションで、そのインタフェースとバインディングは、XML アーチファクトによって定義、記述および検出できます。Web サービスでは、XML ベースのメッセージやインターネット・ベースの製品を使用した他のソフトウェア・アプリケーションとの直接対話がサポートされています。

Web サービスでは次の処理が行われます。

- Web サービスの公開と記述 - Web サービス自体でその機能と属性が定義されるため、他のアプリケーションで認識できます。Web サービスは、Web Services Description Language（WSDL）ファイルを提供することによって、自身の機能を他のアプリケーションから使用できるようにします。
- Web 上での他のサービスによる検索 - Web サービスは、Universal Description, Discover, and Integration（UDDI）レジストリに登録することで、各アプリケーションから検索できます。
- 呼出し - Web サービスを検索して検査すると、リモート・アプリケーションでインターネット標準プロトコルを使用して、サービスを呼び出せます。
- Web サービスのスタイルは、リクエストとレスポンスまたは一方向のどちらかです。同期通信または非同期通信を使用できます。ただし、Web サービス・クライアントと Web サービスの間の情報交換の基本単位は、スタイルや通信のタイプにかかわらず、メッセージとなります。

Web サービスにより標準ベースのインフラストラクチャが提供され、ビジネスにおいて次のことが可能になります。

- 適切な社内ビジネス・プロセスを、他の組織が使用できる付加価値サービスとして提供します。
- 社内のビジネス・プロセスを統合し、ビジネス・パートナのビジネス・プロセスと動的にリンクさせます。

Web サービスの標準

Oracle は、この最新リリースにおいて、Web サービスのインフラストラクチャを拡張し、バージョン 1.4 および 1.5 の Java 2 Enterprise Edition（J2EE）仕様の Web サービスが実装できるようになりました。この項では、このリリースの Web サービスが準拠する標準をリストします。

- Java 2 Enterprise Edition
- Simple Object Access Protocol 1.1 および 1.2
- Web Services Description Language 1.1
- Web Service-Interoperability Basic Profile 1.1
- Web Service Inspection Language 1.0

Java 2 Enterprise Edition

最新リリースの Web サービスは次の Java J2EE 標準に準拠しています。

- JAX-RPC 1.1: クライアント API、メッセージ・ハンドラのサポート、サービス・エンドポイントの実装方法が定義されています。
- EJB 2.1: EJB API が定義されています。標準 JAX-RPC 1.1 では、EJB を Web サービス・エンドポイントとして公開する方法が指定されています。
- SAAJ 1.2: 添付ファイル付きの SOAP メッセージの処理方法が定義されています。
- Enterprise Web Services 1.1 仕様 :（JSR 109 および JSR 921 とも呼ばれる）Web サービスのデプロイおよび実行方法が指定されています。

Simple Object Access Protocol 1.1 および 1.2

Simple Object Access Protocol（SOAP）は、分散環境で情報を交換するための、XML ベースの軽量プロトコルです。SOAP では、リモート・プロシージャ・コール（RPC）やメッセージ指向など、様々なスタイルによる情報交換がサポートされます。RPC スタイルによる情報交換では、同期リクエスト / レスポンス処理が可能です。エンドポイントでは、プロシージャ指向のメッセージを受信して、相関するレスポンス・メッセージで応答します。メッセージ指向の情報交換では、ビジネスや他のスタイルのドキュメントの交換を必要とする組織とアプリケーションがサポートされます。この場合、メッセージは送信されますが、送信者は即時のレスポンスを予期または待機することができません（非同期）。メッセージ指向の情報交換は、ドキュメント・スタイルの交換とも呼ばれます。

SOAP の特性は、次のとおりです。

- プロトコル独立性
- 言語独立性
- プラットフォームおよびオペレーティング・システム独立性
- rpc/encoded または document/literal メッセージ書式のサポート
- 添付ファイルを含む SOAP XML メッセージのサポート（MIME マルチパート構造を使用）

最新リリースの Oracle Application Server Web Services は SOAP 1.2 プロトコルをサポートし、SOAP 1.1 と下位互換性を持っています。

関連資料 :

SOAP 1.1 および 1.2 仕様の詳細は、`http://www.w3.org/TR/SOAP/` を参照してください。

Web Services Description Language 1.1

RPC 指向とメッセージ指向の情報を含むネットワーク・サービスを記述するための XML フォーマット。プログラマまたは自動化された開発ツールは、サービスを記述する WSDL ファイルを作成し、その記述をインターネット経由で使用可能にできます。クライアント・サイドのプログラマと開発ツールは、公開された WSDL 記述を使用して、使用可能な Web サービスに関する情報を取得し、そのサービスにアクセスするクライアント・プロキシやプログラム・テンプレートを構築および作成できます。

関連資料 :

WSDL フォーマットの詳細は、`http://www.w3.org/TR/wsdl` を参照してください。

Web Service-Interoperability Basic Profile 1.1

Web Service-Interoperability（WS-I）は、プラットフォーム、アプリケーションおよびプログラミング言語間での Web サービスの相互運用性の向上を推進する組織です。現行のリリースは、WS-I Basic Profile 1.1 に準拠しています。

Web Service Inspection Language 1.0

Web Services Inspection Language（WS-Inspection または WSIL）は、軽量の Web サービス・ディレクトリ・プロトコルであり、サービスの単一ドキュメント・カタログのための広範なスキーマを提供しています。ドキュメントでは、いくつかのメタデータ（拡張データを含む）とともにサービスがリストされており、他のディレクトリにリンクしています。

通常、WSIL ドキュメントは、特定のサーバーまたはクラスタにデプロイされているサービスをリストします。BPEL プロセスなど、プロバイダとして公開されているものを含みます。WSIL は、SOA コンポーネント間の緊密な統合を可能にし、それにより、BPEL および OWSM のユーザーに、OC4J にデプロイされるサービスを検索する簡単な手段を提供します。

WSIL ドキュメントは、Application Server Control および BPEL と OWSM のツールにより、Web アプリケーションとして表示および管理できます。このアプリケーションは、Oracle の Web サービス・スタックに登録された MBean を利用して、サービスのリストを生成します。

Web アプリケーションは、通常、サーバー・ルートまたは `services` コンテキストに存在します。

`http://`*`yourdomain.com`*`/inspection.wsil`、または

`http://`*`yourdomain.com`*`/services/inspection.wsil`

Web アプリケーションを再デプロイして別のコンテキスト・パスにマッピングすることで、Web アプリケーションを移動できます。

セキュリティを高めるため、Web アプリケーションを削除することで WSIL を無効にすることができます。これにより、サーバーにデプロイされている他のサービスに関する情報が公開されなくなります。

新機能および拡張機能

前述の標準に加えて、最新リリースの OracleAS Web Services には次の新機能および拡張機能が含まれます。

- 認証、整合性および機密保護用の Web Services Security 標準
- Web サービス管理フレームワークと Application Server Control
- Java プラットフォーム用 Web サービス・メタデータ（J2SE 5.0 Web サービス注釈）
- REST Web サービス
- 拡張された Web サービス・テスト・ページ
- 構成およびスクリプティングのための Ant タスク
- シリアライズ用のカスタム型マッピング・フレームワーク
- データベース Web サービス
- SOAP ヘッダーのサポート
- MIME および DIME ドキュメントのサポート
- 添付ファイルとしての MTOM バイナリ・コンテンツ
- メッセージ配信サービスのクオリティ
- HTTP の代替としての JMS トランスポート
- Web サービス・プロバイダのサポート
- WSDL プログラミング構造を記述するための Web Services Invocation Framework

- SOAP メッセージの監査およびロギング
- Oracle BPEL
- Oracle Web Services Manager

認証、整合性および機密保護用の Web Services Security 標準

Organization for the Advancement of Structured Information Standards（OASIS）によって公開および更新されている WS-Security 標準。Web サービスの認証、メッセージの暗号化およびデジタル署名のためのプロファイルを提供します。現行のリリースでは次の機能を持つ WS-Security が実装されています。

- XML 署名
- XML 暗号化
- Username トークン
- X.509 トークン
- SAML トークン

関連資料：

Web サービス・セキュリティの詳細は、『Oracle Application Server Web Services セキュリティ・ガイド』を参照してください。

Web サービス管理フレームワークと Application Server Control

管理フレームワークは、セキュリティ、信頼性、ロギングおよび Web ベースの Application Server Control による監査のための構成および監視機能を提供します。これらの機能は一連の Java Management Extensions（JMX）Management Bean（MBean）によって公開されます。

関連資料：

- Web サービス管理の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの管理」を参照してください。
- MBean で制御可能な Web サービス機能の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービス Mbean」を参照してください。

Java プラットフォーム用 Web サービス・メタデータ（J2SE 5.0 Web サービス注釈）

現行リリースは、J2SE 5.0 Web サービス注釈（Web Services Metadata for the Java Platform（JSR-181）仕様とも呼ばれます）をサポートしています。この仕様には、Web サービスのプログラミングに使用する注釈付き Java 構文が定義されています。

関連項目：

注釈のサポート方法および Oracle による仕様の拡張の詳細は、第 11 章「注釈を使用した Web サービスのアセンブル」を参照してください。

REST Web サービス

現行のリリースは、Representational State Transfer（REST）サービスをサポートしています。このアーキテクチャは、Web アーキテクチャの理念を活用するものです。HTTP のセマンティクスを可能なかぎり使用します。

SOAP Web サービスとは異なり、REST はスタイルであるため、標準やベンダーによるツールのサポートはありません。また、REST Web サービスはメッセージの送信に XML 文書を使用し、SOAP エンベロープは使用しません。

REST Web サービスをアセンブルするには、WebServicesAssembler ツールを使用するか、または J2SE 5.0 Web サービス注釈をソース・ファイルに追加します。アセンブル後は Web サービス・テスト・ページを使用して、デプロイが正常に終了したかを確認し、機能をテストできます。

関連項目：

REST Web サービスの詳細は、第 12 章「REST Web サービスのアセンブル」を参照してください。

拡張された Web サービス・テスト・ページ

Web サービス・テスト・ページの機能が拡張され、Web サービスが正しくデプロイされたかどうかの確認以上のテストが可能になりました。テスト・ページでは、入力値を変えて Web サービスの操作を実行できます。サービスに送信する SOAP リクエストの表示および編集ができます。また、サービスが戻すレスポンスも表示されます。Web サービスでセキュリティおよび信頼性に関する機能を定義している場合は、テスト・ページのエディタを使用し、様々なセキュリティおよび信頼性の値を入力してサービスを実行できます。

また、テスト・ページでは、Web サービス・プロバイダおよび REST Web サービスも実行できます。REST Web サービスの場合、テスト・ページでは JAX-RPC Web サービスの場合と同じ機能が提供されるだけでなく、REST POST 操作および REST GET 操作も起動できます。

関連項目：

Web サービス・テスト・ページの使用方法の詳細は、第 13 章「Web サービス・デプロイのテスト」を参照してください。

構成およびスクリプティングのための Ant タスク

現行のリリースは、Web サービス開発用の Ant タスクを提供しています。これは特に、Web サービスのクライアント / サーバーの開発のスクリプティングおよび自動化を可能にするものです。サンプルの Ant タスクについては、このマニュアルの随所に記載されています。

関連項目：

- 環境にあわせた Ant の設定の詳細は、4-3 ページの「WebServicesAssembler 用の Ant の設定方法」を参照してください。
- Web サービスのアセンブリ・コマンド用の Ant タスクの記述方法の例は、第 18 章「WebServicesAssembler の使用方法」を参照してください。

シリアライズ用のカスタム型マッピング・フレームワーク

複雑なシステムの Web サービスでは、Web サービス・ランタイムで XML に自動的にシリアライズされるネイティブのデータ型以外へのデータ型のマッピングが要求される場合があります。現行リリースでは、カスタムのデータ型をマッピングするためのカスタム型マッピング・フレームワークを提供しています。

関連資料：

Web サービスでの標準外のデータ型の処理の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 値タイプのカスタム・シリアライズ」を参照してください。

データベース Web サービス

現行リリースでは引き続き、PL/SQL の Web サービスとしての公開をサポートするのみでなく、Oracle Database 10*g* における Web サービスへのコールアウト用の OracleAS Web Services の Java ランタイムとしての使用をサポートしています。

現行リリースでは、次のデータベース・アーチファクトを Web サービスとして宣言的に定義できるように、ランタイムおよびツール・サポートが拡張されています。

- PL/SQL ストアド・プロシージャ
- SQL 問合せ
- DML 文
- データベースの Java 仮想マシンにロードされる Java クラス
- Oracle Streams AQ（アドバンスト・キュー）

関連項目：

データベース・アーチファクトの Web サービスとしての実装方法の詳細は、第 10 章「データベース Web サービスのアセンブル」を参照してください。

SOAP ヘッダーのサポート

現行リリースでは、2 つの JAX-RPC 準拠メカニズムを使用した SOAP ヘッダーの高度な操作が可能になっています。

- JAX-RPC ハンドラを使用してプログラムによって SOAP ヘッダーを取得するメカニズム。このメカニズムを使用すると、メインの SOAP メッセージの処理から帯域外の SOAP ヘッダーを処理できます。
- SOAP ヘッダーを Java クラスのメンバー変数に自動的にマッピングするメカニズム。このメカニズムを使用すると、Web サービス実装内で、プログラム的なマッピング・プロセスではなく宣言的なマッピング・プロセスで、SOAP ヘッダーを変数として処理することができます。

関連項目：

- SOAP ヘッダーの詳細は、第 16 章「JAX-RPC ハンドラの使用方法」を参照してください。
- SOAP ヘッダーの処理の詳細は、第 17 章「SOAP ヘッダーの処理」を参照してください。

MIME および DIME ドキュメントのサポート

現行リリースでは、バイナリ・ドキュメントおよび大きな XML 文書の効率的な転送を支援するため、MIME および DIME 添付ファイルをサポートしています。

- Multipurpose Internet Mail Extensions（MIME）添付ファイルは、Java での WSDL の消費、および、MIME 添付ファイルを使用する SOAP を要求する Java Web サービスの WSDL への公開の両方に対応する、WS-I Attachment Profile に準拠しています。
- Direct Internet Message Encapsulation（DIME）添付ファイルは、Microsoft 社互換の、MIME と同様の規格です。Oracle は DIME を他の実装との下位互換の目的でサポートしています。これは Microsoft 社が現在 DIME を添付ファイルとして奨励していないためです。

関連資料：

OracleAS Web Services による MIME および DIME 添付ファイルのサポート方法の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「メッセージ添付ファイルの処理」を参照してください。

添付ファイルとしての MTOM バイナリ・コンテンツ

現在のリリースでは、クライアントとサーバーの間で、MTOM（SOAP Message Transmission Optimization Mechanism）書式を使用してバイナリ・コンテンツを送信できます。MTOM を使用すると、バイナリ・コンテンツを MIME 添付ファイルとして送信でき、ワイヤ上の伝送サイズが小さくなります。バイナリは、セマンティック的には XML ドキュメントの一部です。このことは、WS-Security 署名などの操作をメッセージに対して適用できるので、SWA（SOAP Messages with Attachments）より優る点です。

関連資料：

OracleAS Web Services による MIME および DIME 添付ファイルのサポート方法の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』のバイナリ・データを含む添付ファイルの処理に関する項を参照してください。

メッセージ配信サービスのクオリティ

現行リリースは、OASIS Web Services Reliability（WS-Reliability）を実現でき、最低 1 回のメッセージ配信、重複メッセージの排除（メッセージ配信は最大 1 回）、ただ 1 回のメッセージ配信（配信の保証と重複の排除）およびメッセージ・グループ内のメッセージの順序付けを保証しています。

関連資料：

これらの信頼性機能の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの信頼性の確保」を参照してください。

HTTP の代替としての JMS トランスポート

現行のリリースでは、SOAP メッセージのトランスポートとして、HTTP のかわりに JMS キューを使用できます。JMS は、より高いレベルの信頼できるメッセージ配信機能を SOAP メッセージに対して提供します。

HTTP がトランスポートとして必要な場合を考慮して、現行リリースでは、JMS キューおよびトピックへの SOAP メッセージの追加、および JMS キューおよびトピックからの SOAP メッセージの取得が引き続きサポートされており、この方法で処理されるメッセージを相関させる機能が追加されています。

関連資料：

- JMS を使用した Web サービス開発方法の詳細は、第 9 章「JMS 宛先を使用した Web サービスのアセンブル」を参照してください。
- Web サービス用トランスポート・メカニズムとしての JMS の使用方法の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービス・トランスポートとしての JMS の使用方法」を参照してください。

Web サービス・プロバイダのサポート

プロバイダ API を使用すると、JAX-RPC のような特定のサービス・エンドポイント実装計画に関係のない、Web サービス・エンドポイント用のカスタム処理ロジックを定義できます。プロバイダ・モデルを使用すると、複数のエンドポイントに対して共通の機能を提供できます。プロバイダ・モデルを使用すると、同じ機能を複数の Web サービスに組み込むのでなく、ロジックをランタイムに直接追加できます。

関連資料：

Web サービス・プロバイダの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービス・プロバイダの使用方法」を参照してください。

WSDL プログラミング構造を記述するための Web Services Invocation Framework

Web Services Invocation Framework（WSIF）は、WSDL を使用してプログラム・アーチファクトを記述する汎用性のある拡張可能なメカニズムであり、また、これらのアーチファクトをネイティブのプロトコル経由で呼び出すフレームワークです。現行リリースでは、初期 WSIF の実装がサポートされています。

関連資料：

WSIF の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービス起動フレームワークの使用方法」を参照してください。

SOAP メッセージの監査およびロギング

現行リリースには、インバウンドおよびアウトバウンドの SOAP メッセージをロギング・ファイルおよび監査ファイルに記録する機能が用意されています。Xpath 文を副問合せで使用し、メッセージ全体またはメッセージの一部をログに記録できます。

関連資料 :

メッセージのロギングおよび監査の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「監査メッセージおよびロギング・メッセージ」を参照してください。

Oracle BPEL

BPEL（Business Process Execution Language）は、Organization for the Advancement of Structured Information Standards（OASIS）が公開し、Oracle を含む主要ベンダーが支持している標準です。BPEL はビジネス・プロセス記述のための XML 構文を、（他の）ビジネス・プロセス（通常は Web サービス）の呼出しの組合せによって記述します。したがって、BPEL プロセスの定義は、OASIS によって管理されている XSD に従って構成された XML 文書です。

BPEL におけるビジネス・プロセスは、呼出し（Web サービスへのコールアウト）、受信（外部サービスからのコールイン）、単純な条件付きロジックおよび並行フローまたは並行順序を持つデシジョン・ポイントで構成されます。さらに、このデシジョン・ポイントが起動やデシジョン・ポイントなどから構成されます。ビジネス・プロセスの一環として変数を定義できます。変数は、起動時にビジネス・プロセスに渡されるパラメータから値を割り当てられ、呼出し内でやりとりされ、呼出し結果の値を設定され、ビジネス・プロセスの最終結果として戻されます。ビジネス・プロセス自体も Web サービスとして BPEL（ランタイム）エンジン内に実装されます。

Oracle による BPEL の実装では、WSIF テクノロジを使用して、ビジネス・プロセスを直接呼び出します。BPEL からの実際の呼出しを、Web サービスのオーバーヘッドである SOAP メッセージのマーシャリングおよびアンマーシャリングを行わない、Java クラスの直接コールにすることもできます。

関連資料 :

Oracle BPEL の詳細は、次の Web サイトを参照してください。

```
http://www.oracle.com/technology/products/ias/bpel/
index.html
```

Oracle Web Services Manager

Oracle Web Services Manager は、サービス指向アーキテクチャを管理するための包括的なソリューションです。IT 管理者は、アクセス・ポリシー、ロギング・ポリシー、コンテンツ検証などの Web サービス操作を管理するポリシーを集中的に定義し、これらのポリシーでサービスを包み込むことができます。既存のサービスを変更する必要はありません。また、Oracle Web Services Manager は、監視統計情報を収集してサービス・レベルとセキュリティを保証し、情報を Web ダッシュボードに表示します。結果として、Oracle Web Services Manager を使用すれば、Web サービスの管理と可視性を向上させることができます。

関連資料 :

Oracle Web Services Manager の詳細は、次の Web サイトを参照してください。

```
http://www.oracle.com/technology/products/
webservices_manager/index.html
```

旧リリースの OracleAS Web Services との互換性

OracleAS Web Services の 10.1.3.1 リリースは、10.1.3 リリースと完全に互換性があります。

リリース 9.0.4 または 10.1.2 の Oracle Application Server で稼働するように設計されているアプリケーションは、リリース 10.1.3.1 でも使用できます。ただし、9.0.4 または 10.1.2 の Web サービスは、10.1.3.1 の Application Server にデプロイできますが、このような Web サービスは管理コンソールには表示されません。

OracleAS Web Services 10.1.3.1 のアプリケーションの再デプロイ

下位互換性のため、Oracle Application Server 10*g* リリース 3（10.1.3.1）には、10*g* リリース 2（10.1.2）の Web サービスの実行に必要となる基盤ソフトウェアが含まれています。このため、Oracle Application Server 10*g*（9.0.4）および 10*g* リリース 2（10.1.2）で稼働するように設計された Web サービス・アプリケーションは、リリース 3 でも変更せずに使用できます。

ただし、Web サービスを 10*g* リリース 3（10.1.3.1）向けに再作成することには非常に大きなメリットがあります。たとえば、サービスのクオリティ（QOS）、標準ベースの開発モデル（JAX-RPC）および JMX ベースの管理など、最新機能をすべて利用できます。

関連資料：

既存の Web サービス・アプリケーションの OracleAS Web Services 10.1.3.1 への再デプロイの詳細は、Oracle Application Server のアップグレード・ガイドを参照してください。

無効な機能

- リリース 10.1.2 の Web サービス・スタックは、無効になっています。ただし、リリース 10.1.3.1 ではまだサポートされています。
- `ws.debug` システム・プロパティおよびその動作は無効になっています。
- `oracle.webservices.ClientConstants.WSM_INTERCEPTOR_PIPELINE_CONFIG` プロパティは無効になり、`oracle.webservices.ClientConstants.CLIENT_CONFIG` に置き換えられています。このプロパティの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』のプログラムで管理構成をクライアントに渡す方法に関する項を参照してください。
- `oracle.webservices.OracleStub` のプロパティ `ENCODING_STYLE_PROPERTY` および `OPERATION_STYLE_PROPERTY` は無効になっています。

クラスタリング環境と高可用性

OracleAS Web Services をクラスタリング環境で使用するには、クラスタの各マシンにサービスをインストールする必要があります。Application Server Control を使用すると、サービスを Oracle Application Server クラスタ内のインスタンス・グループに容易にデプロイできます。

標準のステートレスな Web サービスをインストールした場合は、クラスタリング環境へのデプロイによって、リクエストを処理できても状態を共有できない複数のインスタンスが存在するようになります。

ステートフルな Web サービスをインストールした場合は、複数のインスタンスで状態を共有できます。サービスがクラスタ内のすべてのマシンにインストールされていないと、クラスタ・ディスパッチャはサービス・リクエストをそのサービスのないマシンにディスパッチし、サービス呼出しエラーとなる場合があります。

クラスタリング環境をサポートするために、シリアライズ可能 Java クラスを持つステートフル Java Web サービスの場合は、WebServicesAssembler により Web サービスについて生成される J2EE の EAR ファイルの `web.xml` 内に `<distributable>` タグが追加されます。

関連資料：

- インスタンス・グループへのデプロイの詳細は、『Oracle Containers for J2EE デプロイメント・ガイド』の「クラスタ内の OC4J インスタンスへのデプロイ」を参照してください。
- ステートフル Web サービスのアセンブルアセンブルの詳細は、7-10 ページの「Java クラスとステートフル Web サービス」を参照してください。

スタンドアロンでの OC4J と Oracle Application Server 環境での OC4J

デプロイ時には、OC4J は通常単独で、つまり OracleAS Web Services 環境外で使用されます。このような OC4J をスタンドアロン OC4J と呼びます（管理対象外 OC4J と呼ぶこともあります）。このシナリオでは、OC4J は独自の Web リスナーを使用でき、外部の Oracle Application Server プロセスによる管理を受けません。これに対して、完全な Oracle Application Server 環境（管理対象 OC4J と呼ぶこともあります）では、Oracle HTTP Server を Web リスナーとして使用し、Oracle Process Manager and Notification Server（OPMN）を使用して環境を管理します。

関連資料：

- Oracle Application Server 環境とスタンドアロン環境の詳細、および Oracle HTTP Server と OPMN を OC4J とともに使用する方法の詳細は、『Oracle Containers for J2EE 構成および管理ガイド』を参照してください。
- Oracle HTTP Server およびそれに関連する `mod_OC4J` モジュールの一般的な情報は、『Oracle HTTP Server 管理者ガイド』を参照してください。Oracle HTTP Server から OC4J サーブレット・コンテナへの接続はこのモジュール経由で行います。
- OPMN の一般的な情報は、『Oracle Process Manager and Notification Server 管理者ガイド』を参照してください。

2

開発およびドキュメント・ロードマップ

次の各項では、Web サービスの開発手順を示すこのドキュメント内の推奨ロードマップを示します。

- 環境の設定
- ベスト・コーディング・プラクティス
- Web サービスのアーチファクトのアセンブル
- Web サービスのデプロイ
- デプロイされた Web サービスのテスト
- Web サービス・クライアントのアセンブル
- Web サービスへのサービスのクオリティ機能の追加
- Web サービスの拡張機能の追加
- Web サービスの代替方針
- 参照用の章および付録

環境の設定

この章には、OracleAS Web Services に用意されている機能を使用できるように環境を設定する方法が説明されています。

- 第 4 章「始める前に」

ベスト・コーディング・プラクティス

この項に記載されている各章には、Web サービス設計時の考慮事項が記載されています。

次の章には、OracleAS Web Services で使用可能な様々なメッセージ書式が説明されています。その章では、メッセージ書式ごとに長所と短所を説明し、クライアントの機能に応じて使用する書式を提案しています。

- 第 5 章「OracleAS Web Services メッセージ」

『Oracle Application Server Web Services アドバンスト開発者ガイド』の次の章には、相互運用性に関する問題がよく発生する可能性がある領域のいくつかが指摘されています。その章には、様々なプラットフォーム上のアプリケーションと Web サービスとの相互運用性を向上させるための設計上の提案やプログラミング上のガイドラインが記載されています。

- 「相互運用可能な Web サービスの実現」

Web サービスのアーチファクトのアセンブル

OracleAS Web Services を使用すると、Web サービスのアーチファクトをアセンブルする際、Java クラス、EJB、JMS 宛先、データベース・リソース、または J2SE 5.0 JDK の Web サービス注釈機能を使用するソース・ファイルからボトムアップ方式でアセンブルを開始できます。また、トップダウン方式で WSDL からアーチファクトをアセンブルし始めることもできます。OracleAS Web Services では、WebServicesAssembler ツールを使用してアセンブリを実行します。ツールのリファレンス・ガイドについては、第 18 章「WebServicesAssembler の使用方法」を参照してください。

次の各章には、WebServicesAssembler ツールを使用して OracleAS Web Services で可能な様々なタイプの Web サービスのアセンブリを実行する方法が説明されています。

- 第 6 章「WSDL からの Web サービスのアセンブル」
- 第 7 章「Java クラスを使用した Web サービスのアセンブル」
- 第 8 章「EJB を使用した Web サービスのアセンブル」
- 第 9 章「JMS 宛先を使用した Web サービスのアセンブル」
- 第 10 章「データベース Web サービスのアセンブル」
- 第 11 章「注釈を使用した Web サービスのアセンブル」

Web サービスのデプロイ

WebServicesAssembler ツールは、デプロイは実行しませんが、Web サービスを、デプロイ可能な EAR または WAR ファイルにパッケージ化します。このファイルのデプロイは、他の EAR ファイルまたは WAR ファイルを OC4J の実行中インスタンスにデプロイすることとよく似ています。OC4J には、デプロイの実行方法を説明したマニュアルが別途用意されています。

- 『Oracle Containers for J2EE デプロイメント・ガイド』

次の章には、パッケージ化のフォーマットやデプロイに必要なファイルに関する追加情報が説明されています。また、Oracle JDeveloper および Application Server Control ツールによるデプロイのサポートも簡単に説明されています。

- 第 19 章「Web サービスのパッケージ化およびデプロイ」

デプロイされた Web サービスのテスト

次の章には、Web サービスのテスト・ページが記載されています。このページを使用すると、コードを記述しなくても、デプロイが正常かどうかをテストできます。

- 第 13 章「Web サービス・デプロイのテスト」

Web サービス・クライアントのアセンブル

次の各章には、WebServicesAssembler ツールを使用して J2SE および J2EE プラットフォーム用の Web サービス・クライアントをアセンブルする方法が説明されています。

- 第 14 章「J2EE Web サービス・クライアントのアセンブル」
- 第 15 章「J2SE Web サービス・クライアントのアセンブル」

Web サービスへのサービスのクオリティ機能の追加

OracleAS Web Services は、セキュリティ、信頼性、メッセージ・ロギングおよび監査など、サービスのクオリティ機能をサポートしています。次の各章には、これらの機能の実装方法が説明されています。これらの機能を使用するには、Oracle JDeveloper や Application Server Control など、他のツールを使用します。次の各章は、『Oracle Application Server Web Services アドバンスト開発者ガイド』にあります。

- 「Web サービスの管理」
- 「Web サービスのセキュリティの確保」

  この章には、『Oracle Application Server Web Services セキュリティ・ガイド』の内容の概要のみが記載されています。このセキュリティ・ガイドには、メッセージ・レベルのセキュリティによる Web サービスの実装が説明されています。
- 「Web サービスの信頼性の確保」
- 「監査メッセージおよびロギング・メッセージ」

Web サービスの拡張機能の追加

次の各章には、Web サービスのパフォーマンスと機能を拡張する追加機能が記載されています。

- 第 16 章「JAX-RPC ハンドラの使用方法」
- 第 17 章「SOAP ヘッダーの処理」

『Oracle Application Server Web Services アドバンスト開発者ガイド』の次の各章も参照してください。

- 「メッセージ添付ファイルの処理」
- 「Java 値タイプのカスタム・シリアライズ」

Web サービスの代替方針

次の各章には、Web サービスの実装の代替モードが説明されています。

たとえば、Web サービス・コールの実行、SOAP 以外のプロトコル用のクライアントの作成、または HTTP 以外のトランスポート・メカニズムの使用などを行う、独自のインフラストラクチャを記述することができます。

- 第 12 章「REST Web サービスのアセンブル」

『Oracle Application Server Web Services アドバンスト開発者ガイド』の次の各章も参照してください。

- 「Web サービス・トランスポートとしての JMS の使用方法」
- 「Web サービス起動フレームワークの使用方法」
- 「Web サービス・プロバイダの使用方法」

参照用の章および付録

次の各章および付録には、このマニュアルで説明する実装および開発タスクを補足する情報が記載されています。

- 第 1 章「Web サービスの概要」
- 第 3 章「OracleAS Web Services アーキテクチャとライフ・サイクル」
- 第 18 章「WebServicesAssembler の使用方法」
- 付録 A「Web サービス・クライアントの API および JAR」
- 付録 B「WSDL 1.1 API の Oracle 実装」
- 付録 C「oracle-webservices.xml デプロイメント・ディスクリプタ・スキーマ」
- 付録 D「service-ref-mapping スキーマ」
- 付録 E「エラー・メッセージ接頭辞」

『Oracle Application Server Web Services アドバンスト開発者ガイド』の次の参照用の章および付録も参照してください。

- 「Web サービス管理スキーマの概要」
- 「OracleAS Web Services のクライアント・スキーマ」
- 「OracleAS Web Services の信頼性スキーマ」
- 「OracleAS Web Services の監査スキーマ」
- 「OracleAS Web Services のトレース・スキーマ」
- 「JAX-RPC マッピング・ファイル記述子」
- 「Web サービス Mbean」
- 「Java 型の XML および WSDL 型へのマッピング」

3

OracleAS Web Services アーキテクチャとライフ・サイクル

この章では、Oracle Application Server Web Services を構成するコンポーネントおよび公開可能なサービス・アーチファクトの概要を説明します。これらのコンポーネントは Java API for XML-Based RPC（JAX-RPC）で定義されます。この API を使用すると、Java テクノロジの開発者は Simple Object Access Protocol（SOAP）1.1 仕様に従って、XML ベースの RPC 機能を持つ Web アプリケーションおよび Web サービスを構築できます。

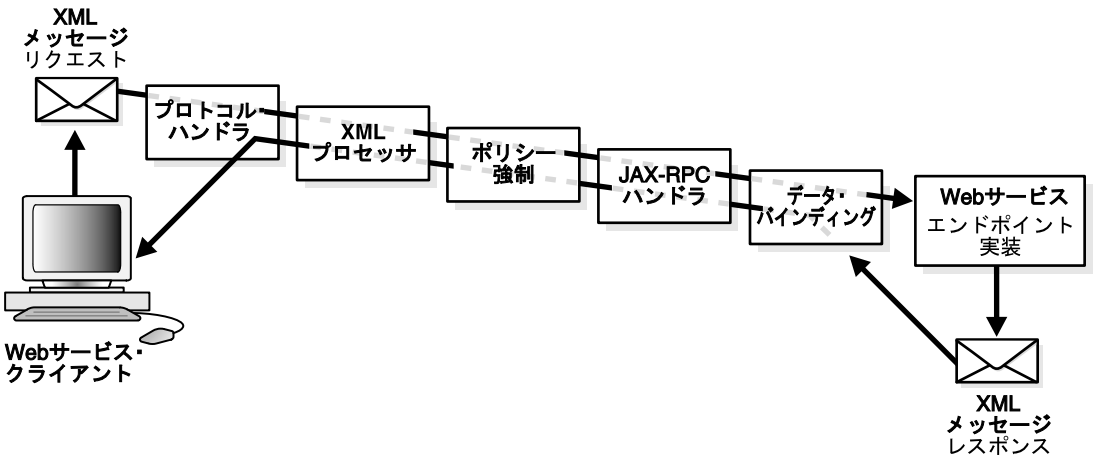
アーキテクチャ

OracleAS Web Services スタックは、パフォーマンス、相互運用性および管理性の 3 つの大きな目標を念頭に置いて設計されています。この項では、SOAP ベースの通信エンドポイントに対してエンタープライズ品質のインフラストラクチャを提供できる Web サービス・ランタイムの作成方法を説明します。

処理コンポーネント

Web サービス・リクエスト処理の各ステップは、ランタイム・インフラストラクチャの論理コンポーネントによって表されます。XML メッセージがシステムに配信される際、エンドポイント実装に配信されるまでに経由するレイヤーは、プロトコル・ハンドラ、XML プロセッサ、ポリシー強制パイプライン、JAX-RPC ハンドラ、データ・バインディングです。レスポンス・メッセージは同じ各インフラストラクチャ内を逆に経由して配信されます。

図 3-1 クライアントからサービスへの XML メッセージ・フロー



この項では、各処理レイヤーの目的を説明します。必要な場合は、システムの機能および構成の詳細が記述されている他の項への参照を示してあります。

プロトコル・ハンドラ

プロトコル・ハンドラは、Web サービス・インフラストラクチャへのエントリ・ポイントを提供します。プロトコル・ハンドラは、トランスポート・プロトコルを介した SOAP メッセージの送受信に使用します。HTTP または JMS を介してメッセージの送受信を行うように、Web サービス・インフラストラクチャを構成できます。

HTTP を介してメッセージを送信する場合は、OracleAS Web Services スタックは Web サービス・メッセージのサービスへの送信に Oracle HTTP クライアント・ライブラリを使用し、クライアントが送信した Web サービス・メッセージの受信に OC4J サーブレット・エンジンを使用します。Oracle サーブレット環境のすべての機能および管理インフラストラクチャを Web サービスで使用できます。たとえば、HTTP または HTTPS を介して暗号化データ・ストリームを使用して Web サービスにアクセスすることができます。

JMS トランスポートは、様々な JMS プロバイダと連携するように構成することができます。JMS トランスポートは、Oracle Application Server 付属の JMS インフラストラクチャを使用してアプリケーション・サーバーに組み込まれています。JMS は、SOAP メッセージの交換時にメッセージ・バス機能のサービス・クオリティを最大限に確保するために使用される場合がよくあります。

XML 処理

SOAP メッセージはトランスポート・レイヤーから取得されると、SOAP with Attachments API for Java（SAAJ）と互換性のある XML メッセージ表現に変換されます。SAAJ メッセージは、パフォーマンス向上と効率的なメモリー使用を目的として Oracle が最適化した XML 解析テクノロジを使用して構成されます。このメッセージは、OC4J が提供する JAX-RPC 準拠の SOAP 処理インフラストラクチャの基礎となります。SAAJ メッセージはインスタンス化されると、処理スタックの次のレイヤーに配信されます。

SOAP リクエストは部分的に XML ではない場合もあります。たとえば、SOAP メッセージが添付ファイル付きで送信される場合があります。添付ファイルが使用されるのは、非 XML コンテンツを SOAP メッセージと一緒にパッケージ化する場合です。SAAJ 実装もまた添付ファイルを処理します。

XML メッセージのデフォルト処理は、メッセージ・ペイロードが SOAP 1.1 または SOAP 1.2 準拠のエンベロープにエンコードされていることを想定していますが、SOAP を使用せずに直接 HTTP を介して XML メッセージを受信し、ディスパッチするように OracleAS Web Services スタックを構成することも可能です。この場合、開発者は、SOAP プロトコルを認識しない既存の HTTP インフラストラクチャと直接統合するアプリケーションを作成できます。HTTP および URL を使用してシステムを記述するメッセージを定義することで、REST アーキテクチャ・スタイルに準拠するようにアプリケーションを構築できます。

XML-over-HTTP メッセージングを使用するようにインフラストラクチャを構成した場合は、メッセージにアプリケーション・メッセージが直接含まれているのか、または SOAP エンベロープがペイロードのトップレベル要素として含まれているのかが、インフラストラクチャで判断されます。メッセージが RAW XML の場合は、処理レイヤーがヘッダーを持たない SOAP エンベロープ内にメッセージをラップします。この SOAP エンベロープは、以降の Web サービス処理要素で処理でき、処理のためエンドポイント実装に配信できます。

XML メッセージは、次に、Web サービス・スタックのポリシー強制メカニズムで処理されます。

ポリシーの強制

OracleAS Web Services スタックは、ランタイム管理ポリシーを強制する管理チェーンを有効化するように、追加情報を使用して構成できます。これらのポリシーには、WS-Reliability、WS-Security、監査およびロギング機能などの Web サービス管理機能が含まれます。これらの機能の設定方法の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの管理」を参照してください。セキュリティ設定の詳細は、『Oracle Application Server Web Services セキュリティ・ガイド』を参照してください。有効化されたポリシーを、サービスに関連付けられた WSDL ドキュメントに追加すると、クライアント・インターセプタ・パイプラインの自動構成がサポートされます。

OracleAS Web Services スタックがサポートしているプロトコルの 1 つに、SOAP メッセージの配信を保証する WS-Reliability 標準があります。信頼性インフラストラクチャでは、WS-Reliability プロトコルに準拠した非同期の確認メッセージをシステム・インフラストラクチャが送受信できる追加機能がサポートされています。これは、Oracle クライアント・インフラストラクチャの拡張としてサポートされており、JAX-RPC の Stub または Call オブジェクトを使用する場合に使用できます。

インターセプタ・チェーンは、OracleAS Web Services 管理エージェント・パイプラインに委任されるようにも構成できます。これにより、OracleAS Web Services 管理製品のサポートがあらかじめ組み込まれ、データ・センター全体での Web サービスに対するポリシー管理をサポートする機能が提供されます。

管理インターセプタは、OC4J 付属のシステム・サービスのランタイム・インフラストラクチャを提供します。アプリケーション固有のインターセプタは、JAX-RPC 1.1 ハンドラ API に適合する形でサポートされます。

JAX-RPC ハンドラ

ハンドラは、アプリケーション固有の SOAP ヘッダーを固有のロールまたはアクター属性に従って処理するように構成されます。JAX-RPC ハンドラは SOAP メッセージの SAAJ 表現にアクセスし、任意のレベルの SOAP メッセージに対して操作を実行できます。

インターセプタと JAX-RPC ハンドラが連携することで、SOAP 処理モデルが実現されます。これにより、Web サービス・エンドポイントが、特定のノード向けの SOAP ヘッダーを選択して処理できます。次に、SOAP メッセージがシステムの他のノードに渡され、SOAP ヘッダーが必要に応じて処理されます。

関連項目：

ハンドラ API の使用方法の詳細は、第 16 章「JAX-RPC ハンドラの使用方法」および第 17 章「SOAP ヘッダーの処理」を参照してください。

データ・バインディング

多くのアプリケーションにおいて、XML ペイロードの各部分は Java オブジェクトに変換され、変換された Java オブジェクトはアプリケーション・フレークワークによって使用されます。この機能は通常、データ・バインディングと呼ばれ、XML データの各部分を Java クラス階層の各メンバーにバインドすることです。データ・バインディングのプロセスは、XML から Java への変換を管理するシリアライズ・フレームワークを介して駆動されます。シリアライズ・フレームワークは拡張可能であり、開発者が XML データから Java オブジェクトへのカスタム型マッピングを定義できます。

OC4J ランタイムには、SAAJ メッセージの処理用に最適化された特別なプロバイダが実装されています。このプロバイダを使用する場合、データ・バインディングは実行されません。プロバイダを使用すると、SOAP メッセージの XML ペイロードを直接操作するアプリケーションを実装できます。

エンドポイント実装

Web サービス・スタックの前述の 4 つのレイヤーを通過後、アプリケーションのビジネス・ロジックを含むエンドポイント実装が起動されます。エンドポイントは通常の Java クラス、Enterprise JavaBean またはプロバイダのいずれかです。JMS エンドポイント構成が有効な場合は、JMS キューもエンドポイントとして使用できます。

OracleAS Web Services コンテナでは、Java クラスが Web サービスとして公開されるために準拠する必要がある要件はわずかです。エンドポイント実装がそのライフ・サイクル全体でコンテナとの間でより複雑な対話を必要とする場合は、エンドポイント実装は、JAX-RPC `ServiceLifecycle` インタフェースを実装できます。このインタフェースでは、サービスの初期化時およびリクエストの処理中に、エンドポイントへの Web サービスのリクエストの詳細が提供されます。

Java クラスにはまた、Web サービス固有の注釈を追加できます。この注釈を使用して、Web サービス操作として公開するメソッド、サービスへのアクセスに使用可能なプロトコルなどを指定する追加構成情報を提供できます。

関連項目：

OracleAS Web Services の J2SE 5.0 JDK Web Service Annotations サポートの詳細は、第 11 章「注釈を使用した Web サービスのアセンブル」を参照してください。

Java Management Extensions（JMX）

管理ポリシーの強制にインターセプタ・パイプラインを使用できる一方で、Java Management Extensions（JMX）標準を使用したシステム管理機能が OracleAS Web Services スタックに公開されています。JMX を使用することで、管理者は OC4J システムの正常動作に関する重要なメトリックを収集し、動作中のシステムの構成を変更できます。JMX のメトリックおよび操作は、Application Server Control ツールの Web サービス・コンソールから使用可能です。

開発ツール

OracleAS Web Services 実装のもう 1 つの主要な機能は、エンドポイント実装の開発およびデプロイを可能にする開発ツールです。Web サービスは、コマンドライン・ツールまたは Java の Integrated Development Environment（IDE）を使用して開発できます。

WebServicesAssembler（WSA）はコマンドライン・ツールであり、Web サービスを WSDL またはエンドポイント実装クラスからデプロイするために必要なアーチファクトの生成に使用します。WSA は、スクリプト化された環境における Web サービスの自動作成に役に立ちます。理由は、WSA は、コマンドラインであることに加えて、その機能を Ant タスクとしても公開するからです。

Oracle JDeveloper は、フル機能を装備した Oracle の Java IDE であり、Web サービスのエンドツーエンド開発に使用可能です。開発者は、Java クラスまたは EJB を作成し、Web サービスとして公開し、Oracle Application Server のインスタンスに自動的にデプロイし、ただちに Web サービスの動作テストを行うことができます。または、Oracle JDeveloper を使用して、WSDL 記述から Web サービスを作成することもできます。WebServicesAssembler と同様に、Oracle JDeveloper も Ant に対応しています。JDeveloper を使用して、クライアントのアセンブルおよびサービスのアセンブルとデプロイを行う Ant スクリプトを作成し、実行できます。

関連項目：

webServicesAssembler ツールを使用して Web サービス・アーチファクトをアセンブルする方法の詳細は、第 18 章「WebServicesAssembler の使用方法」を参照してください。

Web サービス開発のライフ・サイクル

この項では、Web サービスを開発する際の複数の開発段階について説明します。

1. 実装の作成
2. Web サービスのアセンブル
3. クライアントのアセンブル
4. Web サービスのデプロイ
5. Web サービスのテスト
6. デプロイ後タスクの実行

実装の作成

Web サービスとして公開する実装を作成します。OracleAS Web Services では、次のような様々なアーチファクトを Web サービスとして公開できます。

- Java クラス
- Enterprise JavaBeans（EJB）
- JMS キューまたは JMS トピック
- PL/SQL プロシージャ
- SQL 文
- Oracle アドバンスト・キュー
- データベースの Java クラス
- CORBA サーバント・オブジェクト

これらのアーチファクトは、任意のツールまたは IDE を使用して開発できます。Oracle Oracle JDeveloper を使用すると、Java クラス、JMS キューおよび JMS トピック、PL/SQL プロシージャ、CORBA サーバント・オブジェクトおよび EJB を作成できます。

Web サービスのアセンブル

OracleAS Web Services では、Web サービスをアセンブルするための様々なコマンドが提供されています。WSDL から開始するトップダウン方式のコマンドと、Java クラス、EJB、データベース・アーチファクトまたは JMS キューから開始するボトムアップ方式のコマンドがあります。これらのコマンドはコマンドラインでの入力、または Ant スクリプトへのタスクとしての記述が可能です。

Oracle JDeveloper は、Ant に完全対応しているため、Ant スクリプトの構築および実行に使用できます。また、Oracle JDeveloper には、いくつかの Web サービス生成コマンドとまったく同様に動作する設計ウィザードが装備されています。これらのウィザードを使用することで、開発プロセスをスピードアップでき、ビルド・スクリプト作成のステップを省略できます。

最新の Java IDE（Eclipse など）も Ant に対応しています。Oracle JDeveloper を使用しない場合は、これらの IDE を使用して Ant スクリプトを実行できます。

たとえば、WSDL の生成やサービスのクオリティ機能の追加など、他の OracleAS Web Services コマンドを起動する必要がある場合は、コマンドラインまたは Ant タスクで起動できます。これらのコマンドをサポートする設計ウィザードは用意されていませんが、Ant タスクは Oracle JDeveloper で直接実行できます。

関連項目：

Web サービスをアセンブルするコマンドの詳細は、18-4 ページの「Web サービス・アセンブリ・コマンド」を参照してください。

クライアントのアセンブル

OracleAS Web Services には、J2SE および J2EE のクライアント・コードをアセンブルするコマンドが用意されています。

Oracle JDeveloper は、OC4J J2SE Web サービス・クライアントをサポートします。これは、Web サービス WSDL 記述からの Java スタブの作成を可能にするという形で行われます。作成したスタブを使用して、既存の Web サービスにアクセスできます。

Web サービスのデプロイ

Web サービスのデプロイは、Java 言語のコマンドまたは Ant タスクを使用して実行できます。Ant タスクは、ビルド・ファイルへの記述、または Oracle JDeveloper からの直接発行が可能です。また、Oracle JDeveloper には、デプロイ可能な EAR ファイルを構成してデプロイするデプロイ・ウィザードが用意されています。

デプロイは Application Server Control でも実行できます。ただし、このツールは EAR ファイルの構成やそのコンテンツの変更には使用できません。

Web サービスのテスト

OracleAS Web Services スタックでは、デプロイされた Web サービスごとに Web サービス・テスト・ページが提供されます。サービス・エンドポイントのアドレスを Web ブラウザに入力することで、Web サービスによって公開されている操作にアクセスできます。対話式のページを使用して、入力した値による操作を実行できます。

WebServicesAssembler コマンドライン・ツールを使用すると、Web サービスの各メソッドの JUnit フレームワークに関して適切なテストを生成できます。WebServicesAssembler ツールを使用して、`assemble`、`plsqlAssemble` および `genProxy` コマンドに対する JUnit テストを生成すると、Oracle JDeveloper はこれらのテストをデフォルトでインポートできます。

デプロイ後タスクの実行

デプロイ後に実行可能なタスクがいくつか存在します。これらのうちのいくつかは、セキュリティ・ポリシーやロギング・ポリシーの変更など、Web サービスのパフォーマンスの微調整に関するものです。また、キー・ストア構成の変更など、Web サービスの大幅な変更に関するものもあります。これらの変更は、動的に行うことが可能なものもありますが、Web サービスの再デプロイを必要とするものもあります。これらのタスクを実行する場合は、Oracle Application Server Control を使用します。これらのタスクの詳細は、Application Server Control のオンライン・ヘルプを参照してください。

4

始める前に

OC4J の設定および実行方法は、OC4J ディストリビューションの最上位レベルにある `README.txt` ファイルを参照してください。この章は、この `README` ファイルの追補です。この章では、ユーザー環境における Oracle Application Server Web Services の設定の詳細情報について説明します。

この章の内容は、次のとおりです。

- サポート対象のプラットフォーム
- OC4J のインストール方法
- OracleAS Web Services 用の環境の設定方法
- WebServicesAssembler 用の Ant の設定方法
- データベース要件

サポート対象のプラットフォーム

OracleAS Web Services は、次のプラットフォームでサポートされています。

- Redhat Linux
- Solaris Sparc 32 および 64 ビット
- Microsoft Windows: Windows XP、Windows 2000、Windows Server 2003

OC4J のインストール方法

OC4J のインストールおよび実行方法については、`README.txt` ファイルの指示に従ってください。`README.txt` ファイルは、OC4J ディストリビューションの最上位レベルにあります。

OracleAS Web Services 用の環境の設定方法

この項では、OracleAS Web Services を使用するためにインストールする必要があるソフトウェアと定義する必要がある環境変数について説明します。

- Java2 Standard Edition（J2SE）JDK バージョン 1.4.1 以上。
- Java2 Standard Edition（J2SE）JDK バージョン 5.0 の Web サービス注釈機能を使用する場合は、JDK5.0 以上。
- Ant タスクを使用して Web サービスをアセンブルする場合は、Jakarta Ant 1.5.x または 1.6.5（推奨）が必要です。Jakarta Ant 1.6.5 は、OracleAS Web Services ディストリビューションに含まれています。
  - 独自のインストールを使用する場合は、次の Web アドレスから Ant を入手できます。

    `http://jakarta.apache.org/ant/index.html`
  - Ant の独自インストールを使用する場合、Web サービスをアセンブルするための Ant の設定方法については、4-3 ページの「WebServicesAssembler 用の Ant の設定方法」を参照してください。
  - Oracle Application Server に付属している Ant のバージョンをインストールした場合、OracleAS Web Services 用の Ant タスクはすでに構成されています。「Oracle Application Server に付属している Ant 1.6.5 の設定方法」の指示に従い、環境およびビルド・ファイルを適切に変更してください。
- 次の環境変数を定義します。
  - *`ORACLE_HOME`*: OC4J のインストール・ディレクトリを指し示すようにこの変数を定義します。
  - *`JAVA_HOME`*: この変数は、J2SE SDK のインストール・ディレクトリを指すように定義します。OracleAS Web Services は、*`JAVA_HOME`*`/bin/java` が Java VM の場所を示し、*`JAVA_HOME`*`/bin/javac` が Java コンパイラの実行可能ファイルの場所を示しているものと考えます。
- 次のファイルまたはパスをクラスパスに追加します。
  - Web サービスのアセンブリと XML ファイルの処理をより効率的にするには、WebServicesAssembler の JAR（*`ORACLE_HOME`*`/webservices/lib/wsa.jar`）へのパスと XML パーサーの JAR へのパス（*`ORACLE_HOME`*`/lib/xmlparserv2.jar`）をクラスパスに追加します。
  - Ant を使用して Web サービス・プログラムをアセンブルおよびコンパイルする場合、適切な `CLASSPATH` 設定については、Web サービスのコード例に付随する Ant スクリプトを参照してください。コマンドラインで WebServicesAssembler を起動し、生成されたコードをコンパイルする場合、完全なライブラリを `CLASSPATH` に追加できます。

    すべての OracleAS Web Services クライアント JAR ファイルに対する Windows プラットフォームの `set CLASSPATH` コマンドのサンプルは、A-8 ページの

「CLASSPATH コマンドのサンプル」を参照してください。UNIX プラットフォームのクラスパスも同じ方法で設定されます。

– OracleAS Web Services には、J2SE 環境で Web サービス・クライアントを実行するためのライブラリとして別に `wsclient_extended.jar` が用意されています。このライブラリには、J2SE 環境で OracleAS Web Services クライアントに必要となるすべてが含まれています。このライブラリにより、J2SE クライアント・アプリケーションのパッケージ化と配布が簡単になります。`wsclient_extended.jar` ファイルは、Oracle Technology Network の Web サイトの次のアドレスにあります。

  `http://download.oracle.com/otn/java/oc4j/10131/wsclient_extended.zip`

  OC4J Companion CD をインストールした場合、`wsclient_extended.jar` ファイルは *`ORACLE_HOME`*`/webservices/lib` ディレクトリにも置かれます。

  `wsclient_extended.jar` ファイルの詳細は、A-3 ページの「wsclient_extended.jar によるクラスパスの単純化」を参照してください。

- クライアントとサーバー間で信頼性のあるメッセージ機能を有効にする場合、クライアントとサーバーの両方に対して表をインストールする SQL スクリプトを実行する必要があります。

  SQL スクリプトの検索および実行方法については、『Oracle Application Server Web Services アドバンスト開発者ガイド』のクライアントおよびサーバーに対する SQL 表のインストールに関する項を参照してください。

- 信頼性のあるメッセージ機能を使用するか、データベース Web サービスをアセンブルする場合、Oracle データベースをインストールして稼働しておく必要があります。

  このトピックの詳細は、4-7 ページの「データベース要件」を参照してください。

WebServicesAssembler 用の Ant の設定方法

WebServicesAssembler ツールを使用すると、OracleAS Web Services をアセンブルできます。WebServicesAssembler ツールを使用すると、トップダウン方式とボトムアップ方式のどちらを使用して Web サービスを作成するかにかかわらず、サービスの開発およびデプロイに必要なアーチファクトを生成できます。WebServicesAssembler のコマンドは、コマンドラインと Ant タスクのどちらからもコールできます。

この項では、WebServicesAssembler コマンドを Ant タスクからコールできるように環境を設定し、スクリプト・ファイルをビルドする方法について説明します。以前にインストールした Ant インストールを使用するか、または *`ORACLE_HOME`*`/ant` にある Ant を使用することができます。次の各項では、Ant を、インストールしたバージョンに応じて設定する方法について説明します。

> **注意：** このマニュアルのすべての Ant タスク例では、Ant バージョン 1.6.5 以上を使用していることを前提としています。これらのバージョンでは、タスクの名前空間を使用できます。このため、WebServicesAssembler コマンドに対応するすべての Ant のタグおよびサブタグには、接頭辞として `oracle:` 名前空間が付いています。

Oracle Application Server に付属している Ant 1.6.5 の設定方法

次の手順では、*ORACLE_HOME*/ant にある Ant 1.6.5 インストールとともに WebServicesAssembler を使用できるように環境およびビルド・ファイルを設定する方法について説明します。この Ant のバージョンは、Oracle Application Server に付属しているものです。

1. PATH 変数の前に *ORACLE_HOME*/ant/bin を入力します。
2. ビルド・スクリプト（build.xml）を編集します。インポートされる Ant タスクに antlib:oracle 名前空間宣言を追加します。次の例では、*bottomup* がプロジェクトの名前です。

   ```
   <project name="bottomup" default="all" basedir="." xmlns:oracle="antlib:oracle">
   ```

3. oracle: 名前空間を接頭辞として WebServicesAssembler のすべてのタグに追加します。次に例を示します。

   ```
   <oracle:assemble ....>
          <oracle:port ... />
   </oracle:assemble>
   <oracle:genProxy ..../>
   ```

4. 必要に応じて、ant-oracle.properties ファイルをビルド・スクリプトと同じディレクトリにコピーします。

   j2ee/utilities ディレクトリにあるこのプロパティ・ファイルを変更し、ビルド・スクリプトから参照することもできますが、このファイルはテンプレートとして保持しておくことをお薦めします。

5. 必要に応じて、インストール環境にあわせて ant-oracle.properties ファイルを編集します。
6. 必要に応じて、ビルド・スクリプト（build.xml）を編集します。ビルド・スクリプトで ant-oracle.properties ファイルを参照します。次に例を示します。

   ```
   <property file="ant-oracle.properties"/>
   ```

7. 必要に応じて、レポート用として junit Ant タスクを使用する場合、ANT_OPTS システム・プロパティを Xalan TransformerFactoryImpl クラスに設定します。
   - JDK 1.5 を使用している場合、TransformerFactory の ANT_OPTS プロパティを com.sun.org.apache.xalan.internal.xsltc.trax.TransformerFactoryImpl に設定します。次に例を示します。

     ```
     set ANT_OPTS=-Djavax.xml.transform.TransformerFactory=com.sun.org.apache.xalan.
     internal.xsltc.trax.TransformerFactoryImpl
     ```

   - JDK 1.4 を使用している場合、TransformerFactory の ANT_OPTS プロパティを org.apache.xalan.processor.TransformerFactoryImpl に設定します。次に例を示します。

     ```
     set ANT_OPTS=-Djavax.xml.transform.TransformerFactory=org.apache.xalan.
     processor.TransformerFactoryImpl
     ```

1.6.5 より前の Ant インストールを使用した Ant 1.6.5 の設定方法

次の手順では、Ant 1.6.5（以上）より前のインストールとともに WebServicesAssembler を使用できるように環境およびビルド・ファイルを設定する方法について説明します。

1. ディレクトリ *ORACLE_HOME*/j2ee/utilities にナビゲートし、次のファイルが存在することを確認します。
   - ant-oracle.properties: このファイルを使用すると、Oracle Ant タスクを実行するためのキー・プロパティを指定できます。
   - ant-oracle.xml: このファイルを使用すると、Oracle Ant タスクを使用できます。
2. ant-oracle.properties および ant-oracle.xml ファイルをビルド・スクリプト（build.xml）と同じディレクトリにコピーします。

   j2ee/utilities ディレクトリにあるファイルを変更し、ビルド・スクリプトから参照することもできますが、ソース・ファイルはテンプレートとして保持しておくことをお薦めします。また、ant-oracle.xml ファイルを元の場所に残しておく場合、インポート参照をハードコード化し、ファイルのフルパス（c:/oc4j/j2ee/utilities/ant-oracle.xml など）を指定する必要があります。
3. インストール環境にあわせて ant-oracle.properties ファイルを編集します。
4. ビルド・スクリプト（build.xml）を編集します。
   - ビルド・スクリプトに ant-oracle.xml ファイルをインポートします。

     ```
     <!-- Import for OC4J ant integration. -->
     <import file="ant-oracle.xml"/>
     ```
   - インポートされた Ant タスクに antlib:oracle 名前空間の参照を追加します。次の例では、*bottomup* がプロジェクトの名前です。

     ```
     <project name="bottomup" default="all" basedir="."
     xmlns:oracle="antlib:oracle">
     ```
5. oracle: 名前空間を接頭辞として WebServicesAssembler のすべてのコマンドに含めます。次に例を示します。

   ```
   <oracle:deploy ..../>
   <oracle:genProxy ..../>
   ```
6. 必要に応じて、レポート用として junit Ant タスクを使用する場合、ANT_OPTS システム・プロパティを Xalan TransformerFactoryImpl クラスに設定します。
   - JDK 1.5 を使用している場合、TransformerFactory の ANT_OPTS プロパティを com.sun.org.apache.xalan.internal.xsltc.trax.TransformerFactoryImpl に設定します。次に例を示します。

     ```
     set ANT_OPTS=-Djavax.xml.transform.TransformerFactory=com.sun.org.apache.xalan.
     internal.xsltc.trax.TransformerFactoryImpl
     ```
   - JDK 1.4 を使用している場合、TransformerFactory の ANT_OPTS プロパティを org.apache.xalan.processor.TransformerFactoryImpl に設定します。次に例を示します。

     ```
     set ANT_OPTS=-Djavax.xml.transform.TransformerFactory=org.apache.xalan.
     processor.TransformerFactoryImpl
     ```

1.5.2 より前の Ant インストールを使用した Ant 1.5.2 の設定方法

次の手順では、Ant 1.5.2 より前のインストールとともに WebServicesAssembler を使用できるように環境およびビルド・ファイルを設定する方法について説明します。

1. Ant および Java JDK のインストールへのパスが CLASSPATH 環境変数にすでに含まれていることを確認します。
2. `wsa.jar` のパスを CLASSPATH 環境変数に追加します。通常、このパスは次のとおりです。

   OC4J_Home`/webservices/lib/wsa.jar`

   この例では、*OC4J_Home* は、OC4J をインストールしたディレクトリです。
3. この Ant タスクを起動するすべての Ant ビルド・ファイルに次の行を追加します。

   ```
   <taskdef resource="orawsa.tasks" />
   <typedef resource="orawsa.types" />
   ```

   これらの行は、ビルド・ファイル内において、最初の WebServicesAssembler タスクがコールされるより前の場所なら、任意の場所に指定できます。
4. 必要に応じて、レポート用として `junit` Ant タスクを使用する場合、`ANT_OPTS` システム・プロパティを Xalan `TransformerFactoryImpl` クラスに設定します。
   - JDK 1.5 を使用している場合、`TransformerFactory` の `ANT_OPTS` プロパティを `com.sun.org.apache.xalan.internal.xsltc.trax.TransformerFactoryImpl` に設定します。次に例を示します。

     ```
     set ANT_OPTS=-Djavax.xml.transform.TransformerFactory=com.sun.org.apache.xalan.internal.xsltc.trax.TransformerFactoryImpl
     ```
   - JDK 1.4 を使用している場合、`TransformerFactory` の `ANT_OPTS` プロパティを `org.apache.xalan.processor.TransformerFactoryImpl` に設定します。次に例を示します。

     ```
     set ANT_OPTS=-Djavax.xml.transform.TransformerFactory=org.apache.xalan.processor.TransformerFactoryImpl
     ```

Ant タスク用の oracle: 名前空間接頭辞の設定方法

Ant バージョン 1.6.5 以上では、WebServicesAssembler コマンドのすべての Ant タグおよびサブタグの先頭に `oracle:` 接頭辞が必要です。この接頭辞により、これが Oracle Ant タスクであることを Ant インタプリタに通知します。

`oracle:` 接頭辞は、Ant の `build.xml` ファイルの先頭の `project` タグにある接頭辞に対応しています。

```
<project name="myproject" default="all" basedir="." xmlns:oracle="antlib:oracle">
```

`oracle` を接頭辞として使用しない場合は、XML の有効な任意の `QName` 接頭辞に変更できます。たとえば、`oracletags` を接頭辞として使用する場合は、`project` タグの値を変更する必要があります。

```
<project name="myproject" default="all" basedir="." xmlns:oracletags="antlib:oracle">
```

この変更を行う場合、WebServicesAssembler コマンドのすべての Ant タグおよびサブタグは `oracletags:` で始まる必要があります。次に例を示します。

```
<oracletags:assemble ...>
```

データベース要件

次のいずれかのタスクを実行する場合、Oracle データベース（ローカルまたはリモート）をインストールして稼働しておく必要があります。

- データベース Web サービスのアセンブル。OracleAS Web Services で実行可能なデータベース Web サービスの詳細は、第 10 章「データベース Web サービスのアセンブル」を参照してください。
- クライアントとサーバー間における信頼性のあるメッセージ機能の有効化。信頼性のあるメッセージ機能の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの信頼性の確保」を参照してください。

5

OracleAS Web Services メッセージ

Oracle Application Server Web Services は、SOAP 1.1 および 1.2 メッセージをサポートしています。メッセージ書式には、style="document" または style="rpc"、および use="literal" または use="encoded" があります。

この章の項目は、次のとおりです。

OracleAS Web Services のメッセージ書式

この項では、OracleAS Web Services の現行リリースでサポートされているメッセージ書式について説明します。内容は、次のとおりです。

OracleAS Web Services でサポートされているメッセージ書式を理解するには、Web Service Description Language（WSDL）1.1 とワイヤ書式の関係を理解すると役に立ちます。ワイヤ書式とは、Simple Object Access Protocol（SOAP）メッセージまたはペイロードが伝送中に実際に取る表現のことです。メッセージ書式は、WSDL に定義されているバインディングの use および style 属性によって決まります。メッセージ部分を定義する XML スキーマのタイプにより、メッセージ書式が拡張されます。この場合、WSDL は規約だとみなすことができます。WSDL の様々な属性を定義することにより、メッセージのワイヤ書式を変更します。

通常、相互運用性に関して発生する問題は、実行時にワイヤ書式で通知されます。多くの場合、これらの問題は、WSDL を調整し、Web サービスのアーチファクトを再生成することによって解決できます。

メッセージ書式とワイヤ書式の関係は、1 対 1 ではありません。たとえば、ランタイム SOAP メッセージを、rpc-literal スタイルの Web サービスであるサービス B によって生成されるメッセージと同じ種類に見えるようにする XML スキーマを使用して、document-literal スタイルの Web サービスとしてサービス A を定義できます。この場合、サービス A の style および use（すなわち、メッセージ書式）を rpc および literal に変更しても、サービス A はサービス B と同じにはなりません。変更しても、ランタイム SOAP メッセージは、変更内容とはまったく異なるものになってしまいます。サービス A で使用されているスキーマも変更する必要があるのです。

メッセージ書式の概要

次の各項では、OracleAS Web Services でサポートされているメッセージ書式について簡単に説明します。

関連項目：

WSDL のバインディング属性 `use` および `style` の使用の詳細は、xxii ページの「SOAP 関連ドキュメント」および xxii ページの「WSDL 関連ドキュメント」にリストされている SOAP および WSDL の仕様を参照してください。

style="rpc" および style="document"

SOAP ペイロードには、rpc スタイルと document スタイルがあります。通常、rpc スタイルのペイロードは、リモート・プロシージャ・コールまたはメソッド・コールを起動する必要があるときに使用します。rpc スタイルの場合、リクエストの SOAP ボディにある最上位の XML 要素の名前は、常に WSDL 操作の名前になります。これらの名前は、任意のバインディングにおいて一意であるため、曖昧さはありません。通常、SOAP の XML メッセージはメソッド名とパラメータで構成されており、これらはすべて XML で表現されます。

WSDL 操作をオーバーロードする場合、対応する操作バインディングには一意の `SOAPAction` を指定する必要があります。rpc スタイルの SOAP ボディ要素（`<body>`）の構造については、SOAP 1.1 仕様の Section 7 を参照してください。

document スタイルのペイロードの SOAP ボディには、SOAP 1.1 仕様の Section 7 に準拠する必要がない XML が含まれますが、この XML は、メッセージのペイロードを定義するために XML スキーマのグローバル要素を使用します。このスキーマは、WSDL の `type` セクション内に定義するか、そのセクションにインポートします。

use="literal" および use="encoded"

SOAP クライアントおよびサーバーは、WSDL の `binding` の項にある、`use` 属性で指定されたルールに従って、SOAP ペイロードの `<body>` 要素の XML コンテンツを解釈します。このクライアントおよびサーバーは、互いに正しくデータを解釈できるように同じエンコード・ルールに従う必要があります。

use="literal" の場合、入出力メッセージの SOAP ボディをエンコードおよび解釈するルールは、スキーマに完全に基づいて記述されたものになります。

use="encoded" の場合、SOAP ボディの `encodingStyle` 属性は、SOAP 仕様に基づいてメッセージをエンコードおよび解釈するルールを判別します。

- SOAP 1.1 については、SOAP 1.1 仕様内の Section 5「SOAP Encoding」（`http://www.w3.org/TR/2000/NOTE-SOAP-20000508/#_Toc478383512`）を参照してください。
- SOAP 1.2 については、SOAP 1.2 仕様内の Section 3「SOAP Encoding」（`http://www.w3.org/TR/2003/REC-soap12-part2-20030624/#soapenc`）を参照してください。

また、SOAP エンコーディング・スキーマの拡張機能に定義されている型を使用することもできます。

サポートされているメッセージ書式

次の各項では、OracleAS Web Services でサポートされているメッセージ書式について説明します。

- document-literal メッセージ書式
- rpc-encoded メッセージ書式
- rpc-literal メッセージ書式

注意： 単一の Web アプリケーションでの、rpc-encoded と document-literal のような複数のメッセージ書式の使用はサポートされていません。

---

document-literal メッセージ書式

document-literal は、OracleAS Web Services のデフォルトのメッセージ書式です。document-literal 操作には、wrapped と bare という 2 つの一般的なスタイルがあります。

- wrapped スタイルの場合、ラッパー要素のスキーマ定義が、メソッドのパラメータをラップします。メッセージは、SOAP によってエンコードされず、SOAP RPC 表記規則を使用しません。
- bare スタイルの場合、メソッドでは、SOAP ボディにマッピングするパラメータは 1 つのみにする必要があります。メソッドに複数のパラメータがある場合でも、ボディ部分にマッピングできるパラメータは 1 つのみです。その他のパラメータは SOAP ヘッダーにマッピングする必要があります。

document-literal は、WS-I Basic Profile 1.0 および 1.1 に準拠しています。

各 document-literal 操作は、入力メッセージの最上位要素の `QName` を使用して一意に識別されます。.NET Web サービスとの相互運用性が最もよいのは wrapped スタイルの document-literal であるため、OracleAS Web Services のデフォルトのメッセージ書式としてはこの方式を使用することをお薦めします。

関連資料：

相互運用性とメッセージ書式の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「相互運用可能な Web サービスの実現」を参照してください。

利点：

- WS-I に準拠しています。
- 相互運用性をサポートしています。
- SOAP フォルトの処理に適した書式です。
- メッセージ部分の書式は標準の XML スキーマで記述されます。

短所：

- 古いスタックに対する下位互換性が完全ではありません。
- オブジェクト・グラフはサポートしていません。

document-literal メッセージ書式によるサンプル・リクエスト・メッセージ 例 5-1 は、document-literal メッセージ書式のリクエスト・メッセージを示しています。SOAP ボディ（env:Body）の下の XML 要素部分（payloadDocument）は、WSDL のスキーマに定義されているグローバル要素のドキュメント・インスタンスである必要があることに注意してください。

この例は、SOAP 1.1 メッセージに準拠しています。SOAP 1.2 メッセージに準拠するようにこの例を変更するには、xmlns:env の値を http://www.w3.org/2003/05/soap-envelope に変更します。

例 5-1 document-literal メッセージ書式によるリクエスト・メッセージ

```
<env:Envelope
        xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:ns0="http://ws.oracle.com/doc-lit">
        <env:Body>
                <ns0:payloadDocument>
                        <ns0:name>Scott</ns0:name>
                        <ns0:data>Hello</ns0:data>
                </ns0:payloadDocument>
        </env:Body>
</env:Envelope>
```

rpc-encoded メッセージ書式

rpc-encoded メッセージ書式では、SOAP 仕様に定義されているエンコード・ルールを使用します。

- SOAP 1.1 については、SOAP 1.1 仕様内の Section 5「SOAP Encoding」を参照してください。

  http://www.w3.org/TR/2000/NOTE-SOAP-20000508/#_Toc478383512

  SOAP 1.1 rpc-encoded メッセージ書式の encodingStyle 属性は、次の値で表されます。

  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"

- SOAP 1.2 については、SOAP 1.2 仕様の Section 3「SOAP Encoding」を参照してください。

  http://www.w3.org/TR/2003/REC-soap12-part2-20030624/#soapenc

  SOAP 1.2 rpc-encoded メッセージ書式の encodingStyle 属性は、次の値で表されます。

  encodingStyle="http://www.w3.org/2003/05/soap-encoding"

  関連資料：

  相互運用性とメッセージ書式の問題に関する詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「相互運用可能な Web サービスの実現」を参照してください。

利点：

- SOAP 1.1 エンコーディングは、Web サービスの最も一般的なメッセージ書式であるため、古いスタックに対する下位互換性があります。
- オブジェクト・グラフをサポートしています（id および href 属性を使用）。
- 名前空間 http://www.oracle.com/webservices/internal によって示されるその他の Java 型のマッピングのサポートを提供します。サポートされているデータ型については、5-7 ページの表 5-1 を参照してください。

短所：

- WS-I Basic Profile に準拠していません。
- メッセージ・ペイロード全体のスキーマ検証を実行しにくくなります。

rpc-encoded メッセージ書式のサンプル・メッセージ 例 5-2 は、rpc-encoded メッセージ書式を使用するリクエスト・メッセージを示します。SOAP ボディ（env:Body）の下にある送信対象の XML 要素部分（echoString）のタグ名は、対応する WSDL 操作の名前と同じである必要があることに注意してください。env:encodingStyle 属性は、使用されている SOAP エンコード・スタイルを示します。操作要素の下の各 XML 要素部分（stringParam）は、パラメータに対応します。これは、simpleType のインスタンスまたはグローバル型定義である必要があります。グローバル型定義である場合、WSDL のスキーマ内にあるか、SOAP エンコード拡張型の 1 つである必要があります。

OracleAS Web Services に対する rpc-encoded リクエスト・メッセージ（または OracleAS Web Services によって生成されたスタブに対する rpc-encoded レスポンス・メッセージ）は、xsi:type 属性がなくても使用できます。

例 5-2 および例 5-3 は、SOAP 1.1 のメッセージに準拠しています。これらの例を SOAP 1.2 のメッセージに準拠するようにするには、次のようにします。

- xmlns:env の値を http://www.w3.org/2003/05/soap-envelope に変更します。
- env:encodingStyle の値を http://www.w3.org/2003/05/soap-encoding に変更します。

例 5-2 rpc-encoded リクエスト・メッセージ

```
<env:Envelope
        xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:ns0="http://ws.oracle.com/rpc-enc">
        <env:Body>
                <ns0:echoString
                        env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
                        <stringParam>Hello</stringParam>
                </ns0:echoString>
        </env:Body>
</env:Envelope>
```

例 5-3 は、rpc-encoded メッセージ書式を使用するレスポンス・メッセージを示しています。OracleAS Web Services からの rpc-encoded レスポンス・メッセージ（または OracleAS Web Services によって生成されたスタブ内の rpc-encoded リクエスト・メッセージ）には、常に xsi:type 属性が含まれます。

例 5-3 rpc-encoded レスポンス・メッセージ

```
<env:Envelope
        xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:ns0="http://ws.oracle.com/rpc-enc">
        <env:Body>
                <ns0:echoStringResponse
                        env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
                        <stringParam xsi:type="xsd:string">Hello</stringParam>
                </ns0:echoStringResponse>
        </env:Body>
</env:Envelope>
```

rpc-encoded メッセージ書式の xsi:type 属性 SOAP 実装では多くの場合、rpc-encoded メッセージ書式を使用するメッセージでは通常、メッセージ・ペイロードの要素ごとに xsi:type 属性が使用されます。この属性は、オブジェクトのシリアライズおよびデシリアライズに役立ちます。多くの場合、xsi:type 属性はオプションです。xsi:type 属性が必須なのは、要素がスキーマに定義されている要素タイプの派生タイプのインスタンスである場合のみです。インバウンド SOAP メッセージの場合、OracleAS Web Services は、xsi:type 属性の有無に関係なくメッセージを受け入れます。rpc-encoded 書式のアウトバウンド SOAP メッセージの場合、OracleAS Web Services は常に xsi:type 属性を発行します。

例 5-4 および例 5-5 は、SOAP 1.1 のメッセージに準拠しています。これらの例を SOAP 1.2 のメッセージに準拠するようにするには、次のようにします。

- xmlns:env の値を http://www.w3.org/2003/05/soap-envelope に変更します。
- env:encodingStyle の値を http://www.w3.org/2003/05/soap-encoding に変更します。

例 5-4 は、rpc-encoded メッセージ書式を使用した、echo 操作のリクエスト・メッセージを示しています。なお、このコード・サンプルには xsi:type="xsd:string" 属性は含まれていません。

例 5-4 xsi:type 属性がない rpc-encoded リクエスト・メッセージ

```
<env:Envelope
        xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:ns0="http://ws.oracle.com/rpc-enc">
   <env:Body>
      <ns0:echo env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <param>some string</param>
      </ns0:echo>
   </env:Body>
</env:Envelope>
```

例 5-5 は、同じ rpc-encoded 書式のリクエスト・メッセージで、xsi:type 属性があるものを示しています。

例 5-5 xsi:type 属性がある rpc-encoded リクエスト・メッセージ

```
<env:Envelope
        xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:ns0="http://ws.oracle.com/rpc-enc">
   <env:Body>
      <ns0:echo env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <param xsi:type="xsd:string">some string</param>
      </ns0:echo>
   </env:Body>
</env:Envelope>
```

Oracle 固有の型のサポート サポートされている型の完全なリストについては、JAX-RPC 1.1 仕様の 4 ～ 5 章を参照してください。JAX-RPC 1.1 仕様については、次の URL を参照してください。

`http://java.sun.com/webservices/jaxrpc/index.jsp`

OracleAS Web Services の rpc-encoded メッセージ書式でサポートされている Java 型および Oracle 固有型の完全なリストについては、表 5-1 を参照してください。Oracle 固有型に対する Java 型のマッピングのサポートは、次に示す OracleAS Web Services 固有の名前空間で指定されます。

`http://www.oracle.com/webservices/internal`

この名前空間は、JAX-RPC 1.1 によってサポートされていない `Collection` などの標準 Java 型のための、非標準の XML スキーマ定義に対応しています。

注意： `java.util.Collection` ファミリおよび `java.util.Map` ファミリも、use="literal"（すなわち、document-literal および rpc-literal）でサポートされています。これらの Java 型のスキーマ型は別の名前空間で定義されています。これらの Java 型のサポート方法の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 型の XML および WSDL 型へのマッピング」を参照してください。

表 5-1 rpc-encoded 固有のメッセージ書式でサポートされている Java 型

Java 型	Java クラス	マッピング詳細
Java プリミティブ型	boolean、byte、double、float、int、long、short	『Oracle Application Server Web Services アドバンスト開発者ガイド』の OracleAS Web Services による Java プリミティブ型の XML 型へのマッピングに関する項を参照してください。
java.lang オブジェクト型	Boolean、Byte、Double、Float、Integer、Long、Short、String	『Oracle Application Server Web Services アドバンスト開発者ガイド』の OracleAS Web Services による Java 型の XML 型へのマッピングに関する項を参照してください。

表 5-1 rpc-encoded 固有のメッセージ書式でサポートされている Java 型（続き）

Java 型	Java クラス	マッピング詳細
Java 基本型	java.math.BigDecimal java.math.BigInteger java.xml.QName java.util.Calendar java.util.Date java.net.URI	『Oracle Application Server Web Services アドバンスト開発者ガイド』の OracleAS Web Services による Java 型の XML 型へのマッピングに関する項を参照してください。
Java 配列型	サポートされている型の要素からなる 1 次元配列	『Oracle Application Server Web Services アドバンスト開発者ガイド』の OracleAS Web Services による Java 型の XML 型へのマッピングに関する項および Java コレクション・クラスの XML 型へのマッピングに関する項を参照してください。
Java の値のタイプ	サポートされている型のプロパティがある JavaBeans	『Oracle Application Server Web Services アドバンスト開発者ガイド』の Java 値タイプについての OracleAS Web Services のサポートに関する項を参照してください。
添付ファイル **注意：**添付ファイルは相互運用不可能です。	java.awt.Image javax.mail.internet.MimeMultipart javax.xml.transform.Source javax.activation.DataHandler	『Oracle Application Server Web Services アドバンスト開発者ガイド』の「メッセージ添付ファイルの処理」を参照してください。
java.util.Collection	java.util.Collection java.util.List java.util.Set java.util.Vector java.util.Stack java.util.LinkedList java.util.ArrayList java.util.HashSet java.util.TreeSet	『Oracle Application Server Web Services アドバンスト開発者ガイド』の OracleAS Web Services による Java コレクション・クラスの XML 型へのマッピングに関する項および Oracle 固有の XML 型の追加情報に関する項を参照してください。
java.util.Map	java.util.Map java.util.HashMap java.util.TreeMap java.util.Hashtable java.util.Properties	『Oracle Application Server Web Services アドバンスト開発者ガイド』の OracleAS Web Services による Java コレクション・クラスの XML 型へのマッピングに関する項および Oracle 固有の XML 型の追加情報に関する項を参照してください。

`Map` または `Collection` の項目として組み込まれていない値タイプ（ユーザー定義の `MyBean` 型など）を使用する場合、`valueTypeClassName` 引数の値を使用して、WSDL の生成時にこれらの型を WebServicesAssembler ツールに対して宣言する必要があります。

```
java -jar wsa.jar -genWsdl
                  -valueTypeClassName hello.MyBean
                  -valueTypeClassName hello.MyFoo...
```

このコマンドの説明 :

- genWsdl: Java インタフェースをベースにして WSDL を生成します。18-28 ページの「genWsdl」を参照してください。
- valueTypeClassName: java.util.Collection および java.util.Map に使用される JAX-RPC 値タイプの完全修飾クラス名を指定します。18-69 ページの「valueTypeClassName」を参照してください。

これにより、生成された WSDL がこれらの値タイプのスキーマ定義を含めることが可能になります。これによって、ランタイムが、対応するシリアライズ値を正しく生成できるようになります。Web サービスを（Java クラス、EJB、データベース・リソースなどから）ボトムアップ方式でアセンブルするすべての WebServicesAssembler コマンドおよび Ant タスクでは、valueTypeClassName 引数がサポートされています。

関連資料 :

JAX-RPC 値タイプの要件に準拠していない値タイプ・クラスや、デフォルトの JAX-RPC シリアライズ・メカニズムでは処理できない値タイプ・クラスの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 値タイプのカスタム・シリアライズ」を参照してください。

rpc-encoded 書式に対する制限 OracleAS Web Services は、rpc-encoded メッセージ書式と databinding=false の組合せはサポートしていません。

rpc-literal メッセージ書式

rpc-literal メッセージ書式は、WS-I Basic Profile 1.0 および 1.1 に準拠しています。この書式は、rpc スタイルのメッセージ・ペイロード構造を使用しますが、プロシージャによって渡される型を literal 方式で記述する方法もサポートしています。この場合、literal とは、すべてのパラメータ型に対するスキーマはあるが、メッセージ・ボディ自体のペイロードに対するスキーマがないことを意味しています。

関連項目 :

相互運用性とメッセージ書式の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「相互運用可能な Web サービスの実現」を参照してください。

利点 :

- WS-I に準拠しています。
- メッセージ部分の書式は標準の XML スキーマで記述されます。

短所 :

- rpc-literal に対するサポートは、すべてのベンダーにわたって一貫しているわけではありません。
- オブジェクト・グラフはサポートしていません。
- 古いスタックに対する下位互換性がありません。
- Java メソッドのパラメータを WSDL メッセージ部分にマッピングする際に、Java メソッドのパラメータが NULL 値であることを表すことができません。

rpc-literal メッセージ書式によるサンプル・リクエスト・メッセージ 例 5-6 は、rpc-literal メッセージ書式でコード化されたリクエスト・メッセージを示しています。SOAP ボディ（env:Body）の下にある XML 要素部分（echoString）のタグ名は、対応する WSDL 操作の名前と同じである必要があることに注意してください。操作要素の下の各 XML 要素部分（stringParam）は、パラメータに対応しており、simpleType のインスタンスまたはグローバル型定義である必要があります。グローバル型定義である場合は、WSDL のスキーマ内にある必要があります。

この例は、SOAP 1.1 メッセージに準拠しています。SOAP 1.2 メッセージに準拠させる場合は、xmlns:env の値を http://www.w3.org/2003/05/soap-envelope に変更します。

例 5-6 rpc-literal リクエスト・メッセージ

```
<env:Envelope
        xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:ns0="http://ws.oracle.com/rpc-lit">
        <env:Body>
                <ns0:echoString>
                        <stringParam>Hello</stringParam>
                </ns0:echoString>
        </env:Body>
</env:Envelope>
```

Web サービスに対するメッセージ書式の選択方法

WebServicesAssembler ツールには、Web サービスで使用されるメッセージ書式を制御するための各引数が用意されています。これらの引数を使用すると、メッセージ書式として rpc または document（"wrapped" または "bare"）、encoded または literal を指定できます。

- style
- use

次の WebServicesAssembler コマンドを使用すると、use および style 引数を使用して Web サービスのメッセージ書式を指定できます。これらのコマンドの詳細は、第 18 章「WebServicesAssembler の使用方法」を参照してください。

- aqAssemble
- assemble
- corbaAssemble
- dbJavaAssemble
- ejbAssemble
- genWsdl
- plsqlAssemble
- sqlAssemble

次の例では、assemble コマンドを使用して、Web サービスのサーバー・コンポーネントをアセンブルしています。style および use 引数を使用して、メッセージ書式として rpc-literal を使用するよう指定しています。

```
java -jar wsa.jar -assemble
                  -appName $(app.name)
                  -serviceName HelloServiceWSIF
                  -uri $(app.name)
                  -interfaceName oracle.demo.hello.HelloInterface
                  -className oracle.demo.helllo.HelloImpl
                  -input $(service.classes.dir)
                  -output build
                  -ear dist/$(app.name).ear
                  -style rpc
                  -use literal
```

このコマンドの説明 :

- `assemble`: ボトムアップ方式で Java ファイルから Web サービスを生成します。このコマンドにより、デプロイ可能なアーカイブを作成するために必要なすべてのファイルが作成されます。18-8 ページの「assemble」を参照してください。
- `appName`: アプリケーションの名前を指定します。この名前は、通常 `context` や `uri` などの他の引数のベース値として使用されます。18-40 ページの「appName」を参照してください。
- `serviceName`: サービス名を指定します。18-48 ページの「serviceName」を参照してください。
- `uri`: Web サービスに使用する URI を指定します。18-63 ページの「uri」を参照してください。
- `interfaceName`: サービス・エンドポイント・インタフェース（SEI）を格納する Java クラスの名前（パッケージ名を含む）を指定します。18-45 ページの「interfaceName」を参照してください。
- `classname`: Web サービスの実装クラスのクラス名（パッケージ名を含む）を指定します。18-41 ページの「className」を参照してください。
- `input`: `WEB-INF/classes` にコピーされるクラスを格納する JAR またはディレクトリを指定します。この引数は、WebServicesAssembler によって使用されるクラスパスに追加されます。18-45 ページの「input」を参照してください。
- `output`: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
- `ear`: 生成される EAR ファイルの名前と位置を指定します。18-42 ページの「ear」を参照してください。
- `style`: ボトムアップ方式 Web サービス・アセンブリの場合、この引数は、生成される WSDL 内のメッセージ書式の `style` 属性を指定します。18-68 ページの「style」を参照してください。
- `use`: ボトムアップ方式 Web サービス・アセンブリの場合、この引数は、生成される WSDL 内のメッセージ書式の `use` 属性を指定します。18-68 ページの「use」を参照してください。

関連項目：

WebServicesAssembler ツールとその機能の詳細は、
第 18 章「WebServicesAssembler の使用方法」を参照してください。

サービス実装のメッセージ書式の変更

Java サービス・エンドポイント実装は、rpc-encoded、rpc-literal または document-literal 書式で公開できます。ただし、表 5-1 にリストされている `attachment` データ型をサービス・エンドポイント実装で使用する場合は、使用できるのは rpc-encoded のみです。`attachment` 型のいずれも使用しない場合は、rpc-literal または document-literal 書式を使用して相互運用性を向上させることができます。これらのいずれの書式を使用しても、Java オブジェクトと XML 間のマーシャリングでは同じパフォーマンスが実現されます。

メッセージ書式に関する推奨事項

この項では、Web サービスの設計時にメッセージ書式を選択する際の一般的なガイドラインのいくつかについて説明します。メッセージ書式の選択時には、サポートするクライアントの機能要件を考慮してください。想定されるクライアントの機能要件や推奨されるメッセージ書式については、表 5-2 を参照してください。

表 5-2 クライアントの機能に応じた推奨メッセージ書式

クライアントの機能	推奨メッセージ書式
Axis 1.2 クライアント	rpc-encoded または document-literal メッセージ書式を使用します。
BPEL プロセス	literal を使用した document スタイルを使用します。
Oracle リリース 10.1 データベースからのコールアウト	JAX-RPC はすべての書式をサポートします。NULL 値を表す必要がある場合は、rpc-literal データベース・コールアウトを避けてもかまいません。
Oracle リリース 9.2 データベースからのコールアウト	コールアウトは soap.jar に基づいています。rpc-encoded を使用します。
グラフの公開とオブジェクト識別性の保持	rpc-encoded を使用します。rpc-encoded が最も簡単な方式ですが、モデル（スキーマ）が適切に設計されている場合、document-literal でも実現できます。
OmniPortlet	rpc-encoded または document-literal を使用します。Omniportlet API（ウィザード）は rpc-literal をサポートしていません。
Oracle XML Query Service（XQS）統合	literal を使用した rpc または document スタイルを使用します。Oracle XML Query Service は SOAP エンコードをサポートしていません。
WS-I への準拠	literal を使用した rpc または document スタイルを使用します。
XML スキーマ統合	WSDL 操作の SOAP メッセージ部分を記述するために、XML スキーマ定義を再利用することです。この統合は、literal を使用した rpc または document スタイルを使用して実現できます。

SOAP メッセージの操作

OracleAS Web Services は、プログラム的な方法と WebServicesAssembler ツールを使用して Web サービスをボトムアップおよびトップダウン方式でアセンブルする方法の両方で SOAP 1.1 および 1.2 メッセージをサポートしています。

- OraSAAJ API の概要
- OraSAAJ API を使用して SOAP メッセージを処理する方法
- バイナリ・データを含む添付ファイル付きのメッセージの処理方法
- SOAP 要素への XML 要素の変換方法
- Web サービスをボトムアップ方式でアセンブルするための SOAP 1.2 書式メッセージの使用方法
- Web サービスをトップダウン方式でアセンブルするための SOAP 1.2 書式メッセージの使用方法

SOAP with Attachments API for Java（SAAJ）バージョン 1.2 は、SOAP メッセージをプログラム内から表現および操作するモデルとなるものです。標準の SAAJ の API は、SOAP 1.1 をサポートしています。SAAJ の詳細は、次の Web アドレスを参照してください。

```
http://java.sun.com/webservices/saaj/index.jsp
```

Oracle の SAAJ 1.2 API 拡張機能（OraSAAJ）を使用すると、Web サービスが SOAP 1.2 メッセージを操作できるようになります。

OraSAAJ API の概要

OracleAS Web Services は、Java オブジェクトとして添付ファイル付き SOAP メッセージを作成するための仕様である SAAJ 1.2 をサポートしています。ただし、SAAJ 1.2 の標準 API がサポートしているのは SOAP 1.1 メッセージのみです。このため、SOAP 1.2 メッセージに対するプログラムにおけるサポートを提供できるように、OracleAS Web Services には `oracle.webservices.soap` パッケージが含まれています。このパッケージに含まれるインタフェースを使用すると、SOAP 1.2 メッセージ・オブジェクトを操作することや、SOAP 1.2 メッセージ・オブジェクトに情報を追加することが可能になります。

このパッケージのクラス `VersionedMessageFactory` および `VersionedSOAPFactory` は、標準 SAAJ のクラス `MessageFactory` および `SOAPFactory` の拡張機能です。`VersionedMessageFactory` および `VersionedSOAPFactory` クラスのメソッドには、標準の SAAJ の API の使用時に SOAP メッセージのバージョンを指定するための追加パラメータが含まれています。

表 5-3 OraSAAJ API のインタフェースおよびクラス

インタフェース / クラス名	説明
Body12 インタフェース	SOAP 1.2 メッセージの `Body` オブジェクトを表します。
Fault12 インタフェース	SOAP 1.2 の `FaultCode` および `FaultReason` 要素を SOAP 1.2 の `Fault` 要素に追加するためのメソッドを提供します。
FaultCode12 インタフェース	SOAP 1.2 の `FaultValue` および `FaultSubcode` 要素を SOAP 1.2 の `FaultCode` 要素に追加するためのメソッドを提供します。
FaultReason12 インタフェース	SOAP 1.2 の `FaultText` 要素を SOAP 1.2 の `FaultReason` 要素に追加するためのメソッドを提供します。
FaultSubcode12 インタフェース	このインタフェースは、`FaultCode12` インタフェースの拡張機能です。これは、コンパイラによって強制される強い型指定を行うためのマーカー・インタフェースです。
FaultText12 インタフェース	テキスト・ノードのコンテンツおよびロケール情報を `FaultText` 要素に追加するためのメソッドを提供します。
FaultValue12 インタフェース	`FaultValue12` 要素にフォルト・コードを設定するためのメソッドを提供します。
OracleSOAPElement インタフェース	`java.xml.soap.SOAPElement` インタフェースの拡張です。添付ファイル付きメッセージ用のデータ・ハンドラを使用するためのメソッドを提供します。
SOAPVersion インタフェース	プラットフォームで使用可能な SOAP のバージョンを表す各定数を表示します。
VersionedMessageFactory クラス	SOAP メッセージを作成するためのメソッドを提供します。これらのメソッドは、標準の SAAJ のクラス `MessageFactory` の模倣ですが、これらのメソッドには SOAP メッセージのバージョンを指定するための追加パラメータが含まれる点が異なります。あるメソッドは、標準の MIME ヘッダーを使用して空のメッセージを作成します。別のメソッドは、入力ストリームおよび指定された MIME ヘッダーを基にメッセージを作成します。
VersionedSOAPFactory クラス	SOAP 要素を作成するためのメソッドを提供します。これらのメソッドは、標準の SAAJ のクラス `SOAPFactory` の模倣ですが、これらのメソッドには SOAP メッセージのバージョンを指定するための追加パラメータが含まれる点が異なります。

関連資料 :

oracle.webservices.soap パッケージのクラスとメソッドの詳細は、『Oracle Application Server Web Services Java API Reference』を参照してください。

OraSAAJ API を使用して SOAP メッセージを処理する方法

OraSAAJ の拡張機能は、javax.xml.rpc.handler.Handler から使用できます。SOAP 1.2 SAAJ メッセージを操作するには、多くの場合、標準の javax.xml.soap.* クラスが使用されます。ただし、SOAP 1.2 に用意されている機能を使用する場合は、OraSAAJ API を使用する必要があります。

例 5-7 は、標準の javax.xml.soap.* クラスと OraSAAJ クラスを一緒に使用する方法を示しています。このコード例では、SOAP 1.2 メッセージを新規に作成しています。VersionedMessageFactory メソッドは、javax.xml.soap.SoapMessage オブジェクトを戻します。これにより、メッセージに対して getBody や getEnvelope などの標準の javax.xml.soap.* メソッドを使用できるようになります。

addFault メソッドは、SOAP 1.2 フォルトをメッセージに追加します。SOAP 1.2 フォルトを送信するには、OraSAAJ Fault12、FaultCode12、FaultValue12 および FaultReason12 API を使用する必要があります。これは、SOAP 1.2 フォルトには SOAP 1.1 フォルトより多くの情報が含まれているためです。

例 5-7 SAAJ および OraSAAJ API の操作

```
public boolean handleResponse(MessageContext context) {
    ...
    // create a SOAP 1.2 message from scratch.
    // Note the use of VersionedMessageFactory to get a SOAPMessage
    // for a specific version of soap
    SOAPMessage message =
((VersionedMessageFactory)MessageFactory.newInstance()).createVersionedMessage(oracle.w
ebservices.soap.SOAPVersion.SOAP_1_2);
    // Now standard APIs can be used.
    SOAPBody body = message.getSOAPPart().getEnvelope().getBody();
    // However, if you need to send a fault, you must
    // use Oracle specific APIs, because SOAP 1.2
    // faults contain more information than SOAP 1.1 faults.
    // Note the use of Fault12, FaultCode12, and FaultReason12
    SOAPFault fault = body.addFault();
      Fault12 soapFault = (Fault12) fault;
      FaultCode12 faultCode = soapFault.addCode();
      FaultValue12 faultValue = faultCode.addFaultValue();
      QNameAdapter faultCodeQName = new QNameAdapter("http://my.foo.com/",
                                                     "myFaultCode",
                                                     "foo");
      faultValue.setFaultCode(faultCodeQName);
      FaultReason12 faultReason = soapFault.addReason();
      faultReason.addFaultText().setValue("An unknown error occurred");
    ...
}
```

バイナリ・データを含む添付ファイル付きのメッセージの処理方法

oracle.webservices パッケージで提供される OracleSOAPElement インタフェースに含まれるメソッドを使用すると、添付ファイル付きのメッセージを処理できます。

関連項目：

OracleSOAPElement インタフェースおよびそのメソッドを使用してバイナリ・データの添付ファイルを取得および設定する方法の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』のバイナリ・データを含む添付ファイルの処理に関する項を参照してください。

SOAP 要素への XML 要素の変換方法

oracle.webservices パッケージには、XML 要素（org.w3c.dom.Element）を SOAP 要素（javax.xml.soap.SOAPElement）に変換するために SOAPUtil クラスの toSOAPElement メソッドが用意されています。SOAPElement が dom.Element を実装しているのであり、その逆ではないことに注意してください。

例 5-8 は、XML 文書を作成し、SOAP 要素に変換し、標準出力に出力するコード・サンプルを示しています。

例 5-8 SOAP 要素への XML 要素の変換

```
...
try {
   DOMParser parser = new DOMParser();
   parser.parse(new StringReader(
      "<Flds:CustomerGroup xmlns:Flds=\"http://foo.com/foo.xsd\">
      "<Flds:Customer>xyz</Flds:Customer>
      "</Flds:CustomerGroup>"));

   SOAPElement se = SOAPUtil.toSOAPElement(
      parser.getDocument().getDocumentElement());
   ((XMLElement)se).print(System.out);
} catch (Exception ex) {
   ex.printStackTrace();
}
...
```

Web サービスをボトムアップ方式でアセンブルするための SOAP 1.2 書式メッセージの使用方法

ボトムアップ方式での Web サービスの生成において SOAP バージョン 1.2 のメッセージをサポートするため、WebServicesAssembler には、soapVersion 引数が用意されています。使用可能な値は、1.1、1.2 または 1.1,1.2 です。デフォルト値は 1.1 です。

値 1.1,1.2 は、WebServicesAssembler が 2 つのバインディングとともに 2 つのポートを作成することを意味します。1 つのポートとバインディングがバージョン 1.1 をサポートし、もう 1 つのポートとバインディングがバージョン 1.2 をサポートします。各ポートは異なる URL にバインドする必要があります。つまり、同じ URL アドレスを使用して両方のバージョンを同時にサポートすることはできません。

関連項目：

WebServicesAssembler の soapVersion 引数の詳細は、18-65 ページの「soapVersion」を参照してください。

Web サービスをトップダウン方式でアセンブルするための SOAP 1.2 書式メッセージの使用方法

トップダウン方式の Web サービス開発で SOAP 1.2 メッセージをサポートするには、SOAP 1.2 バインディングがある WSDL を提供する必要があります。SOAP 1.2 バインディングがある WSDL には、SOAP 1.2 に固有の URI セットが含まれます。これらの URI のリストについては、表 5-4 を参照してください。

表 5-4 SOAP 1.2 メッセージの URI

URI	説明
`http://schemas.xmlsoap.org/wsdl/soap12`	WSDL の `binding` 要素になる SOAP 1.2 バインディング要素の名前空間。
`http://www.w3.org/2003/05/soap-encoding`	SOAP 1.2 メッセージ用に、含む側の要素のコンテンツのエンコード・ルールを示す。 詳細は、`http://www.w3.org/TR/2003/REC-soap12-part2-20030624/` に記載されている SOAP Version 1.2 Part 2 Recommendation を参照してください。

`http://www.w3.org/2003/05/soap/bindings/HTTP`
および
`http://schemas.xmlsoap.org/soap/http`
	SOAP 1.2 用の HTTP トランスポートが説明されている。これらの URI は両方とも OracleAS Web Services スタックによって受け入れられますが、`schemas.xmlsoap.org` URI の方が `www.w3.org` URI よりも相互運用性があります。このため、Web サービスのボトムアップ方式でのアセンブリ用として WSDL が生成される場合は `schemas.xmlsoap.org` URI が使用されます。

例 5-9 は、SOAP 1.2 メッセージをサポートする WSDL を示しています。SOAP 1.2 をサポートするために必要な URI および要素は、**太字**フォントで示されています。

例 5-9 SOAP 1.2 バインディングがある WSDL のサンプル

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
     name="Rpclitbottomup"
     targetNamespace="http://www.oracle.ws/rpcliteral"
     xmlns="http://schemas.xmlsoap.org/wsdl/"
     xmlns:tns="http://www.oracle.ws/rpcliteral"
     xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
     xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
     xmlns:tns0="http://www.oracle.ws/rpcliteral/schema"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    >
    <types>
        <schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.oracle.ws/rpcliteral/schema"
             elementFormDefault="qualified"
xmlns:tns="http://www.oracle.ws/rpcliteral/schema"
             xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <complexType name="HelloMessage">
                <sequence>
                    <element name="longValue" type="long"/>
                    <element name="age" type="int"/>
                    <element name="greeting" type="string" nillable="true"/>
                    <element name="name" type="string" nillable="true"/>
                    <element name="id" type="decimal" nillable="true"/>
                </sequence>
            </complexType>
        </schema>
```

```
    </types>
    <message name="HelloInterface_hello">
        <part name="msg" type="tns0:HelloMessage"/>
    </message>
    <message name="HelloInterface_helloResponse">
        <part name="result" type="tns0:HelloMessage"/>
    </message>
    <portType name="HelloInterface">
        <operation name="hello" parameterOrder="msg">
            <input message="tns:HelloInterface_hello"/>
            <output message="tns:HelloInterface_helloResponse"/>
        </operation>
    </portType>
    <binding name="HelloInterfacePortBinding" type="tns:HelloInterface">
        <soap12:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="hello">
            <soap12:operation soapAction="http://www.oracle.ws/rpcliteral/hello"
soapActionRequired="false"/>
            <input>
                <soap12:body use="literal" namespace="http://www.oracle.ws/rpcliteral"
parts="msg"/>
            </input>
            <output>
                <soap12:body use="literal" namespace="http://www.oracle.ws/rpcliteral"
parts="result"/>
            </output>
        </operation>
    </binding>
    <service name="Rpclitbottomup">
        <port name="HelloInterfacePort" binding="tns:HelloInterfacePortBinding">
            <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
        </port>
    </service>
</definitions>
```

制限事項

F-2 ページの「OracleAS Web Services のメッセージ」を参照してください。

追加情報

詳細は、次を参照してください。

- WebServicesAssembler ツールを使用した Web サービスのアセンブル方法は、第 18 章「WebServicesAssembler の使用方法」を参照してください。
- Web サービスの相互運用性の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「相互運用可能な Web サービスの実現」を参照してください。
- メッセージの添付ファイル、および Web サービスでの添付ファイルの使用方法の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「メッセージ添付ファイルの処理」を参照してください。
- 標準以外のデータ型の処理の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 値タイプのカスタム・シリアライズ」を参照してください。
- JAX-RPC マッピング・ファイルの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「JAX-RPC マッピング・ファイル記述子」を参照してください。
- OracleAS Web Services でサポートされているデータ型の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 型の XML および WSDL 型へのマッピング」を参照してください。

6

WSDL からの Web サービスのアセンブル

この章では、Web Service Description Language（WSDL）ファイルから始めて Web サービスをアセンブルする方法について説明します。この方法は、トップダウン方式での Web サービスの生成とも呼ばれます。

トップダウン方式でのアセンブリの概要

トップダウン方式での Web サービスのアセンブリでは、公開しようとするビジネス・プロセスがモデリングされている既存の WSDL ファイルからサービスを生成します。

WebServiceAssembler などの開発ツールは、WSDL を使用してサービスのサービス・エンドポイント・インタフェースを生成します。この後で、Java クラスなどのサポートされているアーキテクチャ用にそのサービスを実装できます。実装をコンパイルした後、サービスを生成し、アプリケーション・サーバーにデプロイします。

Web サービスをアセンブルするには、WebServiceAssembler ツール、および J2SE 1.4 SDK ディストリビューションに含まれる `javac` コンパイラなどの Java プラットフォーム・ツールが必要です。

Web サービスをトップダウン方式で生成する場合の一般的な手順は次のとおりです。

1. サービス・エンドポイント・インタフェースを生成します。

   この手順は WebServicesAssembler が実行します。

2. サービスを実装およびコンパイルします。

   この手順は開発者が実行します。

3. サービスをアセンブルします。

   この手順は WebServicesAssembler が実行します。これらの各手順の詳細は、6-4 ページの「トップダウン方式で Web サービスをアセンブルする手順」を参照してください。

関連項目：

Web サービスに割当て可能なメッセージ書式の詳細は、第 5 章「OracleAS Web Services メッセージ」を参照してください。

注意： Web サービスをトップダウン方式でアセンブルする場合には、WebServicesAssembler は、`xsd:choice` または `xsd:group` の XML 型を含む WSDL を使用できません。これらの XML 型を含む WSDL を使用する場合は、WebServicesAssembler の `dataBinding` 引数を `false` に設定し、WSDL ファイルでのスキーマ定義にペイロードが準拠するように `SOAPElement` をコーディングする必要があります。

トップダウン方式での Web サービスのアセンブル方法

この項の内容は、次のとおりです。

前提条件

WebServiceAssembler を使用して Web サービスをトップダウン方式で生成する場合、指定する必要があるのは WSDL と出力ディレクトリのみです。

Web サービスの生成を開始する前に、次の各事項に注意してください。

- WebServicesAssembler には、指定する WSDL についての制限があります。
  - WSDL は、Web Services-Interoperability（WS-I）Basic Profile 1.0 に準拠している必要があります。準拠していない場合でも、WebServiceAssembler には、WSDL の様々な制限を回避するためのコマンドライン引数が用意されています。
  - 実装可能なサービス要素は 1 つのみです。WebServicesAssembler では、一度に 1 つのサービスに対してのみアーチファクトを生成できます。複数のサービスが WSDL で記述されている場合は、コマンドライン引数 `serviceName` を使用して、使用対象のサービスを指定できます。
  - メッセージ書式を WSDL に指定します。トップダウン方式での Web サービス開発では、WebServicesAssembler を使用してメッセージ書式を変更することはできません。
  - WSDL に複数のメッセージ書式を含めることはできません。使用しないメッセージ書式があるバインディングを参照するポートは、WSDL から削除してください。
  - サービス、ポート・タイプ、操作、バインディングまたはポートの名前など、WSDL の名前での National Language Support（「NLS」または「グローバリゼーション・サポート」とも呼ばれます）文字の使用は、サポートされていません。使用すると、コード生成のときに、または Web サービス・テスト・ページで、エラーが発生する場合もあります。
- ラップ・パラメータとアンラップ・パラメータのどちらを使用するかを決定します。このために、WebServicesAssembler には `unwrapParameters` コマンドライン・オプションが用意されています。
- 標準外のデータ型を使用する場合は、その処理方法を `oracle-webservices.xml` デプロイメント・ディスクリプタで必ず定義してください。このファイルを使用して、XML と Java 間でデータを変換するシリアライズ・クラスの名前や、データの Java 表現を記述する Java クラスの名前を識別できます。

  関連項目：

  - WebServicesAssembler のコマンドと引数の詳細は、第 18 章「WebServicesAssembler の使用方法」を参照してください。
  - 標準外のデータ型の使用の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 型の XML および WSDL 型へのマッピング」および 5-7 ページの表 5-1 を参照してください。
  - このファイルの詳細は、付録 C「oracle-webservices.xml デプロイメント・ディスクリプタ・スキーマ」を参照してください。
  - 標準以外のデータ型の処理の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 値タイプのカスタム・シリアライズ」を参照してください。

トップダウン方式で Web サービスをアセンブルする手順

次の手順は、Web サービスをトップダウン方式でアセンブルする方法を例で示しています。この Web サービスはロギング機能を実現するものです。このロギング機能は、`wsdl/LoggingFacility.wsdl` にある WSDL で定義されています。

1. Web サービスの生成ベースとなる WSDL を、WebServiceAssembler の `genInterface` コマンドへの入力として指定します。

   コマンドライン:

   ```
   java -jar wsa.jar -genInterface
                     -output build/src/service
                     -wsdl wsdl/LoggingFacility.wsdl
                     -unwrapParameters false
                     -packageName oracle.demo.topdowndoclit.service
                     -mappingFileName type-mapping.xml
   ```

   Ant タスク:

   ```
   <oracle:genInterface wsdl="wsdl/LoggingFacility.wsdl"
                output= "build/src/service"
                packageName= "oracle.demo.topdowndoclit.service"
                mappingFileName="type-mapping.xml"
                dataBinding="true"
                unwrapParameters="false"
   />
   ```

   このコマンドラインおよび Ant タスクの説明:

   - `genInterface`: 各ポート・タイプのサービス・エンドポイント・インタフェースと、WSDL に定義された複合型に対応する Java 値タイプ・クラス（Bean）を作成します。また、XML スキーマ型と Java 値タイプ・クラス間のマッピングを記述する JAX-RPC マッピング・ファイルも作成されます。18-30 ページの「genInterface」を参照してください。
   - `output`: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
   - `wsdl`: WSDL ドキュメントの絶対ファイル・パス、相対ファイル・パスまたは URL を指定します。18-64 ページの「wsdl」を参照してください。
   - `unwrapParameters`: `false` に設定すると、生成対象のサービス・エンドポイント・インタフェースが、入力パラメータおよび戻り型をラップするラッパーとともに生成されます。18-69 ページの「unwrapParameters」を参照してください。
   - `packageName`: JAX-RPC マッピング・ファイルにパッケージ名が宣言されていない場合に、生成されたクラス用に使用されるパッケージ名を指定します。18-47 ページの「packageName」を参照してください。
   - `mappingFileName`: JAX-RPC マッピング・ファイルを指すファイル位置を指定します。18-46 ページの「mappingFileName」を参照してください。
   - `dataBinding`: `true` の場合、WebServicesAssembler では、スキーマ内の全要素に対して、JAX-RPC デフォルト・タイプ・マッピング規則に従う Java 値タイプを生成しようとします。18-68 ページの「dataBinding」を参照してください。

   少なくとも、WSDL の名前を指定します。`genInterface` の必須およびオプションの引数の詳細は、18-30 ページの「genInterface」を参照してください。

   WSDL に指定されているポート・タイプごとに Java インタフェースが生成され、複合タイプごとに JavaBean が生成されます。生成されたインタフェースが格納されるディレクトリの名前は、`output` および `packageName` 引数の値に基づいて付けられます。この例の場合、生成されたインタフェースは `build/src/service/oracle/demo/topdowndoclit/service` に格納されます。値タイプは指定された出力ディレクトリに格納されますが、パッケージ名は値タイプの名前空間または JAX-RPC マッピング・ファイル `type-mapping.xml` 内の値に基づいて付けられます。

2. 生成されたインタフェースおよびタイプ・クラスをコンパイルします。次に例を示します。

   ```
   javac build/src/service/*.java -destdir build/classes
   ```

3. Web サービスのサービス・エンドポイント実装を作成します。

   Java サービス・エンドポイントには、生成された Java インタフェース内のすべてのメソッドと一致するメソッド・シグネチャが必要です。この例では、`DocLitLoggerImpl.java` ファイルにサービスを作成しているものとします。

4. Java サービス・エンドポイントをコンパイルします。

   たとえば、`Impl` クラスが保存されているディレクトリと同じディレクトリで Java サービス・エンドポイント・インタフェースのソースが生成された場合は、手順 2 と同じコマンドを使用できます。そうでない場合は、`classpath` 引数の値を変更する必要があります。

5. `topDownAssemble` コマンドを使用して WebServiceAssembler ツールを実行し、サービスをアセンブルします。この例において、`doclit_topdown` は、生成された Web サービスのアプリケーション名です。次に例を示します。

   コマンドライン :

   ```
   java -jar wsa.jar -topDownAssemble
              -appName doclit_topdown
              -wsdl ./wsdl/LoggingFacility.wsdl
              -unwrapParameters false
              -className oracle.demo.topdowndoclit.service.DocLitLoggerImpl
              -input build/classes/service
              -output build
              -ear dist/doclit_topdown.ear
              -mappingFileName type-mapping.xml
              -packageName oracle.demo.topdowndoclit.service
              -fetchWsdlImports true
              -classpath ./build/classes/client
   ```

   Ant タスク :

   ```
   <oracle:topDownAssemble
           appName="doclit_topdown"
           wsdl="./wsdl/LoggingFacility.wsdl"
           unwrapParameters="false"
           input="build/classes/service "
           output="build"
           ear="dist/doclit_topdown.ear"
           mappingFileName="type-mapping.xml"
           packageName="oracle.demo.topdowndoclit.service"
           fetchWsdlImports="true"
           >
              <oracle:portType
                 className="oracle.demo.topdowndoclit.service.DocLitLoggerImpl"/>
     </oracle:topDownAssemble>
   ```

   このコマンドおよび Ant タスクの説明 :

   - `topDownAssemble`: WSDL 記述をベースにした Web サービスで必要となるクラスとデプロイメント・ディスクリプタを作成します。これらのファイルは、EAR ファイル、WAR ファイル、ディレクトリのいずれにも格納できます。18-21 ページの「topDownAssemble」を参照してください。
   - `appName`: アプリケーションの名前を指定します。この名前は、通常 `context` や `uri` などの他の引数のベース値として使用されます。18-40 ページの「appName」を参照してください。
   - `wsdl`: WSDL ドキュメントの絶対ファイル・パス、相対ファイル・パスまたは URL を指定します。18-64 ページの「wsdl」を参照してください。

- `unwrapParameters`: `false` に設定すると、生成対象のサービス・エンドポイント・インタフェースが、入力パラメータおよび戻り型をラップするラッパーとともに生成されます。18-69 ページの「unwrapParameters」を参照してください。
- `input`: `WEB-INF/classes` にコピーされるクラスを格納する JAR またはディレクトリを指定します。この引数は、WebServicesAssembler によって使用されるクラスパスに追加されます。18-45 ページの「input」を参照してください。
- `output`: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
- `ear`: 生成される EAR ファイルの名前と位置を指定します。18-42 ページの「ear」を参照してください。
- `mappingFileName`: JAX-RPC マッピング・ファイルを指すファイル位置を指定します。18-46 ページの「mappingFileName」を参照してください。
- `packageName`: JAX-RPC マッピング・ファイルにパッケージ名が宣言されていない場合に、生成されたクラス用に使用されるパッケージ名を指定します。18-47 ページの「packageName」を参照してください。
- `fetchWsdlImports`: WSDL とそれにインポートされるものすべてについてローカル・コピーを作成するかどうかを指定します。18-63 ページの「fetchWsdlImports」を参照してください。

少なくとも、WSDL 名、サービスを実装するクラス名、出力ディレクトリ名を指定します。EAR ファイルおよび EAR 内の WAR ファイルが出力されます。WAR ファイルには、サービスのアーチファクト、実装クラス、Web デプロイメント・ディスクリプタ（`web.xml`）および JAX-RPC デプロイメント・ディスクリプタ（`webservices.xml`）が含まれています。`topDownAssemble` の必須およびオプションの引数の詳細は、18-21 ページの「topDownAssemble」を参照してください。

6. サービスをデプロイします。

   通常の方法で EAR ファイルを OC4J の実行中インスタンスにデプロイします。EAR ファイルのデプロイの詳細は、『Oracle Containers for J2EE デプロイメント・ガイド』を参照してください。次は、デプロイメント・コマンドのサンプルです。

```
java -jar <oc4jHome>/j2ee/home/admin_client.jar deployer:oc4j:localhost:port <user>
<password>
            -deploy
            -file dist/doclit_topdown.ear
            -deploymentName doclit_topdown
            -bindWebApp default-web-site
```

   次のリストは、このコード例のパラメータの説明です。

   - *<oc4j_Home>*: OC4J インストールが含まれているディレクトリ。
   - *<user>*: OC4J のインスタンスのユーザー名。ユーザー名はインストール時に割り当てられます。
   - *<password>*: OC4J のインスタンス用のパスワード。パスワードはインストール時に割り当てられます。
   - `doclit_topdown`: アプリケーションの名前。
   - `default-web-site`: アプリケーションのバインド先の Web サイト。通常、これは `default-web-site` になります。Web サイトを構成するには、*<oc4j_home>*`/j2ee/home/config` の `server.xml` ファイルを参照してください。

7. 必要に応じて、デプロイメントが成功したかどうかを確認します。OracleAS Web Services では、デプロイされた Web サービスごとに Web サービス・テスト・ページが提供されます。Web サービス・テスト・ページへのアクセスおよびその使用方法の詳細は、13-7 ページの「Web サービス・テスト・ページの使用方法」を参照してください。

8. クライアント・コードを生成します。

   - J2SE 環境の場合、genProxy コマンドを使用して WebServicesAssembler ツールを実行し、J2SE Web サービス・クライアントのスタブ（クライアント・プロキシ）を生成します。J2SE Web サービスのクライアント側コードの生成およびアセンブルの詳細は、第 15 章「J2SE Web サービス・クライアントのアセンブル」を参照してください。
   - J2EE 環境の場合、genInterface コマンドを使用して WebServicesAssembler ツールを実行し、J2EE Web サービス・クライアント用の JAX-RPC マッピング・ファイルおよびサービス・エンドポイント・インタフェースを生成します。J2EE Web サービスのクライアント側コードの生成およびアセンブルの詳細は、第 14 章「J2EE Web サービス・クライアントのアセンブル」を参照してください。

   たとえば、次のコマンドは、J2SE クライアント用に使用可能なクライアント・プロキシ（スタブ）をアセンブルします。この例において、doclit_topdown は、生成された Web サービスのアプリケーション名です。

   コマンドライン：

```
java -jar wsa.jar -genProxy
                -wsdl http://localhost:8888/doclit_topdown/doclit_topdown?WSDL
                -unwrapParameters false
                -output build/src/client
                -packageName oracle.demo.topdowndoclit.stubs
                -mappingFileName type-mapping.xml
```

   Ant タスク：

```
<oracle:genProxy
    wsdl="http://localhost:8888/doclit_topdown/doclit_topdown?WSDL"
    unwrapParameters="false"
    output="build/src/client"
    packageName="oracle.demo.topdowndoclit.stubs"
    mappingFileName="type-mapping.xml"
/>
```

   このコマンドおよび Ant タスクの説明：

   - genProxy: J2SE Web サービス・クライアントから使用可能な静的プロキシ・スタブを作成します。18-32 ページの「genProxy」を参照してください。
   - wsdl: WSDL ドキュメントの絶対ファイル・パス、相対ファイル・パスまたは URL を指定します。18-64 ページの「wsdl」を参照してください。
   - unwrapParameters: false に設定すると、生成対象のサービス・エンドポイント・インタフェースが、入力パラメータおよび戻り型をラップするラッパーとともに生成されます。18-69 ページの「unwrapParameters」を参照してください。
   - output: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
   - packageName: JAX-RPC マッピング・ファイルにパッケージ名が宣言されていない場合に、生成されたクラス用に使用されるパッケージ名を指定します。18-47 ページの「packageName」を参照してください。
   - mappingFileName: JAX-RPC マッピング・ファイルを指すファイル位置を指定します。18-46 ページの「mappingFileName」を参照してください。

   少なくとも、WSDL 名および出力ディレクトリ名を指定します。WebServiceAssembler ツールにより、スタブが生成されます。クライアント・アプリケーションはこのスタブを使用して、リモート・サービスでの操作を起動します。genProxy の必須およびオプションの引数の詳細は、18-32 ページの「genProxy」を参照してください。

9. クライアントをコンパイルおよび実行します。

   クライアントをコンパイルする前に、クラスパス上にある適切な JAR をリストします。クライアントのクラスパスで使用可能なすべての JAR ファイルのリストについては、表 A-2「クライアント・サイド・プロキシを使用するクライアントのクラスパス・コンポーネント」を参照してください。個々の JAR をリストするかわりに、クライアントのクラスパス

にクライアント側の JAR として wsclient_extended.jar を含めることができます。この JAR ファイルには、Web サービス・クライアントをコンパイルおよび実行するために必要なすべてのクラスが含まれます。これらのクラスは、表 A-2 にリストされている各 JAR ファイルに含まれているものです。wsclient_extended.jar およびクライアントのクラスパスの詳細は、A-3 ページの「Web サービス・プロキシのクラスパスの設定」を参照してください。

制限事項

F-6 ページの「WSDL からの Web サービスのアセンブル」を参照してください。

追加情報

詳細は、次を参照してください。

- テスト・ページを使用した Web サービス・デプロイのテスト方法は、第 13 章「Web サービス・デプロイのテスト」を参照してください。
- J2EE クライアントの構築方法は、第 14 章「J2EE Web サービス・クライアントのアセンブル」を参照してください。
- J2SE クライアントの構築方法は、第 15 章「J2SE Web サービス・クライアントのアセンブル」を参照してください。
- JAX-RPC ハンドラの詳細は、第 16 章「JAX-RPC ハンドラの使用方法」を参照してください。
- WebServicesAssembler ツールを使用した Web サービスのアセンブル方法は、第 18 章「WebServicesAssembler の使用方法」を参照してください。
- Web サービスのパッケージ化およびデプロイ方法は、第 19 章「Web サービスのパッケージ化およびデプロイ」を参照してください。
- Web サービスの相互運用性の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「相互運用可能な Web サービスの実現」を参照してください。
- Web サービス・クライアントにおけるサービスのクオリティ機能の使用方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの管理」を参照してください。
- Web サービスへのセキュリティの追加方法は、『Oracle Application Server Web Services セキュリティ・ガイド』を参照してください。
- Web サービスへの信頼性の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの信頼性の確保」を参照してください。
- Web サービスへの監査およびロギング構成の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「監査メッセージおよびロギング・メッセージ」を参照してください。
- 標準以外のデータ型の処理の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 値タイプのカスタム・シリアライズ」を参照してください。
- JAX-RPC マッピング・ファイルの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「JAX-RPC マッピング・ファイル記述子」を参照してください。
- OracleAS Web Services でサポートされているデータ型の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 型の XML および WSDL 型へのマッピング」を参照してください。
- Web サービス開発用の Oracle JDeveloper ツールのサポートの詳細は、Oracle JDeveloper のオンライン・ヘルプを参照してください。

7

Java クラスを使用した Web サービスのアセンブル

この章では、Java クラスをベースにしてステートレスおよびステートフルな Web サービスをアセンブルする方法について説明します。アセンブリは、WebServicesAssembler によってボトムアップ方式で実行されます。

ステートレス Web サービスは、複数のコールにわたったローカル状態の保持を行いません。ステートレス Web サービスは、.NET または任意のベンダーの Web サービスと相互運用可能です。サービスは、トランスポート・メカニズムとして HTTP または JMS を使用できます。

対照的に、ステートフル Web サービスは、複数のコールにわたって状態を保持します。このため、メソッド起動の結果は、メソッドのスコープに応じて異なってきます。Oracle Application Server Web Services によってサポートされるステートフル Web サービスは、HTTP ベースです。Java クラスをベースとしたステートフル OracleAS Web Services は、Oracle 固有の拡張機能を備えており、サービス・プロバイダが同じセマンティックを持つスコープを用意しないかぎり、その他のサービスとは相互運用できません。

この章の内容は、次のとおりです。

Web サービス用 Java クラスの記述に対する要件

JAX-RPC を使用し、Java ファイルを使用して Web サービスを作成するには、サービスとして公開するリモート・メソッドを定義するパブリック・インタフェースを用意する必要があります。また、このインタフェースの定義は、`java.rmi.Remote` を拡張して、そのメソッドで `java.rmi.RemoteException` オブジェクトをスローする必要があります。また、このインタフェースはパッケージに含めることも必要です。

また、このパブリック・インタフェースを使用して、Web サービスで使用可能にするパブリック・メソッドのシグネチャ（または、サポートされているデータ型を使用するパブリック・メソッド）をリストすることもできます。つまり、インタフェースを使用して、公開するメソッドをフィルタ処理することができます。

インタフェースの実装は、次の要件も満たす必要があります。

- クラスには、デフォルトのパブリック・コンストラクタを含める必要があります。
- 各クラス・メソッドは、サービス・エンドポイント・インタフェースのメソッドを実装する必要があります。
- クラス・メソッドを final にすることはできません。
- ステートフル Web サービスの場合、クラスは `java.io.Serializable` を実装する必要があります。
- クラスをパッケージに含める必要があります。
- インタフェースのすべてのメソッドは、`java.rmi.RemoteException` をスローする必要があります。また、メソッドはその他の固有の例外を宣言してもかまいません。これらの例外は、`java.lang.Exception` を直接または間接的に拡張したものにする必要がありますが、`RuntimeException` にすることはできません。
- メソッドのパラメータ型および戻り型は、JAX-RPC でサポートされている Java 型である必要があります。

  サポートされている Java 型のリストについては、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 型の XML および WSDL 型へのマッピング」を参照してください。
- ホルダー・クラスは、メソッドのパラメータとして使用できます。これらのホルダー・クラスは、`javax.xml.rpc.holders` パッケージから生成または導出されます。
- 実装クラスに public final static 宣言を含めることはできません。
- サービス・エンドポイント・インタフェースには、リモート参照（`RemoteInterface` を実装するクラス）をパラメータ型または戻り型として含めることはできません。Java 配列または JAX-RPC 値タイプには、リモート参照を要素として含めることはできません。

  インタフェースに関するすべての要件の詳細は、Web アドレス `http://www.jcp.org/aboutJava/communityprocess/final/jsr109/index.html` にある Enterprise Web Services 1.1 の仕様を参照してください。

Web サービス・クライアントがサービス・リクエストを行うと、OC4J はそのクラス内の対応するメソッドを実行します。Web サービスで実行できるアクションには、制限がほとんどありません。Web サービスでは、クライアントに送信するデータが生成されるか、Web サービス・リクエストで指定されたアクションが実行されます。

注意： サービス・インタフェースを提供せず、クラスが準拠クラスの条件の一部を満たしていない場合であっても、OracleAS Web Services は Java クラスを Web サービスとして直接公開できる場合があります。このような場合に Java クラスを Web サービスとして公開するには、`@WebService` などの J2SE 5.0 注釈を使用する必要があります。詳細は、第 11 章「注釈を使用した Web サービスのアセンブル」を参照してください。

また、このような条件の下でアセンブルされた Web サービスは、JAX-RPC 仕様で定義されている適合性または移植性を備えていないことにも注意してください。

関連項目：

ステートレス Web サービス用の Java クラスを記述する方法の詳細は、7-8 ページの「ステートレス Web サービス用の Java 実装の記述方法」を参照してください。また、ステートフル Web サービス用の Java クラスを記述する方法の詳細は、7-14 ページの「ステートフル Web サービス用の Java 実装の記述方法」を参照してください。

Java クラスとステートレス Web サービス

この項の内容は、次のとおりです。

ステートレス Web サービスとしての Java クラスの公開

WebServicesAssembler を使用して、JAX-RPC 1.1 の仕様に準拠する Java クラスから Web サービスをアセンブルできます。Java クラスを Web サービスとして公開することは、軽量のシステムを必要としていて、EJB コンテナが提供するトランザクション機能が不要な場合に、適しています。

WebServicesAssembler では、サービスはボトムアップ方式でアセンブルされます。WebServicesAssembler は、公開する Java クラスをベースとしてアセンブルを行い、WSDL、マッピング・ファイル、実装ファイルおよびデプロイメント・ディスクリプタが含まれた、デプロイ可能な EAR ファイルを生成します。

JAX-RPC を使用するには、サービスとして公開するメソッドが含まれる Java クラスとそのインタフェースを用意する必要があります。

Java クラスをベースとした Web サービスは、Java、.NET またはその他のプログラミング言語で記述されたクライアントから起動できます。Web サービスは、トランスポート・メカニズムとして HTTP または JMS を使用できます。このクライアントは、静的スタブまたは Dynamic Invocation Interface（DII）をベースにすることができます。

関連項目：

クラスとインタフェースの要件の詳細は、7-2 ページの「Web サービス用 Java クラスの記述に対する要件」を参照してください。

Java クラスを使用したステートレス Web サービスのアセンブル方法

この項の内容は、次のとおりです。

前提条件

開始する前に、次のファイルと情報を用意してください。

- サービスとして公開するメソッドが含まれるコンパイル済の Java クラスとそのインタフェースを用意します。Java クラスとそのインタフェースは、Web サービス用の JAX-RPC 標準に準拠する必要があります。

  Java クラスをステートレス Web サービスとして公開する場合は、7-8 ページの「ステートレス Web サービス用の Java 実装の記述方法」を参照してください。Java クラスをステートフル Web サービスとして公開する場合は、7-10 ページの「ステートフル Web サービスとしての Java クラスの公開」を参照してください。

- WebServicesAssembler がサービス・ファイルの生成のみを行うのか、デプロイ可能なアーカイブへのパッケージ化まで行うのかを決定します。ファイルをアーカイブにパッケージ化するには、`ear` 引数を使用します。`ear` を指定しない場合、ファイルは `output` 引数で指定されたディレクトリに格納されます。

  これらの引数の詳細は、18-42 ページの「ear」、18-46 ページの「output」および 18-49 ページの「war」を参照してください。

- Java クラスでその他のメッセージ処理コンポーネントを操作する必要がある場合（たとえば、信頼性機能やセキュリティ機能を実現する場合など）は、メッセージ・ハンドラを指定できます。

  これらのトピックの詳細は、第 17 章「SOAP ヘッダーの処理」および 18-81 ページの「Ant タスクでのハンドラの構成方法」を参照してください。

- Java クラス内のメソッドが標準外のデータ型を使用している場合は、カスタム・シリアライザを指定して処理する必要があります。

  サポートされていないデータ型を使用する場合は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 型の XML および WSDL 型へのマッピング」および 5-7 ページの表 5-1「rpc-encoded 固有のメッセージ書式でサポートされている Java 型」を参照してください。

  カスタム・シリアライズの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 値タイプのカスタム・シリアライズ」を参照してください。

  サポートされているデータ型のリストについては、JAX-RPC 1.1 の仕様を参照してください。この仕様は次の Web サイトで入手できます。

  `http://java.sun.com/webservices/jaxrpc/index.jsp`.

Java クラスを使用したステートレス Web サービスのアセンブル手順

次の手順は、WebServicesAssembler を使用して Java クラスを基にステートレス Web サービスを公開する方法を示しています。

1. Web サービスとして公開するコンパイル済 Java クラスとそのコンパイル済インタフェースを用意します。

   次の例では、`HelloInterface` インタフェースと `HelloImpl` クラスを使用しています。これらのファイルのコード・リストについては、7-5 ページの「Java クラスを使用したステートレス Web サービスのアセンブル手順」を参照してください。

2. `assemble` コマンドを使用して WebServicesAssembler を実行し、サービスのアーチファクトを生成します。この例では、インタフェースと実装クラスは `./build/classes/service` ディレクトリにコンパイル済であるものと想定されています。

   コマンドライン :

```
java -jar wsa.jar -assemble
    -appName hello
    -serviceName HelloService
    -interfaceName oracle.demo.hello.HelloInterface
    -className oracle.demo.hello.HelloImpl
    -input ./build/classes/service
    -output build
    -ear dist/hello.ear
    -uri HelloService
    -targetNamespace http://hello.demo.oracle
    -classpath ./build/classes/service
```

   Ant タスク :

```
<oracle:assemble appName="hello"
        serviceName="HelloService"
        input="./build/classes/service"
        output="build"
        ear="dist/hello.ear"
        targetNamespace="http://hello.demo.oracle"
        >
            <oracle:porttype
                interfaceName="oracle.demo.hello.HelloInterface"
                className="oracle.demo.hello.HelloImpl">
                <oracle:port uri="HelloService" />
            </oracle:porttype>
            <oracle:classpath>
                <pathelement location="build/classes/service"/>
            </oracle:classpath>
</oracle:assemble>
```

   このコマンドおよび Ant タスクの説明 :

   - `assemble`: ボトムアップ方式で Java ファイルから Web サービスを生成します。このコマンドにより、デプロイ可能なアーカイブを作成するために必要なすべてのファイルが作成されます。18-8 ページの「assemble」を参照してください。
   - `appName`: アプリケーションの名前を指定します。この名前は、通常 `context` や `uri` などの他の引数のベース値として使用されます。18-40 ページの「appName」を参照してください。
   - `serviceName`: サービス名を指定します。18-48 ページの「serviceName」を参照してください。
   - `interfaceName`: サービス・エンドポイント・インタフェース（SEI）を格納する Java クラスの名前（パッケージ名を含む）を指定します。18-45 ページの「interfaceName」を参照してください。

- `className`: Web サービスの実装クラスのクラス名（パッケージ名を含む）を指定します。18-41 ページの「className」を参照してください。
- `input`: `WEB-INF/classes` にコピーされるクラスを格納する JAR またはディレクトリを指定します。この引数は、WebServicesAssembler によって使用されるクラスパスに追加されます。18-45 ページの「input」を参照してください。
- `output`: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
- `ear`: 生成される EAR ファイルの名前と位置を指定します。18-42 ページの「ear」を参照してください。
- `uri`: Web サービスに使用する URI を指定します。18-63 ページの「uri」を参照してください。
- `targetNamespace`: 生成される WSDL で使用するターゲット名前空間を指定します。この値は、仕様に準拠している HTTP URL、準拠していない HTTP URL、または URI であってもかまいません。18-66 ページの「targetNamespace」を参照してください。
- `classpath`: WebServicesAssembler に与えられるユーザー作成クラスをすべて含むクラスパスを指定します。この引数を指定する理由の 1 つは、一部の値タイプ・クラスまたは例外を作成済の場合に、WebServicesAssembler が上書きしないようにするためです。18-41 ページの「classpath」を参照してください。

このコマンドの出力は、OC4J インスタンスにデプロイ可能な WAR ファイルの内容が含まれた EAR ファイルです。`dist` ディレクトリには、J2EE Web サービスに準拠したアプリケーション EAR ファイルである `hello.ear` が格納されます。`assemble` の必須およびオプションの引数の詳細は、18-8 ページの「assemble」を参照してください。

3. サービスをデプロイし、アプリケーションをバインドします。

   通常の方法で EAR ファイルを OC4J の実行中インスタンスにデプロイします。EAR ファイルのデプロイの詳細は、『Oracle Containers for J2EE デプロイメント・ガイド』を参照してください。次は、デプロイメント・コマンドのサンプルです。

   ```
   java -jar <OC4J_HOME>/j2ee/home/admin_client.jar deployer:oc4j:localhost:port
   <user> <password>
               -deploy
               -file dist/hello.ear
               -deploymentName hello
               -bindWebApp default-web-site
   ```

   次のリストは、このコード例で使用される変数の説明です。

   - *<OC4J_HOME>*: OC4J インストールが含まれているディレクトリ。
   - *<user>*: OC4J のインスタンスのユーザー名。ユーザー名はインストール時に割り当てられます。
   - *<password>*: OC4J のインスタンス用のパスワード。パスワードはインストール時に割り当てられます。
   - `default-web-site`: アプリケーションのバインド先の Web サイト。通常、これは `default-web-site` になります。Web サイトを構成するには、`<OC4J_HOME>/j2ee/home/config` の `server.xml` ファイルを参照してください。

4. 必要に応じて、デプロイメントが成功したかどうかを確認します。OracleAS Web Services では、デプロイされた Web サービスごとに Web サービス・テスト・ページが提供されます。Web サービス・テスト・ページへのアクセスおよびその使用方法の詳細は、13-7 ページの「Web サービス・テスト・ページの使用方法」を参照してください。

5. クライアント・サイドのコードを生成します。

   - J2SE 環境の場合、`genProxy` コマンドを使用して WebServicesAssembler ツールを実行し、J2SE Web サービス・クライアントのスタブ（クライアント・プロキシ）を生成します。J2SE 環境向けのクライアント・サイド・コードの生成およびアセンブルの詳細は、第 15 章「J2SE Web サービス・クライアントのアセンブル」を参照してください。

- J2EE 環境の場合、genInterface コマンドを使用して WebServicesAssembler ツールを実行し、J2EE Web サービス・クライアント用のマッピング・ファイルおよびサービス・エンドポイント・インタフェースを生成します。クライアント・サイド・コードの生成およびアセンブルの詳細は、第 14 章「J2EE Web サービス・クライアントのアセンブル」を参照してください。

たとえば、次のコマンドは、J2SE クライアントで使用可能なスタブを生成します。

コマンドライン :

```
java -jar wsa.jar -genProxy
                  -output build/src/client/
                  -wsdl http://localhost:8888/hello/HelloService?WSDL
                  -packageName oracle.demo.hello
```

Ant タスク :

```
<oracle:genProxy  wsdl="http://localhost:8888/hello/HelloService?WSDL"
          output="build/src/client"
          packageName="oracle.demo.hello"
/>
```

このコマンドラインおよび Ant タスクの説明 :

- genProxy: J2SE Web サービス・クライアントから使用可能な静的プロキシ・スタブを作成します。18-32 ページの「genProxy」を参照してください。
- output: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
- wsdl: WSDL ドキュメントの絶対ファイル・パス、相対ファイル・パスまたは URL を指定します。18-64 ページの「wsdl」を参照してください。
- packageName: JAX-RPC マッピング・ファイルにパッケージ名が宣言されていない場合に、生成されたクラス用に使用されるパッケージ名を指定します。18-47 ページの「packageName」を参照してください。

このコマンドは、クライアント・プロキシを生成し、これらをディレクトリ build/src/client に格納します。クライアント・アプリケーションはこのスタブを使用して、リモート・サービスでの操作を起動します。genProxy の必須およびオプションの引数の詳細は、18-32 ページの「genProxy」を参照してください。

6. クライアント・アプリケーションを記述します。
7. クライアントをコンパイルおよび実行します。

クライアントをコンパイルする前に、クラスパス上にある適切な JAR をリストします。クライアントのクラスパスで使用可能なすべての JAR ファイルのリストについては、表 A-2「クライアント・サイド・プロキシを使用するクライアントのクラスパス・コンポーネント」を参照してください。個々の JAR をリストするかわりに、クライアントのクラスパスにクライアント側の JAR として wsclient_extended.jar を含めることができます。この JAR ファイルには、Web サービス・クライアントをコンパイルおよび実行するために必要なすべてのクラスが含まれます。これらのクラスは、表 A-2 にリストされている各 JAR ファイルに含まれているものです。wsclient_extended.jar およびクライアントのクラスパスの詳細は、A-3 ページの「Web サービス・プロキシのクラスパスの設定」を参照してください。

ステートレス Web サービス用の Java 実装の記述方法

OracleAS Web Services では、Web サービスとして動作する Java クラスのステートレス実装がサポートされます。ステートレス Java 実装の場合、OracleAS Web Services では、Java クラスの複数インスタンスがプール内に作成されます。リクエストの処理には、任意のインスタンスを使用できます。リクエストの処理が完了すると、オブジェクトは後続のリクエストに使用できるようにプールに戻されます。

ステートレス Web サービス用の Java クラスをアセンブルできるのは、SOAP/HTTP エンドポイントの場合のみであり、SOAP/JMS エンドポイントの場合はできません。

ステートレス Java Web サービスを開発するには、Java インタフェースおよびその実装を定義します。これについては、次の各項で説明します。

- Java インタフェースの定義方法
- Java クラスの定義方法

注意： サービス・インタフェースを提供せず、クラスが準拠クラスの条件の一部を満たしていない場合であっても、OracleAS Web Services は Java クラスを Web サービスとして直接公開できる場合があります。このような場合に Java クラスを Web サービスとして公開するには、`@WebService` などの J2SE 5.0 注釈を使用する必要があります。詳細は、第 11 章「注釈を使用した Web サービスのアセンブル」を参照してください。

また、このような条件の下でアセンブルされた Web サービスは、JAX-RPC 仕様に準拠する適合性または移植性を備えていないことにも注意してください。

Java インタフェースの定義方法

例 7-1 は、ステートレス Web サービスの `HelloInterface.java` インタフェースを示します。JAX-RPC 1.1 の仕様に準拠するには、このインタフェースをパッケージに含める必要があります。また、このインタフェースは `java.rmi.Remote` を拡張して、そのメソッドで `java.rmi.RemoteException` オブジェクトをスローする必要があります。

例 7-1 ステートレス Web サービスのインタフェースの定義

```
package oracle.demo.hello;

import java.rmi.RemoteException;
import java.rmi.Remote;

public interface HelloInterface extends Remote {
    public String sayHello(String name) throws RemoteException;
}
```

Java クラスの定義方法

Web サービスとして公開するインタフェースのメソッドを実装し、Java クラスを作成します。通常、Web サービスの Java クラスでは、1 つ以上のパブリック・メソッドを定義します。JAX-RPC 1.1 の仕様に準拠するには、この実装クラスをパッケージに含める必要があります。また、`java.rmi.Remote` および `java.rmi.RemoteException` をインポートする必要があります。

例 7-2 に、パブリック・クラス `HelloImpl` を示します。このクラスでは、「`Hello` *name*`!`」という文字列を戻すパブリック・メソッド `sayHello` を定義します。この場合、*name* は入力値です。

例 7-2 ステートレス Web サービスのパブリック・クラスの定義

```
package oracle.demo.hello;

import java.rmi.RemoteException;
import java.rmi.Remote;

public class HelloImpl {
     public HelloImpl() {
   }
   public String sayHello(String name) {
     return ("Hello " + name + "!");
   }
}
```

Java クラスの Web サービス実装には、引数を取らないパブリック・コンストラクタを含める必要があることに留意してください。

Java クラスのメソッドの実行時にエラーが発生すると、`RemoteException` がスローされます。この例外に応答して、OracleAS Web Services は Web サービス（SOAP）障害を戻します。実装に Java クラスを使用する Web サービスのエラーを表示するには、標準の J2EE および OC4J 管理機能を使用します。

Web サービスを実装するメソッドを含んだ Java クラスを作成する場合、メソッド、パラメータおよび戻り値には、サポートされている型か、または OracleAS Web Services でサポートされている標準外の型を使用する必要があります。サポートされているデータ型のリストについては、JAX-RPC 1.1 の仕様を参照してください。この仕様は次の Web サイトで入手できます。

`http://java.sun.com/webservices/jaxrpc/index.jsp`

関連資料：

サポートされているデータ型およびサポートされている標準外の型のリストについては、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 型の XML および WSDL 型へのマッピング」および 5-7 ページの表 5-1 を参照してください。

メソッド、パラメータおよび戻り値にサポートされていない型が使用されている場合、次のいずれかの方法でこれらを処理する必要があります。

- インタフェース・クラスを使用して、公開対象のメソッドを、JAX-RPC でサポートされている型およびサポートされている標準外の型を使用しているメソッドのみに制限します。
- カスタム・シリアライザを使用して、サポートされていない型をマップします。

関連資料：

サポートされていない型の使用方法の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 値タイプのカスタム・シリアライズ」を参照してください。

Java クラスとステートフル Web サービス

この項の内容は、次のとおりです。

ステートフル Web サービスとしての Java クラスの公開

OC4J は、Java クラスをベースとしたステートフル Web サービスをサポートしています。このサービスを実装した Java オブジェクトは、HTTP セッションの期間中有効です。これらのサービスは、状態を保持するため Oracle 固有の拡張機能を備えています。これらの拡張機能のため、サービス・プロバイダが同じセマンティックを持つスコープを用意しないかぎり、ステートフル OracleAS Web Services はその他のサービスとは相互運用できません。

OracleAS Web Services によってサポートされるステートフル Web サービスは、HTTP セッション・ベースです。

ステートフル Web サービスのアセンブル方法

この項の内容は、次のとおりです。

前提条件

Java クラスからステートフル Web サービスを生成するための前提条件は、ステートレス Web サービスについて説明した前提条件と同じです。用意する必要がある情報およびファイルの詳細は、7-4 ページの「前提条件」を参照してください。

ステートフル Web サービスのアセンブル手順

次の手順では、WebServicesAssembler を使用して Java クラスからステートフル Web サービスを作成する方法について説明します。このサービスを実装した Java オブジェクトは、HTTP セッションの期間中有効です。

1. Web サービスとして公開する Java クラスとそのインタフェースを用意します。
2. `assemble` コマンドを使用して WebServicesAssembler を実行し、サービスのアーチファクトを生成します。`assemble` の必須およびオプションの引数の詳細は、18-8 ページの「assemble」を参照してください。

   次に例を示します。

   コマンドライン：

```
java -jar wsa.jar -assemble
    -appName counter
    -serviceName counterService
    -interfaceName oracle.demo.count.CounterInterface
    -className oracle.demo.count.CounterImpl
    -input build/classes/service
    -output build
    -ear dist/counter.ear
    -recoverable true
    -timeout 30
    -uri counterService
```

Ant タスク：

```
<oracle:assemble appName="counter"
            serviceName="counterService"
            input="build/classes/service"
            output="build"
            ear="dist/service.ear"
            recoverable="true"
            timeout="30"
        >
        <oracle:porttype
            interfaceName="oracle.demo.count.CounterInterface"
            className="oracle.demo.count.CounterImpl">
            <oracle:port uri="counterService" />
        </oracle:porttype>
        <oracle:classpath>
            <pathelement path="${wsdemo.common.class.path}"/>
            <pathelement location="build/classes/client"/>
        </oracle:classpath>
    </oracle:assemble>
```

このコマンドおよび Ant タスクの説明：

- `assemble`: ボトムアップ方式で Java ファイルから Web サービスを生成します。このコマンドにより、デプロイ可能なアーカイブを作成するために必要なすべてのファイルが作成されます。18-8 ページの「assemble」を参照してください。
- `appName`: アプリケーションの名前を指定します。この名前は、通常 `context` や `uri` などの他の引数のベース値として使用されます。18-40 ページの「appName」を参照してください。
- `serviceName`: サービス名を指定します。18-48 ページの「serviceName」を参照してください。
- `interfaceName`: サービス・エンドポイント・インタフェース（SEI）を格納する Java クラスの名前（パッケージ名を含む）を指定します。18-45 ページの「interfaceName」を参照してください。
- `className`: Web サービスの実装クラスのクラス名（パッケージ名を含む）を指定します。18-41 ページの「className」を参照してください。
- `input`: `WEB-INF/classes` にコピーされるクラスを格納する JAR またはディレクトリを指定します。この引数は、WebServicesAssembler によって使用されるクラスパスに追加されます。18-45 ページの「input」を参照してください。
- `output`: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
- `ear`: 生成される EAR ファイルの名前と位置を指定します。18-42 ページの「ear」を参照してください。
- `recoverable`: セッション状態のアプリケーションをリカバリ可能にするかどうかを指定します。この引数は、サービスが、セッション・スコープを持つステートフル Web サービスとして公開されているときにのみ使用できます。18-50 ページの「recoverable」を参照してください。
- `timeout`: セッションの開始からタイムアウトまでの秒数を指定します。18-51 ページの「timeout」を参照してください。
- `uri`: Web サービスに使用する URI を指定します。18-63 ページの「uri」を参照してください。

このコマンドは、デプロイ可能なアーカイブを作成するために必要なすべてのファイルを生成します。出力の `build` ディレクトリには、EAR ファイルおよび Java クラス用にそれぞれディレクトリが生成されます。`dist` ディレクトリには、J2EE Web サービスに準拠したアプリケーション EAR ファイルである `counter.ear` が格納されます。

コマンドラインの timeout 引数に注意してください。この引数は、HTTP セッションの開始からタイムアウトまでの秒数を示すのみでなく、暗黙的に session 引数を true に設定します。session が true である場合は、サービス・インスタンスが HTTP セッションにおいて保持されます。

recoverable 引数は、このステートフル・アプリケーションを分散可能とする必要があることを指定します。リカバリ可能とは、対話しているノードがダウンした場合に、サービスが分散環境内でリカバリできることを意味します。つまり、Web サービスの状態も分散可能である必要があります。

3. サービスをデプロイし、アプリケーションをバインドします。

   通常の方法で EAR ファイルを OC4J の実行中インスタンスにデプロイします。EAR ファイルのデプロイの詳細は、『Oracle Containers for J2EE デプロイメント・ガイド』を参照してください。次は、デプロイメント・コマンドのサンプルです。

```
java -jar <OC4J_HOME>/j2ee/home/admin_client.jar deployer:oc4j:localhost:port
<user> <password>
          -deploy
          -file dist/counter.ear
          -deploymentName counter
          -bindWebApp default-web-site
```

   次のリストは、このコード例で使用されているパラメータの説明です。

   - *<OC4J_HOME>*: OC4J のインストールが含まれているディレクトリ。
   - *<user>*: OC4J のインスタンスのユーザー名。ユーザー名はインストール時に割り当てられます。
   - *<password>*: OC4J のインスタンス用のパスワード。パスワードはインストール時に割り当てられます。
   - default-web-site: アプリケーションのバインド先の Web サイト。通常、これは default-web-site になります。Web サイトを構成するには、*<OC4J_HOME>*/j2ee/home/config の server.xml ファイルを参照してください。

4. （オプション）デプロイが成功したことを確認します。

   OracleAS Web Services では、デプロイされた Web サービスごとに Web サービス・テスト・ページが提供されます。Web サービス・テスト・ページへのアクセスおよびその使用方法の詳細は、13-7 ページの「Web サービス・テスト・ページの使用方法」を参照してください。

5. クライアント・コードを生成します。

   - J2SE 環境の場合、genProxy コマンドを使用して WebServicesAssembler ツールを実行し、スタブ（クライアント・プロキシ）を生成します。J2SE 環境向けのクライアント・サイド・コードの生成およびアセンブルの詳細は、第 15 章「J2SE Web サービス・クライアントのアセンブル」を参照してください。
   - J2EE 環境の場合、genInterface コマンドを使用して WebServicesAssembler ツールを実行し、J2EE Web サービス・クライアント用のマッピング・ファイルおよびサービス・エンドポイント・インタフェースを生成します。クライアント・サイド・コードの生成およびアセンブルの詳細は、第 14 章「J2EE Web サービス・クライアントのアセンブル」を参照してください。

   たとえば、次のコマンドは、J2SE クライアント用に使用可能なクライアント・プロキシ（スタブ）を生成します。

   コマンドライン：

```
java -jar wsa.jar -genProxy
                  -output build/src/client/
                  -wsdl http://localhost:8888/counter/counterService?WSDL
                  -packageName oracle.demo.count
```

Ant タスク：

```
<oracle:genProxy wsdl="http://stadp54.us.oracle.com:8888/counter/counter?WSDL"
    output="build/src/client"
    packageName="oracle.demo.count"/>
```

このコマンドおよび Ant タスクの説明：

- genProxy: J2SE Web サービス・クライアントから使用可能な静的プロキシ・スタブを作成します。18-32 ページの「genProxy」を参照してください。
- output: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
- wsdl: WSDL ドキュメントの絶対ファイル・パス、相対ファイル・パスまたは URL を指定します。18-64 ページの「wsdl」を参照してください。
- packageName: JAX-RPC マッピング・ファイルにパッケージ名が宣言されていない場合に、生成されたクラス用に使用されるパッケージ名を指定します。18-47 ページの「packageName」を参照してください。

このコマンドは、クライアント・プロキシを生成し、これらをディレクトリ build/src/client に格納します。クライアント・アプリケーションはこのスタブを使用して、リモート・サービスでの操作を起動します。genProxy の必須およびオプションの引数の詳細は、18-32 ページの「genProxy」を参照してください。

注意： UNIX や Linux などのシステムでは、コマンドラインで必要に応じて URL を引用符（" "）で囲んでください。

---

6. クライアント・アプリケーションを記述します。

   スタブ、DII コールまたはエンドポイント・クライアント・インスタンス上で javax.xml.rpc.Call.SESSION_MAINTAIN_PROPERTY ランタイム・プロパティを true に設定して、クライアントが確実にセッションに参加できるようにします。

   このプロパティを直接設定するかわりに、OracleAS Web Services に用意されている、setMaintainSession(boolean) メソッドがあるラッパー・クラスを使用することもできます。セッションを保持するには、このメソッドを true に設定します。このようなプロパティの設定は、クライアント内でラッパーによって行います。たとえば、クライアント・コードで次のように入力します。

   ```
   CounterServicePortClient c = new CounterServicePortClient();
   //sets Maintain Session to true, as the endpoint is stateful.
   c.setMaintainSession(true);
   ```

7. クライアントをコンパイルおよび実行します。

   クライアントをコンパイルする前に、クラスパス上にある適切な JAR をリストします。クライアントのクラスパスで使用可能なすべての JAR ファイルのリストについては、表 A-2「クライアント・サイド・プロキシを使用するクライアントのクラスパス・コンポーネント」を参照してください。個々の JAR をリストするかわりに、クライアントのクラスパスにクライアント側の JAR として wsclient_extended.jar を含めることができます。この JAR ファイルには、Web サービス・クライアントをコンパイルおよび実行するために必要なすべてのクラスが含まれます。これらのクラスは、表 A-2 にリストされている各 JAR ファイルに含まれているものです。wsclient_extended.jar およびクライアントのクラスパスの詳細は、A-3 ページの「Web サービス・プロキシのクラスパスの設定」を参照してください。

ステートフル Web サービス用の Java 実装の記述方法

ステートフル Web サービスの Java 実装は、ステートレス・サービスの実装と同じ要件を満たす必要があります。

関連項目：

Java 実装に対する要件の詳細は、7-2 ページの「Web サービス用 Java クラスの記述に対する要件」を参照してください。

WebServicesAssembler を使用すると、ステートフル Java 実装に対するコール、セッションおよびエンドポイントの各スコープを定義できます。

- `call` スコープ : WebServicesAssembler の callScope 引数を `true` に設定すると、コールごとに新しいクラス・インスタンスが作成されます。インスタンスは、コールが終了するたびにガベージ・コレクションされます。`callScope` のデフォルト値は `false` です。
- session スコープ : WebServicesAssembler の session 引数を `true` に設定すると、クラス・インスタンスは HTTP セッションに格納されます。これが適用されるのは、HTTP トランスポートの場合のみです。セッション・タイムアウトは、`timeout` 引数によってチューニングできます。

  `session` のデフォルト値は `false` です。ただし、WebServicesAssembler で `timeout` 引数が設定されている場合は、`session` は自動的に `true` に設定されます。
- endpoint スコープ : これはデフォルトの動作であり、Web サービス・アプリケーションに対するスコープです。サービス・エンドポイント実装のクラス・インスタンスは、エンドポイントごとにシングルトン・インスタンスです。

ステートフル Java Web サービスの開発手順については、次の各項で説明します。

- Java インタフェースの定義方法
- Java クラスの定義方法

Java インタフェースの定義方法

例 7-3 は、ステートフル Web サービスの `CountInterface.java` インタフェースを示します。また、この例は、サービス・クラスがサービス・エンドポイント・インタフェースを直接実装する必要がないことも示しています。JAX-RPC 1.1 の仕様に準拠するには、このインタフェースをパッケージに含める必要があります。また、このインタフェースは `java.rmi.Remote` を拡張して、そのメソッドで `java.rmi.RemoteException` オブジェクトをスローする必要があります。

例 7-3 ステートフル Web サービスのインタフェースの定義

```
package oracle.demo.count;

import java.rmi.RemoteException;
import java.rmi.Remote;

public interface CounterInterface extends Remote {
    // gets the current counter value
    public int getCurrentCounter() throws RemoteException;
}
```

Java クラスの定義方法

Web サービスとして公開するインタフェースのメソッドを実装し、Java クラスを作成します。通常、Web サービスの Java クラスでは、1 つ以上のパブリック・メソッドを定義します。JAX-RPC 1.1 の仕様に準拠するには、この実装クラスをパッケージに含める必要があります。また、`java.rmi.Remote` および `java.rmi.RemoteException` をインポートする必要があります。

例 7-4 に、パブリック・クラス `CounterImpl` を示します。このクラスは、カウントを初期化し、パブリック・メソッド `getCurrentCounter` を定義しています。

例 7-4 ステートフル Web サービスのパブリック・クラスの定義

```
package oracle.demo.count;

import java.rmi.RemoteException;
import java.rmi.Remote;

public class CounterImpl implements java.io.Serializable {
    private int counter = 0;

    public CounterImpl() {
    }

  public int getCurrentCounter() {
      System.out.println("Current counter value is: " + (++counter));
      return (counter);
   }
}
```

Web サービスのパッケージ化およびデプロイ

Java クラスを公開する Web サービスのパッケージについては、19-3 ページの「Java クラスをベースにした Web サービスのパッケージ化」を参照してください。

関連資料：

Web モジュールのデプロイメントの詳細は、『Oracle Containers for J2EE デプロイメント・ガイド』を参照してください。

Web サービスとして Java クラスを公開するためのツールのサポート

Oracle JDeveloper を使用して、J2EE 準拠の Java クラス・ファイルを Web サービスとして作成、変更およびデプロイできます。Oracle JDeveloper で Java クラスを作成する場合、モデリング・ツールおよびウィザードを使用します。このウィザードを使用して、次のタスクを実行できます。

- Java クラスおよびインタフェースをプロジェクトにインポートまたは作成します。
- Web サービスとして公開する Java クラスをパッケージ化およびデプロイします。

関連資料：

Oracle JDeveloper を使用して Java クラスを作成し、Web として公開する方法の詳細は、Oracle JDeveloper のオンライン・ヘルプを参照してください。

制限事項

F-7 ページの「Java クラスからの Web サービスのアセンブル」を参照してください。

追加情報

詳細は、次を参照してください。

- テスト・ページを使用した Web サービス・デプロイのテスト方法は、第 13 章「Web サービス・デプロイのテスト」を参照してください。
- J2EE クライアントの構築方法は、第 14 章「J2EE Web サービス・クライアントのアセンブル」を参照してください。
- J2SE クライアントの構築方法は、第 15 章「J2SE Web サービス・クライアントのアセンブル」を参照してください。
- JAX-RPC ハンドラの詳細は、第 16 章「JAX-RPC ハンドラの使用方法」を参照してください。
- WebServicesAssembler ツールを使用した Web サービスのアセンブル方法は、第 18 章「WebServicesAssembler の使用方法」を参照してください。
- Web サービスのパッケージ化およびデプロイ方法は、第 19 章「Web サービスのパッケージ化およびデプロイ」を参照してください。
- クライアントのアセンブルに必要な JAR ファイルの詳細は、付録 A「Web サービス・クライアントの API および JAR」を参照してください。
- Web サービスの相互運用性の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「相互運用可能な Web サービスの実現」を参照してください。
- Web サービス・クライアントにおけるサービスのクオリティ機能の使用方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの管理」を参照してください。
- Web サービスへのセキュリティの追加方法は、『Oracle Application Server Web Services セキュリティ・ガイド』を参照してください。
- Web サービスへの信頼性の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの信頼性の確保」を参照してください。
- Web サービスへの監査およびロギング構成の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「監査メッセージおよびロギング・メッセージ」を参照してください。
- 標準以外のデータ型の処理の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 値タイプのカスタム・シリアライズ」を参照してください。
- JAX-RPC マッピング・ファイルの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「JAX-RPC マッピング・ファイル記述子」を参照してください。
- OracleAS Web Services でサポートされているデータ型の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 型の XML および WSDL 型へのマッピング」を参照してください。
- Web サービス開発用の Oracle JDeveloper ツールのサポートの詳細は、Oracle JDeveloper のオンライン・ヘルプを参照してください。

8

EJB を使用した Web サービスのアセンブル

この章では、WebServicesAssembler ツールを使用してバージョン 2.1 の Enterprise JavaBeans（EJB）を Web サービスとして公開する方法について説明します。

この章の内容は、次のとおりです。

Web サービスとしての EJB の公開

WebServicesAssembler を使用して、J2EE 1.4 標準に準拠する Web サービスとしてバージョン 2.1 の EJB を公開できます。EJB の多くは、中間層のビジネス機能をカプセル化するために記述されます。エンタープライズ・アプリケーションで EJB を使用する場合、EJB を Web サービスとして公開できます。

注意： WebServicesAssembler を使用して、バージョン 3.0 の EJB を Web サービスとして公開することはできません。

EJB コンポーネントは設計上、分散コンピューティング用であり、Web サービスとして公開する場合に適しています。EJB 仕様では、宣言的トランザクション、スレッド管理およびロール・ベースのセキュリティがサポートされています。これらの利点は、EJB コンポーネントを Web サービスとして使用する場合に活用できます。Web サービスとして公開した EJB は、従来の RMI EJB クライアントや SOAP プロトコルからもアクセスできます。J2EE 1.4 では、Web サービスとして公開できるのはステートレス・セッション Bean のみです。

設計上、SOAP および Web サービスはステートレスです。このため、ステートレス・セッション Bean は、Web サービスを公開する媒体として理想的です。ステートレス・セッション Bean は、与信の確認、銀行口座への請求、または発注に使用できます。その他のリモート・アプリケーションから使用されるビジネス機能を実装するセッション Bean は、Web サービスとして公開する場合に最適です。

JAX-RPC を使用した EJB Web サービスを記述するには、サービスを実装する EJB を記述し、そのインタフェースを用意する必要があります。EJB には、クライアントが Web サービス・リクエストを行うときに呼び出すことができるビジネス・ロジックを含める必要があります。

EJB Web サービスは、Java、.NET またはその他のプログラミング言語で記述されたクライアントから呼び出せる点で、その他の Web サービスと同じです。EJB Web サービスのクライアントは、静的スタブ、動的プロキシまたは Dynamic Invocation Interfaces（DII）を利用できます。

EJB を Web サービスとして公開するための一般的な手順は、次のとおりです。

1. ステートレス EJB コンポーネントのサービス・エンドポイント・インタフェースを作成します。

   この手順は開発者が実行します。

2. サービスのアーチファクトをアセンブルします。これを行うには、WSDL ファイルおよびマッピング・ファイルを生成し、アプリケーションをデプロイ可能なアーカイブにパッケージ化する必要があります。

   この手順は WebServicesAssembler を使用して実行できます。

   関連項目：

   これらの各ステップの詳細は、「EJB から Web サービスをアセンブルする手順」を参照してください。

バージョン 2.0 の EJB の操作

この章では、バージョン 2.1 の EJB を Web サービスとして公開することに焦点を当てていますが、バージョン 2.0 の EJB も公開可能です。バージョン 2.1 の EJB を操作するための Oracle Application Server Web Services のすべての機能は、バージョン 2.0 の EJB でも使用できます。OracleAS Web Services および WebServicesAssembler は、バージョン 2.0 の EJB を検出し、それらが正しく処理されているかどうかを確認できます。EJB のリモート・インタフェース・メソッドとして、Web サービスとして公開するメソッドを定義する必要があります。

関連項目：

J2EE Web サービスでの EJB に対する要件の詳細は、8-7 ページの「Web サービス用の EJB を記述する方法」を参照してください。

バージョン 3.0 の EJB の操作

WebServicesAssembler ツールを使用して、バージョン 3.0 の EJB を Web サービスとして公開することはできません。ただし、OracleAS Web Services がサポートする J2SE 5.0 Web サービス注釈を使用してこれらの EJB を公開することはできます。

関連項目：

注釈を使用してバージョン 3.0 の EJB を Web サービスとして公開する方法の詳細は、第 11 章「注釈を使用した Web サービスのアセンブル」および「バージョン 3.0 の EJB から Web サービスをアセンブルするために注釈を使用する手順」を参照してください。

EJB での Web サービスのアセンブル方法

この項の内容は、次のとおりです。

前提条件

開始する前に、次のファイルと情報を用意してください。

- 公開するビジネス機能を含んだ Enterprise JavaBean とそのインタフェースを記述します。EJB とそのインタフェースは、EJB 2.1 標準および Web サービス用の J2EE 1.4 標準に準拠する必要があります。

  これらの要件の詳細は、8-7 ページの「Web サービス用の EJB を記述する方法」を参照してください。

- WebServicesAssembler がサービス・ファイルの生成のみを行うのか、サービス・ファイルの生成とデプロイ可能なアーカイブへのパッケージ化を行うのかを決定します。ファイルをアーカイブにパッケージ化するには、`ear` 引数を使用します。`ear` を指定しない場合、ファイルは `output` 引数で指定されたディレクトリに格納されます。

  出力引数およびパッケージ化引数の詳細は、18-42 ページの「ear」、18-46 ページの「output」および 18-46 ページの「war」を参照してください。

- EJB 内のメソッドが標準外のデータ型を使用している場合は、カスタム・シリアライザを指定して処理する必要があります。

  標準ではないデータ型の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 型の XML および WSDL 型へのマッピング」および 5-7 ページの表 5-1を参照してください。

  カスタム・シリアライザの作成方法の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 値タイプのカスタム・シリアライズ」を参照してください。

- EJB でその他のメッセージ処理コンポーネントを操作する必要がある場合（たとえば、SOAP ヘッダー情報を処理する場合）は、メッセージ・ハンドラを指定できます。

  これらのトピックの詳細は、第 17 章「SOAP ヘッダーの処理」および 18-81 ページの「Ant タスクでのハンドラの構成方法」を参照してください。

EJB から Web サービスをアセンブルする手順

次の手順では、WebServicesAssembler を使用してセッション Bean を Web サービスとして公開する方法について説明します。

1. Web サービスとして公開する EJB とそのサービス・エンドポイント・インタフェースを記述します。
2. `ejb-jar.xml` デプロイメント・ディスクリプタを調べます。

   `<service-endpoint>` 要素がファイルにない場合、この要素とその値を入力します。この要素は、この Web サービスのサービス・エンドポイント・インタフェースを識別します。次の `ejb-jar.xml` のフラグメントでは、`<service-endpoint>` 要素は太字で強調表示されています。

```
<enterprise-beans>
    <session>
        <ejb-name>HelloServiceBean</ejb-name>
        <service-endpoint>oracle.demo.ejb.HelloServiceIntf</service-endpoint>
        <ejb-class>oracle.demo.ejb.HelloserviceBean</ejb-class>
        <session-type>Stateless</session-type>
        <transaction-type>Container</transaction-type>
    </session>
</enterprise-beans>
```

3. `ejbAssemble` コマンドを使用して WebServicesAssembler ツールを実行し、サービスのアーチファクトを生成します。次に例を示します。

   コマンドライン：

```
java -jar wsa.jar -ejbAssemble
                -appName helloServices-ejb
                -ear dist/helloServices-ejb.ear
                -output build
                -targetNamespace http://oracle.j2ee.ws/ejb/Hello
                -typeNamespace http://oracle.j2ee.ws/ejb/Hello/types
                -input dist/HelloServiceejb.jar
                -ejbName HelloServiceBean
```

   Ant タスク：

```
<oracle:ejbAssemble appName="ejbApp"
          targetNamespace="http://oracle.j2ee.ws/ejb/Hello"
          ear="dist/ejbApp.ear"
          output="build"
          typeNamespace="http://oracle.j2ee.ws/ejb/Hello/types"
          input dist/HelloServiceejb.jar
          ejbName HelloServiceBean
/>
```

   このコマンドおよび Ant タスクの説明：

   - `ejbAssemble`: EJB を Web サービスとして公開できる EAR または EJB JAR を作成します。バージョン 2.1 EJB の、有効な JAR を入力として指定する必要があります。システムでは、WSDL と固有の `oracle-webservices.xml` デプロイメント・ディスクリプタが作成されます。18-14 ページの「ejbAssemble」を参照してください。
   - `appName`: アプリケーションの名前を指定します。この名前は、通常 `context` や `uri` などの他の引数のベース値として使用されます。18-40 ページの「appName」を参照してください。

- `ear`: 生成される EAR ファイルの名前と位置を指定します。18-42 ページの「ear」を参照してください。
- `output`: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
- `targetNamespace`: 生成される WSDL で使用するターゲット名前空間を指定します。この値は、仕様に準拠している HTTP URL、準拠していない HTTP URL、または URI であってもかまいません。18-66 ページの「targetNamespace」を参照してください。
- `typeNamespace`: 生成される WSDL 内のスキーマ型に使用する型名前空間を指定します。指定した名前は必ず使用され、破棄されません。18-67 ページの「typeNamespace」を参照してください。
- `input`: `WEB-INF/classes` にコピーされるクラスを格納する JAR またはディレクトリを指定します。この引数は、WebServicesAssembler によって使用されるクラスパスに追加されます。18-45 ページの「input」を参照してください。
- `ejbName`: Web サービスとして公開する EJB の名前を指定します。これはクラス名ではありません。`ejb-jar.xml` ファイルの `<ejb-name>` タグに指定された EJB の一意の名前です。18-44 ページの「ejbName」を参照してください。

このコマンドは、WSDL ファイルおよびマッピング・ファイルを生成し、アプリケーションをデプロイ可能なアーカイブ `dist/helloServices-ejb.ear` にパッケージ化することにより、EJB 2.1 Web サービスをアセンブルします。このアーカイブには、`helloService-ejb.jar` ファイルが含まれます。このファイルには、EJB 実装クラス、生成された WSDL ファイルおよびマッピング・ファイル、標準 Web サービス・ディスクリプタ・ファイル `webservices.xml` および Oracle 固有のデプロイメント・ディスクリプタ・ファイル `oracle-webservices.xml` が格納されています。`ejbAssemble` コマンドの詳細は、18-14 ページの「ejbAssemble」を参照してください。

4. サービスをデプロイし、アプリケーションをバインドします。

   EAR ファイルは、OC4J の実行中インスタンスにデプロイされます。デプロイメントの詳細は、『Oracle Containers for J2EE デプロイメント・ガイド』を参照してください。次は、デプロイメント・コマンドのサンプルです。

   ```
   java -jar <OC4J_HOME>/j2ee/home/admin_client.jar deployer:oc4j:localhost:port
   <user> <password>
               -deploy
               -file dist/ejbApp.ear
               -deploymentName ejbApp
               -bindWebApp default-web-site
   ```

   次のリストは、このコード例のパラメータの説明です。

   - *<OC4J_HOME>*: OC4J のインストールが含まれているディレクトリ。
   - *<user>*: OC4J のインスタンスのユーザー名。ユーザー名はインストール時に割り当てられます。
   - *<password>*: OC4J のインスタンス用のパスワード。パスワードはインストール時に割り当てられます。
   - *ejbApp*: アプリケーションの名前。
   - `default-web-site`: アプリケーションのバインド先の Web サイト。通常、これは `default-web-site` になります。Web サイトを構成するには、*<OC4J_HOME>*`/j2ee/home/config` の `server.xml` ファイルを参照してください。

   デプロイメントを行うと、この Web サービスの WSDL が次の Web アドレスとして利用可能になります。*context-root* および *endpoint-address-uri* の値は、`META-INF/oracle-webservices.xml` ファイルにあります。

   `http://`*host*`:`*port*`/`*context-root*`/`*endpoint-address-uri*

5. （オプション）デプロイが成功したことを確認します。

   OracleAS Web Services では、デプロイされた Web サービスごとに Web サービス・テスト・ページが提供されます。Web サービス・テスト・ページへのアクセスおよびその使用方法の詳細は、13-7 ページの「Web サービス・テスト・ページの使用方法」を参照してください。

6. クライアント・サイドのコードを生成します。

   - J2SE 環境の場合、genProxy コマンドを使用して WebServicesAssembler ツールを実行し、J2SE Web サービス・クライアントのスタブ（クライアント・プロキシ）を生成します。スタブの生成およびアセンブルの詳細は、第 15 章「J2SE Web サービス・クライアントのアセンブル」を参照してください。
   - J2EE 環境の場合、genInterface コマンドを使用して WebServicesAssembler ツールを実行し、J2EE Web サービス・クライアント用の JAX-RPC マッピング・ファイルおよびサービス・エンドポイント・インタフェースを生成します。J2EE Web サービスのクライアントの生成およびアセンブルの詳細は、第 14 章「J2EE Web サービス・クライアントのアセンブル」を参照してください。

   たとえば、次のコマンドと Ant タスクは、J2SE クライアントで使用可能なスタブを生成します。クライアント・プロキシは、ディレクトリ build/src/client に格納されます。クライアント・アプリケーションはこのスタブを使用して、リモート・サービスでの操作を起動します。genProxy の必須およびオプションの引数の詳細は、18-32 ページの「genProxy」を参照してください。

   コマンドライン：

   ```
   java -jar wsa.jar -genProxy
                     -output build/src/client/
                     -wsdl http://localhost:8888/hello/HelloService?WSDL
                     -packageName oracle.demo.hello
   ```

   Ant タスク：

   ```
   <oracle:genProxy
             wsdl="http://localhost:8888/hello/HelloService?WSDL"
             output="build/src/client"
             packageName="oracle.demo.hello"
   />
   ```

   このコマンドラインおよび Ant タスクの説明：

   - genProxy: J2SE Web サービス・クライアントから使用可能な静的プロキシ・スタブを作成します。18-32 ページの「genProxy」を参照してください。
   - output: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
   - wsdl: WSDL ドキュメントの絶対ファイル・パス、相対ファイル・パスまたは URL を指定します。18-64 ページの「wsdl」を参照してください。
   - packageName: JAX-RPC マッピング・ファイルにパッケージ名が宣言されていない場合に、生成されたクラス用に使用されるパッケージ名を指定します。18-47 ページの「packageName」を参照してください。

7. クライアントをコンパイルおよび実行します。

   クライアントをコンパイルする前に、クラスパス上にある適切な JAR をリストします。クライアントのクラスパスで使用可能なすべての JAR ファイルのリストについては、表 A-2「クライアント・サイド・プロキシを使用するクライアントのクラスパス・コンポーネント」を参照してください。個々の JAR をリストするかわりに、クライアントのクラスパスにクライアント側の JAR として wsclient_extended.jar を含めることができます。この JAR ファイルには、Web サービス・クライアントをコンパイルおよび実行するために必要なすべてのクラスが含まれます。これらのクラスは、表 A-2 にリストされている各 JAR ファイルに含まれているものです。wsclient_extended.jar およびクライアントのクラスパスの詳細は、A-3 ページの「Web サービス・プロキシのクラスパスの設定」を参照してください。

Web サービス用の EJB を記述する方法

JAX-RPC を使用した EJB ベースの Web サービスを記述するには、サービスを実装する EJB を記述し、そのインタフェースを用意する必要があります。EJB には、クライアントが Web サービス・リクエストを行うと OracleAS Web Services から呼び出されるビジネス機能を含める必要があります。

この項では、「`HELLO!! You just said:`*phrase*」という文字列を戻す EJB Web サービスを記述する方法について説明します。この場合、*phrase* は、クライアントからの入力です。この EJB Web サービスは単一の `String` パラメータを含むクライアント・リクエストを受信し、Web サービス・クライアントに戻すレスポンスを生成します。

Web サービス用に J2EE 1.4 準拠の EJB 実装を記述するには、次のタスクを実行する必要があります。

- EJB サービス・エンドポイント・インタフェースを記述する方法
- Web サービス用の EJB インタフェースを実装する方法

EJB サービス・エンドポイント・インタフェースを記述する方法

JAX-RPC を使用し、EJB を使用して Web サービスを作成するには、EJB に準拠するパブリック・サービス・エンドポイント・インタフェースを記述する必要があります。ステートレス・セッション Bean のサービス・エンドポイント・インタフェースを作成するための要件は、Enterprise Web Services 1.1 仕様の「Section 5.3.2.1」を参照してください。この仕様は、次の Web サイトから入手できます。

`http://www.jcp.org/en/jsr/detail?id=921`

このインタフェースは `java.rmi.Remote` を拡張する必要があり、すべてのメソッドは `java.rmi.RemoteException` をスローする必要があります。サービス・エンドポイント・インタフェースに定義する EJB メソッドのパラメータまたは戻り型として使用できるのは、標準外のデータ型および JAX-RPC 値タイプである Java プリミティブおよびクラスのみです。

JAX-RPC 値タイプの例としては、`java.lang.String` や `java.lang.Double` などのプリミティブ以外の値タイプ、および `java.awt.Image` や `javax.xml.transform.Source` など、Multipurpose Internet Mail Extensions（MIME）タイプに対応する Java 値タイプがあります。

サービス・エンドポイント・インタフェースではカスタム Java データ型を使用できますが、これらを処理するためのシリアライザも用意する必要があります。

関連資料：

- 標準ではないデータ型の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 型の XML および WSDL 型へのマッピング」および 5-7 ページの表 5-1 を参照してください。
- カスタム・データ型の使用方法とそのシリアライズの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 値タイプのカスタム・シリアライズ」を参照してください。

例 8-1 は、サービス・エンドポイント・インタフェースのサンプルです。

例 8-1 サービス・エンドポイント・インタフェースのサンプル

```
package oracle.demo.ejb;

import java.rmi.Remote;
import java.rmi.RemoteException;
/**
 * This is an Enterprise Java Bean Service Endpoint Interface
 */
public interface HelloServiceInf extends java.rmi.Remote {
    /**
     * @param phrase java.lang.String
```

```
     * @return java.lang.String
     * @throws String The exception description.
     */
    java.lang.String sayHello(java.lang.String phrase)
          throws java.rmi.RemoteException;
}
```

Web サービス用の EJB インタフェースを実装する方法

Web サービスで公開するビジネス機能を実装し、Enterprise JavaBean を作成します。

例 8-2 で説明する HelloServiceBean は、セッション Bean のサンプルです。このクラスでは、「HELLO!! You just said: *phrase*」を戻すパブリック・メソッド sayHello を定義します。この場合、*phrase* は、クライアントからの入力です。通常、Web サービスの JavaBean では、1 つ以上のパブリック・メソッドを定義します。

Web サービスに使用する Enterprise JavaBean は、次の要件に準拠する任意の Java クラスです。

- 引数を取らないコンストラクタがある必要があります。
- 使用するすべてのプロパティはアクセッサを介して公開する必要があります。

> **関連資料：**
>
> EJB のパラメータおよび戻り型に対して JAX-RPC がサポートするデータ型の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 型の XML および WSDL 型へのマッピング」および 5-7 ページの表 5-1 を参照してください。

JAX-RPC 標準に準拠するには、HelloServiceBean のすべてのメソッドが java.rmi.RemoteException をスローする必要があります。また、バージョン 2.1 の EJB 仕様および Enterprise Web Services 1.1 仕様のすべての要件も満たす必要があります。

例 8-2 HelloService セッション Bean のサンプル

```
package oracle.demo.ejb;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.Properties;
import javax.ejb.*;

/**
 * This is a Session Bean Class.
 */
public class HelloServiceBean implements SessionBean {
    public String sayHello(String phrase) {
        return "HELLO!! You just said :" + phrase;
    }

    public void setSessionContext(javax.ejb.SessionContext ctx)
            throws java.rmi.RemoteException {
    }

    public void ejbActivate() throws java.rmi.RemoteException {
    }

    public void ejbCreate()
            throws javax.ejb.CreateException,
                   java.rmi.RemoteException {
    }
    public void ejbPassivate() throws java.rmi.RemoteException {
    }
```

```
    public void ejbRemove() throws java.rmi.RemoteException {
    }
}
```

EJB を公開する Web サービスのパッケージ化およびデプロイ

EJB を公開する Web サービスのパッケージの構造については、19-3 ページの「EJB をベースにした Web サービスのパッケージ化」を参照してください。

関連資料：

EJB のデプロイメントの詳細は、『Oracle Containers for J2EE デプロイメント・ガイド』を参照してください。

注意： WebServicesAssembler は、2.1 EJB の構成が正しいかどうかを検出できません。構成が正しいかどうかは、2.1 EJB Web サービスのデプロイ時およびサーバー・サイド・コードの生成時にのみ検出できます。

トランスポート・レベルで EJB ベースの Web サービスを保護する方法

`oracle-webservices.xml` デプロイメント・ディスクリプタの `<ejb-transport-security-constraint>` 要素および `<ejb-transport-login-config>` 要素を使用して、バージョン 2.1 または 3.0 の EJB のトランスポート・レベルのセキュリティ制約を構成できます。これらの要素の詳細は、C-15 ページの「トランスポート・レベルでの EJB ベースの Web サービスの保護」を参照してください。

関連資料：

EJB に対するトランスポート・レベルのセキュリティの実現、およびトランスポート・レベルで保護されている Web サービスにアクセスするクライアントを記述する方法の詳細は、『Oracle Application Server Web Services セキュリティ・ガイド』の「EJB をベースとする Web サービスに対するトランスポート・レベル・セキュリティの追加」および「トランスポート・レベルで保護された Web サービスへのアクセス」を参照してください。

Web サービスとして EJB を公開するためのツールのサポート

Oracle JDeveloper では、モデリング・ツールおよびウィザードを使用して、J2EE に準拠した EJB を作成、変更およびデプロイできます。EJB ウィザードを使用して、次のタスクを実行できます。

- ステートレス・セッション Bean を含む、複数タイプの Enterprise JavaBeans 用としてエンタープライズ Bean クラスを作成します。
- EJB オブジェクトを作成するために必要なホーム・インタフェースを生成します。`ejbCreate()` メソッドを含めると、EJB を Oracle Applications Server に即座にデプロイでき、デプロイ用のメソッドを手動でコード化しなくても済みます。
- ホーム・インタフェース・メソッドの選択を可能にし、デフォルトのメソッドを作成します。
- リモート・インタフェースを生成します。
- リモート・インタフェース・メソッドの選択を可能にします。
- Web サービスとして公開する EJB をデプロイします。

Oracle JDeveloper を使用して EJB を作成し、Web サービスとして公開する方法の詳細は、Oracle JDeveloper のオンライン・ヘルプを参照してください。

制限事項

F-8 ページの「EJB からの Web サービスのアセンブル」を参照してください。

追加情報

詳細は、次を参照してください。

- テスト・ページを使用した Web サービス・デプロイのテスト方法は、第 13 章「Web サービス・デプロイのテスト」を参照してください。
- J2EE クライアントの構築方法は、第 14 章「J2EE Web サービス・クライアントのアセンブル」を参照してください。
- J2SE クライアントの構築方法は、第 15 章「J2SE Web サービス・クライアントのアセンブル」を参照してください。
- JAX-RPC ハンドラの使用方法は、第 16 章「JAX-RPC ハンドラの使用方法」を参照してください。
- WebServicesAssembler ツールを使用した Web サービスのアセンブル方法は、第 18 章「WebServicesAssembler の使用方法」を参照してください。
- Web サービスのパッケージ化およびデプロイ方法は、第 19 章「Web サービスのパッケージ化およびデプロイ」を参照してください。
- クライアントのアセンブルに必要な JAR ファイルの詳細は、付録 A「Web サービス・クライアントの API および JAR」を参照してください。
- Web サービスの相互運用性の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「相互運用可能な Web サービスの実現」を参照してください。
- Web サービス・クライアントにおけるサービスのクオリティ機能の使用方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの管理」を参照してください。
- Web サービスへのセキュリティの追加方法は、『Oracle Application Server Web Services セキュリティ・ガイド』を参照してください。
- EJB ベースの Web サービスへのトランスポート・レベルのセキュリティの追加方法は、C-15 ページの「トランスポート・レベルでの EJB ベースの Web サービスの保護」を参照してください。

  また、『Oracle Application Server Web Services セキュリティ・ガイド』の「トランスポート・レベル・セキュリティの Web サービスへの追加」および「トランスポート・レベルで保護された Web サービスへのアクセス」も参照してください。
- Web サービスへの信頼性の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの信頼性の確保」を参照してください。
- Web サービスへの監査およびロギング構成の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「監査メッセージおよびロギング・メッセージ」を参照してください。
- 標準以外のデータ型の処理の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 値タイプのカスタム・シリアライズ」を参照してください。
- JAX-RPC マッピング・ファイルの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「JAX-RPC マッピング・ファイル記述子」を参照してください。
- OracleAS Web Services でサポートされているデータ型の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 型の XML および WSDL 型へのマッピング」を参照してください。
- Web サービス開発用の Oracle JDeveloper ツールのサポートの詳細は、Oracle JDeveloper のオンライン・ヘルプを参照してください。

9

JMS 宛先を使用した Web サービスのアセンブル

この章では、JMS 宛先を Web サービスとして公開する方法について説明します。JMS エンドポイント Web サービスは、JMS 宛先（キューまたはトピック）を WSDL の document-literal スタイルの操作として公開します。この操作には、送信モードと受信モードがあります。

JMS エンドポイント Web サービスは、Java エンドポイントの Web サービスの特殊ケースとみなすことができます。JMS エンドポイント Web サービスでは、JMS エンドポイントが JMS メッセージ・オブジェクトの送受信という Web サービス操作を実行します。

注意： データベースのキューまたは Oracle Streams アドバンスト・キューイング（AQ）をベースにした Web サービスと JMS 宛先をベースにした Web サービスには、いくつかの相違があります。AQ の Web サービスは、データベース内のキューの構成をベースにしています。JMS 宛先の Web サービスは、中間層にある JMS プロバイダの構成をベースにします。JMS キューは、バックエンドのデータソース内にあります。このデータソースには、データベース、ファイル・ベースのシステムまたはその他のデータ・リポジトリなどがあります。

データベース内のキューまたは AQ から Web サービスを構成する場合は、10-23 ページの「Oracle Streams AQ から Web サービスをアセンブルする方法」を参照してください。

JMS をトランスポート・メカニズムとして使用して Web サービスと通信する場合は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービス・トランスポートとしての JMS の使用方法」を参照してください。

JMS エンドポイント Web サービスの概要

OracleAS Web Services を使用すると、JMS 宛先とメッセージをやりとりする Web サービス・エンドポイントを作成できます。JMS Web サービス・エンドポイントは、特定の JMS 宛先または宛先のペアとの間でメッセージを送受信するように構成されています。

JMS エンドポイント Web サービスには、次の操作のいずれかを持たせることができます。

- send: XML ペイロード（SOAP の body 要素）が、該当する JMS 宛先に送信されます。送信操作を構成して、各送信メッセージに JMS メッセージ・プロパティを設定し、JMS の返信先、優先度、有効期限などを指定できます。
- receive: メッセージが、該当する JMS 宛先から取得され、JMS メッセージ・ボディを使用して SOAP レスポンス・メッセージ・ボディのペイロードが作成されます。
- 両方 : サービスは両方の操作を提供できます。

JMS エンドポイント Web サービスを構成して、メッセージ ID、相関 ID および返信先 JMS メッセージ・プロパティを SOAP ヘッダーとして送信できます。この構成により、生成された WSDL およびスキーマでメッセージ・プロパティのヘッダーおよびタイプが明示的に宣言され、Web サービス・クライアントがこれらを使用できるようになります。

- 宛先が JMS キューである場合、send 操作の起動は、エンキューを行うことを意味します。receive 操作の起動は、デキューを行うことを意味します。
- 宛先がトピックである場合、send 操作はパブリッシュを行うこと、receive 操作はサブスクライブを行うことを意味します。

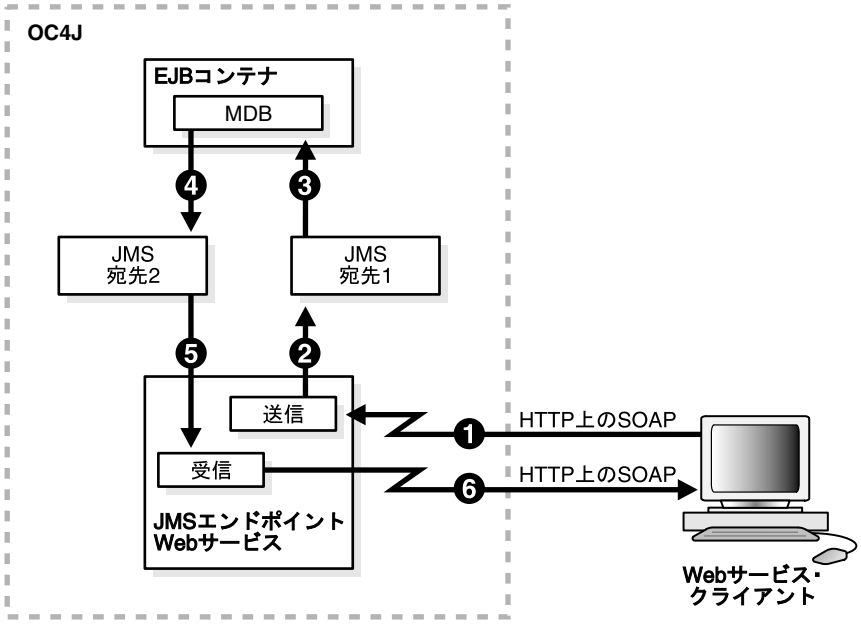
個々の JMS エンドポイント Web サービスでは、サービス開発者の決定に従って send 操作のみ、receive 操作のみまたは両方の操作をサポートできます。

JMS エンドポイント Web サービスは、JMS メッセージ・タイプとして javax.jms.ObjectMessage を使用します。このメッセージは内容として、XML フラグメントの String 表現または javax.xml.soap.SOAPElement のインスタンスを保持します。

送信を行う JMS エンドポイント Web サービス用として生成される WSDL は、Web Service-Interoperability（WS-I）Basic Profile 1.0 に準拠しており、相互運用性があります。

たとえば、図 9-1 に示す MDB ベースの JMS エンドポイント Web サービス・アプリケーションは、JMS エンドポイント Web サービスからみると、メッセージの `send` 操作と `receive` 操作の両方を処理します。また、この図に示す MDB は、JMS の宛先をリスニングするように構成されています。

図 9-1 MDB ベースの JMS エンドポイント Web サービス



次の手順は、図 9-1 に示す MDB ベースの JMS エンドポイント Web サービス・アプリケーションの処理内容を示します。

1. Web サービス・クライアントが、JMS エンドポイント Web サービス上で `send` 操作を起動するための SOAP リクエストを送信します。
2. JMS エンドポイント Web サービスが受信メッセージを処理し、それを JMS 宛先である JMS 宛先 `1` にダイレクトします。
3. EJB コンテナが、JMS 宛先 `1` をリスニングする MDB を起動します。
4. MDB は、メッセージを処理すると、JMS 宛先 `2` で新しいメッセージを生成します。メッセージの生成および使用には、1 つ以上の MDB が関係する場合があります。たとえば、ある MDB が JMS 宛先 `1` をリスニングするのみでなく、JMS 宛先 `2` へのメッセージ送信も行う場合があります。
5. （矢印 5 および 6）Web サービス・クライアントが、SOAP リクエスト送信の結果として、JMS エンドポイント Web サービス上で `receive` 操作を実行し、メッセージを受信します。JMS エンドポイント Web サービスが JMS 宛先からのメッセージを使用し、SOAP レスポンス・メッセージに入れ、送信対象となるその SOAP レスポンス・メッセージをクライアントに渡します。

JMS エンドポイント Web サービスのアセンブル手順

次の手順では、WebServicesAssembler を使用して JMS エンドポイント Web サービスをアセンブルする方法について説明します。

1. `jmsAssemble` コマンドまたは Ant タスクを使用して WebServicesAssembler を実行し、Web サービスの EAR ファイルを生成します。

   `jmsAssemble` コマンドの場合は、JMS `send` 操作に対するコネクションを取得するために使用する JMS コネクション・ファクトリの JNDI 名（`sendConnectionFactoryLocation`）、またはすべての `send` 操作 JMS メッセージのデフォルトの返信先として使用する JMS コネクション・ファクトリの JNDI 名（`replyToConnectionFactoryLocation`）を指定する必要があります。jmsAssemble コマンドおよびその必須とオプションの引数の詳細は、18-16 ページの「jmsAssemble」を参照してください。

   次の例では、`jms/ws/mdb/theQueueConnectionFactory` は、JMS の `send` 操作用に JMS キューへのコネクションを生成する JMS コネクション・ファクトリの JNDI 名です。

   - `jms/ws/mdb/theQueue`: `send` 操作による SOAP メッセージ・ペイロードの送信先である JMS キューの JNDI 名。
   - `jms/ws/mdb/logQueueConnectionFactory`: 返信先キュー用に使用される JMS コネクション・ファクトリの JNDI 名。
   - `jms/ws/mdb/logQueue`: すべての send メッセージにデフォルトの返信先として設定される JMS キューの JNDI 名。この例では `linkReceiveWithReplyTo` 引数が有効であるため、この返信先は、JMS エンドポイント Web サービスの `receive` 操作でのメッセージの取得にも使用されます。

   このコマンドによって生成される J2EE の EAR ファイルには、WSDL や生成されるファイル `web.xml` ファイルなど、JMS エンドポイント Web サービスの構成情報が含まれます。

   コマンドライン：

```
java -jar wsa.jar -jmsAssemble
     -sendConnectionFactoryLocation jms/ws/mdb/theQueueConnectionFactory
     -sendQueueLocation jms/ws/mdb/theQueue
     -replyToConnectionFactoryLocation jms/ws/mdb/logQueueConnectionFactory
     -replyToQueueLocation jms/ws/mdb/logQueue
     -linkReceiveWithReplyTo true
     -targetNamespace http://oracle.j2ee.ws/jms-doc
     -typeNamespace http://oracle.j2ee.ws/jms-doc/types
     -serviceName JmsService
     -appName jms_service
     -context jms_service
     -input ./build/mdb_service.jar
     -uri JmsService
     -output ./dist
```

   Ant タスク：

```
<oracle:jmsAssemble
         linkReceiveWithReplyTo="true"
         targetNamespace="http://oracle.j2ee.ws/jms-doc"
         typeNamespace="http://oracle.j2ee.ws/jms-doc/types"
         serviceName="JmsService"
         appName="jms_service"
         context="jms_service"
         input="./build/mdb_service.jar"
         uri="JmsService"
         output="./dist"
         sendConnectionFactoryLocation="jms/ws/mdb/theQueueConnectionFactory"
         sendQueueLocation="jms/ws/mdb/theQueue"
         replyToConnectionFactoryLocation="jms/ws/mdb/logQueueConnectionFactory"
         replyToQueueLocation="jms/ws/mdb/logQueue"/>
```

このコマンドおよび Ant タスクの説明 :

- `jmsAssemble`: JMS 宛先（キューまたはトピック）を Web サービスとして公開します。18-16 ページの「jmsAssemble」を参照してください。
- `sendConnectionFactoryLocation`: JMS `send` 操作のためのコネクションの取得に使用する JMS `ConnectionFactory` の JNDI 名を指定します。`ConnectionFactory` のタイプは、`send` 宛先との整合性が必要とされます。18-59 ページの「sendConnectionFactoryLocation」を参照してください。
- `sendQueueLocation`: JMS `send` 操作に使用する JMS キューの JNDI 名を指定します。デフォルト値はありません。18-60 ページの「sendQueueLocation」を参照してください。
- `replyToConnectionFactoryLocation`: すべての `send` 操作 JMS メッセージのデフォルトの返信先として使用する JMS コネクション・ファクトリの JNDI 名を指定します。`ConnectionFactory` のタイプは、返信先との整合性が必要とされます。18-59 ページの「replyToConnectionFactoryLocation」を参照してください。
- `replyToQueueLocation`: すべての `send` 操作 JMS メッセージのデフォルトの返信先として使用する JMS キューの JNDI 名を指定します。18-59 ページの「replyToQueueLocation」を参照してください。
- `linkReceivedWithReplyTo`: `receive` 操作を返信先にリンクするかどうかを決定します。18-58 ページの「linkReceiveWithReplyTo」を参照してください。
- `targetNamepace`: 生成される WSDL で使用するターゲット名前空間を指定します。この値は、仕様に準拠している HTTP URL、準拠していない HTTP URL、または URI であってもかまいません。18-66 ページの「targetNamespace」を参照してください。
- `typeNamespace`: 生成される WSDL 内のスキーマ型に使用する型名前空間を指定します。指定した名前は必ず使用され、破棄されません。18-67 ページの「typeNamespace」を参照してください。
- `serviceName`: サービス名を指定します。18-48 ページの「serviceName」を参照してください。
- `appName`: アプリケーションの名前を指定します。この名前は、通常 `context` や `uri` などの他の引数のベース値として使用されます。18-40 ページの「appName」を参照してください。
- `context`: Web アプリケーションのルート・コンテキストを指定します。18-62 ページの「context」を参照してください。
- `input`: `WEB-INF/classes` にコピーされるクラスを格納する JAR またはディレクトリを指定します。この引数は、WebServicesAssembler によって使用されるクラスパスに追加されます。18-45 ページの「input」を参照してください。
- `uri`: Web サービスに使用する URI を指定します。18-63 ページの「uri」を参照してください。
- `output`: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。

2. すべての JMS 宛先をデプロイします。
3. サービスをデプロイし、アプリケーションをバインドします。

   通常の方法で EAR ファイルを OC4J の実行中インスタンスにデプロイします。EAR ファイルのデプロイの詳細は、第 19 章「Web サービスのパッケージ化およびデプロイ」および『Oracle Containers for J2EE デプロイメント・ガイド』を参照してください。次は、デプロイメント・コマンドのサンプルです。

```
java -jar <OC4J_HOME>/j2ee/home/admin_client.jar deployer:oc4j:localhost:port
<user> <password>
            -deploy
            -file dist/jms_service.ear
            -deploymentName jms_service
            -bindWebApp default-web-site
```

次のリストは、このコード例のパラメータの説明です。

- *<oc4jHome>*: OC4J インストールが含まれているディレクトリ。
- *<user>*: OC4J のインスタンスのユーザー名。ユーザー名はインストール時に割り当てられます。
- *<password>*: OC4J のインスタンス用のパスワード。パスワードはインストール時に割り当てられます。
- `default-web-site`: アプリケーションのバインド先の Web サイト。通常、これは `default-web-site` になります。Web サイトを構成するには、*<OC4J_HOME>*`/j2ee/home/config` の `server.xml` ファイルを参照してください。

4. （オプション）デプロイが成功したことを確認します。

   OracleAS Web Services では、デプロイされた Web サービスごとに Web サービス・テスト・ページが提供されます。Web サービス・テスト・ページへのアクセスおよびその使用方法の詳細は、13-7 ページの「Web サービス・テスト・ページの使用方法」を参照してください。

5. クライアント・サイド・コードを生成します。

   JMS エンドポイント Web サービスの WSDL からクライアント・サイドのプロキシを生成する方法と、その他の WSDL からプロキシを生成する方法は同じです。JMS エンドポイント Web サービスの WSDL は、WS-I Basic Profile 1.0 に準拠した WSDL ツールで使用できますので、相互運用性があります。たとえば、.NET WSDL ツールを使用して C# クライアント・スタブを生成することにより、Oracle JMS エンドポイント Web サービスと通信できます。

   - J2SE 環境の場合、`genProxy` コマンドを使用して WebServicesAssembler ツールを実行し、J2SE Web サービス・クライアントのスタブ（クライアント・プロキシ）を生成します。J2SE 環境向けのクライアント・サイド・コードの生成およびアセンブルの詳細は、第 15 章「J2SE Web サービス・クライアントのアセンブル」を参照してください。
   - J2EE 環境の場合、`genInterface` コマンドを使用して WebServicesAssembler ツールを実行し、J2EE Web サービス・クライアント用のマッピング・ファイルおよびサービス・エンドポイント・インタフェースを生成します。クライアント・サイド・コードの生成およびアセンブルの詳細は、第 14 章「J2EE Web サービス・クライアントのアセンブル」を参照してください。

   たとえば、次のコマンドは、J2SE クライアントで使用可能なスタブを生成します。

   コマンドライン：

```
java -jar wsa.jar -genProxy
                  -output build/src/client/
                  -wsdl http://localhost:8888/hello/JmsService?WSDL
                  -packageName oracle.demo.jms_service
```

   Ant タスク：

```
<oracle:genProxy  wsdl="http://localhost:8888/hello/JmsService?WSDL"
          output="build/src/client"
          packageName="oracle.demo.jms_service"
/>
```

   このコマンドラインおよび Ant タスクの説明：

   - `genProxy`: J2SE Web サービス・クライアントから使用可能な静的プロキシ・スタブを作成します。18-32 ページの「genProxy」を参照してください。
   - `output`: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
   - `wsdl`: WSDL ドキュメントの絶対ファイル・パス、相対ファイル・パスまたは URL を指定します。18-64 ページの「wsdl」を参照してください。

- `packageName`: JAX-RPC マッピング・ファイルにパッケージ名が宣言されていない場合に、生成されたクラス用に使用されるパッケージ名を指定します。18-47 ページの「packageName」を参照してください。

このコマンドは、クライアント・プロキシを生成し、これらをディレクトリ `build/src/client` に格納します。クライアント・アプリケーションはこのスタブを使用して、リモート・サービスでの操作を起動します。`genProxy` の必須およびオプションの引数の詳細は、18-32 ページの「genProxy」を参照してください。

6. クライアントをコンパイルおよび実行します。

   クライアントをコンパイルする前に、クラスパス上にある適切な JAR をリストします。クライアントのクラスパスで使用可能なすべての JAR ファイルのリストについては、表 A-2「クライアント・サイド・プロキシを使用するクライアントのクラスパス・コンポーネント」を参照してください。個々の JAR をリストするかわりに、クライアントのクラスパスにクライアント側の JAR として `wsclient_extended.jar` を含めることができます。この JAR ファイルには、Web サービス・クライアントをコンパイルおよび実行するために必要なすべてのクラスが含まれます。これらのクラスは、表 A-2 にリストされている各 JAR ファイルに含まれているものです。`wsclient_extended.jar` およびクライアントのクラスパスの詳細は、A-3 ページの「Web サービス・プロキシのクラスパスの設定」を参照してください。

メッセージ処理と応答メッセージ

JMS エンドポイント Web サービスは、受信した SOAP メッセージを処理し、ペイロード（SOAP メッセージの `body` 要素）を JMS 宛先に格納します。この項では、JMS エンドポイント Web サービスからの JMS メッセージを使用および処理するために、開発者が知る必要のある詳細を説明します。

JMS エンドポイント Web サービスに関連する JMS メッセージの内容には、`javax.xml.soap.SOAPElement`（これは `org.w3c.dom.Element` のサブクラスでもあります）のインスタンスか、または XML ペイロードの文字列表現である `java.lang.String` のいずれかがなることができます。JMS エンドポイント Web サービスでは、JMS 宛先にメッセージを置く前に、特定の JMS メッセージ・ヘッダー値を設定できます。JMS エンドポイント Web サービスのアセンブル時に指定したオプションの構成引数の値に応じて、JMS エンドポイント Web サービスにより次の JMS メッセージ・ヘッダーが設定されます。

```
JMSType
JMSReplyTo
JMSExpiration
JMSPriority
JMSDeliveryMode
```

JMS エンドポイント Web サービスは、`replyToTopicLocation` 引数または `replyToQueueLocation` 引数で指定された値を、`JMSReplyTo` ヘッダーに設定します（これらのいずれかが、その JMS エンドポイント Web サービスに対して構成済になっている必要があります）。`replyToConnectionFactoryLocation` 引数で指定した値は、メッセージに標準文字列プロパティとして設定されます。プロパティ名は `OC4J_REPLY_TO_FACTORY_NAME` です。

例 9-1 に、onMessage() メソッドが JMS エンドポイント Web サービスの send 操作で生成されたメッセージの reply-to 情報を取得することを示すコード部分を示します。

例 9-1 送信操作によって生成されたメッセージの返信先情報の取得

```
...
public void onMessage(Message inMessage) {
    // Do some processing
    ObjectMessage msg = null;
    String  factoryName;
    Destination dest;
    Element  el;
    try {
      // Message should be of type objectMessage
      if (inMessage instanceof ObjectMessage) {
        // retrieve the object
        msg = (ObjectMessage) inMessage;
        el = (Element)msg.getObject();
        System.out.println("MessageBean2::onMessage() => Message received: " );
        ((XMLElement)el).print(System.out);
        processElement(el);
        factoryName = inMessage.getStringProperty("OC4J_REPLY_TO_FACTORY_NAME");
        dest = inMessage.getJMSReplyTo();
...
```

制限事項

F-8 ページの「JMS 宛先を使用した Web サービスのアセンブル」を参照してください。

追加情報

詳細は、次を参照してください。

- テスト・ページを使用した Web サービス・デプロイのテスト方法は、第 13 章「Web サービス・デプロイのテスト」を参照してください。
- J2EE Web サービス・クライアントの構築方法は、第 14 章「J2EE Web サービス・クライアントのアセンブル」を参照してください。
- J2SE Web サービス・クライアントの構築方法は、第 15 章「J2SE Web サービス・クライアントのアセンブル」を参照してください。
- JAX-RPC ハンドラの詳細は、第 16 章「JAX-RPC ハンドラの使用方法」を参照してください。
- WebServicesAssembler ツールを使用した Web サービスのアセンブル方法は、第 18 章「WebServicesAssembler の使用方法」を参照してください。
- Web サービスのパッケージ化およびデプロイ方法は、第 19 章「Web サービスのパッケージ化およびデプロイ」を参照してください。
- クライアントのアセンブルに必要な JAR ファイルの詳細は、付録 A「Web サービス・クライアントの API および JAR」を参照してください。
- Web サービスの相互運用性の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「相互運用可能な Web サービスの実現」を参照してください。
- Web サービス・クライアントにおけるサービスのクオリティ機能の使用方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの管理」を参照してください。
- Web サービスへのセキュリティの追加方法は、『Oracle Application Server Web Services セキュリティ・ガイド』を参照してください。

- Web サービスへの信頼性の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの信頼性の確保」を参照してください。
- Web サービスへの監査およびロギング構成の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「監査メッセージおよびロギング・メッセージ」を参照してください。
- 標準以外のデータ型の処理の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 値タイプのカスタム・シリアライズ」を参照してください。
- JAX-RPC マッピング・ファイルの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「JAX-RPC マッピング・ファイル記述子」を参照してください。
- OracleAS Web Services でサポートされているデータ型の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 型の XML および WSDL 型へのマッピング」を参照してください。
- Web サービス開発用の Oracle JDeveloper ツールのサポートの詳細は、Oracle JDeveloper のオンライン・ヘルプを参照してください。

10

データベース Web サービスのアセンブル

この章では、Oracle データベースと対話できる Oracle Application Server Web Services を開発する方法について説明します。OracleAS Web Services がデータベースと対話するには、コールインおよびコールアウトの 2 つの方法があります。Web サービスのコールインとは、データベース・リソースにアクセスする Web サービスを提供することです。Web サービスは OC4J インスタンス上で動作し、JDBC を介してデータベース・リソースにアクセスします。

Web サービスのコールアウトとは、データベース内から外部 Web サービスを起動することです。起動は、データベース内で実行される SQL、PL/SQL または Java コードから行われます。

この章の項目は、次のとおりです。

データベース Web サービスの概要

Web サービスのテクノロジを使用すると、Web を介することで、プラットフォーム、言語またはデータ・フォーマットにかかわらず、アプリケーション間の対話が可能になります。XML、SOAP、WSDL、UDDI、WS-Security および WS-Reliability などの主要要素は、ソフトウェア業界全体で採用されています。Web サービス・テクノロジは、通常中間層のアプリケーション・サーバーに実装およびデプロイされるサービスを指しています。ただし、異機種間で接続されていない環境では、ストアド・プロシージャ、データおよびメタデータへの Web サービス・インタフェース経由でのアクセス要求が増大しています。データベース Web サービス・テクノロジは、Web サービスに対するデータベースからのアプローチです。アプローチには 2 方向あります。

- データベース・リソースへの Web サービスとしてのアクセス（データベースのコールイン）
- データベース自身からの外部 Web サービスの使用（データベースのコールアウト）

データベースのコールイン

Oracle データベースを Web サービス・プロバイダとして使用することで、Java ストアド・プロシージャ、PL/SQL パッケージ、アドバンスト・キュー、事前定義済 SQL 問合せおよび DML への投資を有効活用できます。クライアント・アプリケーションは、標準 Web サービス・プロトコルを使用して、Oracle データベースへのデータの問合せおよび取得、ならびにストアド・プロシージャの起動ができます。Oracle 固有のデータベース接続プロトコルへの依存性はありません。アプリケーションはキャッシュされている任意の OC4J 接続を使用できます。このアプローチは異機種間で接続されていない分散環境では特に有効です。

データベース Web サービスは OracleAS Web Services の一部であるため、整合性のある均一な開発 / デプロイ環境に統合できます。Web サービスで公開されたデータベースと Web サービス・クライアントの間で交換されるメッセージを使用することで、OracleAS Web Services が提供するセキュリティ、信頼性、監査およびロギングなどの管理機能をすべて活用できます。

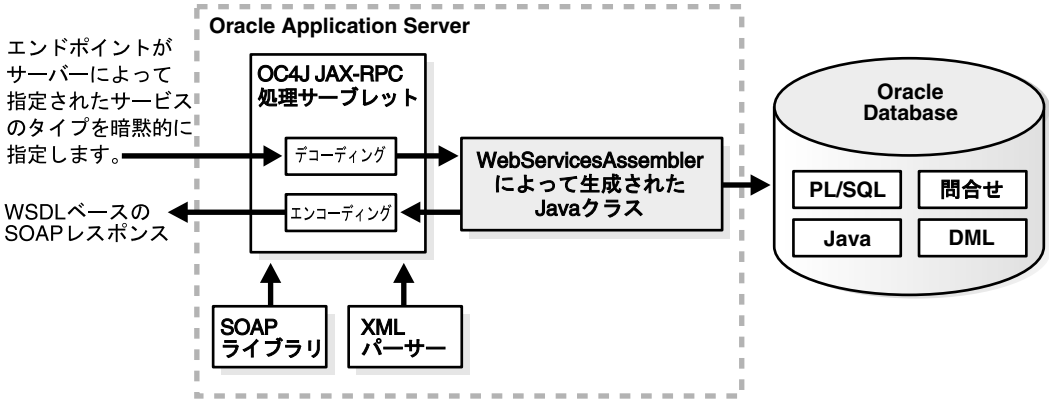
図 10-1 は Web サービスのコールインの説明図です。次の各ステップは、そのプロセスの説明です。

1. データベース・サービスのタイプを問うリクエストがアプリケーション・サーバーに受信されます。サービス・エンドポイントが、リクエストされたサービスのタイプを暗黙的に指定します。
2. OC4J JAX-RPC 処理サーブレットは SOAP ライブラリおよび XML パーサーを参照し、リクエストをデコードします。
3. サーブレットは、公開対象データベース操作に対応するように WebServicesAssembler によって生成されたクラスに、リクエストを渡します。WebServicesAssembler はこれらのクラスを、Oracle JPublisher を暗黙的にコールして生成します。生成されたクラスは、データベースの PL/SQL パッケージ、問合せ、DML、AQ ストリームまたは Java クラスを表すことができます。
4. データベースはレスポンスを OC4J JAX-RPC 処理サーブレットに渡します。このサーブレットは SOAP ライブラリおよび XML パーサーを参照してレスポンスをエンコードします。
5. WSDL に従って生成された SOAP レスポンスがクライアントに戻されます。

関連項目：

PL/SQL パッケージ、SQL 問合せ、DML 文、Oracle AQ ストリームまたはサーバー・サイド Java クラスのデータベース操作の Web サービスとしての公開の詳細は、10-7 ページの「データベース・リソースを公開する Web サービスの開発」を参照してください。

図 10-1 Web サービスによるデータベースへのコールイン



データベースのコールアウト

Web サービスが組み込まれるように、リレーショナル・データベースの格納、索引付けおよび検索機能を拡張することができます。データベースから Web サービスをコールし、株価、為替レートまたは天気情報などのオンデマンドで生成される動的なデータの追跡、集約、リフレッシュおよび問合せが可能です。サービス・コンシューマとしてのデータベースの使用例として、複数のサプライヤからの在庫情報の取得およびローカルの在庫データベースの更新を行う、事前定義済データベース・ジョブからの外部 Web サービスのコールがあげられます。もう 1 つの例として、Web クローラがあります。たとえば、複数の仕入先の製品 / 価格情報を照合するようにスケジュールされたデータベース・ジョブがそれです。

図 10-2 はデータベースのコールアウトの説明図です。

- SQL および PL/SQL コールの仕様 : （Oracle JPublisher で生成された）ユーザー定義のファンクション・コールを使用して、SQL 文またはビュー内から直接、または変数を介して Web サービスを起動します。
- `UTL_DBWS` PL/SQL パッケージを使用した、動的な Web サービスの起動。`Call` オブジェクトは WSDL をベースにして動的に作成でき、作成後に Web サービス操作を起動できます。

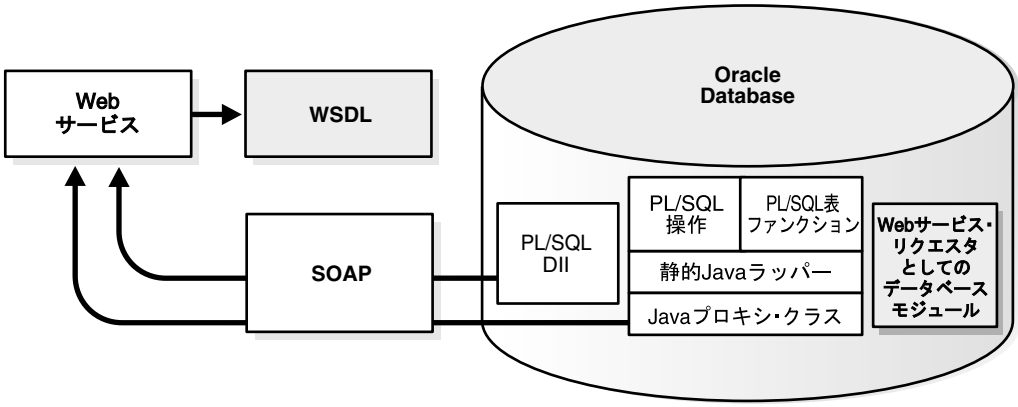
  『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』には、`UTL_DBWS` PL/SQL パッケージの使用方法についての詳細な説明があります。
- Pure Java 静的プロキシ・クラス : Oracle JPublisher を使用して、JAX-RPC を使用するクライアント・プロキシ・クラスを事前生成します。このメソッドを使用すると、Web サービスを UDDI レジストリで検索しなくてもその位置があらかじめ特定されているため、Web サービスの起動が容易になります。クライアント・プロキシ・クラスは、パラメータのマーシャリングおよびアンマーシャリングを含め、SOAP リクエストを構築するためのすべての作業を行います。
- Dynamic Invocation Interface（DII）over JAX-RPC を使用する Pure Java: 動的起動では、クライアント・プロキシを使用せずに SOAP リクエストを作成し、サービスにアクセスできます。

使用するメソッドは、起動元が SQL または PL/SQL か、それとも Java クラスであるかによって異なり、また、サービスが事前に認識されているか（静的起動）または実行時に初めて認識されるか（DII）によっても異なります。

関連項目：

OracleAS Web Services が提供するデータベースからの PL/SQL および Java コールアウトに対するサポートの詳細は、10-36 ページの「データベースにおける Web サービス・クライアントの開発」を参照してください。

図 10-2 データベース内からの Web サービスのコール



SQL と XML の間の型マッピング

次の項では、Web サービスが事前に認識されている場合（静的起動）の、コールインおよびコールアウトで使用する SQL と XML の間の型マッピングについて説明します。

- Web サービスのコールインで使用される SQL から XML への型マッピング
- Web サービスのコールアウトで使用される XML から SQL への型マッピング

Web サービスを実行時に認識させる場合は、Dynamic Invocation Interface（DII）または UTL_DBWS PL/SQL パッケージのみを使用できます。JAX-RPC DII の使用方法の詳細は、次の Web アドレスの API の記述を参照してください。

```
http://java.sun.com/j2ee/1.4/docs/#api
```

関連資料：

UTL_DBWS パッケージの使用方法の詳細は、『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

Web サービスのコールインで使用される SQL から XML への型マッピング

データベース Web サービスのコールインでは、PL/SQL ストアド・プロシージャや SQL 文などの SQL 操作は、1 つ以上の Web サービス操作にマッピングされます。SQL 操作のパラメータは SQL 型から XML 型にマッピングされます。

注意： 1 つ以上の操作にマッピングされる可能性がある理由は、OracleAS Web Services が SQL 値に対する XML データ表現の選択候補（SQL 結果セットの異なる表現など）を追加で提供する場合があるためです。

表 10-1 は、Web サービスのコールインで使用される SQL から XML への型マッピングを示します。最初の列に記載されているのは SQL 型です。表の 2 番目の列 **XML 型（literal）**は、`use` 属性のデフォルトである `literal` 値に関する SQL から XML への型マッピングを示します。3 番目の列 **XML 型（encoded）**は、`use` 属性の `encoded` 値に関するマッピングを示します。`literal` 値および `encoded` 値は、SOAP メッセージ・ボディのエンコーディング・ルールのことです。

関連項目：

SOAP メッセージ・ボディのエンコーディング・ルールの詳細は、5-3 ページの「use="literal" および use="encoded"」を参照してください。

表 10-1 Web サービスのコールインで使用される SQL から XML への型マッピング

SQL 型	XML 型（literal）	XML 型（encoded）
INT	int	int
INTEGER	int	int
FLOAT	double	double
NUMBER	decimal	decimal
VARCHAR2	string	string
DATE	dateTime	dateTime
TIMESTAMP	dateTime	dateTime
BLOB	byte[]	byte[]
CLOB	String	String
LONG	String	String
RAW	byte[]	byte[]
プリミティブ PL/SQL 索引付き表	Array	Array
PL/SQL Boolean	boolean	boolean
PL/SQL 索引付き表	complexType	complexType
PL/SQL レコード	complexType	complexType
REF CURSOR（*name*Beans）	Array	Array
REF CURSOR（*name*XML）	any	text_xml
REF CURSOR（*name*XMLRowSet）	swaRef	text_xml
SQL オブジェクト	complexType	complexType
SQL 表	complexType	complexType
SYS.XMLTYPE	any	text_xml

注意： National Language Support（「NLS」または「グローバリゼーション・サポート」とも呼ばれます）文字が `SQL SYS.XMLTYPE` の値で使用されている場合は、正しく処理されない場合があります。

REF CURSOR を戻す問合せまたは PL/SQL ファンクションは、*name*Beans、*name*XMLRowSet および *name*XML の 3 つのメソッドにマッピングされます。*name* は問合せまたは PL/SQL ファンクションの名前です。

- *name*Beans: このメソッドは配列を戻します。各要素は XSD 複合型のインスタンスであり、カーソルの 1 つの行を表します。複合型のサブ要素はその行の列に対応します。
- *name*XMLRowSet: このメソッドは、XML 形式の OracleWebRowSet インスタンスを含んだ swaRef または text_xml レスポンスを戻します。swaRef MIME 形式の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の MIME 添付ファイルの処理に関する項を参照してください。
- *name*XML: このメソッドは、Oracle XDB 行セットを含んだ XML any または text_xml のレスポンスを戻します。

OUT および IN OUT PL/SQL パラメータの両方が、WSDL ファイルの IN OUT パラメータにマッピングされます。

関連項目：

- REF CURSOR と同じ型の結果セットを戻す例は、10-15 ページの「SQL 問合せまたは DML 文から Web サービスをアセンブルする方法」を参照してください。
- PL/SQL パラメータを WSDL に生成してクライアント・コードからアクセスする方法の例は、10-14 ページの「クライアント・コードから PL/SQL の IN および IN OUT パラメータにアクセスする方法」を参照してください。

表 10-1 には、2 つの異なるマッピングが記載されている点に注意してください。use が literal の場合と encoded の場合です。デフォルトのマッピングは literal です。データベース Web サービスにとっては、encoded を使用する必要性は特にありません。encoded 用のマッピングが提供されているのは、use="encoded" 設定が必要になった場合に備えてです。特に指定のないかぎり、この章の説明では、use="literal" 設定を使用していることを前提にしています。

数値型に関しての SQL から XML へのマッピングの変更

表 10-1 は、コールインで使用する SQL から XML への型マッピングを定義しています。数値型のマッピングは、Oracle JPublisher による SQL 型の Java 型へのマッピング方法によって決定されています。Oracle JPublisher のオプション -numbertypes=objectjdbc が、WebServicesAssembler ツールによってデフォルトで使用されています。このため、SQL の数値型に対応する XML 型は、生成される WSDL ファイルではすべて NULL 値可能として宣言されます。

Oracle JPublisher のマッピングを変更し、それによって XML 型を変更するには、WebServicesAssembler の jpubProp 引数を使用できます。たとえば、次の引数をデータベースの WebServicesAssembler の Ant タスクで指定すると、SQL の数値型に対して生成される XML 型は、NULL 値可能として宣言されることはなくなります。

```
jpubProp="numbertypes=jdbc"
```

一方で、oracle または bigdecimal を numbertypes オプションのターゲットとして指定すると、SQL 数値型に対して生成される XML 型はすべて decimal となり、NULL 値可能になります。

Web サービスのコールアウトで使用される XML から SQL への型マッピング

データベース Web サービスのコールアウトでは、XML 型が SQL 型にマッピングされます。表 10-2 に、コールアウトで使用される XML から SQL への型マッピングを示します。

表 10-2 Web サービスのコールアウトで使用される XML から SQL への型マッピング

XML 型	SQL 型
int	NUMBER
float	NUMBER
double	NUMBER
decimal	NUMBER
dateTime	DATE
String	VARCHAR2
byte[]	RAW
complexType	SQL OBJECT
Array	SQL TABLE
text_xml	XMLType

データベース・リソースを公開する Web サービスの開発

この項では、PL/SQL ストアド・プロシージャ、SQL 文、Oracle Streams AQ キューおよびサーバー・サイド Java クラスとして実装される Web サービスの開発方法について説明します。

- Web サービスのコールインのライフ・サイクルの使用方法
- Web サービスのコールインに対する WebServicesAssembler のサポート
- PL/SQL パッケージからの Web サービスのアセンブル方法
- SQL 問合せまたは DML 文から Web サービスをアセンブルする方法
- Oracle Streams AQ から Web サービスをアセンブルする方法
- サーバー・サイド Java クラスを Web サービスとしてアセンブルする方法

Web サービスのコールインのライフ・サイクルの使用方法

データベース Web サービスのコールイン・アプリケーションの作成は、ボトムアップ・プロセスです。多くの場合、既存のデータベース・アプリケーション（PL/SQL パッケージまたは Java アプリケーションなど）または操作スクリプト（SQL 問合せ、DML または AQ など）の再利用を考えるはずです。また、Web サービスとして公開するリソースをデータベースに移入することもできます。Web サービスのコールインは、通常次の各ステップを取ります。

1. 公開するデータベース・リソースを決定し、使用可能にします。

   たとえば、リソースは次のいずれかの方法で提供できます。

   - PL/SQL パッケージのデータベースへのロード
   - SQL 問合せまたは DML 文 Web サービスで使用するスキーマの作成
   - Java クラスのデータベースへのロード（データベースのサーバー・サイド Java Web サービス用）

2. WebServicesAssembler ツールを実行し、指定されたリソースをベースにして Web サービスをアセンブルします。

   なお、Oracle JDeveloper を使用しても Web サービスをアセンブルできます。

3. Web サービスのアセンブルで PL/SQL ラッパーが生成された場合は、データベースにロードします。

   PL/SQL レコードまたは表タイプ `INDEX BY` が PL/SQL パッケージに含まれている場合、PL/SQL Web サービスをアセンブルすると、ラッパーが生成されます。生成されたラッパーはデータベースにロードする必要があります。サーバー・サイド Java のコールインをアセンブルすると、常に PL/SQL ラッパーが生成されます。`sysUser` 引数が設定されている場合、WebServicesAssembler はラッパーを自動的にロードします。詳細は、18-55 ページの sysUser を参照してください。

   なお、Oracle JDeveloper を使用しても PL/SQL ラッパーをデータベースにロードできます。

4. OC4J データソースを構成し、Web サービス実装を構成する Oracle JPublisher 生成の Java クラスが、データベースおよび公開するリソースに確実に接続できるようにします。

   J2EE データソース・ファイルにデータソース・エントリを追加し、Web サービス・アプリケーションがデータベースに接続できるようにします。

5. Web サービス・アプリケーションを OC4J の実行中インスタンスにデプロイします。

6. 必要に応じて、デプロイメントが成功したかどうかを確認します。OracleAS Web Services では、デプロイされた Web サービスごとに Web サービス・テスト・ページが提供されます。

   Web サービス・テスト・ページへのアクセスおよびその使用方法の詳細は、13-7 ページの「Web サービス・テスト・ページの使用方法」を参照してください。

7. WebServicesAssembler ツールを使用して Web サービス・クライアント・プロキシを生成し、クライアント・アプリケーションに組み込みます。

8. クライアント・プロキシを使用して Web サービスを起動します。

Web サービスのコールインに対する WebServicesAssembler のサポート

WebServicesAssembler ツールの次の引数は、Web サービスのコールイン・タイプのすべてに対して必須です。

- appName: アプリケーション名
- dataSource: 実行時に使用されるデータソース JNDI 名
- dbConnection: コード生成時に使用されるデータベース接続 URL
- dbUser: コード生成時に使用されるデータベース・ユーザーおよびパスワード

`appName` 引数には、Web サービス・アプリケーション名を指定します。`dataSource` 引数は、アクセス対象のデータベース用に、データソースの JNDI 位置を定義します。実行時には、Web サービスのコードは、このデータソースを介してデータベースにアクセスします。

WebServicesAssembler ツールは、Web サービス作成時に、`dbConnection` および `dbUser` 引数を使用してデータベースへの接続を定義します。これらの引数の値はコード生成時に使用され、実行時には使用されません。`dbConnection` 引数は、公開対象のリソースに関する情報へのアクセスに使用します。たとえば、PL/SQL パッケージ、スキーマまたは問合せに関する情報へのアクセスに使用します。

Web サービスの作成時に使用するデータベースは、実行時に使用するデータベースと同じである必要はありません。ただし、どちらにも公開対象のデータベース・リソースを含むスキーマがある必要があります。

実行時には、データベース Web サービスはこのデータソースからデータベースへの接続を取得します。データベース Web サービスは、接続が失われるとデータベースへの再接続を行います。接続の状態が無効になると、Web サービスは接続の再確立を試行します。Web サービスは、再接続に失敗すると、フォルトを戻します。その次の回にクライアントが Web サービスを起動すると、Web サービスはデータベースへの接続を試行します。したがって、接続失敗の処理が原因で Web サービスが一時的にフォルトを戻しても、その後正常に接続できる場合があります。

次の引数はオプションであり、Web サービスのすべてのコールインのケースで使用できます。

- context: Web アプリケーションのルート・コンテキスト
- debug: 詳細な診断メッセージを表示
- ear: 生成される EAR の名前および場所
- jpubProp: Oracle JPublisher コード生成を微調整するための Oracle JPublisher オプションを指定
- output: 生成されるファイルの格納場所
- portName: 生成される WSDL 内のポート名
- serviceName: 生成される WSDL 内のサービス名のローカル部分
- style: 生成される WSDL 内で使用されるメッセージ書式の `style` 部分
- uri: デプロイ・ディスクリプタで Web サービス用に使用される URI
- use: 生成される WSDL で使用されるメッセージ書式の `use` 部分

公開対象リソースがデータベースに移入されていることが、すべてのコールインタイプに共通の前提条件になります。WebServicesAssembler は Oracle JPublisher を使用し、データベース・リソースにアクセスする Java コードを生成します。`jpubProp` 引数は、コマンドラインまたは Ant タスクで複数回使用でき、Oracle JPublisher にオプションを渡します。

関連資料：

Oracle JPublisher オプションのリスト、および PL/SQL、SQL 型、SQL 文およびサーバー・サイド Java のクライアント・サイド Java ラッパーへのマッピング方法は、『Oracle Database JPublisher ユーザーズ・ガイド』を参照してください。

PL/SQL パッケージからの Web サービスのアセンブル方法

`plsqlAssemble` コマンドを使用して、Web サービスを PL/SQL ストアド・プロシージャからアセンブルします。生成される Web サービスでは、各 Web サービス操作が 1 つの PL/SQL ストアド・プロシージャに対応します。

この項の内容は、次のとおりです。

- 前提条件
- PL/SQL パッケージから Web サービスをアセンブルする手順
- サンプル PL/SQL パッケージ
- PL/SQL のファンクション名が Web サービスの操作名にマップされる方法
- クライアント・コードから PL/SQL の IN および IN OUT パラメータにアクセスする方法
- クライアント・コードから SQL の XMLType 型にアクセスする方法

前提条件

開始する前に、次のファイルと情報を用意してください。

- Web サービスとして公開する PL/SQL パッケージ

関連項目：

この例で使用しているストアド・プロシージャの詳細は、10-13 ページの「サンプル PL/SQL パッケージ」を参照してください。

- Web サービス・アプリケーションの名前
- JDBC データソース用の JNDI 位置
- JDBC データベースの接続 URL、ユーザー名およびパスワード

PL/SQL パッケージから Web サービスをアセンブルする手順

次の手順は、PL/SQL パッケージ echo_plsql に対する Web サービスのアセンブル方法について説明しています。

1. 「前提条件」の項で説明した PL/SQL パッケージおよび情報を WebServicesAssembler の plsqlAssemble コマンドに入力します。

   コマンドライン :

   ```
   java -jar wsa.jar
       -plsqlAssemble
       -appName Echo
       -sql echo_plsql
       -dataSource jdbc/OracleManagedDS
       -dbConnection jdbc:oracle:thin:@stacd15:1521:lsqlj
       -dbUser scott/tiger
       -style rpc
       -use encoded
   ```

   Ant タスク :

   ```
   <oracle:plsqlAssemble
       dbUser="scott/tiger"
       sql="echo_plsql"
       dbConnection="jdbc:oracle:thin:@stacd15:1521:1sqlj"
       dataSource="jdbc/OracleManagedDS"
       appName="EchoPlsql"
       style="rpc"
       use="encoded"
   />
   ```

   このコマンドおよび Ant タスクの説明 :

   - plsqlAssemble: ストアド・プロシージャとファンクションを含む PL/SQL パッケージから Web サービスを生成します。このコマンドを使用するには、データベースに接続する必要があります。18-17 ページの「plsqlAssemble」を参照してください。
   - appName: アプリケーションの名前を指定します。この名前は、通常 context や uri などの他の引数のベース値として使用されます。18-40 ページの「appName」を参照してください。
   - sql: PL/SQL のパッケージ名を指定します。18-54 ページの「sql」を参照してください。
   - dataSource: 実行時に Web サービスによって使用されるデータソースの JNDI 位置を指定します。18-53 ページの「dataSource」を参照してください。
   - dbConnection: データベースの JDBC URL を指定します。18-53 ページの「dbConnection」を参照してください。
   - dbUser: *user*/*password* の形式でデータベースのスキーマとパスワードを指定します。18-53 ページの「dbUser」を参照してください。
   - style: ボトムアップ方式 Web サービス・アセンブリの場合、この引数は、生成される WSDL 内のメッセージ書式の style 属性を指定します。18-68 ページの「style」を参照してください。
   - use: ボトムアップ方式 Web サービス・アセンブリの場合、この引数は、生成される WSDL 内のメッセージ書式の use 属性を指定します。18-68 ページの「use」を参照してください。

   plsqlAssemble コマンドで使用可能な引数は、18-17 ページの「plsqlAssemble」を参照してください。

   デフォルトでは、WebServicesAssembler は document-wrapped スタイルを使用してサービスを生成します。ただし、document-wrapped スタイルを使用する JAX-RPC クライアントは、IN OUT パラメータを直接サポートしません。WebServicesAssembler はそのかわりに、IN パラメータと IN OUT パラメータを個別にパッケージ化します。この例で使用され

ている PL/SQL パッケージは、IN OUT パラメータを含んでおり、plsqlAssemble コマンドには -style rpc 引数が含まれています。パラメータおよび各種ドキュメント・スタイルの詳細は、10-14 ページの「クライアント・コードから PL/SQL の IN および IN OUT パラメータにアクセスする方法」を参照してください。

このコマンドにより、Web サービス・アプリケーション EchoPlsql.ear が生成されるとともに、オプションで次の PL/SQL スクリプトが生成されます。

- Echo_plsql_wrapper.sql: PL/SQL レコードおよび INDEX BY 表をサポートするために生成される PL/SQL ラッパー
- Echo_plsql_dropper.sql: ラッパー・スクリプトで作成される型およびパッケージを削除する PL/SQL スクリプト

2. Web サービスの生成中に作成された PL/SQL ラッパーをすべてデータベースにインストールします。

   PL/SQL Web サービスのアセンブリでは PL/SQL ラッパーが生成されない場合もあります。生成された場合は、Web サービスの実行前にデータベースの適切なユーザー・スキーマにロードする必要があります。

   ラッパーは自動または手動でロードできます。ラッパーを自動でロードするには、次の行を plsqlAssemble コマンドに追加します。

   -jpubProp plsqload（コマンドラインの場合）、または

   jpubprop="plsqlload"（Ant タスクの場合）

   Web サービスのアセンブリ後にラッパー・パッケージを手動でロードするには、SQL*PLUS を使用します。次のコマンドラインは、ラッパー・パッケージをロードするためのサンプル SQL*PLUS コマンドです。

   ```
   SQL>@Echo_plsql_wrapper.sql
   ```

3. サービスを OC4J の実行中インスタンスにデプロイし、アプリケーションをバインドします。

   ステップ 1 の -dataSource 引数で参照しているデータソースがこの OC4J インスタンスに設定されている必要があります。

   次のコマンドラインは、サンプルのデプロイおよびバインドのコマンドです。

   ```
   % java –jar <J2EE_HOME>/admin_client.jar deployer:oc4j:localhost:port admin welcome
   –deploy -file dist/echo.ear –deploymentName echo
   ```

   ```
   % java –jar <J2EE_HOME>/admin_client.jar deployer:oc4j:localhost:port admin welcome
   –bindWebApp plsql plsql-web default-web-site /plsql
   ```

   この例において、<*J2EE_HOME*> は、J2EE のインストール・ディレクトリです。

   EAR ファイルのデプロイの詳細は、『Oracle Containers for J2EE デプロイメント・ガイド』を参照してください。

4. （オプション）デプロイが成功したことを確認します。

   OracleAS Web Services では、デプロイされた Web サービスごとに Web サービス・テスト・ページが提供されます。Web サービス・テスト・ページへのアクセスおよびその使用方法の詳細は、13-7 ページの「Web サービス・テスト・ページの使用方法」を参照してください。

5. クライアント・サイド・コードを生成します。

   - J2SE 環境の場合、genProxy コマンドを使用して WebServicesAssembler ツールを実行し、J2SE Web サービス・クライアントのスタブ（クライアント・プロキシ）を生成します。J2SE 環境向けのクライアント・サイド・コードの生成およびアセンブルの詳細は、第 15 章「J2SE Web サービス・クライアントのアセンブル」を参照してください。
   - J2EE 環境の場合、genInterface コマンドを使用して WebServicesAssembler ツールを実行し、J2EE Web サービス・クライアント用の JAX-RPC マッピング・ファイルおよびサービス・エンドポイント・インタフェースを生成します。クライアント・サイ

ド・コードの生成およびアセンブルの詳細は、第 14 章「J2EE Web サービス・クライアントのアセンブル」を参照してください。

たとえば、次のコマンドは genProxy コマンドを使用して J2SE クライアントを build/classes/client ディレクトリに生成します。

```
% java -jar wsa.jar -genProxy
                    -wsdl http://locahost:8888/plsql/echo?WSDL
                    -output build/src/client
                    -mappingFileName ./mapping.xml
                    -packageName oracle.demo.db.plsql.stub
                    -unwrapParameters true
```

このコマンドの説明 :

- genProxy: J2SE Web サービス・クライアントから使用可能な静的プロキシ・スタブを作成します。18-32 ページの「genProxy」を参照してください。
- wsdl: WSDL ドキュメントの絶対ファイル・パス、相対ファイル・パスまたは URL を指定します。18-64 ページの「wsdl」を参照してください。
- output: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
- mappingFileName: JAX-RPC マッピング・ファイルを指すファイル位置を指定します。18-46 ページの「mappingFileName」を参照してください。
- packageName: JAX-RPC マッピング・ファイルにパッケージ名が宣言されていない場合に、生成されたクラス用に使用されるパッケージ名を指定します。18-47 ページの「packageName」を参照してください。
- unwrapParameters: この引数は document-literal 操作に対してのみ設定でき、その他のメッセージ書式では無視されます。unwrapParameters を true に設定すると (デフォルト)、戻り型とレスポンス・タイプがアンラップされます。通常はこの方が使いやすくなります。特にタイプが単純な場合はそうです。18-69 ページの「unwrapParameters」を参照してください。

6. クライアントをコンパイルおよび実行します。

クライアントをコンパイルする前に、クラスパス上にある適切な JAR をリストします。クライアントのクラスパスで使用可能なすべての JAR ファイルのリストについては、表 A-2「クライアント・サイド・プロキシを使用するクライアントのクラスパス・コンポーネント」を参照してください。個々の JAR をリストするかわりに、クライアントのクラスパスにクライアント側の JAR として wsclient_extended.jar を含めることができます。この JAR ファイルには、Web サービス・クライアントをコンパイルおよび実行するために必要なすべてのクラスが含まれます。これらのクラスは、表 A-2 にリストされている各 JAR ファイルに含まれているものです。wsclient_extended.jar およびクライアントのクラスパスの詳細は、A-3 ページの「Web サービス・プロキシのクラスパスの設定」を参照してください。

次のコマンドラインは、サンプルのコンパイルおよび実行のコマンドです。

```
 % javac -classpath path:
         <ORACLE_HOME>/webservices/lib/wsclient_extended.jar:
         :<ORACLE_HOME>/webservices/lib/jaxrpc-api.jar EchoClient.java


% java -classpath path:
       <ORACLE_HOME>/webservices/lib/wsclient_extended.jar:
       <ORACLE_HOME>/webservices/lib/jaxrpc-api.jar:
       <J2EE_HOME>/lib/jax-qname-namespace.jar:
       <J2EE_HOME>/lib/activation.jar:
       <J2EE_HOME>/lib/mail.jar:
       <J2EE_HOME>/lib/http_client.jar:
       <ORACLE_HOME>/lib/xmlparserv2.jar EchoClient
```

この例において、<J2EE_HOME> は J2EE のインストール・ディレクトリであり、<ORACLE_HOME> に OracleAS Web Services がインストールされています。

サンプル PL/SQL パッケージ

例 10-1 は、Web サービスとして公開可能な、データベース内の PL/SQL パッケージを示しています。このパッケージには、各種 SQL および PL/SQL データ型を使用するプロシージャおよびファンクションが含まれています。

例 10-1 サンプル PL/SQL パッケージ

```
create or replace type address as object(
     street varchar2(30), city varchar2(30), state varchar2(2), zip varchar2(5));
create or replace type employee as object (eid int, efirstname varchar(30),
     elastname varchar(30), addr address, salary float);
create table employees (eid int, emp employee);
create table employee_espp (eid int, status int);
create table employee_accounts (eid int, accounts sys.xmltype);
create table employee_biodata (eid int, biodata CLOB);
create table employee_image (eid int, image BLOB);
create table employee_hiredate(eid int, hiredate TIMESTAMP);

create or replace package echo_plsql as
procedure set_object (emp IN employee);
function get_object1(id IN int) return employee;
function get_object2(id IN int) return address;
function hold_varchar(id IN int, firstname OUT varchar2, lastname OUT varchar2) return
float;
procedure set_boolean(id IN int , status IN boolean);
function get_boolean(id IN int) return boolean;
procedure hold_float_inout(id IN int, newsalary IN OUT float);
procedure clear_object (id IN int);
procedure set_clob (id int, biodata IN CLOB);
function get_clob(id IN int) return CLOB;
procedure set_blob(id int, image IN BLOB);
function get_blob(id IN int) return BLOB;
procedure set_xmltype(id IN number, accounts sys.xmltype);
function get_xmltype(id IN number) return sys.xmltype;
procedure set_date(id IN int, hiredate IN TIMESTAMP);
function get_date(id IN int) return TIMESTAMP;
TYPE rec is RECORD (emp_id int, manager_id int);
TYPE index_tbl is TABLE OF rec INDEX BY BINARY_INTEGER;
function echo_rec(mrec rec) return rec;
function echo_index_tbl(mtbl index_tbl) return index_tbl;
end echo_plsql;
```

PL/SQL のファンクション名が Web サービスの操作名にマップされる方法

WebServicesAssembler は、PL/SQL のファンクションまたはプロシージャを Web サービス操作にマッピングします。このとき、通常は名前が変更されます。通常、PL/SQL 名のアンダースコアが取り除かれ、アンダースコアの次の文字が大文字になります。たとえば、例 10-1 の PL/SQL ファンクション `echo_index_tbl` に注目してみます。このファンクションは Web サービス操作 `echoIndexTbl` にマッピングされます。例 10-2 の WSDL フラグメントでは、PL/SQL ファンクション `echo_index_tbl` が Web サービス操作 `echoIndexTbl` として表現されています。

例 10-2 PL/SQL ファンクションのマッピングを示す WSDL フラグメント

```
<operation name="echoIndexTbl" parameterOrder="EchobaseIndexTblBase_1">
<input message="tns:Echo_echoIndexTbl"/>
<output message="tns:Echo_echoIndexTblResponse"/>
</operation>
```

クライアント・コードから PL/SQL の IN および IN OUT パラメータにアクセスする方法

10-13 ページの例 10-1 の PL/SQL パラメータ OUT および IN OUT は、XML では IN OUT パラメータとして表現されており、holdVarchar のホルダー・パラメータとして使用されています。例 10-3 の WSDL フラグメントのエントリは holdVarchar 操作を示しています。2 番目および 3 番目のパラメータは入力メッセージおよび出力メッセージの両方で使用されており、どちらのパラメータも IN OUT パラメータであることがわかります。

例 10-3 IN OUT パラメータを示す WSDL フラグメント

```
<operation name="holdVarchar"
   parameterOrder="Integer_1 String_2 String_3">
  <input message="tns:Echo_holdVarchar"/>
  <output message="tns:Echo_holdVarcharResponse"/>
</operation>
<message name="Echo_holdVarchar">
<part name="Integer_1" type="xsd:int"/>
<part name="String_2" type="xsd:string"/>
<part name="String_3" type="xsd:string"/>
</message>
<message name="Echo_holdVarcharResponse">
<part name="result" type="ns1:double"/>
<part name="String_2" type="xsd:string"/>
<part name="String_3" type="xsd:string"/>
</message>
```

JAX-RPC クライアント・コードの IN OUT パラメータにアクセスするには、JAX-RPC ホルダーを使用する必要があります。たとえば、例 10-4 のコードは戻り値を firstName.value および lastName.value として取得します。ここで firstName および lastName はどちらも String ホルダーです。ホルダーの実際の値には、println 文で示しているようにメンバー value を使用してアクセスします。

例 10-4 JAX-RPC ホルダーを使用する、クライアント・コードの IN OUT パラメータへのアクセス

```
System.out.println("holdVarchar");
StringHolder firstName = new StringHolder("Tom");
StringHolder lastName = new StringHolder("Gordon");
System.out.println("Holder returned:  empid="
+ ci.holdVarchar(id, firstName, lastName)
+ ", name="
+ firstName.value
+ "."
+ lastName.value);
```

plsqlAssemble コマンドラインに -style rpc が指定されている点に注意してください。rpc スタイルではホルダーがサポートされています。デフォルトの document-wrapped スタイルではホルダーはサポートされていません。

Web サービスがデフォルトの document-wrapped スタイルを使用して作成されている場合は、異なる holdVarchar シグネチャが生成されています。OUT 引数は属性として戻り値に取得されます。

次の WSDL セグメントは document-wrapped スタイルにおける holdVarchar 操作を示しています。戻り型 EchoUser_holdVarchar_Out では、属性 lastnameOut および firstnameOut に、PL/SQL パラメータ firstname および lastname の OUT 値が記録されます。

例 10-5 document-wrapped スタイルで処理される IN OUT パラメータを示す WSDL フラグメント

```
<operation name="holdVarchar" parameterOrder="Integer_1">
            <input message="tns:Echo_holdVarchar"/>
            <output message="tns:Echo_holdVarcharResponse"/>
 </operation>

<message name="Echo_holdVarchar">
  <part name="Integer_1" type="xsd:int"/>
</message>
<message name="Echo_holdVarcharResponse">
  <part name="result" type="tns:Echo_holdVarchar_Out"/>
</message>

<complexType name="Echo_holdVarchar_Out">
     <sequence>
         <element name="return" type="double" nillable="true"/>
        <element name="lastnameOut" type="string" nillable="true"/>
         <element name="firstnameOut" type="string" nillable="true"/>
     </sequence>
</complexType>
```

クライアント・コードから SQL の XMLType 型にアクセスする方法

10-13 ページの例 10-1 の SQL `XMLType` は、XML any 型にマッピングされます。このマッピングは、例 10-6 の WSDL フラグメントの `getXmltype` 操作で示されています。

例 10-6 SQL XMLType の text_xml へのマッピングを示す WSDL フラグメント

```
<message name="Echo_getXmltypeResponse">
<part name="result" type="ns2:any"/>
</message>
<operation name="getXmltype" parameterOrder="BigDecimal_1">
<input message="tns:Echo_getXmltype"/>
<output message="tns:Echo_getXmltypeResponse"/>
</operation>
```

WebServicesAssembler は、XML any を Java 型 `org.w3c.org.dom.Element` にマッピングするプロキシを生成します。したがって、Java クライアントは SQL `XMLType` インスタンスに `Element` インスタンスとしてアクセスします。

SQL 問合せまたは DML 文から Web サービスをアセンブルする方法

`sqlAssemble` コマンドを使用して、Web サービスを SQL 文から生成します。文には SQL 問合せおよび DML（Data Manipulation Language）文を含むことができます。

PL/SQL Web サービスの生成とは異なり、SQL 文をアセンブルしても、PL/SQL ラッパーは生成されません。PL/SQL ラッパーは PL/SQL レコードまたは `INDEX BY` 表タイプを処理する場合のみ生成されます。これらのタイプは SQL 文では使用されません。

この項の内容は、次のとおりです。

- 前提条件
- SQL 文または問合せから Web サービスをアセンブルする手順
- サンプルの SQL 文
- SQL 問合せが Web サービスの操作名にマッピングされる方法
- DML 操作が Web サービス操作にマッピングされる方法

前提条件

開始する前に、次のファイルと情報を用意してください。

- SQL 文または問合せ。次の例で使用されている SQL 文は、10-18 ページの例 10-7「サンプルの SQL 文」でも使用されています。

  複数の `sqlstatement` 引数をコマンドラインまたは Ant タスクに指定できます。`sqlstatement` 引数の書式の詳細は、18-54 ページの「sqlstatement」を参照してください。

- Web サービス・アプリケーションの名前。
- JDBC データソース用の JNDI 位置。この情報は実行時に使用されます。
- JDBC データベースの接続 URL。この情報はコンパイル時に使用されます。
- 問合せまたは文のベースとするスキーマの名前およびパスワード。この情報はコンパイル時に使用されます。

SQL 文または問合せから Web サービスをアセンブルする手順

次の手順では、`sqlAssemble` コマンドを使用し、SCOTT スキーマに基づいて問合せおよび文の Web サービスをアセンブルします。

1. 「前提条件」の項で説明した SQL 文または問合せ、それらの対象となるデータベースの名前およびパスワード、およびその他の情報を、WebServicesAssembler の `sqlAssemble` コマンドに対する入力として使用します。

   たとえば、次のコマンドは Web サービス・アプリケーション `query.ear` を生成します。

   コマンドライン：

   ```
   java -jar wsa.jar -sqlAssemble
      -appName query
      -dataSource jdbc/OracleManagedDS
      -sqlstatement "getEmpCount=select ename, sal from emp where
                        sal>:{mysal NUMBER}"
      -sqlstatement "getEmpBySal=select ename, sal from emp where
                        sal>:{mysal NUMBER}"
      -sqlstatement "updateEmp=update emp SET sal=sal+500 where
                        ename=:{myname VARCHAR}"
      -dbConnection jdbc:oracle:thin:@stacd15:1521:lsqlj
      -dbUser scott/tiger
   ```

   Ant タスク：

   ```
   <oracle:sqlAssemble
           appName="query"
           dataSource="jdbc/OracleManagedDS"
           dbConnection="jdbc:oracle:thin:@dsunrde22:1521:sqlj"
           dbUser="scott/tiger">
           <sqlstatement="getEmpCount=select ename, sal from emp where sal>:{mysal
   NUMBER}"/>
           <sqlstatement="getEmpBySal=select ename, sal from emp where sal>:{mysal
   NUMBER}"/>
           <sqlstatement="updateEmp=update emp SET sal=sal+500 where  ename=:{myname
   VARCHAR}"/>
   />
   ```

   このコマンドおよび Ant タスクの説明：

   - `sqlAssemble`: SQL 問合せやデータ操作言語（DML）などの SQL 文から Web サービスを生成します。このコマンドを使用するには、データベースに接続する必要があります。18-19 ページの「sqlAssemble」を参照してください。
   - `appName`: アプリケーションの名前を指定します。この名前は、通常 `context` や `uri` などの他の引数のベース値として使用されます。18-40 ページの「appName」を参照してください。

- `dataSource`: 実行時に Web サービスによって使用されるデータソースの JNDI 位置を指定します。18-53 ページの「dataSource」を参照してください。
- `dbConnection`: データベースの JDBC URL を指定します。18-53 ページの「dbConnection」を参照してください。
- `dbUser`: *user*/*password* の形式でデータベースのスキーマとパスワードを指定します。18-53 ページの「dbUser」を参照してください。
- `sqlStatement`: Web サービスとして公開する DML 文または SQL 問合せを指定します。18-54 ページの「sqlstatement」を参照してください。

2. サービスを OC4J の実行中インスタンスにデプロイし、アプリケーションをバインドします。

   `-dataSource` 引数で参照しているデータソースがこの OC4J インスタンスに設定されている必要があります。

   次のコマンドラインは、サンプルのデプロイおよびバインドのコマンドです。

```
% java -jar <J2EE_HOME>/admin_client.jar deployer:oc4j:localhost:port admin welcome
-deploy -file dist/query.ear -deploymentName query

% java -jar <J2EE_HOME>/admin_client.jar deployer:oc4j:localhost:port admin welcome
-bindWebApp plsql plsql-web default-web-site /query
```

   このサンプル・コマンドラインの *<J2EE_HOME>* は、J2EE のインストール・ディレクトリです。

   EAR ファイルのデプロイの詳細は、『Oracle Containers for J2EE デプロイメント・ガイド』を参照してください。

3. 必要に応じて、デプロイメントが成功したかどうかを確認します。OracleAS Web Services では、デプロイされた Web サービスごとに Web サービス・テスト・ページが提供されます。Web サービス・テスト・ページへのアクセスおよびその使用方法の詳細は、13-7 ページの「Web サービス・テスト・ページの使用方法」を参照してください。

4. クライアント・サイド・コードを生成します。

   - J2SE 環境の場合、`genProxy` コマンドを使用して WebServicesAssembler ツールを実行し、J2SE Web サービス・クライアントのスタブ（クライアント・プロキシ）を生成します。J2SE 環境向けのクライアント・サイド・コードの生成およびアセンブルの詳細は、第 15 章「J2SE Web サービス・クライアントのアセンブル」を参照してください。
   - J2EE 環境の場合、`genInterface` コマンドを使用して WebServicesAssembler ツールを実行し、J2EE Web サービス・クライアント用の JAX-RPC マッピング・ファイルおよびサービス・エンドポイント・インタフェースを生成します。クライアント・サイド・コードの生成およびアセンブルの詳細は、第 14 章「J2EE Web サービス・クライアントのアセンブル」を参照してください。

   たとえば、次のコマンドは `genProxy` を使用して J2SE クライアントのコードを生成します。

```
% java -jar wsa.jar -genProxy
                    -wsdl http://locahost:8888/query/query?WSDL
                    -output build/src/client
                    -mappingFileName ./mapping.xml
                    -packageName oracle.demo.db.query.stub
                    -unwrapParameters true
```

   このコマンドはディレクトリ `build/src/client` にクライアントを生成します。

   このコマンドの説明 :

   - `genProxy`: J2SE Web サービス・クライアントから使用可能な静的プロキシ・スタブを作成します。18-32 ページの「genProxy」を参照してください。
   - `wsdl`: WSDL ドキュメントの絶対ファイル・パス、相対ファイル・パスまたは URL を指定します。18-64 ページの「wsdl」を参照してください。

- `output`: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
- `mappingFileName`: JAX-RPC マッピング・ファイルを指すファイル位置を指定します。18-46 ページの「mappingFileName」を参照してください。
- `packageName`: JAX-RPC マッピング・ファイルにパッケージ名が宣言されていない場合に、生成されたクラス用に使用されるパッケージ名を指定します。18-47 ページの「packageName」を参照してください。
- `unwrapParameters`: この引数は document-literal 操作に対してのみ設定でき、その他のメッセージ書式では無視されます。`unwrapParameters` を `true` に設定すると(デフォルト)、戻り型とレスポンス・タイプがアンラップされます。通常はこの方が使いやすくなります。特にタイプが単純な場合はそうです。18-69 ページの「unwrapParameters」を参照してください。

5. クライアントをコンパイルおよび実行します。

   クライアントをコンパイルする前に、クラスパス上にある適切な JAR をリストします。クライアントのクラスパスで使用可能なすべての JAR ファイルのリストについては、表 A-2「クライアント・サイド・プロキシを使用するクライアントのクラスパス・コンポーネント」を参照してください。個々の JAR をリストするかわりに、クライアントのクラスパスにクライアント側の JAR として `wsclient_extended.jar` を含めることができます。この JAR ファイルには、Web サービス・クライアントをコンパイルおよび実行するために必要なすべてのクラスが含まれます。これらのクラスは、表 A-2 にリストされている各 JAR ファイルに含まれているものです。`wsclient_extended.jar` およびクライアントのクラスパスの詳細は、A-3 ページの「Web サービス・プロキシのクラスパスの設定」を参照してください。

   次のコマンドラインは、サンプルのコンパイルおよび実行のコマンドです。

```
% javac -classpath path:
     <ORACLE_HOME>/webservices/lib/wsclient_extended.jar:
     :<ORACLE_HOME>/webservices/lib/jaxrpc-api.jar QueryClient.java

% java -classpath path
     <ORACLE_HOME>/webservices/lib/wsclient_extended.jar:
     <ORACLE_HOME>/webservices/lib/jaxrpc-api.jar:
     <J2EE_HOME>/lib/jax-qname-namespace.jar:
     <J2EE_HOME>/lib/activation.jar:
     <J2EE_HOME>/lib/mail.jar:
     <J2EE_HOME>/lib/http_client.jar:
     <ORACLE_HOME>/webservices/lib/commons-logging.jar:
     <ORACLE_HOME>/lib/xmlparserv2.jar QueryClient
```

この例において、*<J2EE_HOME>* は J2EE のインストール・ディレクトリであり、*<ORACLE_HOME>* は OC4J のインストール・ディレクトリです。

サンプルの SQL 文

例 10-7 は、Web サービスとして公開される SQL 文の定義を示しています。

例 10-7 サンプルの SQL 文

```
getEmpCount=select ename, sal from emp where sal>:{mysal NUMBER}
getEmpBySal=select ename, sal from emp where sal>:{mysal NUMBER}
updateEmp=update emp SET sal=sal+500 where  ename=:{myname VARCHAR}
```

SQL 問合せが Web サービスの操作名にマッピングされる方法

SQL 問合せは、Web サービスとして公開されると、3 つのサービス操作にマッピングされます。たとえば、例 10-7 の getEmpBySal 問合せにより、次のサービス操作が生成されます。

- getEmpBySal**Beans**: 配列を戻します。配列要素は、問合せ結果の行内の列に対応した各属性を持ったオブジェクト・タイプです。
- getEmpBySal**XMLRowSet**: WebRowSet 形式の問合せ結果が入った XML 文書を戻します。
- getEmpBySal**XML**: Oracle XDB 行セット書式の問合せ結果が入った XML 文書を戻します。

1 つの問合せから 3 つの操作が生成されるので、便利です。戻り値の違いは形式のみです。なお、オリジナルの問合せ名に Beans、XMLRowSet および XML を付加するネーミング規則になっています。

例 10-8 の WSDL フラグメントでは、WSDL ファイルの 3 つの操作の戻り型を説明しています。

例 10-8 SQL 問合せのサービス操作を示す WSDL フラグメント

```
<complexType name="getEmpBySalBeansResponse">
<sequence>
<element name="result" type="tns:Query_getEmpBySalRowUser" nillable="true"
     minOccurs="0" maxOccurs="unbounded"/>
</sequence>
</complexType>
<complexType name="Query_getEmpBySalRowUser">
<sequence>
<element name="ename" type="string" nillable="true"/>
<element name="sal" type="decimal" nillable="true"/>
</sequence>
</complexType>

<complexType name="getEmpBySalXMLRowSetResponse">
<sequence>
<element name="result" type="ns1:swaRef" nillable="true"/>
</sequence>
</complexType>

<complexType name="getEmpBySalXMLResponse">
<sequence>
<element name="result" type="xsd:any" nillable="true"/>
</sequence>
</complexType>
```

getEmpBySalXMLRowSetResponse および getEmpBySalXMLResponse の 2 つのメソッドは swaRef 型のパラメータを持ちます。この型に対して、アセンブラはクライアント・プロキシに javax.xml.soap.AttachmentPart を生成します。戻される問合せ結果にアクセスするクライアント・コードを例 10-9 に示します。

例 10-9 swaRef 型に対して戻される問合せ結果への、クライアント・コードからのアクセス

```
import oracle.jdbc.pool.OracleDataSource;
import oracle.jdbc.rowset.OracleWebRowSet;
import javax.xml.soap.AttachmentPart;
import org.w3c.org.Element;
import javax.xml.transform.dom.*;
import java.io.*;
   ...

/* Access the query result as Oracle XDB RowSet */
Element element = eme.getEmpBySalXML(new BigDecimal(500));
DOMSource doms = new javax.xml.transform.doc.DOMSource(element);
buf = new jav.io.ByteArrayOutputStream();
StreamResult streamr = new StreamResult(buf);
trnas.transform(doms, streamr);
```

```
System.out.println(buf, toString());


/* Access the query result as Oracle WebRowSet */
ap = eme.getEmpBySalXMLRowSet(new BigDecimal(500));
source = (Source) ap.getContent();
trans = TransformerFactory.newInstance().newTransformer();
buf = new ByteArrayOutputStream();
streamr = new StreamResult(buf);
trans.transform(source, streamr);
InputStream istream = new ByteArrayInputStream(buf.toString().getBytes());
OracleWebRowSet rowset = new OracleWebRowSet();
System.setProperty("http.proxyHost", "www-proxy.us.oracle.com");
System.setProperty("http.proxyPort", "80");
         System.setProperty("javax.xml.parsers.DocumentBuilderFactory",
"oracle.xml.jaxp.JXDocumentBuilderFactory");
rowset.readXml(new InputStreamReader(istream));
rowset.writeXml(new PrintWriter(System.out));
```

例 10-9 のコードは、問合せ結果を Oracle XDB 行セット（ROWSET）および Oracle Web 行セット（OracleWebRowSet）の 2 つの形式で表示します。例 10-10 では問合せ結果を Oracle XDB 行セットで表示しています。例 10-11 では結果を WebRowSet 形式で出力しています。実際には、oracle.jdbc.rowset.OracleWebRowSet API を使用して、例 10-9 の可変行セットである OracleWebRowSet インスタンスにアクセスします。

関連資料：

これらのデータ型の詳細は、『Oracle Database JDBC 開発者ガイドおよびリファレンス』を参照してください。

例 10-10 問合せ結果（Oracle XDB 行セット書式）

```
<ROWSET>
<ROW num="1">
<ENAME>SMITH</ENAME><SAL>800</SAL>
</ROW>
<ROW num="2">
<ENAME>ALLEN</ENAME><SAL>1600</SAL>
</ROW>
<ROW num="3">
<ENAME>WARD</ENAME><SAL>1250</SAL>
</ROW>
</ROWSET>
```

例 10-11 問合せ結果（JDBC Web 行セット書式）

```
<?xml version="1.0" encoding="UTF-8"?>
 <webRowSet xmlns="http://java.sun.com/xml/ns/jdbc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/jdbc
http://java.sun.com/xml/ns/jdbc/webrowset.xsd">
   <properties>
<command></command>
<concurrency>1007</concurrency>
<datasource></datasource>
<escape-processing>true</escape-processing>
<fetch-direction>1002</fetch-direction>
<fetch-size>10</fetch-size>
<isolation-level>2</isolation-level>
<key-columns>
</key-columns>
<map>
</map>
```

```
<max-field-size>0</max-field-size>
<max-rows>0</max-rows>
<query-timeout>0</query-timeout>
<read-only>false</read-only><rowset-type>1005</rowset-type>
<show-deleted>false</show-deleted>
<table-name></table-name>
<url>jdbc:oracle:thin:@stacd15:1521:lsqlj1</url>
<sync-provider>

<sync-provider-name>com.sun.rowset.providers.RIOptimisticProvider</sync-provider-name>
  <sync-provider-vendor>Sun Microsystems Inc.</sync-provider-vendor>
  <sync-provider-version>1.0</sync-provider-version>
  <sync-provider-grade>2</sync-provider-grade>
  <data-source-lock>1</data-source-lock>
</sync-provider>
   </properties>
   <metadata>
<column-count>2</column-count>
<column-definition>
  <column-index>1</column-index>
  <auto-increment>false</auto-increment>
  <case-sensitive>true</case-sensitive>
  <currency>false</currency>
  <nullable>1</nullable>
  <signed>true</signed>
  <searchable>true</searchable>
  <column-display-size>10</column-display-size>
  <column-label>ENAME</column-label>
  <column-name>ENAME</column-name>
  <schema-name></schema-name>
  <column-precision>0</column-precision>
  <column-scale>0</column-scale>
  <table-name></table-name>
  <catalog-name></catalog-name>
  <column-type>12</column-type>
  <column-type-name>VARCHAR2</column-type-name>
              </column-definition>

<column-definition>
  <column-index>2</column-index>
  <auto-increment>false</auto-increment>
  <case-sensitive>true</case-sensitive>
  <currency>false</currency>
  <nullable>1</nullable>
  <signed>true</signed>
  <searchable>true</searchable>
  <column-display-size>10</column-display-size>
  <column-label>SAL</column-label>
  <column-name>SAL</column-name>
  <schema-name></schema-name>
  <column-precision>0</column-precision>
  <column-scale>0</column-scale>
  <table-name></table-name>
  <catalog-name></catalog-name>
  <column-type>2</column-type>
  <column-type-name>NUMBER</column-type-name>
</column-definition>
   </metadata>
   <data>
<currentRow>
  <columnValue>SMITH</columnValue>
             <columnValue>800</columnValue>
</currentRow>
```

```
<currentRow>
  <columnValue>ALLEN</columnValue>
            <columnValue>1600</columnValue>
     </currentRow>
     <currentRow>
       <columnValue>WARD</columnValue>
            <columnValue>1250</columnValue>
     </currentRow>
   </data>
 </webRowSet>
```

DML 操作が Web サービス操作にマッピングされる方法

DML 文は UPDATE、DELETE または INSERT SQL 文です。sqlAssemble コマンドを使用して、DML 文を Web サービスの操作として公開できます。DML 操作は正常終了すると自動的にコミットされ、エラーがあった場合は自動的にロールバックされます。

例 10-12 は、例 10-7 の DML 文を公開する WSDL フラグメントです。

```
updateEmp=update emp SET sal=sal+500 where ename=:{myname VARCHAR}
```

この DML 文は 2 つの操作として公開されています。この例では、updateEmp により DML 文が実行されますが、updateEmpiS によっても同じものがバッチ・モードで実行されます。バッチ操作では、オリジナルの各 DML 引数の配列を使用します。配列の各要素がバッチの 1 実行として使用されます。バッチ操作の結果は、バッチで更新される行の合計数になります。

例 10-12 DML 操作のレスポンス・タイプを示す WSDL フラグメント

```
    <message name="SqlStmts_updateEmp">
      <part name="salary" type="xsd:string"/>
  </message>
  <message name="SqlStmts_updateSchemaResponse">
      <part name="result" type="xsd:int"/>
  </message>
  <message name="SqlStmts_updateEmpiS">
      <part name="salary" type="tns:ArrayOfstring"/>
  </message>
  <message name="SqlStmts_updateEmpiSResponse">
      <part name="result" type="xsd:int"/>
  </message>
```

Oracle Streams AQ から Web サービスをアセンブルする方法

Oracle Streams アドバンスト・キューイングは、Oracle データベースが提供する非同期メッセージング・システムです。アドバンスト・キュー（AQ）を Web サービスとして公開することにより、クライアントはメッセージをデータベース内の受信者、つまり、同じ Web サービスの別のクライアントに送信できます。

WebServicesAssembler ツールは、データベース内の既存の AQ から Web サービスを生成できます。AQ はシングル・コンシューマまたは複数のコンシューマを持つことができます。シングル・コンシューマは、通常のいわゆるキューのことです。複数のコンシューマを持つ AQ は、通常のいわゆるトピックのことです。各 Oracle Streams AQ はキュー表に格納されています。キュー表は、格納しているすべての AQ のペイロード・タイプを定義し、AQ がキューのみをサポートするか、またはトピックもサポートするのかも定義します。生成される Java コードでは、Oracle Streams AQ JMS API が使用されます。

WebServicesAssembler がキューおよびトピックに関して公開する Web サービス操作の例は、それぞれ 10-27 ページの例 10-14 および 10-29 ページの例 10-15 を参照してください。アセンブラは Oracle Streams AQ および AQ JMS API をベースにして操作を作成します。

この項の内容は、次のとおりです。

- 前提条件
- Oracle AQ からの Web サービスをアセンブルする手順
- サンプルの AQ キュー / トピックの宣言
- WebServicesAssembler で生成されるキューの Web サービスのサンプル
- WebServicesAssembler で生成されるトピックに対する Web サービスのサンプル
- Web サービスとして公開されている AQ キューにクライアント・コードからアクセスする方法
- JMS を使用した Oracle AQ Queue にアクセスする方法

関連資料：

Oracle Streams AQ および AQ JMS API の詳細は、『Oracle Streams Advanced Queuing Java API Reference』を参照してください。

前提条件

開始する前に、次のファイルと情報を用意してください。

- AQ が存在するデータベースに対するデータベース接続 URL。WebServicesAssembler は、JDBC のかわりに JMS キュー・インスタンスを使用して Oracle AQ にアクセスするオプションを用意しています。詳細は、10-31 ページの「JMS を使用した Oracle AQ Queue にアクセスする方法」を参照してください。
- AQ が存在するスキーマの名前、およびスキーマにアクセスするためのユーザー名とパスワード。これはコンパイル時に使用されます。
- Web サービスとして公開するキューまたはトピックの名前。シングル・キューのみまたはトピックをパブリッシュして、Web サービスで公開できます。サンプルのキューおよびトピックは 10-26 ページの「サンプルの AQ キュー / トピックの宣言」を参照してください。
- Web サービス・アプリケーションの名前。
- データソースの JNDI 名。この情報は実行時に使用されます。

Oracle AQ からの Web サービスをアセンブルする手順

次の手順は、WebServicesAssembler を使用して Oracle AQ キューから Web サービスをアセンブルする方法を示しています。

1. 「前提条件」の項で説明したファイルおよびその他の情報を、WebServicesAssembler の `-aqAssemble` コマンドに入力します。

   たとえば、次のコマンドは Web サービス・アプリケーションを作成し、`queue.ear` ファイルを現行ディレクトリに生成します。WebServicesAssembler ツールは、実行時にキューにアクセスする Java ファイルを生成します。10-26 ページの「サンプルの AQ キュー / トピックの宣言」に、AQ `sample_queue` の宣言の例を示します。

   コマンドライン :

   ```
   java -jar $ORACLE_HOME/webservices/lib/wsa.jar
     -aqAssemble
     -appName queue
     -dataSource jdbc/OracleManagedDS
     -portName assembleQueuePort
     -sql sample_queue
     -dbConnection jdbc:oracle:thin:@stacd15:1521:lsqlj
     -dbUser scott/tiger
   ```

   Ant タスク :

   ```
   <aqAssemble
      appName="queue"
      dataSource="jdbc/OracleManagedDS"
      sql="sample_queue"
      portName="assembleQueuePort"
      dbConnection="jdbc:oracle:thin:@stacd15:1521:lsqlj"
      dbUser="scott/tiger"
   />
   ```

   宣言内のサンプル・トピック `sample_topic` は、`sample_queue` と同じようにパブリッシュできます（ただし、WebServicesAssembler の別個の起動においてです）。異なる点は、`aqAssemble` コマンドの `sql` 引数および `appName` 引数の値のみです。

   このコマンドおよび Ant タスクの説明 :

   - `aqAssemble`: データベースのアドバンスト・キューから Web サービスを生成します。このコマンドを使用するには、データベースに接続する必要があります。18-6 ページの「aqAssemble」を参照してください。
   - `appName`: アプリケーションの名前を指定します。この名前は、通常 `context` や `uri` などの他の引数のベース値として使用されます。18-40 ページの「appName」を参照してください。
   - `dataSource`: 実行時に Web サービスによって使用されるデータソースの JNDI 位置を指定します。18-53 ページの「dataSource」を参照してください。
   - portName: 18-47 ページの「portName」を参照してください。
   - `sql`: PL/SQL のパッケージ名を指定します。18-54 ページの「sql」を参照してください。
   - `dbConnection`: データベースの JDBC URL を指定します。18-53 ページの「dbConnection」を参照してください。
   - `dbUser`: *user*/*password* の形式でデータベースのスキーマとパスワードを指定します。18-53 ページの「dbUser」を参照してください。

2. サービスを OC4J の実行中インスタンスにデプロイし、アプリケーションをバインドします。

   この手順は、AQ が OC4J インスタンスのデータソースとして設定されていることを想定しています。

   次のコマンドラインは、サンプルのデプロイおよびバインドのコマンドです。

   ```
   % java -jar <J2EE_HOME>/admin_client.jar deployer:oc4j:localhost:port admin welcome
   -deploy -file dist/queue.ear -deploymentName queue

   % java -jar <J2EE_HOME>/admin_client.jar deployer:oc4j:localhost:port admin welcome
   -bindWebApp queue queue-web default-web-site /queue
   ```

   この例において、*<J2EE_HOME>* は、J2EE のインストール・ディレクトリです。

   EAR ファイルのデプロイの詳細は、『Oracle Containers for J2EE デプロイメント・ガイド』を参照してください。

3. （オプション）デプロイが成功したことを確認します。

   OracleAS Web Services では、デプロイされた Web サービスごとに Web サービス・テスト・ページが提供されます。Web サービス・テスト・ページへのアクセスおよびその使用方法の詳細は、13-7 ページの「Web サービス・テスト・ページの使用方法」を参照してください。

4. クライアント・サイド・コードを生成します。

   - J2SE 環境の場合、`genProxy` コマンドを使用して WebServicesAssembler ツールを実行し、J2SE Web サービス・クライアントのスタブ（クライアント・プロキシ）を生成します。J2SE 環境向けのクライアント・サイド・コードの生成およびアセンブルの詳細は、第 15 章「J2SE Web サービス・クライアントのアセンブル」を参照してください。
   - J2EE 環境の場合、`genInterface` コマンドを使用して WebServicesAssembler ツールを実行し、J2EE Web サービス・クライアント用の JAX-RPC マッピング・ファイルおよびサービス・エンドポイント・インタフェースを生成します。クライアント・サイド・コードの生成およびアセンブルの詳細は、第 14 章「J2EE Web サービス・クライアントのアセンブル」を参照してください。

   たとえば、次のコマンドは `genProxy` を使用して J2SE クライアントのコードを生成します。

   ```
   % java -jar wsa.jar -genProxy
                       -wsdl http://locahost:8888/queue/queue?WSDL
                       -output build/src/client
                       -mappingFileName ./mapping.xml
                       -packageName oracle.demo.db.queue.stub
                       -unwrapParameters true
   ```

   このコマンドの説明 :

   - `genProxy`: J2SE Web サービス・クライアントから使用可能な静的プロキシ・スタブを作成します。18-32 ページの「genProxy」を参照してください。
   - `wsdl`: WSDL ドキュメントの絶対ファイル・パス、相対ファイル・パスまたは URL を指定します。18-64 ページの「wsdl」を参照してください。
   - `output`: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
   - `mappingFileName`: JAX-RPC マッピング・ファイルを指すファイル位置を指定します。18-46 ページの「mappingFileName」を参照してください。
   - `packageName`: JAX-RPC マッピング・ファイルにパッケージ名が宣言されていない場合に、生成されたクラス用に使用されるパッケージ名を指定します。18-47 ページの「packageName」を参照してください。

- unwrapParameters: この引数は document-literal 操作に対してのみ設定でき、その他のメッセージ書式では無視されます。unwrapParameters を true に設定すると(デフォルト)、戻り型とレスポンス・タイプがアンラップされます。通常はこの方が使いやすくなります。特にタイプが単純な場合はそうです。18-69 ページの「unwrapParameters」を参照してください。

5. クライアントをコンパイルおよび実行します。

   クライアントをコンパイルする前に、クラスパス上にある適切な JAR をリストします。クライアントのクラスパスで使用可能なすべての JAR ファイルのリストについては、表 A-2「クライアント・サイド・プロキシを使用するクライアントのクラスパス・コンポーネント」を参照してください。個々の JAR をリストするかわりに、クライアントのクラスパスにクライアント側の JAR として wsclient_extended.jar を含めることができます。この JAR ファイルには、Web サービス・クライアントをコンパイルおよび実行するために必要なすべてのクラスが含まれます。これらのクラスは、表 A-2 にリストされている各 JAR ファイルに含まれているものです。wsclient_extended.jar およびクライアントのクラスパスの詳細は、A-3 ページの「Web サービス・プロキシのクラスパスの設定」を参照してください。

   次のコマンドラインは、サンプルのコンパイルおよび実行のコマンドです。

```
% javac -classpath path
<ORACLE_HOME>/webservices/lib/wsclient_extended.jar:
:<ORACLE_HOME>/webservices/lib/jaxrpc-api.jar QueueClient.java

% java -classpath path
<ORACLE_HOME>/webservices/lib/wsclient_extended.jar:
<ORACLE_HOME>/webservices/lib/jaxrpc-api.jar:
<J2EE_HOME>/lib/jax-qname-namespace.jar:
<J2EE_HOME>/lib/activation.jar:<J2EE_HOME>/lib/mail.jar:
<J2EE_HOME>/lib/http_client.jar:
<ORACLE_HOME>/webservices/lib/commons-logging.jar:
<ORACLE_HOME>/lib/xmlparserv2.jar QueueClient
```

この例において、<*J2EE_HOME*> は J2EE のインストール・ディレクトリであり、<*ORACLE_HOME*> は OracleAS Web Services のインストール・ディレクトリです。

サンプルの AQ キュー / トピックの宣言

例 10-13 の PL/SQL スクリプトは、キュー sample_queue およびトピック sample_topic を定義しています。キューのペイロード・タイプは queue_message であり、SQL オブジェクト・タイプです。トピックのペイロード・タイプは topic_message であり、同様に SQL オブジェクト・タイプです。

例 10-13 サンプルのキュー / トピック宣言

```
create type scott.queue_message as object (
   Subject               VARCHAR2(30),
   Text                  VARCHAR2(80));
create type scott.topic_message as object (
   Subject               VARCHAR2(30),
   Text                  VARCHAR2(80));
BEGIN
  dbms_aqadm.create_queue_table (
          Queue_table            => 'scott.queue_queue_table',
          Queue_payload_type     => 'scott.queue_message');
  dbms_aqadm.create_queue(
    queue_name  => 'scott.sample_queue',
    queue_table => 'scott.queue_queue_table' );
  dbms_aqadm.start_queue(queue_name => 'scott.sample_queue');

  dbms_aqadm.create_queue_table (
    Queue_table            => 'scott.topic_queue_table',
    Multiple_consumers => TRUE,
```

```
    Queue_payload_type     => 'scott.topic_message');
  dbms_aqadm.create_queue(
    queue_name  => 'scott.sample_topic',
    queue_table => 'scott.topic_queue_table' );
  dbms_aqadm.start_queue(queue_name => 'scott.sample_topic');
END;
/
```

WebServicesAssembler で生成されるキューの Web サービスのサンプル

「サンプルの AQ キュー / トピックの宣言」で記述されているキューに対して WebServicesAssembler によって公開される Web サービス操作を、例 10-14 にリストします。

この例では、send 操作がペイロードをキューにエンキューしています。ペイロード・タイプは、<send/> 要素に示されているように、複合型 tns:QueueMessageUser です。この型は SQL 型 QUEUE_MESSAGE に対応しています。receive 操作はキューからのペイロードを戻します。<receiveResponse/> 要素は、戻されるペイロードのタイプが tns:QueueMessage であることを示しています。メッセージがキューに追加されるまで操作はブロックされます。receiveNoWait 操作はキューからのペイロードを戻します。キューにメッセージが存在しない場合は、操作は待機せずに NULL を戻します。receive2 操作は 2 つの引数を持ちます。

- xsd:string 型の selector
- xsd:boolean 型の noWait

selector は AQ 規則で指定されているフィルタ条件です。条件を満たすメッセージのみを戻す receive 操作が可能です。たとえば、JMSPriority < 3 AND PRICE < 300 を selector に指定すると、優先度 3 以上、および属性 PRICE が 300 未満のメッセージのみが公開されます。パラメータ noWait が true の場合、操作はブロックされません。

例 10-14 キューに対して公開される Web サービス操作

```
<operation name="receive">
  <input message="tns:SampleQueue_receive" />
  <output message="tns:SampleQueue_receiveResponse" />
  </operation>
<operation name="receive2">
  <input message="tns:SampleQueue_receive2" />
  <output message="tns:SampleQueue_receive2Response" />
  </operation>
<operation name="receiveNoWait">
  <input message="tns:SampleQueue_receiveNoWait" />
  <output message="tns:SampleQueue_receiveNoWaitResponse" />
  </operation>
<operation name="send">
  <input message="tns:SampleQueue_send" />
  <output message="tns:SampleQueue_sendResponse" />
  </operation>

<complexType name="receive">
  <sequence />
  </complexType>
<complexType name="receiveResponse">
<sequence>
  <element name="result" type="tns:QueueMessageUser" nillable="true" />
  </sequence>
  </complexType>
<complexType name="QueueMessageUser">
<sequence>
  <element name="text" type="string" nillable="true" />
  <element name="subject" type="string" nillable="true" />
  </sequence>
  </complexType>
<complexType name="receive2">
```

```
<sequence>
  <element name="String_1" type="string" nillable="true" />
  <element name="boolean_2" type="boolean" />
  </sequence>
  </complexType>
<complexType name="receive2Response">
<sequence>
  <element name="result" type="tns:QueueMessageUser" nillable="true" />
  </sequence>
  </complexType>
<complexType name="receiveNoWait">
  <sequence />
  </complexType>
<complexType name="receiveNoWaitResponse">
<sequence>
  <element name="result" type="tns:QueueMessageUser" nillable="true" />
  </sequence>
  </complexType>
<complexType name="send">
<sequence>
  <element name="QueueMessageUser_1" type="tns:QueueMessageUser" nillable="true" />
  </sequence>
  </complexType>
<complexType name="sendResponse">
  <sequence />
  </complexType>
```

WebServicesAssembler で生成されるトピックに対する Web サービスのサンプル

10-26 ページの「サンプルの AQ キュー / トピックの宣言」で記述されているトピックに対して WebServicesAssembler によって公開される Web サービス操作を、例 10-15 にリストします。

この例では、publish 操作によりペイロードがトピックに入力されます。引数は、例 10-15 で示しているように、tns:TopicMessageUser などのペイロード・タイプです。このメッセージは、すべてのトピック・サブスクライバに受信されます。

publish2 操作は、ペイロードを受信者リストのすべてのサブスクライバに送信します。この操作は次の引数を取ります。

- tns:TopicMessageUser 型の payload
- String 配列型の recipients

publish3 操作は、ペイロードをトピックにブロードキャストします。この操作は次の引数を取ります。

- payload: 送信するメッセージ。
- deliveryMode（xsd:int 型）: 指定できるインタフェース値は javax.jms.DeliveryMode.PERSISTENT または javax.jms.DeliveryMode.NON_PERSISTENT のどちらかです。ただし、このリリースでは DeliveryMode.PERSISTENT のみがサポートされます。インタフェース javax.jms.DeliveryMode は JMS API のものです。
- priority（xsd:int 型）: メッセージの優先度を指定します。0 から 9 の値を指定できます。0 が優先度最低、9 が最高になります。
- timeToLive（xsd:long 型）: メッセージの寿命をミリ秒単位で指定します。0 は無制限を意味します。

receive 操作は、受信者に送信されるメッセージを戻します。この操作は 1 つの引数 receiver を取ります。

receiveNoWait 操作は、指定された受信者に待機なしで送信されるメッセージを戻します。

receive2 操作は、指定された受信者に送信されるフィルタ処理済メッセージを戻します。この操作は次の引数を取ります。

- receiver（xsd:string 型）: フィルタ処理済メッセージの受信者。
- selector（xsd:string 型）: AQ 規則で指定されているフィルタ条件。

receive3 操作は、指定された受信者に対するフィルタ済のペイロードを戻します。この操作は次の引数を取ります。

- receiver（xsd:string 型）: フィルタ処理済メッセージの受信者。
- selector（xsd:string 型）: AQ 規則で指定されているフィルタ条件。
- timeout（xsd:long 型）: 操作のタイムアウト時間をミリ秒単位で指定します。0 はタイムアウトなしを意味します。

subscribe 操作は、ユーザーをトピックにサブスクライブします。Web サービスのサポート基盤となる接続には、コンシューマをサブスクライブするために必要な権限が付与されている必要があります。権限がない場合は、この操作は無効です。

unsubscribe 操作は、ユーザーをトピックからサブスクライブ解除します。この場合も、Web サービスのサポート基盤となる接続には、コンシューマをサブスクライブ解除するために必要な権限が付与されている必要があります。権限がない場合は、この操作は無効です。

関連資料 :

コンシューマのサブスクライブおよびサブスクライブ解除に必要な権限の詳細は、『Oracle Streams Advanced Queuing Java API Reference』を参照してください。

例 10-15 トピックに対して公開される Web サービス操作

```
<operation name="publish">
  <input message="tns:SampleTopic_publish" />
  <output message="tns:SampleTopic_publishResponse" />
  </operation>
<operation name="publish2">
  <input message="tns:SampleTopic_publish2" />
  <output message="tns:SampleTopic_publish2Response" />
  </operation>
<operation name="publish3">
  <input message="tns:SampleTopic_publish3" />
  <output message="tns:SampleTopic_publish3Response" />
  </operation>
<operation name="receive">
  <input message="tns:SampleTopic_receive" />
  <output message="tns:SampleTopic_receiveResponse" />
  </operation>
<operation name="receive2">
  <input message="tns:SampleTopic_receive2" />
  <output message="tns:SampleTopic_receive2Response" />
  </operation>
<operation name="receive3">
  <input message="tns:SampleTopic_receive3" />
  <output message="tns:SampleTopic_receive3Response" />
  </operation>
<operation name="receiveNoWait">
  <input message="tns:SampleTopic_receiveNoWait" />
  <output message="tns:SampleTopic_receiveNoWaitResponse" />
  </operation>
<complexType name="publish">
<sequence>
  <element name="TopicMessageUser_1" type="tns:TopicMessageUser" nillable="true" />
  </sequence>
  </complexType>
```

```
<complexType name="TopicMessageUser">
<sequence>
  <element name="text" type="string" nillable="true" />
  <element name="subject" type="string" nillable="true" />
  </sequence>
  </complexType>
<complexType name="publishResponse">
  <sequence />
  </complexType>
<complexType name="publish2">
<sequence>
  <element name="TopicMessageUser_1" type="tns:TopicMessageUser" nillable="true" />
  <element name="arrayOfString_2" type="string" nillable="true" minOccurs="0"
maxOccurs="unbounded" />
  </sequence>
  </complexType>
<complexType name="publish2Response">
  <sequence />
  </complexType>
<complexType name="publish3">
<sequence>
  <element name="TopicMessageUser_1" type="tns:TopicMessageUser" nillable="true" />
  <element name="int_2" type="int" />
  <element name="int_3" type="int" />
  <element name="long_4" type="long" />
  </sequence>
  </complexType>
<complexType name="publish3Response">
  <sequence />
  </complexType>
<complexType name="receive">
<sequence>
  <element name="String_1" type="string" nillable="true" />
  </sequence>
  </complexType>
<complexType name="receiveResponse">
<sequence>
  <element name="result" type="tns:TopicMessageUser" nillable="true" />
  </sequence>
  </complexType>
<complexType name="receive2">
<sequence>
  <element name="String_1" type="string" nillable="true" />
  <element name="String_2" type="string" nillable="true" />
  </sequence>
  </complexType>
<complexType name="receive2Response">
<sequence>
  <element name="result" type="tns:TopicMessageUser" nillable="true" />
  </sequence>
  </complexType>
<complexType name="receive3">
<sequence>
  <element name="String_1" type="string" nillable="true" />
  <element name="String_2" type="string" nillable="true" />
  <element name="long_3" type="long" />
  </sequence>
  </complexType>
<complexType name="receive3Response">
<sequence>
  <element name="result" type="tns:TopicMessageUser" nillable="true" />
  </sequence>
  </complexType>
```

```
<complexType name="receiveNoWait">
<sequence>
  <element name="String_1" type="string" nillable="true" />
  </sequence>
  </complexType>
<complexType name="receiveNoWaitResponse">
<sequence>
  <element name="result" type="tns:TopicMessageUser" nillable="true" />
  </sequence>
  </complexType>
```

Web サービスとして公開されている AQ キューにクライアント・コードからアクセスする方法

例 10-16 は、公開されている Web サービスにアクセスする JAX-RPC Web サービス・クライアントのサンプルです。10-28 ページの「WebServicesAssembler で生成されるトピックに対する Web サービスのサンプル」は、キューとトピックによって公開される操作を示しています。

例 10-16 Web サービスとして公開される AQ キューにアクセスするクライアント・コード

```
SampleQueuePortClient queue = new SampleQueuePortClient();
     QueueMessageUser m;
     queue.send(new QueueMessageUser( "star chopper", "sample 1"));
     queue.send(new QueueMessageUser("easy blocks", "sample 2"));
     queue.send(new QueueMessageUser("back to future", "sample 3"));
     m = queue.receive();
     while (m != null) {
         System.out.println("Message received from SampleQueue: " + m.getSubject() + ":
" + m.getText());
         m = queue.receiveNoWait();
          }
```

このクライアントは次のレスポンスを戻します。

```
Message received from SampleQueue: sample 1: star chopper
Message received from SampleQueue: sample 2: easy blocks
Message received from SampleQueue: sample 3: back to future
```

JMS を使用した Oracle AQ Queue にアクセスする方法

デフォルトでは、Web サービス・インタフェース・コードはデータソースを使用して JDBC 接続を取得します。次に、この接続を使用してデータベース内のキューにアクセスします。

JDBC のかわりに JMS を使用してキューにアクセスすることもできます。WebServicesAssembler ツールには、`aqAssemble` コマンドに対して次のような特別な引数が用意されており、これらを使用して JMS キュー・インスタンスを使用することで、公開される Oracle AQ にアクセスできます。

- aqConnectionLocation: 公開される AQ に接続する Oracle Streams AQ JMS キュー・コネクションの JNDI 位置
- aqConnectionFactoryLocation: 公開される AQ に対する Oracle Streams AQ JMS キューのコネクション・ファクトリの JNDI 位置

`aqAssemble` コマンドで `dataSource` 引数を指定するかわりに、前述のパラメータのいずれかを指定できます。Web サービスの実行時に、JMS キューが JDBC ベースのキューのかわりに使用されます。

サーバー・サイド Java クラスを Web サービスとしてアセンブルする方法

dbJavaAssemble コマンドを使用して、Oracle データベース内の Java VM の Java クラスを起動する Web サービスを生成します。Web サービス操作として公開できるのは、静的メソッドまたはインスタンス・メソッドです。インスタンス・メソッドは、セッションのデフォルト・インスタンスまたはシングルトン・インスタンスを介して起動できます。

公開する Java クラスには、次のパラメータ型および戻り型を指定できます。

- プリミティブ型（char を除く）
- シリアライズ可能型（つまり、java.io.Serializable を実装するクラス）
- サポートされている型の属性を持つ JavaBeans
- JDBC 型、つまり oracle.sql.* 型
- サポートされている型の配列

この項の内容は、次のとおりです。

前提条件

開始する前に、次の情報を用意してください。

- サーバー・サイド Java クラスの完全修飾クラス名
- コード生成時に使用されるデータベース接続 URL
- コード生成時に使用される、Java クラスを搭載するスキーマの名前およびパスワード
- Web サービス・アプリケーションの名前
- 実行時に使用されるデータソース JNDI 名

サーバー・サイド Java クラスからの Web サービスをアセンブルする手順

次の手順は、WebServicesAssembler を使用してサーバー・サイド Java クラスから Web サービスをアセンブルする方法を示しています。

1. 「前提条件」の項で説明した情報を、WebServicesAssembler の dbJavaAssemble コマンドに入力します。

   例を示します。次の dbJavaAssemble コマンドでは、サーバー・サイド・クラス oracle.sqlj.checker.JdbcVersion は SQLJ サーバー・サイド・トランスレータに含まれています。このコマンドは、クラス javacallin.ear に対する Web サービス・アプリケーションをアセンブルします。また、PL/SQL ラッパーおよび Java ストアド・プロシージャのラッパーも生成します。Java ストアド・プロシージャのラッパーの目的は、サーバー・サイド Java クラスのシグネチャ型を、PL/SQL ストアド・プロシージャに公開できる型に変換することです。sysUser 引数が宣言されているため、WebServicesAssembler により、生成されたラッパーはデータベースに自動的にロードされます。

コマンドライン :

```
java -jar wsa.jar
  -dbJavaAssemble
  -appName javacallin
  -dbJavaClassName oracle.sqlj.checker.JdbcVersion
  -dbConnection jdbc:oracle:thin:@stacd15:1521:lsqlj
  -dataSource jdbc/OracleManagedDS
  -dbUser scott/tiger
  -sysUser sys/knl_test7
```

Ant タスク :

```
<oracle:dbJavaAssemble
   appName="javacallin"
   dbJavaClassName="oracle.sqlj.checker.JdbcVersion"
   dbConnection="jdbc:oracle:thin:@stacd15:1521:lsqlj"
   dataSource="jdbc/OracleManagedDS"
   dbUser="scott/tiger"
   sysUser="sys/knl_test7"
/>
```

このコマンドおよび Ant タスクの説明 :

- `dbJavaAssemble`: Oracle データベースにある Java VM の Java クラスから Web サービスを生成します。このコマンドを使用するには、データベースに接続する必要があります。18-12 ページの「dbJavaAssemble」を参照してください。
- `appName`: アプリケーションの名前を指定します。この名前は、通常 `context` や `uri` などの他の引数のベース値として使用されます。18-40 ページの「appName」を参照してください。
- `dbJavaClassName`: Web サービスとして公開するサーバー・サイド Java クラスの名前を指定します。18-53 ページの「dbJavaClassName」を参照してください。
- `dbConnection`: データベースの JDBC URL を指定します。18-53 ページの「dbConnection」を参照してください。
- `dataSource`: 実行時に Web サービスによって使用されるデータソースの JNDI 位置を指定します。18-53 ページの「dataSource」を参照してください。
- `dbUser`: *user*/*password* の形式でデータベースのスキーマとパスワードを指定します。18-53 ページの「dbUser」を参照してください。
- `sysUser`: `SYS` 権限のあるユーザーの名前とパスワードを、`dbSysUser/syspassword` の形式で指定します。この引数を使用すると、コード生成時に PL/SQL および Java ラッパー・コードがデータベースに自動的にインストールされます。18-55 ページの「sysUser」を参照してください。

実行時には、Web サービスのコードが JDBC を使用して PL/SQL ラッパーを起動し、ここから Java ストアド・プロシージャ・ラッパーが起動され、さらに最終的に当該のサーバー・サイド・クラスが起動されます。10-35 ページの例 10-18 に、このコマンドで生成される Web サービス操作をいくつか示します。

2. サービスを OC4J の実行中インスタンスにデプロイし、アプリケーションをバインドします。

このステップは、ステップ 1 に指定されているデータソースが OC4J のインスタンスにインストールされていることを前提にしています。

次のコマンドラインは、サンプルのデプロイおよびバインドのコマンドです。

```
% java -jar <J2EE_HOME>/admin_client.jar deployer:oc4j:localhost:port admin welcome
-deploy -file dist/javacallin.ear -deploymentName javacallin

% java -jar <J2EE_HOME>/admin_client.jar deployer:oc4j:localhost:port admin welcome
-bindWebApp javacallin javacallin-web default-web-site /javacallin
```

この例において、<*J2EE_HOME*> は、J2EE のインストール・ディレクトリです。

EAR ファイルのデプロイの詳細は、『Oracle Containers for J2EE デプロイメント・ガイド』を参照してください。

3. （オプション）デプロイが成功したことを確認します。

   OracleAS Web Services では、デプロイされた Web サービスごとに Web サービス・テスト・ページが提供されます。Web サービス・テスト・ページへのアクセスおよびその使用方法の詳細は、13-7 ページの「Web サービス・テスト・ページの使用方法」を参照してください。

4. クライアント・サイド・コードを生成します。

   - J2SE 環境の場合、genProxy コマンドを使用して WebServicesAssembler ツールを実行し、J2SE Web サービス・クライアントのスタブ（クライアント・プロキシ）を生成します。J2SE 環境向けのクライアント・サイド・コードの生成およびアセンブルの詳細は、第 15 章「J2SE Web サービス・クライアントのアセンブル」を参照してください。
   - J2EE 環境の場合、genInterface コマンドを使用して WebServicesAssembler ツールを実行し、J2EE Web サービス・クライアント用の JAX-RPC マッピング・ファイルおよびサービス・エンドポイント・インタフェースを生成します。クライアント・サイド・コードの生成およびアセンブルの詳細は、第 14 章「J2EE Web サービス・クライアントのアセンブル」を参照してください。

   たとえば、次のコマンドは genProxy を使用して J2SE クライアントのコードを生成します。

```
% java -jar wsa.jar -genProxy
                    -wsdl http://locahost:8888/javacallin/javacallin?WSDL
                    -output build/src/client
                    -mappingFileName ./mapping.xml
                    -packageName oracle.demo.db.queue.stub
                    -unwrapParameters true
```

   このコマンドの説明：

   - genProxy: J2SE Web サービス・クライアントから使用可能な静的プロキシ・スタブを作成します。18-32 ページの「genProxy」を参照してください。
   - wsdl: WSDL ドキュメントの絶対ファイル・パス、相対ファイル・パスまたは URL を指定します。18-64 ページの「wsdl」を参照してください。
   - output: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
   - mappingFileName: JAX-RPC マッピング・ファイルを指すファイル位置を指定します。18-46 ページの「mappingFileName」を参照してください。
   - packageName: JAX-RPC マッピング・ファイルにパッケージ名が宣言されていない場合に、生成されたクラス用に使用されるパッケージ名を指定します。18-47 ページの「packageName」を参照してください。
   - unwrapParameters: この引数は document-literal 操作に対してのみ設定でき、その他のメッセージ書式では無視されます。unwrapParameters を true に設定すると（デフォルト）、戻り型とレスポンス・タイプがアンラップされます。通常はこの方が使いやすくなります。特にタイプが単純な場合はそうです。18-69 ページの「unwrapParameters」を参照してください。

5. クライアントをコンパイルおよび実行します。

   クライアントをコンパイルする前に、クラスパス上にある適切な JAR をリストします。クライアントのクラスパスで使用可能なすべての JAR ファイルのリストについては、表 A-2「クライアント・サイド・プロキシを使用するクライアントのクラスパス・コンポーネント」を参照してください。個々の JAR をリストするかわりに、クライアントのクラスパスにクライアント側の JAR として wsclient_extended.jar を含めることができます。この JAR ファイルには、Web サービス・クライアントをコンパイルおよび実行するために必要なすべてのクラスが含まれます。これらのクラスは、表 A-2 にリストされている各 JAR ファイルに含まれているものです。wsclient_extended.jar およびクライアントのク

ラスパスの詳細は、A-3 ページの「Web サービス・プロキシのクラスパスの設定」を参照してください。

次のコマンドラインは、サンプルのコンパイルおよび実行のコマンドです。

```
% javac -classpath path
<ORACLE_HOME>/webservices/lib/wsclient_extended.jar:
:<ORACLE_HOME>/webservices/lib/jaxrpc-api.jar JavacallinClient.java

% java -classpath path
<ORACLE_HOME>/webservices/lib/wsclient_extended.jar:
<ORACLE_HOME>/webservices/lib/jaxrpc-api.jar:
<J2EE_HOME>/lib/jax-qname-namespace.jar:
<J2EE_HOME>/lib/activation.jar:<J2EE_HOME>/lib/mail.jar:
<J2EE_HOME>/lib/http_client.jar:
<ORACLE_HOME>/webservices/lib/commons-logging.jar:
<ORACLE_HOME>/lib/xmlparserv2.jar JavacallinClient
```

この例において、*<J2EE_HOME>* は J2EE のインストール・ディレクトリであり、*<ORACLE_HOME>* は OC4J のインストール・ディレクトリです。

サンプルのサーバー・サイド Java クラス

例 10-17 には、Oracle9*i* および 10*g* データベースとともに配布されているサーバー・サイド Java クラス `oracle.sqlj.checker.JdbcVersion` の 2 つの API が示されています。`dbJavaAssemble` コマンドはこれらの API を Web サービスとして公開します。

例 10-17 サンプルのサーバー・サイド Java クラス

```
public class oracle.sqlj.checker.JdbcVersion extends java.lang.Object {
    public static int getDriverMajorVersion();
    public static int getDriverMinorVersion();
    ...
}
```

サーバー・サイド Java クラスから生成される Web サービス操作のサンプル

例 10-18 の WSDL フラグメントは、例 10-17 の `JdbcVersion` API の `getDriverMajorVersion` および `getDriverMinorVersion` に対して生成された Web サービス操作です。

例 10-18 サーバー・サイド Java クラスに対して生成される操作を示す WSDL フラグメント

```
<complexType name="getDriverMajorVersion">
  <sequence />
  </complexType>
<complexType name="getDriverMajorVersionResponse">
<sequence>
  <element name="result" type="decimal" nillable="true" />
  </sequence>
  </complexType>
<complexType name="getDriverMinorVersion">
  <sequence />
  </complexType>
<complexType name="getDriverMinorVersionResponse">
<sequence>
  <element name="result" type="decimal" nillable="true" />
  </sequence>
  </complexType>

<portType name="JdbcVersion">
<operation name="getDriverMajorVersion">
  <input message="tns:JdbcVersion_getDriverMajorVersion" />
  <output message="tns:JdbcVersion_getDriverMajorVersionResponse" />
```

```
  </operation>
<operation name="getDriverMinorVersion">
  <input message="tns:JdbcVersion_getDriverMinorVersion" />
  <output message="tns:JdbcVersion_getDriverMinorVersionResponse" />
  </operation>
  </portType>
```

注意： サーバー・サイド Java クラスは、Web サービスから起動せずに、JDBC から起動することもできます。JDBC から起動する場合、『Oracle Database JPublisher ユーザーズ・ガイド』で、データベース・サーバー・サイド Java を起動するプロキシ・クラスの生成方法を参照してください。

データベースにおける Web サービス・クライアントの開発

この項の内容は、次のとおりです。

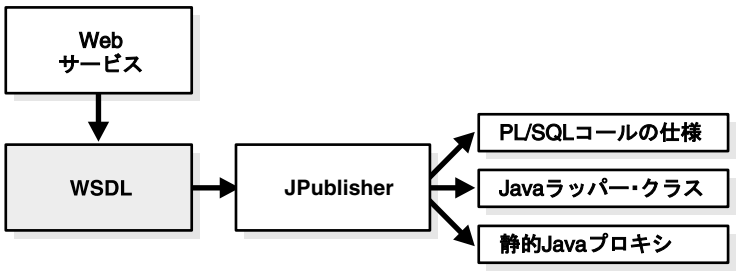
Web サービスのコールアウトの概要

Oracle JPublisher は、PL/SQL および Java Web サービスのコールアウトを、必要なスタブ・コードの作成によってサポートしています。これにより、データベース内で PL/SQL または Java を使用して Web サービス・クライアントを起動できるようになります。Web サービスのコールアウト用に、WSDL のファイルすなわち位置を Oracle JPublisher に指定します。Oracle JPublisher により、PL/SQL ラッパー、および PL/SQL ラッパーを実装する必要なデータベース・サーバー・サイド Java クラスが生成されます。生成される PL/SQL ラッパーには、各 Web サービス操作用の PL/SQL プロシージャまたはファンクションが格納されます。

オプションとして、Oracle JPublisher には、Java クライアント・プロキシのみを生成する機能もあります。これらの Java クライアント・プロキシは、データベースの Java コードによる Web サービスのコールアウトに使用できます。

図 10-3 に、Oracle JPublisher が生成可能なスタブ・コードを示します。

図 10-3 Web サービスのコールアウト・スタブの作成



注意： 実行時にのみ使用可能な WSDL をベースにした外部 Web サービスの起動を動的に構成する必要がある場合は、JAX-RPC Dynamic Invocation Interface API for Java または PL/SQL `UTL_DBWS` パッケージを使用します。

Oracle JPublisher が生成するクライアント・プロキシは、OracleAS Web Services 10.1.3 の Java プロキシに対して生成される、単純化されたクライアント・コードをベースにしています。したがって、Oracle JPublisher が生成する Java および PL/SQL クライアントは OracleAS Web Services 10.1.3.1 によって完全にサポートされます。さらに、Oracle JPublisher は、OracleAS Web Services 9.0.4 スタイルの Web サービス・クライアントも生成できます。

Web サービスのコールアウトには、これらのユーティリティおよびツールが必要です。

- データベースのリリースが 9.2 以降
- データベース Web サービスをコールアウトするユーティリティ

  指示に従って、JAR および SQL ファイルをデータベースにロードします。これらのユーティリティは Oracle Database Web Services の Web サイトから入手可能です。

  `http://www.oracle.com/technology/tech/webservices/database.html`

- Oracle JPublisher 10*g*

  Oracle JPublisher をインストールしていない場合は、JDBC、SQLJ および Oracle JPublisher のダウンロード Web サイトから入手できます。

  `http://www.oracle.com/technology/tech/java/java_db/index.html`

Oracle JPublisher のこれらのオプションは、Web サービスのコールアウトには必須です。

- `proxywsdl`: 起動する Web サービスに対する WSDL ファイルの URL
- `user`: PL/SQL ラッパーの生成対象となるデータベース・スキーマ（およびパスワード）

これらは Oracle JPublisher のオプション・パラメータです。

- `httpproxy`: WSDL ファイルにアクセスするための HTTP プロキシ・ホストおよびポート
- `sysuser`: SYSDBA 権限を持つデータベース・ユーザー（およびパスワード）
- `proxyopts`: `proxywsdl` 固有のオプションのリスト
- `dir`: 生成されたすべてのファイルが格納されるディレクトリ

`sysUser` 引数を使用することで、Oracle JPublisher が生成済ファイルをデータベースにロードできるようになります。この引数が宣言されていない場合は、PL/SQL から Web サービスを起動するために、生成済ファイルを手動でデータベースにロードする必要があります。

関連資料：

Web サービスのコールアウトに関連したオプション（`proxywsdl`、`proxyopts` および `httpproxy` など）や例については、『Oracle Database JPublisher ユーザーズ・ガイド』を参照してください。

データベースから Web サービスをコールする方法

Web サービス・コールアウト・ユーティリティを使用すると、クライアントはデータベースから Web サービスにアクセスできます。Web サービス・コールアウトは、データベース内の PL/SQL クライアント、SQL 文および Java で使用できます。このユーティリティは、Oracle JPublisher 10*g* リリース 2（10.2）および OracleAS Web Services 10*g* リリース 3（10.1.3.1）が基になっています。

ターゲットが Oracle Database 10*g*（リリース 10.1 または 10.2）か、Oracle Database 10*g* より前かによって、2 つのバージョンのユーティリティを使用できます。

コールアウト・ユーティリティは、次の Web サイトの「Database as Web Services consumer: Calling-out external Web services」から入手できます。

`http://www.oracle.com/technology/sample_code/tech/java/jsp/dbwebservices.html`

次の各項では、データベース内のクライアントが Web サービス・コールアウトを実行する方法について説明します。

- 静的プロキシおよび JPublisher を使用して Web サービス・コールアウトを実行する方法
- DII および SYS.UTL_DBWS ユーティリティを使用して Web サービス・コールアウトを実行する方法

静的プロキシおよび JPublisher を使用して Web サービス・コールアウトを実行する方法

Oracle JPublisher のコマンドライン・オプション -proxywsdl を使用すると、Web サービスの WSDL ファイルからデータベース・サイドの Java ラッパーおよび PL/SQL ラッパーを生成できます。JPublisher が Web サービス・クライアント用のラッパーを生成してデータベースにロードするには、dbwsa.jar ファイルと dbwsclient.jar ファイルが、それぞれ、クラスパス内およびデータベース内部に存在している必要があります。

次の手順に従って、Oracle JPublisher がサポートする Web サービス・コールアウト用の環境とデータベースを設定します。この手順は、1 回のみ実行する必要があります。

1. まだシステムにない場合は、Oracle JPublisher 10*g* リリース 2（10.2）をダウンロードしてインストールします。

   Oracle JPublisher のリリースは、次の Web サイトから入手できます。

   ```
   http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/
   index.html
   ```

2. dbwsa.jar を、ディレクトリ ORACLE_HOME¥sqlj¥lib（Windows の場合）または ORACLE_HOME/sqlj/lib（Solaris の場合）に追加します。

3. Java VM および Java コンパイラとして適切な JDK を設定します。

   JDK のバージョンは、ターゲット・データベースの Java VM と同じである必要があります。

   - Oracle Database 10*g*（リリース 10.1 または 10.2）の場合は、JDK 1.4 を使用します。
   - Oracle Database 9.2 の場合は、JDK 1.3 を使用します。

4. dbwsa.jar ファイルを、クラスパス環境変数に追加します。

5. SYS スキーマ、または Web サービス・クライアントが起動されたスキーマに、dbwsclient.jar ファイルをロードします。

   たとえば、次の loadjava コマンドは、dbwsclient.jar ファイルを SYS スキーマにロードします。

   ```
   %loadjava -u sys/change_on_install -r -v -f -s -grant public -noverify -genmissing
   dbwsclient.jar
   ```

   次の loadjava コマンドは、dbwsclient.jar ファイルを特定のスキーマにロードする方法を示しています。

   ```
   % loadjava -u scott/tiger -r -v -f -noverify -genmissing dbwsclient.jar
   ```

例

次の例は、Web サービス・クライアント用の Java ラッパーおよび PL/SQL ラッパーを生成し、SQL 文を使用してそれを起動する方法を示しています。この例では、次の一般的な手順に従います。

1. 起動する Web サービスを識別します。

2. 適切なオプションを指定して Oracle JPublisher をコールし、クライアント・プロキシ、PL/SQL ラッパーおよび Java ラッパーを生成して、データベースにそれらをロードします。

   これを行うための Oracle JPublisher コマンドは、必須の -proxywsdl オプションと -user オプションを含みます。また、オプションの -endpoint、-httpproxy、-sysuser、-dir および -proxyopts の各オプションを含む場合もあります。次に例を示します。

   ```
   % jpub -user=username/password -sysuser=superuser_name/superuser_password
   -proxywsdl=WSDL_URL -endpoint=Web_services_endpoint
   ```

   Web サービスがすでに http://localhost:8888/javacallout/javacallout にデプロイされているものとします。

次のコマンドは、Web サービス・クライアントおよびその Java ラッパーと PL/SQL ラッパーをサブディレクトリ tmp に作成した後、データベースにラッパーをロードします。

```
% jpub -user scott/tiger -sysuser sys/change_on_install
-proxywsdl=sample/javacallout.wsdl
-endpoint=http://localhost:8888/javacallout/javacallout -dir=tmp
```

このコマンドは次の出力を生成します。

```
tmp/HelloServiceEJBJPub.java
tmp/plsql_wrapper.sql
tmp/plsql_dropper.sql
tmp/plsql_grant.sql
tmp/plsql_revoke.sql
Executing tmp/plsql_dropper.sql
Executing tmp/plsql_wrapper.sql
Executing tmp/plsql_grant.sql
Loading tmp/plsql_proxy.jar
```

3. データベースの内部から Web サービスを起動します。

   tmp/plsql_wrapper.sql で提供されている PL/SQL ファンクションを起動できます。各 PL/SQL ファンクションは、Web サービスの操作に対応しています。たとえば、次のエンドポイントで Web サービスを利用できるものとします。

   ```
   http://localhost:8888/javacallout/javacallout
   ```

   この場合は、次の SQL コマンドを発行できます。

   ```
   SQL> select jpub_plsql_wrapper.sayhello('hello') from dual;
   ```

   コマンドは次の出力を戻します。

   ```
   JPUB_PLSQL_WRAPPER.SAYHELLO('HELLO')
   -----------------------------------
   HELLO!! You just said :hello
   ```

Oracle JPublisher のコールアウト・サポートの詳細は、『Oracle Database JPublisher ユーザーズ・ガイド』の Dynamic Invocation Interface と SYS.UTL_DBWS ユーティリティを使用した Web サービス・コールアウトに関する項を参照してください。この資料は、次の Web サイトから入手できます。

```
http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/index.html
```

DII および SYS.UTL_DBWS ユーティリティを使用して Web サービス・コールアウトを実行する方法

PL/SQL ユーザーは、PL/SQL のパッケージ SYS.UTL_DBWS を使用して Web サービスを起動できます。このパッケージでは、JAX-RPC Dynamic Invocation Interface（DII）API 用のラッパー・ファンクションが提供されています。

次の手順は、Oracle Database 10*g* リリース 1 以降に適用されます。この手順では、SYS.UTL_DBWS を使用する Web サービス・コールアウト用にデータベースを設定し、ダウンロードで提供される最新の機能でユーティリティを更新します。

1. 「静的プロキシおよび JPublisher を使用して Web サービス・コールアウトを実行する方法」の説明に従って、dbwsclient.jar ファイルをデータベースにロードします。
2. SYS として SQL スクリプト utl_dbws_decl.sql および utl_dbws_body.sql を実行します。

   SYS.UTL_DBWS を使用して Web サービスをコールできるようになります。

例

コールアウト・ユーティリティのダウンロードには、SQL スクリプト `samples/test-plsql-dii.sql` および `samples/test-plsql-dii2.sql` が含まれます。どちらのスクリプトも、`javacallout.wsdl` で定義されている `sayHello` 操作を起動します。2 つのスクリプトで WSDL ファイルとの通信を調べてください。どちらのスクリプトも、次の出力を生成します。`PL/SQL DII client return` という文章は、クライアント・コードによって生成されます。

```
PL/SQL DII client return HELLO!! You just said :hello
```

`SYS.UTL_DBWS` ユーティリティの詳細は、次の Web サイトから入手できる『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

```
http://www.oracle.com/technology/documentation/database10g.html
```

データベース・リソースを公開する Web サービスに対するツールのサポート

Oracle JDeveloper を使用すると、Oracle データベースに保存されている PL/SQL パッケージのプログラム・ユニットをベースにして、Web サービスを作成できます。Oracle JDeveloper の各ウィザードを使用すると、次のタスクを実行できます。

- PL/SQL パッケージのデータベースへの作成
- PL/SQL Web サービスの作成
- PL/SQL Web サービスのデプロイ
- Web サービスを使用するスタブの作成

Oracle JDeveloper を使用して PL/SQL パッケージ・ユニットを作成し、Web サービスとして公開する方法の詳細は、Oracle JDeveloper のオンライン・ヘルプを参照してください。

制限事項

F-8 ページの「データベース・リソースからの Web サービスの開発」を参照してください。

追加情報

詳細は、次を参照してください。

- テスト・ページを使用した Web サービス・デプロイのテスト方法は、第 13 章「Web サービス・デプロイのテスト」を参照してください。
- J2SE クライアントの構築方法は、第 15 章「J2SE Web サービス・クライアントのアセンブル」を参照してください。
- J2EE クライアントの構築方法は、第 14 章「J2EE Web サービス・クライアントのアセンブル」を参照してください。
- JAX-RPC ハンドラの詳細は、第 16 章「JAX-RPC ハンドラの使用方法」を参照してください。
- WebServicesAssembler ツールを使用した Web サービスのアセンブル方法は、第 18 章「WebServicesAssembler の使用方法」を参照してください。
- Web サービスのパッケージ化およびデプロイ方法は、第 19 章「Web サービスのパッケージ化およびデプロイ」を参照してください。
- クライアントのアセンブルに必要な JAR ファイルの詳細は、付録 A「Web サービス・クライアントの API および JAR」を参照してください。
- Web サービスの相互運用性の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「相互運用可能な Web サービスの実現」を参照してください。

- Web サービス・クライアントにおけるサービスのクオリティ機能の使用方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの管理」を参照してください。
- Web サービスへのセキュリティの追加方法は、『Oracle Application Server Web Services セキュリティ・ガイド』を参照してください。
- Web サービスへの信頼性の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの信頼性の確保」を参照してください。
- Web サービスへの監査およびロギング構成の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「監査メッセージおよびロギング・メッセージ」を参照してください。
- 標準外のデータ型の処理方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 値タイプのカスタム・シリアライズ」を参照してください。
- JAX-RPC マッピング・ファイルの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「JAX-RPC マッピング・ファイル記述子」を参照してください。
- OracleAS Web Services でサポートされているデータ型の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 型の XML および WSDL 型へのマッピング」を参照してください。
- Web サービス開発用の Oracle JDeveloper ツールのサポートの詳細は、Oracle JDeveloper のオンライン・ヘルプを参照してください。

11

注釈を使用した Web サービスのアセンブル

この章では、Java クラスから Web サービスを短時間で開発するための注釈の使用方法を説明します。Web サービスは、トランスポート・メカニズムとして HTTP または JMS を使用できます。注釈機能を使用できるのは、Java クラスからのボトムアップ方式で Web サービスを開発する場合のみです。

この章の内容は、次のとおりです。

OracleAS Web Services および J2SE 5.0 Web サービス注釈

標準の Java メカニズムで Web サービスを設計、開発およびデプロイするには、膨大な量の情報を提供する必要があります。たとえば、Java クラスをベースにしたサービスをデプロイするためには、そのクラス自体、関連するサービス・エンドポイント・インタフェース、および場合によってはメタデータ・ファイルが必要になります。

関連項目：

Java クラスを使用した Web サービスの生成の詳細は、第 7 章「Java クラスを使用した Web サービスのアセンブル」を参照してください。

Oracle Application Server Web Services は、J2SE 5.0 Web サービス注釈（Web Services Metadata for the Java Platform（JSR-181）仕様とも呼ばれます）をサポートしています。次の Web サイトから入手できるこの仕様では、Web サービスのプログラミングに使用する注釈付き Java 構文が定義されています。

```
http://www.jcp.org/en/jsr/detail?id=181
```

J2SE 5.0 Web サービス注釈を使用するメリットとデメリットを次のリストに記載します。

J2SE 5.0 Web サービス注釈を使用するメリット

- コミュニティの標準をベースにすることになります。
- J2SE 5.0 JDK でサポートされています。メタデータがクラス・ファイルに直接格納されるため、移植が可能になります。
- 言語レベルでの IDE のサポート。J2SE 5.0 JDK をサポートする IDE は、J2SE 5.0 Web サービス注釈を認識し、オートコンプリートや構文チェックなどの機能を提供できます。

J2SE 5.0 Web サービス注釈のデメリット

- J2SE 5.0 JDK の使用が必須になります。
- 制限事項の詳細は、F-9 ページの「注釈を使用した Web サービスのアセンブル」を参照してください。

注釈付き Java 構文を使用するには、J2SE 5.0 JDK が必要になります。また、J2SE 5.0 JDK には、注釈の定義方法、構文、および新しい注釈の作成方法など、言語レベルでの注釈のサポートを記述したドキュメントが付属しています。J2SE 5.0 JDK は、次の Web サイトから入手できます。

```
http://java.sun.com/j2se/1.5.0/download.jsp
```

J2SE 5.0 JDK には、Java メタデータ注釈という機能があり、固有のタグを Java クラスに追加できます。これらのタグは、クラスのバイト・コードにコンパイルされ、サード・パーティの注釈プロセッサからアクセス可能になります。Web Services Metadata for the Java Platform 仕様は、この機能を Web サービスのメタデータ・タグ用にまで拡張したものです。OracleAS Web Services はこの仕様をサポートしており、開発者は注釈を使用して Web サービス生成プロセスをカスタマイズできます。

OracleAS Web Services で Web サービス注釈を使用するには、J2SE 5.0 JDK の JVM が必要です。WebServicesAssembler ツールは、J2SE 5.0 注釈が含まれる Java クラスから Web サービスを生成できます。このツールを J2SE 5.0 JDK の JVM で使用すると、`assemble` および `genWsdl` コマンドが、指定された Java クラスの注釈を処理します。

EJB 3.0 仕様は、EJB 用のメタデータ・タグを追加できるように、Web Services Metadata for the Java Platform 仕様を拡張したものです。この仕様は、Web Services Metadata for the Java Platform のタグをサポートしており、EJB を Web サービスとして公開できるようにします。EJB 3.0 注釈は、EJB を含む EAR ファイルが動作中の Oracle Application Server にデプロイされたときに処理されます。セッション Bean で実装された、J2SE 5.0 JDK 注釈付きのインタフェースがすべて処理されます。WebServicesAssembler ツールは、EJB 3.0 注釈の処理には使用できない点に注意してください。

サポートされる注釈

次の各項では、OracleAS Web Services でサポートされている注釈について説明します。

- サポートされる J2SE 5.0 JDK 注釈
- Oracle による J2SE 5.0 JDK 注釈への追加

サポートされる J2SE 5.0 JDK 注釈

OracleAS Web Services では、Web Services Metadata for the Java Platform 仕様で記述されている注釈セットをすべて使用できます。表 11-1 は、これらの注釈の簡単な説明です。Web Services Metadata for the Java Platform 仕様では、これらの注釈についてのさらに詳しい情報が提供されています。

また、表 11-1 には、`@WebServiceRef` 注釈についての簡単な説明もあります。OracleAS Web Services では、`@WebServiceRef` 注釈を使用して、アプリケーション・クライアント・コンテナ内のバージョン 3.0 EJB、サーブレットまたは Java クラスから、Web サービス・エンドポイントを参照できます。また、`@WebServiceRef` 注釈のサポートには、`ejb-jar.xml` デプロイメント・ディスクリプタでの `service-ref` ターゲットのインジェクションのサポートも含まれます。この注釈の詳細は、Java API for XML-based Web Services 2.0（JSR-224）仕様を参照してください。

`http://www.jcp.org/en/jsr/detail?id=224`

表 11-1 OracleAS Web Services がサポートする J2SE 5.0 JDK 注釈

J2SE 5.0 JDK 注釈	説明
javax.jws.HandlerChain	Web サービスと外部で定義されているハンドラ・チェーンを関連付けます。
javax.jws.Oneway	特定の Web メソッドが入力メッセージのみを持ち、出力を持たないことを示します。
javax.jws.WebMethod	メソッドが Web サービス操作として公開されることを示します。
javax.jws.WebParam	Web サービスのメッセージ部分および XML 要素に対する個別のパラメータのマッピングをカスタマイズします。
javax.jws.WebResult	WSDL 部分および XML 部分に対する戻り値のマッピングをカスタマイズします。
javax.jws.WebService	Web サービスを実装している Java クラス、または Web サービス・インタフェースを定義している Java インタフェースを指定します。
javax.jws.soap.SOAPBinding	SOAP メッセージ・プロトコルに対する Web サービスのマッピングを指定します。
javax.jws.soap.SOAPMessageHandlers	Web サービスでビジネス・メソッドの前または後に実行する SOAP プロトコル・ハンドラのリストを指定します。
javax.xml.ws.WebServiceRef	アプリケーション・クライアント・コンテナ内のバージョン 3.0 EJB、Java クラスまたはサーブレットから Web サービス・エンドポイントへの参照を宣言します。 `WebServiceRef` 注釈のプロパティを使用すると、WebServicesAssembler が JAX-RPC マッピング・ファイルの名前を生成する方法を制御できます。詳細は、「WebServiceRef 注釈がマッピング・ファイル名を決定する方法」を参照してください。

OracleAS Web Services での WebServiceRef の使用

Java API for XML-based Web Services 2.0（JSR-224）仕様では、使用して、アプリケーション・クライアント・コンテナ内のバージョン 3.0 EJB、サーブレットまたは Java クラスから Web サービス・エンドポイントを参照するための、`@WebServiceRef` 注釈が定義されています。表 11-2 は、`@WebServiceRef` 注釈で使用できるプロパティです。

表 11-2 WebServiceRef 注釈のプロパティ

プロパティ	説明
mappedName	JNDI ルックアップのための固有名。
name	Web サービスの JNDI ルックアップ名。この値は、`java:comp/env` 名前空間で見つかります。
type	Java クラス・オブジェクトとしてのリソースの型。デフォルトの値は `javax.xml.rpc.Service` です。
value	Java クラス・オブジェクトとしてのサービスの型。デフォルトでは、インジェクション・ターゲットのクラスです。
wsdlLocation	Web サービスの WSDL 記述の URL。この注釈が存在する場合は、参照されている生成されたサービス・インタフェースの `@WebService` 注釈で指定されている WSDL URL をオーバーライドします。

次の各項では、OracleAS Web Services での `@WebServiceRef` 注釈の使用方法についてさらに詳しく説明します。

- WebServiceRef 注釈がマッピング・ファイル名を決定する方法
- WebServiceRef 注釈での複数ポートを持つ WSDL の指定
- WebServiceRef 注釈の例

WebServiceRef 注釈がマッピング・ファイル名を決定する方法 Java クラスまたはバージョン 3.0 EJB に存在する `@WebServiceRef` ごとに、JAX-RPC マッピング・ファイルがデプロイされるアーカイブに含まれる必要があります。マッピング・ファイルの生成は WebServicesAssembler が行いますが、Web サービスに対する適切な場所にマッピング・ファイルが存在することはユーザーが確認する必要があります。マッピング・ファイルが適切な場所にない場合は、手動でコピーする必要があります。

- Web サービスがバージョン 3.0 EJB に基づいている場合は、生成される JAX-RPC マッピング・ファイルは `META-INF` ディレクトリのすぐ下に存在する必要があります。
- Web サービスが Java クラスに基づいている場合は、生成される JAX-RPC マッピング・ファイルは `WEB-INF` ディレクトリのすぐ下に存在する必要があります。

WebServicesAssembler は、次の手順でマッピング・ファイルの名前を判別します。名前の割当てに成功するまで、この手順に従います。最後のステップが失敗すると、プロセス全体が失敗し、名前は割り当てられません。

1. インジェクション・ターゲット・クラスがサービス・エンドポイント・インタフェース（SEI）で、`@WebService` 注釈と `serviceName` プロパティが指定されている場合は、デプロイ・システムは次の名前でマッピング・ファイルのロードを試みます。

   `[@WebService.serviceName]Service-java-wsdl-mapping.xml`

   この例で、`[@WebService.serviceName]` は `serviceName` プロパティの値を表しています。

2. インジェクション・ターゲット・クラスがサービス・エンドポイント・インタフェースで、@WebServiceRef 注釈と serviceName プロパティが指定されている場合は、デプロイ・システムは次の名前でマッピング・ファイルのロードを試みます。

   [*@WebServiceRef.name*]Service-java-wsdl-mapping.xml

   この例で、[*@WebServiceRef.name*] は name プロパティの値を表します。

3. インジェクション・ターゲット・クラスがサービス・エンドポイント・インタフェースで、@WebServiceRef 注釈と wsdlLocation プロパティが指定されている場合は、マッピング・ファイルのベース名は、wsdlLocation プロパティで参照されている WSDL ファイルと同じです。

   [*base name of WSDL file*]Service-java-wsdl-mapping.xml

   たとえば、WSDL ファイルが MyService.wsdl という名前である場合、そのサービスに対するマッピング・ファイルは MyServiceService-java-wsdl-mapping.xml である必要があります。これは、genWSDL コマンドまたは genInterface コマンドを実行すると WebServicesAssembler が生成する名前と整合しています。したがって、WebServicesAssembler が生成するマッピング・ファイルをコピーすると、必然的に動作するはずです。

注意： wsdlLocation プロパティは、リモート WSDL を指し示すことはできません。WSDL ファイルは、マッピング・ファイルと同じアーカイブに存在する必要があります。

WebServiceRef 注釈での複数ポートを持つ WSDL の指定 Java API for XML-based Web Services 2.0 仕様によると、@WebServiceRef に対して指定されている WSDL が複数のポートを備えている場合は、その注釈に対するインジェクション・ターゲットは javax.xml.rpc.Service 型である必要があります。サービスがこのターゲットにインジェクトされると、getPort(QName portName, Class serviceEndpointInterface) をコールして、呼出し用に正確なポートを取得できます。このシグネチャでは、portName は使用する特定のポートの名前であり、serviceEndpointInterface はサービス・エンドポイント・インタフェース・クラス・ファイルの名前です。

WebServiceRef 注釈の例 ここでは、@WebServiceRef 注釈の使用例を示します。

例 11-1 では、サービス・エンドポイント・インタフェースに対する参照はフィールドにインジェクトされ、さらに setter メソッドにインジェクトされています。この場合、マッピング・ファイルの名前は SimpleService-java-wsdl-mapping.xml です。

例 11-1 フィールドおよび setter メソッドへの参照のインジェクト

```
@Stateless(name="WSRefBean")
public class StatelessEJBClient implements StatelessEJBClientInterface{


@WebServiceRef(name="Simple",wsdlLocation="META-INF/SimpleEJBServiceService.wsdl")
    private SimpleEJBService svc;

    private SimpleEJBService svc2;

    public String echoString(String str) throws RemoteException {
        if(svc == null){
            return "fail1";
        }
        if(svc2 == null){
            return "fail2";
        }
        String test = svc.echoString("Success");
        test = svc2.echoString(test);
        return str + test;
```

```
    }


@WebServiceRef(name="Simple",wsdlLocation="META-INF/SimpleEJBServiceService.wsdl")
    public void setSvc2(SimpleEJBService service){
        svc2 = service;
    }
}
```

例 11-2 は前の例とよく似ています。唯一違うところは、インジェクションが、実際のサービス・エンドポイント・インタフェースではなく javax.xml.Service オブジェクトに対して行われていることです。前に説明したように、wsdlLocation に基づいて、マッピング・ファイルは（EJB なので）META-INF に存在することが予想され、名前は SimpleService-java-wsdl-mapping.xml である必要があります。

例 11-2 javax.xml.Service オブジェクトへの参照のインジェクション

```
@Stateless(name="WSRefBean3")
public class StatelessEJBClient3 implements StatelessEJBClientInterface3{


@WebServiceRef(name="Simple",wsdlLocation="META-INF/SimpleEJBServiceService.wsdl")
    private Service svc;

    public String echoString(String test) throws RemoteException {
        if(svc == null){
            return "fail";
        }
        try{
            SimpleEJBService port =
(SimpleEJBService)svc.getPort(SimpleEJBService.class);
            return port.echoString(test);
        }catch(Exception ex){
            ex.printStackTrace();
            throw new RemoteException(ex.getMessage(),ex);
        }
    }
}
```

例 11-3 では、インジェクションに ejb-jar.xml デプロイメント・ディスクリプタを使用します。この技法を使用するときは、ejb-jar.xml で version="3.0" を指定する必要があります。このデプロイメント・ディスクリプタは、@WebServiceRef と組み合せて使用できます。

インジェクションに ejb-jar.xml デプロイメント・ディスクリプタを使用するときは、<jaxrpc-mapping-file> 要素でマッピング・ファイルを直接指定することもできます。この場合、マッピング・ファイルの名前は CustomSessionBeanService-java-wsdl-mapping.xml です。

例 11-3 ejb-jar.xml デプロイメント・ディスクリプタを使用する参照のインジェクション

```
<ejb-jar version="3.0">
    <enterprise-beans>
        <session>
            <ejb-name>WSRefBean2</ejb-name>
            <ejb-class>oracle.j2ee.tests.ejb.impl.StatelessEJBClient2</ejb-class>
            <session-type>Stateless</session-type>
            <service-ref>
                <service-ref-name>CustomSessionEJB</service-ref-name>
                <service-interface>javax.xml.rpc.Service</service-interface>

<service-ref-type>oracle.j2ee.tests.ejb.impl.CustomSession</service-ref-type>
                <wsdl-file>META-INF/CustomSessionBeanService.wsdl</wsdl-file>
```

```
<jaxrpc-mapping-file>META-INF/CustomSessionBeanService-java-wsdl-mapping.xml</jaxrpc-mapping-file>

                <injection-target>

<injection-target-class>oracle.j2ee.tests.ejb.impl.StatelessEJBClient2</injection-target-class>
                    <injection-target-name>svc</injection-target-name>
                </injection-target>
            </service-ref>
        </session>
    </enterprise-beans>
</ejb-jar>
```

Oracle による J2SE 5.0 JDK 注釈への追加

この項では、J2SE 5.0 JDK の Java メタデータ注釈機能で読取りおよび処理が可能な Oracle 固有の注釈を説明します。

- Deployment 注釈
- Schema 注釈
- WSIFEJBBinding 注釈

Deployment 注釈

`@Deployment` 注釈（`oracle.webservices.annotations.Deployment`）は、エンドポイント実装またはサービス・エンドポイント・インタフェースにデプロイ属性を設定するために使用します。

表 11-3 は、`@Deployment` 注釈のプロパティの説明です。すべてのプロパティはオプションです。

注意： 次の表のプロパティは、デプロイメント・ディスクリプタによって、または WebServicesAssembler で渡される引数値によって、オーバーライドされる可能性があります。詳細は、11-13 ページの「WebServicesAssembler で注釈値をオーバーライドする方法」を参照してください。

表 11-3 Deployment 注釈のプロパティ

プロパティ	データ型	説明
contextPath	String	この値は、Web サービスのデフォルトのコンテキストを指定します。各エンドポイント・クラスが `@Deployment` 注釈を持つ複数の Web サービスを WAR ファイルにパッケージする場合は、`contextPath` プロパティの値がすべてのサービスで同じである必要があります。これは、`contextPath` が Web サービスのデプロイに対してグローバルであるためです。Web サービス間で `contextPath` の値が異なる場合、各 Web サービスの動作は予測不可能になります。`contextPath` プロパティが 1 つのみ設定されている場合は、そのプロパティが、同じ EAR または WAR ファイル内にあるすべての Web サービスのルート・コンテキストになります。 このプロパティの値は、EAR ファイルでデプロイされる `oracle-webservices.xml` デプロイメント・ディスクリプタの `<context-root>` 要素に、有効な値を指定することでオーバーライドできます。デプロイメント・ディスクリプタ内で `<context-root>` を設定すると、`Deployment` 注釈を持つすべての Web サービスの `contextPath` プロパティをオーバーライドできます。 **デフォルト：** EAR ファイルの名前。

表 11-3 Deployment 注釈のプロパティ（続き）

プロパティ	データ型	説明
portName	String	この値は、生成される WSDL に設定される portName 要素を指定します。Web サービスごとに異なる portName 値を指定できます。 portName プロパティの値は、EAR ファイルでデプロイされる oracle-webservices.xml デプロイメント・ディスクリプタの <port-component [name]> 属性に値を入力することで、オーバーライドできます。 **デフォルト：**バージョン 3.0 EJB の場合は、この値は EJB の名前から取得されます。この名前は、EJB の @Stateless.name プロパティで指定できます。@Stateless.name プロパティが設定されていない場合、portName のデフォルト値は EJB の単純名です。 他のすべての Web サービスの場合、portName のデフォルト値は次のパターンで生成されます。 [*protocol*] [*binding_and_version*] たとえば、HttpSoap11 または HttpSoap12 のようになります。
restSupport	Boolean	このブール値は、サービスが REST Web サービスであるかどうかを示します。true の場合、ポートは REST スタイルの GET および POST のリクエストおよびレスポンスをサポートします。 **デフォルト：**false
uriPath	String	この値は、contextPath プロパティの値に追加されます。Web サービスごとに異なる uriPath 値を指定できます。 このプロパティの値は、対応する <port-component [name]> に対する <endpoint-address-uri> 要素に URI を指定することで、オーバーライドできます。これらの要素は、EAR ファイルでデプロイされる oracle-webservices.xml デプロイメント・ディスクリプタ内の同じサービス要素に指定しておく必要があります。 **注意：**web.xml デプロイメント・ディスクリプタの <url-pattern> の値と、uriPath の値が一致しない場合は、<url-pattern> の値が優先されます。 **デフォルト：**バージョン 3.0 EJB の場合、uriPath のデフォルトは @Stateless.name プロパティです。他の Web サービスの場合のデフォルトは、単純な（修飾されていない）クラス名です。

次に、@Deployment 注釈を使用するインタフェースの例を示します。この例の Web サービスは、デプロイされてから、URL http://$HOST:$PORT/ejb30/ejb30-simple によってアクセスできるようになります。

```
...
@WebService(name="CustomSession",
            targetNamespace="http://ejb.oracle.com/targetNamespace",
            serviceName="CustomSessionBeanService")
@Deployment(contextPath="ejb30",
            uriPath="ejb30-simple",
            portName="Custom")
public interface CustomSessionIntf extends Remote{
...
```

Schema 注釈

@Schema 注釈（oracle.webservices.annotations.Schema）を使用すると、WSDL でのスキーマ要素の生成方法を構成できます。これはオプションの注釈で、オプションのプロパティ elementFormDefault と targetNamespace のみを持ちます。

elementFormDefault プロパティを使用すると、WSDL の elementFormDefault 属性の値を変更できます。この属性の値は、ローカルに宣言されている要素を、インスタンス・ドキュメント内ではターゲット名前空間で修飾する必要があるかどうかを示します。この属性の値が unqualified である場合は、ローカルに宣言された要素をターゲット名前空間で修飾することはできません。この属性の値が qualified である場合は、ローカルに宣言された要素をターゲット名前空間で修飾する必要があります。

@Schema.elementFormDefault プロパティは、@Schema.ElementFormDefault.QUALIFIED と @Schema.ElementFormDefault.UNQUALIFIED という 2 つの値を持つ列挙です。デフォルトでは、OracleAS Web Services ツールは、WSDL 内のすべてのスキーマの elementFormDefault 属性を qualified に設定します。@Schema.elementFormDefault プロパティを使用することで、この動作を変更できます。

targetNamespace プロパティを使用すると、生成されるスキーマの targetNamespace 属性を設定できます。この名前空間は、すべての型に対するデフォルトになります。

表 11-4 は、@Schema 注釈のプロパティをまとめたものです。

> **注意：** 次の表のプロパティは、デプロイメント・ディスクリプタによって、または WebServicesAssembler で渡される引数値によって、オーバーライドされる可能性があります。詳細は、11-13 ページの「WebServicesAssembler で注釈値をオーバーライドする方法」を参照してください。

表 11-4 Schema 注釈のプロパティ

プロパティ	データ型	説明
elementFormDefault	enum、値は QUALIFIED および UNQUALIFIED	WSDL の elementFormDefault 属性の値を変更できます。このプロパティには、次の値を指定できます。 ■ QUALIFIED: WSDL の elementFormDefault 属性の値を qualified に変更します。ローカルに宣言された要素は、ターゲット名前空間で修飾されます。 ■ UNQUALIFIED: WSDL の elementFormDefault 属性の値を、unqualified に変更します。ローカルに宣言されたすべての要素は、ターゲット名前空間で修飾されません。 **デフォルト：** QUALIFIED
targetNamespace	String	生成されるスキーマの targetNamespace 属性を設定します。この名前空間は、型に対するデフォルトになります。 **デフォルト：** 空の文字列。ただし、その場合は、WSDL の同じ targetNamespace がデフォルトになります。

WSIFEJBBinding 注釈

@WSIFEJBBinding 注釈（oracle.webservices.annotations.@WSIFEJBBinding）は、バージョン 3.0 EJB 用の WSDL に Web Services Invocation Framework（WSIF）のバインディングを生成します。WSIF バインディングは、単一のポートに対してのみ適用されます。この注釈および関連するプロパティは、WSDL に拡張機能を追加し、Web サービスとして公開された 3.0 EJB に WSIF クライアントがアクセスできるようにします。

注意： OracleAS Web Services では、WSIF バインディングのみを定義する WSDL ドキュメントは使用できません。SOAP バインディングも定義する必要があります。

`WSIFEJBBinding` 注釈を使用して Web サービスをアセンブルするときは、常に `SoapBinding` 注釈を含め、適切なスタイルと使用に設定し、意図したメッセージ・エンコーディングを取得する必要があります。

関連資料：

『Oracle Application Server Web Services アドバンスト開発者ガイド』の次の項では、WSIF、EJB および WSIF クライアントについて詳細に説明されています。

- WSIF と EJB に関する項では、EJB および WSDL に対する WSIF 拡張に基づいて、Web サービスで WSIF の機能を使用する方法について説明されています。
- 「WSIF クライアントの記述」では、WSIF バインディングのあるサービス用のクライアントを作成する方法について説明されています。

表 11-5 は、`@WSIFEJBBinding` 注釈に対して指定できるプロパティです。`className` プロパティと `jndiName` プロパティは必須です。

注意： 次の表のプロパティは、デプロイメント・ディスクリプタによって、または WebServicesAssembler で渡される引数値によって、オーバーライドされる可能性があります。詳細は、11-13 ページの「WebServicesAssembler で注釈値をオーバーライドする方法」を参照してください。

表 11-5 WSIFEJBBinding 注釈のプロパティ

プロパティ	データ型	説明
className	String	（必須）EJB ホーム・インタフェースのクラス名を指定します。 **デフォルト：** WebServicesAssembler は、親 Ant タスクのクラス名を使用します（指定されている場合）。
initialContextFactory	String	初期コンテキストを提供するファクトリの名前を指定します。 **デフォルト：** `com.evermind.server.rmi.RMIInitialContextFactory` です。
jndiName	String	（必須）EJB の JNDI 名を指定します。 **デフォルト：** `@Stateless.name` を問い合せることで見つかった EJB の名前、または EJB クラスの単純名。
jndiProviderURL	String	JNDI プロバイダの URL を指定します。 **デフォルト：** 空の文字列です。
name	String	Web サービスでの WSIF ポートの名前を指定します。この名前を使用して、WSDL の `port` 要素で WSIF ポートを識別します。 **デフォルト：** なし。

次の例は、バージョン 3.0 EJB に対するインタフェース・ファイルで使用されている @WSIFEJBBinding 注釈を示しています。

```
...
@WebService(name="Hello",
            targetNamespace="http://ejb.oracle.com/targetNamespace",
            serviceName="CustomHelloBeanService")
@WSIFEJBBinding(
   className="oracle.demo.hello.HelloBean",
   jndiName="Hello",
   initialContextFactory="com.evermind.server.rmi.RMIInitialContextFactory",
   jndiProviderUrl="ormi://localhost:23791/ejb30wsif")
public interface CustomHelloBeanIntf extends Remote {
...
```

ファイルをコンパイルすると、WSDL での EJB WSIF バインディングと、次の値を持つ EJB ポートが生成されます。

```
...
<port name="WsifEJBPort" binding="tns:WsifEjbBinding">
    <ejb:address className="oracle.demo.hello.HelloBean"
        jndiName="Hello"
        initialContextFactory="com.evermind.server.rmi.RMIInitialContextFactory"
        jndiProviderURL="ormi://localhost:23791/ejb30wsif"/>
</port>
...
```

注釈の使用方法

この項の内容は、次のとおりです。

- Java クラスから Web サービスをアセンブルするために注釈を使用する手順
- バージョン 3.0 の EJB から Web サービスをアセンブルするために注釈を使用する手順
- 注釈をオーバーライドする方法
- 注釈付きの Java ファイルのサンプル

Java クラスから Web サービスをアセンブルするために注釈を使用する手順

J2SE 5.0 注釈付きの Java クラスから Web サービスを生成する手順は、次のとおりです。

1. Web サービス・メタデータ注釈を実装クラスに追加します。

   例 11-4 は、注釈付きの Java ファイルのサンプルです。

   a. 少なくとも、@WebService 注釈がクラス内に存在する必要があります。

   b. サービス・エンドポイント・インタフェースを参照する場合は、実装クラス内で @WebService 注釈に endpointInterface プロパティを追加します。サービス・エンドポイント・インタフェースにも @WebService 注釈がある場合は、各注釈はこちらの開始ポイントで処理されるようになります。

   c. @WebMethod の注釈を、Web サービスで公開する各メソッドに追加します。サービス・エンドポイント・インタフェースに注釈を追加すると、そのすべてのメソッドが @WebMethod 注釈の有無に関係なく公開される点に注意してください。

注意： サービス・エンドポイント・インタフェースに `@WebService` 注釈を入力し、`@WebMethod` 注釈を入力しないと、入力した `@WebMethod` 注釈の数に関係なく、すべてのメソッドが公開されます。

実装 Bean に `@WebService` 注釈を入力し、`@WebMethod` 注釈を入力しないと、少なくとも 1 つのメソッドを公開する必要があるため、検証例外が発生します。Web Services Metadata for the Java Platform 仕様の 4.1 項と 4.2 項を参照してください。

---

2. 注釈付きのクラスをコンパイルします。

   クラスは JDK 5.0 対応のコンパイラでコンパイルする必要があります。

3. WebServicesAssembler の `assemble` コマンドを使用して、Web サービス・アーチファクト（WSDL、デプロイ・ディスクリプタなど）を生成します。

   a. 実装クラスは、`assemble` コマンドの `className` 引数の値として指定します。これはすべての注釈がサービス・エンドポイント・インタフェースにある場合も同様です。注釈として実装クラスへの追加が必須なのは、`@WebService` です。

   b. 注釈付き Java クラスから WSDL のみを生成する場合は、`genWsdl` コマンドを使用し、実装クラスを `className` 引数の値として指定します。

バージョン 3.0 の EJB から Web サービスをアセンブルするために注釈を使用する手順

Web サービスは、デプロイ時に EJB 3.0 準拠の Bean からアセンブルできます。Web サービス・エンドポイントを EJB 3.0 Bean から生成する手順は、次のとおりです。

1. `@Stateless` 注釈を Bean に追加します。

   ステートレス EJB セッション Bean のみがサポートされています。

   `@Stateless` 注釈の `name` プロパティに、Web サービス・エンドポイントの名前を指定します。このプロパティを指定しない場合、デフォルトのエンドポイント名は、EJB Bean クラスの非修飾クラス名になります。

2. Bean が実装するインタフェースに、`@WebService` 注釈を追加します。

   a. インタフェースに注釈を入力します。Bean クラスに直接入力することはできません。

   b. 少なくとも、`@WebService` 注釈をインタフェースに含める必要があります。

   c. Web サービスで公開する Bean 実装の各メソッドに、`@WebMethod` 注釈を追加します。

注意： サービス・エンドポイント・インタフェースに `@WebService` 注釈を入力し、`@WebMethod` 注釈を入力しないと、入力した `@WebMethod` 注釈の数に関係なく、すべてのメソッドが公開されます。

実装 Bean に `@WebService` 注釈を入力し、`@WebMethod` 注釈を入力しないと、少なくとも 1 つのメソッドを公開する必要があるため、検証例外が発生します。Web Services Metadata for the Java Platform 仕様の 4.1 項と 4.2 項を参照してください。

---

   d. EJB Bean クラスにこのインタフェースを実装します。

3. （オプション）`@Deployment` 注釈を、`uriPath` および `portName` プロパティに値を指定して、インタフェースに追加します。

   EJB 3.0 Bean のポート名およびデフォルト URI が、デフォルトの EJB 名になります。この名前が Bean の `@Stateless` 注釈に設定されます。Bean 名が `@Stateless` 注釈に指定されていない場合は、かわりに Bean のこの短縮クラス名が使用されます。`@Deployment` 注

釈の uriPath および portName プロパティに値を指定することで、この指定をオーバーライドできます。

4. EJB クラスをコンパイルし、EAR ファイルにパッケージ化します。EAR ファイルを OC4J の実行中インスタンスにデプロイします。

注釈をオーバーライドする方法

この項では、デプロイメント・ディスクリプタまたは WebServicesAssembler を使用した、Java ファイルの注釈値のオーバーライド方法を説明します。

- WebServicesAssembler で注釈値をオーバーライドする方法
- デプロイメント・ディスクリプタで Deployment 注釈の値をオーバーライドする方法

WebServicesAssembler で注釈値をオーバーライドする方法

WebServicesAssembler の assemble および genWsdl コマンドにコマンドライン引数を渡すことで、同じ機能を実行する Java クラス・ファイルの注釈をオーバーライドできます。たとえば、portName 引数を assemble または genWsdl コマンドに渡すと、渡した値が @Deployment.portName 注釈の値をオーバーライドします。

表 11-6 注釈の値をオーバーライドできる WebServicesAssembler の引数

WebServicesAssembler の引数	オーバーライドされる注釈
context	@Deployment.contextPath
portName	@Deployment.portName
portTypeName	@WebService.name（WSDL で portType 名を導出するには、@WebService.name 注釈が使用されます。Web Services Metadata for the Java Platform 仕様の 5.11 項を参照してください。）
qualifiedElementForm	@Schema.elementFormDefault
restSupport	@Deployment.restSupport
serviceName	@WebService.serviceName
targetNamespace	@WebService.targetNamespace
typeNamespace	@Schema.targetNamespace
uri	@Deployment.uriPath

デプロイメント・ディスクリプタで Deployment 注釈の値をオーバーライドする方法

Oracle 固有の @Deployment 注釈で指定されたプロパティをオーバーライドするには、META-INF/oracle-webservices.xml（Web モジュールの場合は WEB-INF/oracle-webservices.xml）の EAR ファイルに、オーバーライドする値を指定するデプロイメント・ディスクリプタをパッケージします。

oracle-webservices.xml デプロイメント・ディスクリプタを使用して、@Deployment 注釈プロパティをオーバーライドする場合は、ディスクリプタの各 <webservice-description> 要素が、Web サービスの serviceName をベースにしてデプロイされる Web サービスと一致する必要があります。serviceName は、@WebService.serviceName 注釈で指定します。

@Deployment 注釈を使用する複数の Web サービスをアセンブルしようとしており、ただ 1 つのサービスのプロパティをオーバーライドするデプロイメント・ディスクリプタを指定する場合は、その他のサービスは影響を受けません。デプロイメント・ディスクリプタには、オーバーライドするプロパティのみを指定します。

たとえば、次にあげる oracle-webservices.xml デプロイメント・ディスクリプタからの XML フラグメントの値は、@Deployment 注釈のプロパティをオーバーライドします。このフラグメントでは、<port-component name> 属性および <endpoint-address-uri> 要素に対して値を指定しているため、注釈の portName および uriPath プロパティがオーバーライドされます。<context-root> 要素はこの例では指定されていないため、注釈の contextPath プロパティはオーバーライドされません。

```
<webservice-description name="CustomSessionBeanService">
    <port-component name="CustomSession">
          <endpoint-address-uri>/custom-session</endpoint-address-uri>
    </port-component>
</webservice>
```

もちろん、クラスに注釈を付けるかわりに oracle-webservices.xml デプロイメント・ディスクリプタのみを使用してデプロイメント・プロパティを指定することも可能です。

注釈付きの Java ファイルのサンプル

Java クラスから Web サービスを生成する際、J2SE 5.0 注釈を次のいずれかに入力できます。

- エンドポイント・クラスのみ
- エンドポイント・クラスおよびサービス・エンドポイント・インタフェース

注釈をエンドポイント・クラスおよびサービス・エンドポイント・インタフェースの両方に入力する場合は、エンドポイント・クラスには最小限の注釈のみを入力する必要があります。

例 11-4 に、Web サービス・メタデータ注釈付きの実装クラスを示します。この例の中で @WebService、@WebMethod および @WebParam の各注釈がどのように使用されているかに注目してください。

- @WebService 注釈を使用すると、serviceName および targetNamespace プロパティに値を指定できます。これらのプロパティは、生成される WSDL ファイル内の wsdl:service name および wsdl:targetNamespace 要素に値を移入するために使用されます。targetNamespace プロパティは、WSDL およびスキーマ型に対するデフォルトの名前空間を提供します。

  注意： スキーマの要素および型に対するターゲット名前空間は、@WebService.targetNamespace で指定されている名前空間または WSDL の生成された名前空間と同じです。document-literal Web サービスの場合は、@WebParam.targetNamespace 注釈プロパティを使用することで、操作のパラメータを他の名前空間に割り当てることができます。次に例を示します。

  ```
  @WebParam(name="param", targetNamespace="http://mytns")
  ```

  この場合、@WebParam.name プロパティを設定する必要があることに注意してください。document-literal ラップ・サービスの場合は、ラッパー要素は常に WSDL と同じターゲット名前空間に作成されます。

  オプションとして、サービス・エンドポイント・インタフェースの完全修飾クラス名を使用して、endpointInterface プロパティを指定できます。この場合、次の注釈を追加します。

  - @WebService 注釈は、エンドポイント・クラスにのみ入力し、endpointInterface に正しい値を入力します。endpointInterface が使用されている場合は、エンドポイント・クラスの @WebService 注釈を除いて、Web サービス関連の注釈がすべて無視されます。
  - @WebMethod および @WebParam 注釈を、サービス・エンドポイント・インタフェースに入力します。

- @WebMethod 注釈は、Web サービスとしてアセンブルするメソッドを指定します。注釈が付けられたこれらのメソッドは、生成される WSDL 内の wsdl:operation 要素にアセンブルされます。operationName プロパティの値は、@WebMethod 注釈に指定されている場合、wsdl:operation の操作名として使用されます。値が指定されていない場合は、メソッドの名前がデフォルトで使用されます。
- @WebParam 注釈は、wsdl:operations に対するメッセージ部分またはメッセージ・パラメータを指定します。Mode を、オプションとして各パラメータに指定できます。INOUT または OUT モードは、javax.xml.rpc.holders.Holder インタフェースを実装するパラメータに対してのみ指定できます。Holder インタフェースを実装するパラメータは、Mode.OUT が注釈で明示的に指定されていない場合、デフォルトとして INOUT パラメータになります。それ以外のパラメータは、すべて IN パラメータとして使用する必要があります。

例 11-4 J2SE 5.0 Web サービス・メタデータ注釈付きの Java ファイルのサンプル

```
package oracle.webservices.examples.annotations;

import java.rmi.RemoteException;
import javax.jws.*;
import javax.jws.WebParam.Mode;

@WebService (
    serviceName = "annotatedBank",
    targetNamespace = "http://service.annotatedBank"
)
public class BankServiceImpl {
    @WebMethod (operationName="create-account")
    public String createAccount( @WebParam(name="accountName") String acctName,
                                 float initBalance )
        throws RemoteException, AccountException {
        return m_bank.addNewAccount(acctName,initBalance);
    }

    @WebMethod
    public void deposit( @WebParam(name="accountID", mode=Mode.IN) String acctID,
                         float amount )
        throws RemoteException, AccountException {
        Account theAccount = m_bank.getAccount(acctID);
        if ( theAccount == null ) {
            throw new AccountException("No account found for " + acctID);
        }
        theAccount.deposit(amount);
    }
    //class truncated…
}
```

制限事項

F-9 ページの「注釈を使用した Web サービスのアセンブル」を参照してください。

追加情報

詳細は、次を参照してください。

- テスト・ページを使用した Web サービス・デプロイのテスト方法は、第 13 章「Web サービス・デプロイのテスト」を参照してください。
- J2SE クライアントの構築方法は、第 15 章「J2SE Web サービス・クライアントのアセンブル」を参照してください。
- J2EE クライアントの構築方法は、第 14 章「J2EE Web サービス・クライアントのアセンブル」を参照してください。
- JAX-RPC ハンドラの詳細は、第 16 章「JAX-RPC ハンドラの使用方法」を参照してください。
- クライアントのアセンブルに必要な JAR ファイルの詳細は、付録 A「Web サービス・クライアントの API および JAR」を参照してください。
- Web サービスの相互運用性の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「相互運用可能な Web サービスの実現」を参照してください。
- Web サービス・クライアントにおけるサービスのクオリティ機能の使用方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの管理」を参照してください。
- Web サービスへのセキュリティの追加方法は、『Oracle Application Server Web Services セキュリティ・ガイド』を参照してください。
- Web サービスへの信頼性の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの信頼性の確保」を参照してください。
- Web サービスへの監査およびロギング構成の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「監査メッセージおよびロギング・メッセージ」を参照してください。
- 標準以外のデータ型の処理の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 値タイプのカスタム・シリアライズ」を参照してください。
- JAX-RPC マッピング・ファイルの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「JAX-RPC マッピング・ファイル記述子」を参照してください。
- OracleAS Web Services でサポートされているデータ型の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 型の XML および WSDL 型へのマッピング」を参照してください。
- Web サービス開発用の Oracle JDeveloper ツールのサポートの詳細は、Oracle JDeveloper のオンライン・ヘルプを参照してください。

12

REST Web サービスのアセンブル

この章では、Oracle Application Server Web Services が REST Web サービスの生成と使用をサポートする方法を説明します。この章の内容は、次のとおりです。

REST Web サービスの理解

REST（Representational State Transfer）Web サービス・アーキテクチャは、World Wide Web Consortium（W3C）が策定した Web アーキテクチャに準拠し、そのアーキテクチャの理念を使用したものです。HTTP のセマンティクスを可能なかぎり使用します。REST Web サービスはメッセージの送信に XML 文書を使用し、SOAP エンベロープは使用しません。SOAP Web サービスとは異なり、REST はスタイルであるため、標準やベンダーによるツールのサポートはありません。

Oracle Application Server Web Services を使用し、REST を使用してエンドポイントを公開すると、そのエンドポイントは SOAP ポートとしても公開されます。OracleAS Web Services は、HTTP コマンドのサポートを GET および POST に制限しています。REST Web サービスは、他の OracleAS Web Services と同様にデプロイします。

注意： OracleAS Web Services は、use つまりエンコーディング・メカニズムが literal（`use="literal"`）の場合にのみ、REST Web サービスをアセンブルできます。REST Web サービスは、メッセージ書式が encoded の場合はサポートされません。

セキュリティおよび信頼性などの Web サービス管理機能は、REST Web サービスの起動（REST または SOAP）では使用できません。これは、通常この情報の保持に使用する SOAP ヘッダーが、REST Web サービスの呼出しでは使用できないためです。

REST Web サービスに対する WebServicesAssembler のサポート

この項の内容は、次のとおりです。

REST Web サービスをアセンブルできるコマンド

WebServicesAssembler を使用して、REST Web サービスの機能を、HTTP をプロトコルとして使用できる Web アプリケーションに追加できます。これには、Java クラス、EJB およびデータベース・リソースに基づいて作成される Web サービス・アプリケーションがあります。WebServicesAssembler のブール型引数 `restSupport` を使用すると、次のいずれかのコマンドで REST Web サービスをアセンブルできます。

- aqAssemble
- assemble
- corbaAssemble
- dbJavaAssemble
- ejbAssemble
- plsqlAssemble
- sqlAssemble
- topDownAssemble

REST Web サービスをサポートする注釈

Oracle 固有の `@Deployment` 注釈には、オプションのブール型属性 `restSupport` を適用できます。属性の値が `true` の場合は、注釈が適用されたポートは、REST スタイルの `GET` および `POST` のリクエストおよびレスポンスをサポートします。

関連項目：

`restSupport` 属性および `Deploy` タグの詳細は、11-7 ページの「Oracle による J2SE 5.0 JDK 注釈への追加」を参照してください。

REST Web サービスのアセンブル方法

この項の内容は、次のとおりです。

トップダウン方式での REST Web サービスのアセンブル手順

次の手順は、WSDL からの REST Web サービスのアセンブル方法を例で示しています。この例では、トップダウン方式による Web サービスのアセンブルに必要な手順のアウトラインのみを示しています。

関連項目：

トップダウン方式で WSDL から Web サービスをアセンブルする手順の各ステップの詳細は、第 6 章「WSDL からの Web サービスのアセンブル」を参照してください。

1. Web サービス生成のベースとなる WSDL を、WebServicesAssembler の `genInterface` コマンドへの入力として指定します。

   この例で使用される WSDL を例 12-1 に示します。次は、`genInterface` コマンドのサンプルです。

   ```
   java -jar wsa.jar -genInterface
                     -output build/src/service
                     -wsdl wsdl/MovieFacility.wsdl
                     -unwrapParameters false
   ```

   このコマンドの説明：

   - `genInterface`: 各ポート・タイプのサービス・エンドポイント・インタフェースと、WSDL に定義された複合型に対応する Java 値タイプ・クラス（Bean）を作成します。また、XML スキーマ型と Java 値タイプ・クラス間のマッピングを記述する JAX-RPC マッピング・ファイルも作成されます。18-30 ページの「genInterface」を参照してください。
   - `output`: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
   - `wsdl`: WSDL ドキュメントの絶対ファイル・パス、相対ファイル・パスまたは URL を指定します。18-64 ページの「wsdl」を参照してください。
   - `unwrapParameters`: この引数は document-literal 操作に対してのみ設定でき、その他のメッセージ書式では無視されます。`false` に設定すると、生成対象のサービス・エンドポイント・インタフェースが、入力パラメータおよび戻り型をラップするラッパーとともに生成されます。18-69 ページの「unwrapParameters」を参照してください。

2. 生成されたインタフェースおよびタイプ・クラスをコンパイルします。
3. 提供する Web サービスに対する Java サービス・エンドポイントを実装します。

4. Java サービス・エンドポイントをコンパイルします。
5. WebServicesAssembler ツールの `topDownAssemble` コマンドを実行し、サービスを生成します。

   `restSupport` 引数を `true` に設定します。次に例を示します。

```
java -jar wsa.jar -topDownAssemble
            -wsdl ./wsdl/MovieFacility.wsdl
            -unwrapParameters false
            -className oracle.webservices.examples.rest.RpcLitMovieImpl
            -input build/classes/service
            -output build
            -ear dist/rpclit_topdown.ear
            -restSupport true
```

   このコマンドの説明 :

   - `topDownAssemble`: WSDL 記述をベースにした Web サービスで必要となるクラスとデプロイメント・ディスクリプタを作成します。これらのファイルは、EAR ファイル、WAR ファイル、ディレクトリのいずれにも格納できます。18-21 ページの「topDownAssemble」を参照してください。
   - `wsdl`: WSDL ドキュメントの絶対ファイル・パス、相対ファイル・パスまたは URL を指定します。18-64 ページの「wsdl」を参照してください。
   - `unwrapParameters`: この引数は document-literal 操作に対してのみ設定でき、その他のメッセージ書式では無視されます。`false` に設定すると、生成対象のサービス・エンドポイント・インタフェースが、入力パラメータおよび戻り型をラップするラッパーとともに生成されます。18-69 ページの「unwrapParameters」を参照してください。
   - `className`: Web サービスの実装クラスのクラス名（パッケージ名を含む）を指定します。18-41 ページの「className」を参照してください。
   - `input`: `WEB-INF/classes` にコピーされるクラスを格納する JAR またはディレクトリを指定します。この引数は、WebServicesAssembler によって使用されるクラスパスに追加されます。18-45 ページの「input」を参照してください。
   - `output`: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
   - `ear`: 生成される EAR ファイルの名前と位置を指定します。18-42 ページの「ear」を参照してください。
   - `restSupport`: Representational State Transfer（REST）のサポートをこの Web サービスに対して有効にするかどうかを指定します。18-48 ページの「restSupport」を参照してください。

6. サービスをデプロイします。

   通常の方法で EAR ファイルを OC4J の実行中インスタンスにデプロイします。EAR ファイルのデプロイの詳細は、『Oracle Containers for J2EE デプロイメント・ガイド』を参照してください。

7. （オプション）デプロイが成功したことを確認します。

   OracleAS Web Services では、デプロイされた Web サービスごとに Web サービス・テスト・ページが提供されます。テスト・ページを使用すると、REST POST または GET リクエストを生成して起動できます。Web サービス・テスト・ページへのアクセスおよびその使用方法の詳細は、13-7 ページの「Web サービス・テスト・ページの使用方法」を参照してください。

注意： REST Web サービスは、J2EE または J2SE クライアントを使用しません。ただし、各 REST エンドポイントは同時に SOAP エンドポイントでもあるため、J2EE または J2SE クライアントを、これらのエンドポイントに対してアセンブルできます。GET または POST REST メッセージの作成方法の例が必要な場合は、Web サービス・テスト・ページを使用してください。

REST Web サービス操作へのアクセス方法

例 12-1 は、「トップダウン方式での REST Web サービスのアセンブル手順」で prc-literal サービスのアセンブルに使用された WSDL フラグメントです。

例 12-1 RPC-Literal Web サービス向けの WSDL フラグメント

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns0="http://www.oracle.com/rest/doc/types"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://www.oracle.com/rest" name="rest-service"
targetNamespace="http://www.oracle.com/rest"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/">
    <types>
        <schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:soap11-enc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://www.oracle.com/rest/doc/types" elementFormDefault="qualified"
targetNamespace="http://www.oracle.com/rest/doc/types">
            <complexType name="Movie">
                <sequence>
                    <element name="Title" type="xsd:string"/>
                    <element name="Director" type="xsd:string"/>
                    <element name="Year" type="xsd:int" />
                </sequence>
            </complexType>
            <complexType name="ArrayOfMovie">
                <sequence>
                    <element name="Movie" type="tns:Movie"
                                maxOccurs="unbounded" />
                </sequence>
            </complexType>
        </schema>
    </types>
    <message name="FindMoviesRequest">
        <part name="TitleWords" type="xsd:string" />
        <part name="Year" type="xsd:int" />
    </message>
    <message name="FindMoviesResponse">
        <part name="Movies" type="tns0:ArrayOfMovie" />
    </message>
    <message name="AddMovieRequest">
        <part name="Movie" type="tns0:Movie" />
    </message>
    <message name="AddMovieResponse">
        <part name="Added" type="xsd:boolean" />
    </message>
    <portType name="MovieDB">
         <operation name="findMovies">
             <input message="tns:FindMoviesRequest" />
             <output message="tns:FindMoviesResponse" />
         </operation>
```

```
        <operation name="addMovie">
            <input message="tns:AddMovieRequest" />
            <output message="tns:AddMovieResponse" />
        </operation>
    </portType>

    <binding name="HttpSoap11Binding" type="tns:MovieDB">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="findMovies">
            <soap:operation soapAction="http://www.oracle.com/rest/findMovies"/>
            <input>
                <soap:body use="literal" parts="TitleWords Year"
namespace="http://www.oracle.com/rest"/>
            </input>
            <output>
                <soap:body use="literal" parts="Movies"
namespace="http://www.oracle.com/rest"/>
            </output>
        </operation>
        <operation name="addMovie">
            <soap:operation soapAction="http://www.oracle.com/rest/addMovie"/>
            <input>
                <soap:body use="literal" parts="Movie"
namespace="http://www.oracle.com/rest"/>
            </input>
            <output>
                <soap:body use="literal" parts="Added"
namespace="http://www.oracle.com/rest"/>
            </output>
        </operation>
    </binding>
    <service name="rest-service">
        <port name="HttpSoap11" binding="tns:HttpSoap11Binding">
            <soap:address location="http://localhost:8888/webservice/webservice/"/>
        </port>
    </service>
</definitions>
```

前述の WSDL を使用し、「トップダウン方式での REST Web サービスのアセンブル手順」に記載の手順を実行することで、次のようなインタフェースが生成されます。

```
interface MovieDb {
    public Movie[] findMovies (String titleWords, int year);
    public boolean addMovie (Movie movie);
}
```

生成されたインタフェース内の最初のメソッドは、単純なパラメータのみを持ちます。このメソッドは、HTTP GET を使用して起動できます。次に例を示します。

```
http://{yourhost}/{context-path}/{service-url}/findMovies?TitleWords=Star+Wars&Year=1977
```

この問合せ文字列は、次の XML レスポンスを返します。

```
<ns0:findMoviesResponse xmlns:ns0="http://www.oracle.com/rest">
   <Movies>
      <ns1:Movie xmlns:ns1="http://www.oracle.com/rest/doc/types">
         <ns1:Title>tim</ns1:Title>
         <ns1:Director>tim</ns1:Director>
         <ns1:Year>1978</ns1:Year>
      </ns1:Movie>
   </Movies>

</ns0:findMoviesResponse>
```

生成されたインタフェース内の addMovie メソッドは、複雑なパラメータを取るため、HTTP POST によってのみ起動できます。たとえば、次の XML メッセージを独自の Web サービスの URL、http://{*yourhost*}/{*context-path*}/{*service-url*} に対して POST できます。

```
<ns0:addMovie xmlns:ns0="http://www.oracle.com/rest">
   <Movies>
      <ns1:Movie xmlns:ns1="http://www.oracle.com/rest/doc/types">
         <ns1:Title>tim</ns1:Title>
         <ns1:Director>tim</ns1:Director>
         <ns1:Year>1978</ns1:Year>
      </ns1:Movie>
   </Movies>
</ns0:addMovie>
```

ボトムアップ方式での REST Web サービスのアセンブル手順

次の手順は、Java クラスからの REST Web サービスのアセンブル方法を例で示しています。この例では、ボトムアップ方式による Web サービスのアセンブルに必要な手順のアウトラインのみを示しています。

> **関連項目：**
>
> Java クラスで Web サービスをアセンブルする手順の各ステップの詳細は、第 7 章「Java クラスを使用した Web サービスのアセンブル」を参照してください。

1. Web サービスとして公開するコンパイル済 Java クラスとそのコンパイル済インタフェースを用意します。

   例 12-2 は、この例で使用している StringTools インタフェースです。

2. restSupport 引数を true にした assemble コマンドを指定して、サービスのアーチファクトを生成します。

   ```
   java -jar wsa.jar -assemble
              -appName tools
              -serviceName StringTools
              -interfaceName oracle.webservices.examples.rest.StringTools
              -className oracle.webservices.examples.StringToolsImpl
              -input ./build/classes/service
              -output build
              -use literal
              -ear dist/tools.ear
              -uri StringToolsService
              -restSupport true
   ```

   このコマンドの説明：

   - assemble: ボトムアップ方式で Java ファイルから Web サービスを生成します。このコマンドにより、デプロイ可能なアーカイブを作成するために必要なすべてのファイルが作成されます。18-8 ページの「assemble」を参照してください。
   - appName: アプリケーションの名前を指定します。この名前は、通常 context や uri などの他の引数のベース値として使用されます。18-40 ページの「appName」を参照してください。
   - serviceName: サービス名を指定します。18-48 ページの「serviceName」を参照してください。
   - interfaceName: サービス・エンドポイント・インタフェース（SEI）を格納する Java クラスの名前（パッケージ名を含む）を指定します。18-45 ページの「interfaceName」を参照してください。
   - className: Web サービスの実装クラスのクラス名（パッケージ名を含む）を指定します。18-41 ページの「className」を参照してください。

- `input`: `WEB-INF/classes` にコピーされるクラスを格納する JAR またはディレクトリを指定します。この引数は、WebServicesAssembler によって使用されるクラスパスに追加されます。18-45 ページの「input」を参照してください。
- `output`: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
- `use`: ボトムアップ方式 Web サービス・アセンブリの場合、この引数は、生成される WSDL 内のメッセージ書式の `use` 属性を指定します。18-68 ページの「use」を参照してください。
- `ear`: 生成される EAR ファイルの名前と位置を指定します。18-42 ページの「ear」を参照してください。
- `uri`: Web サービスに使用する URI を指定します。18-63 ページの「uri」を参照してください。
- `restSupport`: Representational State Transfer（REST）のサポートをこの Web サービスに対して有効にするかどうかを指定します。18-48 ページの「restSupport」を参照してください。

3. サービスをデプロイし、アプリケーションをバインドします。

   通常の方法で EAR ファイルを OC4J の実行中インスタンスにデプロイします。EAR ファイルのデプロイの詳細は、『Oracle Containers for J2EE デプロイメント・ガイド』を参照してください。

4. 必要に応じて、デプロイメントが成功したかどうかを確認します。OracleAS Web Services では、デプロイされた Web サービスごとに Web サービス・テスト・ページが提供されます。

   テスト・ページを使用すると、REST POST または GET リクエストを生成して起動できます。Web サービス・テスト・ページへのアクセスおよびその使用方法の詳細は、13-7 ページの「Web サービス・テスト・ページの使用方法」を参照してください。

> **注意：** REST Web サービスは、J2EE または J2SE クライアントを使用しません。ただし、各 REST エンドポイントは同時に SOAP エンドポイントでもあるため、J2EE または J2SE クライアントを、これらのエンドポイントに対してアセンブルできます。GET または POST REST メッセージの作成方法の例が必要な場合は、Web サービス・テスト・ページを使用してください。

REST Web サービス操作へのアクセス方法

例 12-2 は、前述のステップ 2 で REST Web サービスのアセンブルに使用されていた `StringTools` インタフェースです。

例 12-2 REST Web サービスのアセンブルに使用するインタフェース

```
interface StringTools {
package oracle.webservices.examples.rest;
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface StringTools extends Remote{
    public String appendStrings (String a, String b) throws RemoteException;
    public String toUpperCase (String c) throws RemoteException;
    public String concatArrayOfStrings (String s[]) throws RemoteException;
```

このインタフェースの最初の 2 つのメソッド、`appendStrings` および `toUpperCase` は、単純なパラメータを使用しています。このため、これらは、REST Web サービス操作として公開すると、HTTP GET を使用してアクセスできます。次の例は、アセンブリ時に `document` スタイルを指定した場合の、`appendStrings` 操作のコールを示します。

```
http://{yourserver}/{context-path}/{service-URL}/appendStrings?String_1=Hello+&String_2
=World
```

この問合せ文字列は、次の XML レスポンスを返します。

```
<appendStringsResponseElement xmlns="http://oracle.webservices.examples.rest/">
     <result>Hello World</result>
</appendStringsResponseElement>
```

このインタフェース内の 3 番目のメソッド `concatArrayOfStrings` は、複雑なパラメータを取ります。このため、これは、REST Web サービス操作として公開しても、HTTP GET ではコールできません。HTTP POST によってのみ、コールできます。次に例を示します。

```
<ns1:concatArrayOfStringsElement
xmlns:ns1="http://oracle.webservices.examples.rest/">
            <ns1:arrayOfString_1>a,</ns1:arrayOfString_1>
            <ns1:arrayOfString_1>b.</ns1:arrayOfString_1>
</ns1:concatArrayOfStringsElement>
<concatArrayOfStringsRequest xmlns="http://oracle.webservices.examples.rest/">
```

このリクエスト文字列は、次の XML コードを返します。

```
<ns0:concatArrayOfStringsResponseElement
xmlns:ns0="http://oracle.webservices.examples.rest/">
   <ns0:result>a,b.</ns0:result>
</ns0:concatArrayOfStringsResponseElement>
```

REST Web サービスのテスト方法

Web サービス・テスト・ページを使用して、REST Web サービスが正常にデプロイされたかどうかをテストできます。

関連項目：

Web サービス・テスト・ページの使用方法の詳細は、13-14 ページの「REST Web サービスのテストの起動」を参照してください。

REST でのデプロイメント・ディスクリプタへの追加

REST サポートが Web サービスに対して有効になっている場合は、オプションのブール型 `<rest-support>` サブ要素が、`oracle-webservices.xml` デプロイメント・ディスクリプタの `<port-component>` 要素に追加されます。`<rest-support>` が `true` に設定されると、サブ要素が適用されるポートは、REST スタイルの `GET` および `POST` のリクエストおよびレスポンスをサポートします。デフォルト値は `false` です。

関連項目：

`<rest-support>` サブ要素の追加情報は、C-11 ページの「<rest-support>」を参照してください。

REST がリクエストとレスポンスを作成する方法

次の各項では、REST Web サービス・リクエストのクライアント・サイドでの構築方法、およびサーバー・サイドでの処理方法を説明します。

HTTP GET リクエスト

SOAP エンドポイントが次の URL にデプロイされていると仮定します。

```
http://example.com/my-app/my-service
```

このエンドポイントが REST 対応の場合は、次の URL への HTTP GET リクエストが受け入れられます。

```
http://example.com/my-app/my-service/{operationName}?{param1}={value1}&{param2}={value2}
```

この例において、{*operationName*} はサービスに対する WSDL 内の操作名の 1 つです。RPC-literal 操作の場合は、{*param1*}、{*param2*} などが、操作の入力である `wsdl:message` 内で定義される各部分の名前になります。ただし、これらの各部分は `simpleTypes`（`xsd:int` など）である必要があります。

注意： ブラウザによっては、HTTP GET URL のサイズを制限している場合があります（一般的には 2000 文字未満）。パラメータ数を制限し、パラメータの名前と値を短くすることで、URL サイズを小さくするように注意してください。

---

document-literal 操作では、メッセージはパラメータを 1 つのみ持ちます。複数パラメータをシミュレートするため、WSDL には、スキーマで `sequence` として定義されているパラメータを指定します。sequence の各メンバーをパラメータと考えることができます。この場合、{*param1*}、{*param2*} などが、`message` 部分ではなく sequence 型のメンバーになります。RPC の場合と同様に、これらのメンバーは `simpleTypes` である必要があります。

例 12-3 は、`addNumbers` という名前の操作に対して定義されたリクエスト・メッセージです。

例 12-3 操作に対する GET リクエスト

```
<wsdl:message name="AddNumbersRequest">
      <wsdl:part name="a" type="xsd:int" />
      <wsdl:part name="b" type="xsd:int" />
</wsdl:Message>
```

このリクエストは、次の URL を指定した GET で起動できます。

```
http://{yourhost}/{context-path}/{service-url}/addNumbers?a=23&b=24
```

例 12-4 は、OracleAS Web Services プラットフォームが GET リクエストからサーバー・サイドで作成する SOAP エンベロープを示します。このメッセージは、他の受信 SOAP リクエストと同様に処理されます。

例 12-4 GET リクエストから作成される SOAP エンベロープ

```
<soap:Envelope>
    <soap:Body>
        <ns:addNumbers>
            <ns:a>23</ns:a>
            <ns:b>24</ns:b>
        </ns:addNumbers>
    <soap:Body>
<soap:Envelope>
```

例 12-5 は、addNumbers サービスが document-literal 操作として定義されている場合に送信されるリクエスト・メッセージです。

例 12-5 document-literal のラップ操作に対する GET リクエストのサンプル

```
<wsdl:message name="AddNumbersRequest">
     <wsdl:part name="params" type="tns:AddNumbersRequstObject" />
</wsdl:Message>
```

例 12-6 は、AddNumbersRequestObject の定義を示しています。これはスキーマでも同じ定義になります。

例 12-6 document-literal のラップ操作の XML 定義

```
<xsd:complexType name="AddNumbersRequestObject">
    <xsd:complexContent>
        <xsd:sequence>
            <xsd:element name="a" type="xsd:int"/>
            <xsd:element name="b" type="xsd:int"/>
        </xsd:sequence>
    </xsd:complexContent>
</xsd:complexType>
```

この操作は、次の URL を指定した GET リクエストによって起動できます。これは、例 12-3 の RPC-literal リクエストに対して使用したものと同じ URL である点に注意してください。

http://{*yourhost*}/{*context-path*}/{*service-url*}/addNumbers?a=23&b=24

HTTP POST リクエスト

REST Web サービスでは、単純な XML メッセージの HTTP POST リクエストはサポートされていますが、SOAP エンベロープの HTTP POST リクエストはサポートされていません。REST リクエストでは、添付ファイル付きのメッセージはサポートされていません。REST Web サービスでは SOAP リクエストもサポートされているため、指定されたリクエストが SOAP または REST のどちらであるかを実装側で判断する必要があります。

SOAP サービスは、POST リクエストを受信すると SOAPAction ヘッダーを検索します。ヘッダーが存在する場合は、実装はこのリクエストを SOAP リクエストであると判断します。存在しない場合は、リクエストの root 要素の QName が検索されます。それが SOAP エンベロープの QName の場合は、メッセージは SOAP リクエストとして処理されます。そうでない場合は、REST リクエストとして処理されます。

REST リクエストは、リクエスト・ドキュメントを SOAP エンベロープでラップすることで処理されます。SOAP 1.2 リクエストの Content-Type ヘッダーではない HTTP ヘッダーのみが、そのまま受信されます。この Content-Type ヘッダーは、SOAP 1.2 に対する適切なコンテンツ・タイプである application/soap+xml に変更されます。

たとえば、例 12-7 の REST リクエストは、例 12-8 の SOAP エンベロープにラップされます。

例 12-7 REST リクエスト

```
<ns:addNumbers>
    <ns:a>23</ns:a>
    <ns:b>24</ns:b>
</ns:addNumbers>
```

例 12-8 のリクエストは、通常の SOAP リクエストとして処理されます。

例 12-8 REST リクエストをラップする SOAP エンベロープ

```
<soap:Envelope>
       <soap:Body>
              <ns:addNumbers>
                     <ns:a>23</ns:a>
                     <ns:b>24</ns:b>
              </ns:addNumbers>
          </soap:Body>
</soap:Envelope>
```

REST レスポンス

REST リクエストとして処理される（GET または POST）リクエストに対するレスポンスも、REST スタイルである必要があります。OracleAS Web Services プラットフォームは、サーバー上の SOAP レスポンスをクライアントに送信する前に、REST レスポンスに変換します。REST レスポンスは、SOAP 本体の最初の子要素をルート要素として持つ XML 文書になります。たとえば、例 12-9 の SOAP エンベロープが、サーバー上にあると想定します。

例 12-9 SOAP レスポンス

```
<soap:Envelope>
    <soap:Body>
        <ns0:result xmlns:nso="…">
            <ns:title>How to Win at Poker</ns:title>
            <ns:author>John Doe</ns:author>
        </ns0:result>
    </soap:Body>
</soap:Envelope>
```

例 12-10 は、クライアントに返送されるレスポンスを示します。Content-Type が常に `text/xml` である点に注意してください。SOAP ヘッダーや添付ファイルは、クライアントに返送されません。

例 12-10 REST レスポンス

```
<ns0:result xmlns:ns0="…">
       <ns:title>How to Win at Poker</ns:title>
       <ns:author>John Doe</ns:author>
</ns0:result>
```

REST Web サービスに対するツールのサポート

Web サービスの REST 機能を有効化するオプションが、Oracle JDeveloper の **Java Web サービスの作成**ウィザードに用意されています。Oracle JDeveloper を使用して Web サービスの REST 機能を有効化する方法の詳細は、Oracle JDeveloper のオンライン・ヘルプを参照してください。

制限事項

F-9 ページの「REST Web サービスのアセンブル」を参照してください。

追加情報

詳細は、次を参照してください。

- WSDL からの Web サービスのアセンブル方法は、第 6 章「WSDL からの Web サービスのアセンブル」を参照してください。
- Java クラスからの Web サービスのアセンブル方法は、第 7 章「Java クラスを使用した Web サービスのアセンブル」を参照してください。
- EJB からの Web サービスのアセンブル方法は、第 8 章「EJB を使用した Web サービスのアセンブル」を参照してください。
- データベース・リソースからの Web サービスのアセンブル方法は、第 10 章「データベース Web サービスのアセンブル」を参照してください。
- REST Web サービスのテスト方法は、第 13 章「Web サービス・デプロイのテスト」を参照してください。

13

Web サービス・デプロイのテスト

この章では、JAX-RPC または REST の Web サービスのデプロイをテストする方法について説明します。Web サービスの操作を起動できれば、Web サービスは正常にデプロイされています。テストを行うため、Oracle Application Server Web Services ではテスト・ページが提供されています。Web サービス・テスト・ページを使用すると、次のタスクを実行できます。

- Web サービスのデプロイ済サービス記述（WSDL）を表示する
- 値を変えて Web サービスの操作を実行する
- Web サービスのセキュリティと信頼性に対する値を変えて操作を実行する
- リクエストに対する HTTP 認証の値を提供する
- ストレス・テストを実行する

次の各項では、Web サービス・テスト・ページにアクセスして使用する方法を詳しく説明します。

- Web サービスの操作にアクセスしてテストする手順
- Web サービス・テスト・ページへのアクセス方法
- Web サービス・テスト・ページの使用方法
- Web サービス・テスト・ページを無効にする方法
- Web サービスの WSDL を直接取得する方法

関連資料：

REST Web サービスの詳細は、第 12 章「REST Web サービスのアセンブル」を参照してください。

Web サービスの操作にアクセスしてテストする手順

次の手順では、JAX-RPC または REST の Web サービスのテスト・ページの機能にアクセスして使用する方法について説明します。後の項では各ステップについてさらに詳しく説明します。

1. Web サービス用テスト・ページにアクセスします。

   テスト・ページには、Application Server Control または Web ブラウザからアクセスできます。このステップの詳細は、13-2 ページの「Web サービス・テスト・ページへのアクセス方法」を参照してください。

2. **「操作」**ドロップダウン・リストから、テストする操作を選択します。

3. テストする Web サービス操作に対するパラメータ値を入力します。

   このステップの詳細は、13-9 ページの「テスト・ページでの値の編集」を参照してください。

4. （オプション）セキュリティおよび信頼性メッセージ機能を実行するためのパラメータ値を入力します（JAX-RPC Web サービスの場合のみ）。

   このステップの詳細は、13-10 ページの「WS-Security および信頼性メッセージ機能のテスト」を参照してください。

5. （オプション）Web サービスにアクセスするための HTTP 認証の値を入力します。

   このステップの詳細は、13-12 ページの「Web サービス・テストに対する HTTP 認証の有効化」を参照してください。

6. （オプション）ストレス・テストを有効にするためのパラメータを入力します。

   このステップの詳細は、13-12 ページの「Web サービス操作のストレス・テスト」を参照してください。

7. **「起動」**ボタンをクリックして、サービス・エンドポイントにテスト・メッセージを送信します。

   JAX-RPC または REST の Web サービスに対するリクエストは、SOAP リクエストとして起動できます。REST Web サービスの場合は、SOAP リクエスト、HTTP POST 操作または HTTP GET 操作として起動できます。このステップの詳細は、13-13 ページの「Web サービスのテストの起動」を参照してください。

Web サービス・テスト・ページへのアクセス方法

JAX-RPC または REST の Web サービスに対する URI を Web ブラウザに直接入力して、または Application Server Control を使用して、テスト・ページにアクセスできます。

- Web ブラウザからの Web サービス・テスト・ページへのアクセス
- Application Server Control からの Web サービス・テスト・ページへのアクセス
- SSL で保護された Web サービス・テスト・ページへのアクセス

Web ブラウザからの Web サービス・テスト・ページへのアクセス

JAX-RPC または REST の Web サービスに対するテスト・ページに直接アクセスするには、サービス・エンドポイントのアドレスを Web ブラウザに入力します。このアドレスの書式は、次のとおりです。

`http://`*`host:port/context-root/service`*

表 13-1 で、アドレスのコンポーネントについて説明します。

表 13-1 テスト・ページにアクセスするための URL コンポーネント

URL コンポーネント	説明
context-root	Web サービスに関連する Web モジュールについて `<context-root>` 要素に指定されている値。この値を確認するには、Web サービスの EAR ファイル内の `META-INF/application.xml` を参照してください。
host	OracleAS Web Services を実行する Web サービスのサーバーのホスト名。
port	OracleAS Web Services を実行する Web サービスのサーバーのポート名。
service	Web サービスに関連するサーブレットについて `<url-pattern>` 要素に指定されている値。これはサービス名です。この値を確認するには、Web サービスの WAR ファイル内の `WEB-INF/web.xml` を参照してください。

Application Server Control からの Web サービス・テスト・ページへのアクセス

Application Server Control を使用して、特定の JAX-RPC または REST Web サービスに対するテスト・ページにアクセスできます。次の手順は、テスト・ページに移動するためのツールの使用方法の概要です。詳細は、Application Server Control のオンライン・ヘルプの Web サービス・ページに関するトピックを参照してください。

1. Application Server Control を開きます。
2. **「OC4J: ホーム」**ページの**「Web サービス」**タブをクリックします。
3. Web サービスの表で、テストする Web サービス・アプリケーションとポートに対応する**「選択」**ラジオ・ボタンをクリックします。
4. **「サービスのテスト」**ボタンをクリックして、**「Web サービスのテスト : *port_name*」**ページを開きます。

   Enterprise Manager で「Web サービスのテスト」ページが表示され、現在の環境で定義されている Web サイトがリストされます。
5. **「選択」**ラジオ・ボタンをクリックして、Web サービスに対する Web サイト・リスナーを選択します。

   Enterprise Manager は、選択されたアプリケーションと Web サイトに対して定義されているコンテキスト・ルートに関する情報に基づいて、URL を生成します。「URL」フィールドに URL が表示されます。
6. **「Web サービスのテスト」**ボタンをクリックして、Web サービス・テスト・ページを開きます。

SSL で保護された Web サービス・テスト・ページへのアクセス

Web サービス・テスト・ページが SSL（Secure Sockets Layer）で保護されている場合は、使用するためには追加の構成手順を実行する必要があります。手順は、OracleAS Web Services が Oracle HTTP Server で実行しているかどうかによって異なります。

- OracleAS Web Services をスタンドアロン・アプリケーションとして実行している場合は、キーストアおよびトラストストアの構成プロパティを `opmn.xml` ファイルに追加するのみで済みます。これは、Oracle Process Manager and Notification Server（OPMN）に対する主な構成ファイルです。
- OracleAS Web Services を Oracle HTTP Server で実行している場合は、`opmn.xml` ファイルの編集に加えて、Oracle ウォレットでユーザー証明書をエクスポートし、それを信頼できる証明書として OC4J キーストアにインポートする必要があります。

関連資料：

- OPMN の詳細は、『Oracle Process Manager and Notification Server 管理者ガイド』を参照してください。
- キーストアおよびトラストストアのプロパティの詳細は、『Oracle Containers for J2EE セキュリティ・ガイド』を参照してください。

次の各項では、SSL で保護されたテスト・ページにアクセスするために必要な変更を行う方法について説明します。

- OracleAS Web Services がスタンドアロンで実行している場合の SSL で保護されたテスト・ページへのアクセス
- OracleAS Web Services が Oracle HTTP Server 上で実行している場合の SSL で保護されたテスト・ページへのアクセス

OracleAS Web Services がスタンドアロンで実行している場合の SSL で保護されたテスト・ページへのアクセス

OracleAS Web Services をスタンドアロン・アプリケーションとして実行している場合は、SSL で保護された Web サービス・テスト・ページにアクセスするには次の手順に従います。

1. インストールで `opmn.xml` ファイルを検索します。

   `opmn.xml` ファイルは次の場所にあります。

   ORACLE_HOME`/opmn/conf/opmn.xml`

   ORACLE_HOME は、OracleAS Web Services をインストールしたディレクトリです。

   ファイルはすでに SSL 用に構成されているものとします。

2. `opmn.xml` の OC4J `start-parameters` セクションに次のプロパティを追加し、ファイルを保存します。これらのプロパティの詳細は、表 13-2 を参照してください。これらのプロパティを使用している `opmn.xml` ファイルのサンプルは、例 13-1 を参照してください。
   - `-Djavax.net.ssl.trustStore=`<OC4J キーストア・ファイルのフルパス >
   - `-Djavax.net.ssl.trustStorePassword=`<OC4J キーストアのパスワード >
   - `-Djavax.net.ssl.keyStore=`<OC4J キーストア・ファイルのフルパス >
   - `-Djavax.net.ssl.keyStorePassword=`<OC4J キーストアのパスワード >
3. OPMN を再起動します。

   たとえば、次のコマンドは、すべての OPMN プロセスを停止してから再起動します。

   ```
   prompt> opmnctl startall
   ```
4. ブラウザに Web サービスのテスト・ページの URL を入力します。

5. サービスをテストするための値を入力し、「**起動**」ボタンをクリックします。

予想される SOAP メッセージのレスポンスを受け取ります。

表 13-2 は、キーストアとトラストストアについて Oracle HTTPS がサポートするプロパティです。

表 13-2 キーストアとトラストストアのプロパティ

プロパティ名	説明
javax.net.ssl.keyStore	キーストアとして使用するキーストア・ファイルまたはウォレット・ファイルの場所と名前を指定します。 このプロパティを設定することで、特定の接続に対して使用する資格証明を含む、Oracle Wallet Manager からエクスポートされたテキスト・ウォレット・ファイルを示すことができます。 HTTPS 接続に対して他の資格証明が設定されていない場合は、最初のハンドシェークが行われるときに、このプロパティで示されているファイルが開かれます。このファイルの読取りでエラーが発生すると、接続は失敗し、`IOException` がスローされます。 このプロパティが設定されていない場合は、アプリケーションが、証明書チェーンに信頼できる証明書が含まれることを確認する必要があります。
javax.net.ssl.keyStorePassword	このプロパティを設定することで、キーストア（キーストア・ファイルまたはウォレット・ファイル）を開くために必要なパスワードを示すことができます。
javax.net.ssl.trustStore	このプロパティは javax.net.ssl.keyStore と同じように使用しますが、トラストストア（クライアントが暗黙的に受け入れる信頼できる認証局を含むファイル）として使用するキーストア・ファイルまたはウォレット・ファイルの場所と名前を指定します。
javax.net.ssl.	このプロパティは `javax.net.ssl.keyStorePassword` と同じように使用しますが、トラストストア（キーストア・ファイルまたはウォレット・ファイル）を開くために必要なパスワードを指定します。

例 13-1 は、キーストアおよびトラストストア・プロパティを含む opmn.xml ファイルのフラグメントを示しています。この例で、`<ias-component id="default-group">` は OC4J 開始パラメータ・セクションの開始位置を示しています。

例 13-1 キーストアとトラストストアのプロパティを含む opmn.xml のフラグメント

```
<opmn>
 ...
  <ias-component id="default-group">
   <process-type id="home" module-id="OC4J" status="enabled">
    <module-data>
      <category-id="start-parameters">
        <data-id="java-options" value=".. -Djavax.net.ssl.trustStore=<full path to OC4J
keystore file>, -Djavax.net.ssl.trustStorePassword=<OC4J keystore password>,
-Djavax.net.ssl.keyStore=<full path to OC4J keystore file>,
-Djavax.net.ssl.keyStorePassword=<OC4J keystore password>, ...>
      </category>
       ...
    </module-data>
   </process-type>
  </ias-component>
 ...
</opmn>
```

OracleAS Web Services が Oracle HTTP Server 上で実行している場合の SSL で保護されたテスト・ページへのアクセス

OracleAS Web Services を Oracle HTTP Server 上で実行している場合は、SSL で保護された Web サービス・テスト・ページにアクセスするには次の手順に従います。

注意： この例では、Oracle Wallet Manager と Java keytool ユーティリティを使用して、証明書のエクスポートとインポートを行います。これらのユーティリティの使用方法については、このマニュアルでは説明しません。詳細は、次の資料を参照してください。

- Oracle Wallet Manager については、『Oracle Database Advanced Security 管理者ガイド』の Oracle Wallet Manager の使用方法に関する項を参照してください。このマニュアルは、Oracle Server Technologies グループから入手できます。
- Java keytool ユーティリティについては、次の Web サイトで「keytool - Key and Certificate Management Tool」を参照してください。

  `http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/keytool.html`

---

1. インストールで `opmn.xml` ファイルを検索します。

   `opmn.xml` ファイルは次の場所にあります。

   `ORACLE_HOME``/opmn/conf/opmn.xml`

   `ORACLE_HOME` は、OracleAS Web Services をインストールしたディレクトリです。

   ファイルはすでに SSL 用に構成されているものとします。

2. `opmn.xml` の `OC4J start-parameters` セクションに次のプロパティを追加し、ファイルを保存します。これらのプロパティについては、前の項の表 13-2 を参照してください。これらのプロパティを使用している `opmn.xml` ファイルのサンプルは、例 13-1 を参照してください。

   - `-Djavax.net.ssl.trustStore=<`OC4J キーストア・ファイルのフルパス `>`
   - `-Djavax.net.ssl.trustStorePassword=<`OC4J キーストアのパスワード `>`
   - `-Djavax.net.ssl.keyStore=<`OC4J キーストア・ファイルのフルパス `>`
   - `-Djavax.net.ssl.keyStorePassword=<`OC4J キーストアのパスワード `>`

3. Oracle Wallet Manager を使用して、Oracle HTTP Server のウォレット・ファイルからユーザー証明書をエクスポートします。

   Oracle Wallet Manager は次の場所にあります。

   `ORACLE_HOME``/bin/owm`

   次はウォレット・ファイルのパスの例です。

   `ORACLE_HOME``/Apache/Apache/conf/ssl.wlt/default/ewallet.p12`

   証明書のエクスポート方法は、Oracle Wallet Manager のマニュアルを参照してください。

4. Java keytool ユーティリティを使用して、証明書を OC4J キーストアに信頼できる証明書としてインポートします。

5. OPMN を再起動します。

   たとえば、次のコマンドは、すべての OPMN プロセスを停止してから再起動します。

   `prompt``> opmnctl startall`

6. ブラウザに Web サービスのテスト・ページの URL を入力します。
7. サービスをテストするための値を入力し、「**起動**」ボタンをクリックします。

予想される SOAP メッセージのレスポンスを受け取ります。

Web サービス・テスト・ページの使用方法

Oracle Application Server Web Services では、デプロイされた Web サービスごとにテスト・ページが提供されます。テスト・ページを使用すると、JAX-RPC Web サービスまたは REST サービスで公開されている操作を実行できます。

- Web サービス WSDL の表示
- テスト・ページでの値の編集
- WS-Security および信頼性メッセージ機能のテスト
- Web サービス・テストに対する HTTP 認証の有効化
- Web サービス操作のストレス・テスト
- Web サービスのテストの起動

図 13-1 は、`HelloServiceEJB` JAX-RPC Web サービスの `sayHello` 操作に対する Web サービス・テスト・ページです。

図 13-1 Web サービス・テスト・ページ

Web サービス WSDL の表示

JAX-RPC または REST Web サービスの WSDL を表示するには、「**Service Description**」リンクをクリックします。Web ブラウザの「**ファイル**」→「**名前を付けて保存**」操作を使用して、WSDL をローカルに保存できます。

テスト・ページに戻るには、「**戻る**」ボタンをクリックします。

図 13-2 は、JAX-RPC Web サービスの WSDL のフラグメントです。

図 13-2 JAX-RPC Web サービスの WSDL フラグメント

```
<?xml version="1.0" encoding="UTF-8" ?>
- <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:ns1="http://oracle.j2ee.ws/ejb/Hello/types"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:tns="http://oracle.j2ee.ws/ejb/Hello" name="HelloServiceEJB"
    targetNamespace="http://oracle.j2ee.ws/ejb/Hello">
  - <types>
    - <schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:soap11-
        enc="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:tns="http://oracle.j2ee.ws/ejb/Hello/types"
        elementFormDefault="qualified"
        targetNamespace="http://oracle.j2ee.ws/ejb/Hello/types">
      - <complexType name="sayHello">
        - <sequence>
            <element name="String_1" nillable="true" type="string" />
          </sequence>
        </complexType>
      - <complexType name="sayHelloResponse">
        - <sequence>
            <element name="result" nillable="true" type="string" />
          </sequence>
        </complexType>
        <element name="sayHelloElement" type="tns:sayHello" />
        <element name="sayHelloResponseElement" type="tns:sayHelloResponse" />
      </schema>
    </types>
  - <message name="HelloServiceInf_sayHello">
      <part name="parameters" element="ns1:sayHelloElement" />
```

図 13-3 は、REST Web サービスの WSDL のフラグメントです。

図 13-3 REST Web サービスの WSDL フラグメント

```
<?xml version="1.0" encoding="UTF-8" ?>
- <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns0="http://oracle.demo/types/"
    xmlns:java="http://schemas.xmlsoap.org/wsdl/java/"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:format="http://schemas.xmlsoap.org/wsdl/formatbinding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:tns="http://oracle.demo/" name="HelloWebService"
    targetNamespace="http://oracle.demo/">
  - <types>
    - <schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:soap11-
        enc="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:tns="http://oracle.demo/types/"
        targetNamespace="http://oracle.demo/types/" elementFormDefault="qualified">
      - <complexType name="sayHelloTo">
        - <sequence>
            <element name="name" type="string" nillable="true" />
          </sequence>
        </complexType>
      - <complexType name="sayHelloToResponse">
        - <sequence>
```

テスト・ページでの値の編集

テスト・ページでは、JAX-RPC または REST Web サービスで公開されている操作をテストできます。デフォルトでは、操作の編集可能なパラメータと属性が HTML 形式で表示されます。フォーム上のオプションのパラメータと属性は、チェック・ボックスで示されます。パラメータに値を設定するには、チェック・ボックスを選択します。

注意： HTML フォームで属性とパラメータを区別するために、属性の前には「**@**」記号が付いています。

テストする Web サービス操作を選択するには、**「操作」**ドロップダウン・リストを使用します。

フォームの「**+**」リンクは、構造のコピーを追加できることを示します。

フォームの「**x**」リンクは、不要な構造のコピーを削除できることを示します。

XML ソースとしてのテスト・ページの編集方法

JAX-RPC または REST Web サービス操作に対する値を HTML フォームで入力するかわりに、XML ソース・コードに直接入力できます。そのためには、テスト・ページで**「XML ソース」**ラジオ・ボタンを選択します。

注意：

- XML ソースに値を入力する場合は、属性の前に @ 記号を付ける必要はありません。
- XML ソースで操作に対する値を入力して、**HTML フォーム**・モードに切り替えた場合、入力した値は保持されません。フォームは、すべての値がクリアされて表示されます。

WS-Security および信頼性メッセージ機能のテスト

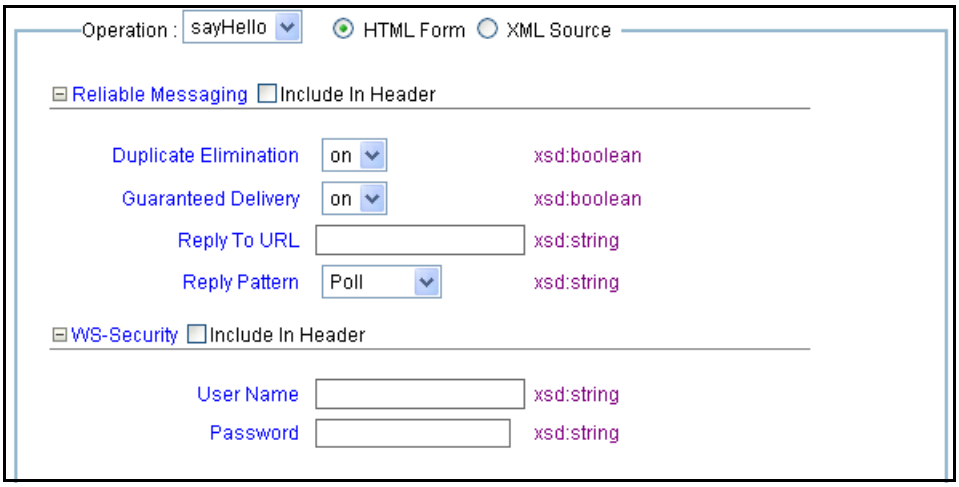
セキュリティおよび信頼性に対する値を変えて、JAX-RPC Web サービス操作を起動できます。これらのパラメータを公開するには、**「信頼できるメッセージング」**および**「WS セキュリティ」**の隣のノードを展開します。

注意： OracleAS Web Services によって実装される REST Web サービスは、セキュリティまたは信頼性の機能をサポートしていません。REST Web サービスのリクエストに対してセキュリティまたは信頼性のオプションを指定した場合は、無視されます。

図 13-4 は、HTML フォームでのセキュリティおよび信頼性に対するパラメータとデフォルト値を示しています。

ヒント： パラメータに値を指定した後、**「XML ソース」**ラジオ・ボタンをクリックすると、SOAP リクエストに挿入された信頼性およびセキュリティのヘッダーを確認できます。

図 13-4 テスト・ページでの WS-Security および信頼性メッセージ・コンポーネント



信頼できるメッセージのパラメータ

信頼性機能をテストに含めることを示すには、**「信頼できるメッセージング」**ノードを展開します。信頼性 SOAP ヘッダーがリクエストの SOAP エンベロープに挿入されます。

次のパラメータの設定を変更できます。

- **重複削除** : この機能をオンにすると、**「重複削除」**信頼性ヘッダーがメッセージに挿入されます。これにより、送信されるメッセージの重複が削除されます。デフォルト値は**オン**です。
- **保証付き配信** : この機能をオンにすると、**「保証付き配信」**信頼性ヘッダーがメッセージに挿入されます。これにより、メッセージの受信が確認されます。デフォルト値は**オン**です。
- **URL への返信** : 非同期確認が必要なメッセージに関する確認および障害の送信先となる URL を示します。通常、この URL は、リスナーがポートをリスニングするクライアントのホスト名です。
- **返信パターン** : クライアントがエンドポイントと対話する方法を指定します。この値には、**コールバック**（非同期確認 / 障害）、**レスポンス**（同期確認 / 障害）または**ポーリング**（確認または障害のポーリングの必要あり）があります。デフォルト値は**ポーリング**です。

関連資料 :

これらの信頼性機能の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの信頼性の確保」を参照してください。

WS-Security のパラメータ

セキュリティ機能をテストに含めることを示すには、**「WS セキュリティ」**チェック・ボックスを選択します。セキュリティ SOAP ヘッダーが SOAP エンベロープに挿入されます。

次のパラメータの設定を変更できます。

- ユーザー名
- パスワード

関連資料 :

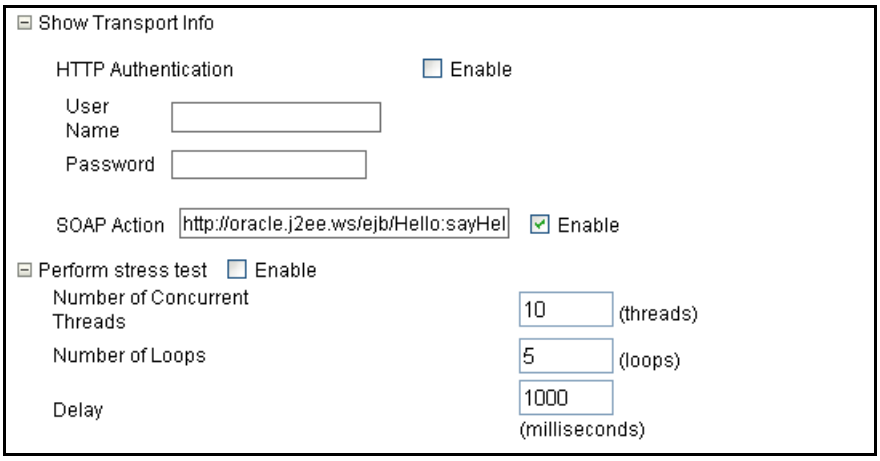
OracleAS Web Services で使用可能なセキュリティ機能の詳細は、『Oracle Application Server Web Services セキュリティ・ガイド』を参照してください。

Web サービス・テストに対する HTTP 認証の有効化

「トランスポート情報の表示」ノードを展開すると、JAX-RPC または REST Web サービスに対する HTTP 認証オプションが表示されます。テストしている HTTP サービスがパスワードで保護されている場合は、**「トランスポート情報の表示」**の下のパラメータを使用すると、ユーザー名とパスワードを提供できるようになります。SOAP リクエストに対する特殊なフィルタを提供する必要のあるサービスの場合は、**「SOAP アクション」**に値を指定できます。

図 13-5 は、テスト・ページの**「トランスポート情報の表示」**パラメータです。

図 13-5 テスト・ページでの HTTP 認証およびストレス・テストのパラメータ



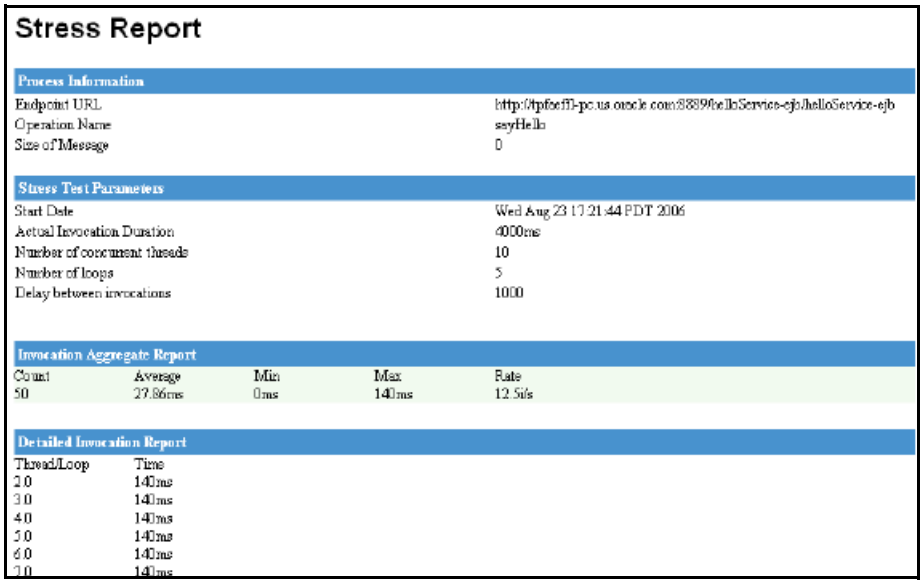
Web サービス操作のストレス・テスト

「ストレス・テストの実行」ノードを展開すると、JAX-RPC または REST Web サービス操作の連続的な一連の起動を作成および構成するためのオプションが表示されます。**「ストレス・テストの実行」**のオプションは図 13-5 に示されています。

- **同時スレッド数**: 起動を送信する同時スレッドの数です。デフォルトは 10 スレッドです。
- **ループ数**: 操作を起動する回数です。デフォルトはスレッドごとに 5 回です。
- **遅延**: 操作の起動の間で待機するミリ秒数です。デフォルト値は 1000 ミリ秒（1 分）です。

テストを起動すると、ストレス・レポート・ページが返されます。レポート・ページでは、テストされたサービス・エンドポイントと操作、送信されたメッセージのサイズ、実行された同時スレッドの数、スレッドごとの実行回数および各操作起動の間の遅延が示されます。図 13-6 は、サンプルのストレス・テスト・レポートのフラグメントです。

図 13-6 サンプルのストレス・テスト・レポート



Stress Report

Process Information

Endpoint URL	http://tpfbeff-pc.us.oracle.com:8889/helloService-ejb/helloService-ejb
Operation Name	sayHello
Size of Message	0

Stress Test Parameters

Start Date	Wed Aug 23 17:21:44 PDT 2006
Actual Invocation Duration	4000ms
Number of concurrent threads	10
Number of loops	5
Delay between invocations	1000

Invocation Aggregate Report

Count	Average	Min	Max	Rate
50	27.86ms	0ms	140ms	12.5/s

Detailed Invocation Report

Thread/Loop	Time
2 0	140ms
3 0	140ms
4 0	140ms
5 0	140ms
6 0	140ms
7 0	140ms

Web サービスのテストの起動

操作でテストするパラメータ値と機能を指定したなら、Web サービス・エンドポイントにメッセージを送信できます。JAX-RPC Web サービスの場合は、メッセージは SOAP リクエストとしてサービスに送信されます。REST Web サービスの場合は、SOAP リクエスト、XML REST GET 操作または POST 操作のいずれかとしてメッセージを送信できます。

- JAX-RPC Web サービスのテストの起動
- REST Web サービスのテストの起動

JAX-RPC Web サービスのテストの起動

SOAP リクエストとして JAX-RPC Web サービス・エンドポイントにメッセージを送信するには、「**起動**」ボタンをクリックします。テスト・ページにサービスからのレスポンスが表示されます。レスポンスは、書式設定された XML（デフォルト）または RAW XML（ワイヤ書式）で表示できます。

図 13-7 JAX-RPC Web サービスからのレスポンス

REST Web サービスのテストの起動

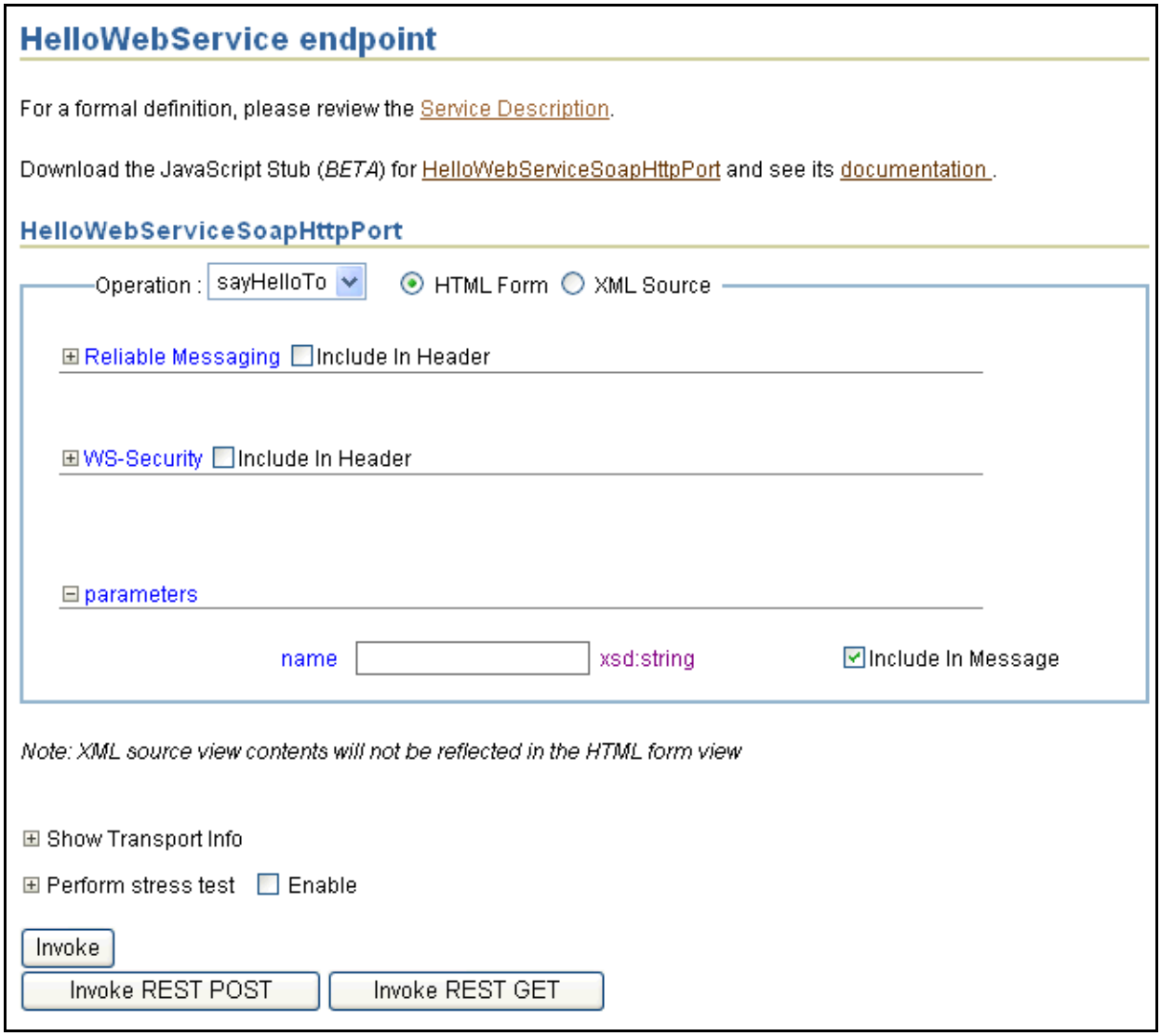
REST Web サービス用のテスト・ページでは、XML REST POST 操作または GET 操作として REST サービスにテスト・メッセージを送信できます。さらに、OracleAS Web Services では、SOAP リクエストとしてメッセージを送信することもできます。

テスト・ページにある次のボタンを使用して、テスト・メッセージに対する Web サービス操作を起動できます。

- 起動 : SOAP リクエストとしてリクエストを起動します。
- REST POST の起動 : REST POST リクエストを生成して起動します。
- REST GET の起動 : GET URL にページをリダイレクトします。

図 13-8 は、REST に対応した Web サービス操作用のテスト・ページです。

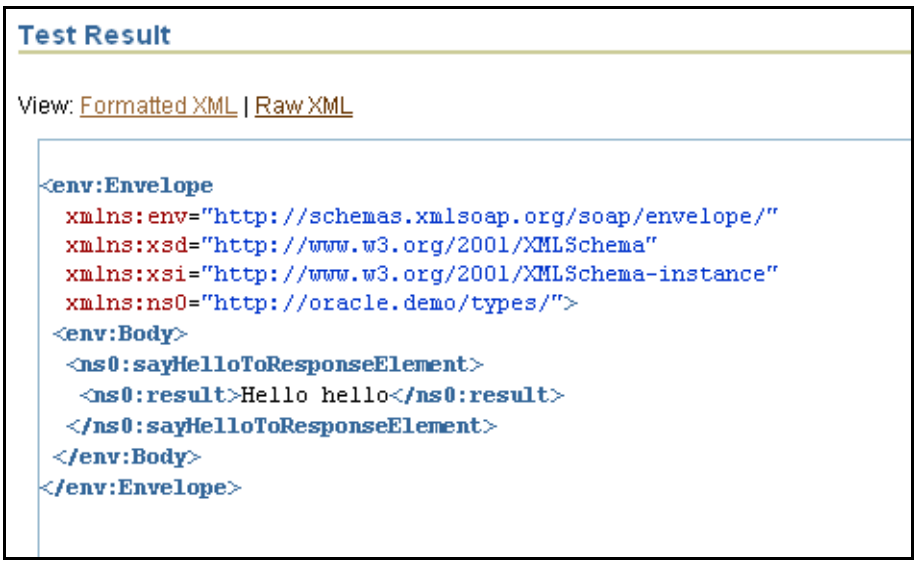
図 13-8 REST Web サービス操作用のテスト・ページ



起動 HTTP 経由で SOAP メッセージとしてサービスに XML REST リクエストを送信するには、「**起動**」ボタンをクリックします。サービスは、SOAP レスポンス・メッセージをテスト・ページ・アプリケーションに戻します。レスポンスは、書式設定された XML（デフォルト）または RAW XML（ワイヤ書式）で表示できます。

図 13-9 は、書式設定された XML で表示された REST Web サービスからの SOAP レスポンスです。

図 13-9 書式設定された XML の SOAP メッセージとして表示された REST サービスからのレスポンス



REST POST の起動 REST Web サービスにメッセージを送信するには、テスト・ページの「**REST POST の起動**」をクリックします。レスポンスがテスト・ページ・アプリケーションに戻されます。レスポンスは、書式設定された XML（デフォルト）または RAW XML（ワイヤ書式）で表示できます。

注意：

- OracleAS Web Services によって実装される REST Web サービスは、セキュリティまたは信頼性の機能をサポートしていません。REST Web サービス・リクエストにこれらのオプションを指定した場合は、無視されます。
- REST Web サービスのレスポンスには、SOAP の `Envelope` 要素または `Body` 要素は含まれません。

図 13-10 は、REST POST 操作からの REST Web サービス・レスポンスです。

図 13-10 書式設定された XML での REST POST 操作からのレスポンス

REST GET の起動 Web ブラウザでの HTML GET コマンドとしてサービスにリクエストを送信するには、テスト・ページの**「REST GET の起動」**をクリックします。レスポンスは、テスト・ページ・アプリケーションではなくブラウザに表示されます。

注意 :

- OracleAS Web Services によって実装される REST Web サービスは、セキュリティまたは信頼性の機能をサポートしていません。REST Web サービス・リクエストにこれらのオプションを指定した場合は、無視されます。
- REST Web サービスのレスポンスには、SOAP の `Envelope` 要素または `Body` 要素は含まれません。

図 13-11 は、REST GET 操作からの REST Web サービス・レスポンスです。

図 13-11 REST GET 操作からのレスポンス

Web サービス・テスト・ページを無効にする方法

テスト・ページを無効にすることで、Web サービスを公開するアプリケーションの詳細が外部から見られにくくなり、セキュリティが向上します。

テスト・ページを無効にするための一般的な手順は、次のとおりです。

1. デプロイの前に、JAR ファイルから `oracle-webservices.xml` デプロイメント・ディスクリプタを抽出します。
2. デプロイメント・ディスクリプタの `<webservice-description>` セクションに、`false` に設定したブール型要素 `<expose-testpage>` を挿入します。

   例 13-2 は、この手順を示したものです。
3. 編集済の `oracle-webservices.xml` デプロイメント・ディスクリプタで、JAR ファイルを置き換えます。
4. 通常どおり JAR ファイルをデプロイします。

> **注意：** `ddFileName` コマンドでデプロイメント・ディスクリプタを渡して Web サービスを再アセンブルすると、このエントリは上書きされます。

例 13-2 は、テスト・ページを無効にした `oracle-webservices.xml` デプロイメント・ディスクリプタのフラグメントです。

例 13-2 oracle-webservices.xml ファイルでの Web サービス・テスト・ページの無効化

```
<oracle-webservices>
  ...
  <webservice-description name="with-test-page-blocked">
    <expose-testpage>false</expose-testpage>
     ...
   </webservice-description>
</oracle-webservices>
```

Web サービスの WSDL を直接取得する方法

Web サービス・テスト・ページを使用しないで Web サービスの WSDL ファイルを取得する場合は、直接取得できます。

WSDL を取得するには、Web サービスの URL を使用して問合せ文字列を追加します。WSDL サービス記述を取得するための URL の書式は、次のとおりです。

`http://`*`host:port/context-root/service`*`?WSDL`

`WSDL` は大文字でも小文字でもかまいません。URL コンポーネントの詳細は、13-3 ページの表 13-1 を参照してください。

この URL では、WSDL 記述が *service*`.wsdl` の形式で戻されます。*service*`.wsdl` の記述には、指定した URL にある Web サービス *service* の WSDL が含まれています。この WSDL を使用すると、その Web サービスにアクセスするクライアント・アプリケーションを作成できます。

制限事項

F-10 ページの「Web サービス・デプロイのテスト」を参照してください。

追加情報

Web サービス・テスト・ページを使用できる Web サービスの詳細は、次の章を参照してください。

- 第 6 章「WSDL からの Web サービスのアセンブル」
- 第 7 章「Java クラスを使用した Web サービスのアセンブル」
- 第 8 章「EJB を使用した Web サービスのアセンブル」
- 第 10 章「データベース Web サービスのアセンブル」
- 第 12 章「REST Web サービスのアセンブル」
- 『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービス・プロバイダの使用方法」

Web Services Security および Web Services Reliability の詳細は、次のマニュアルを参照してください。

- Web サービスへのセキュリティの追加方法は、『Oracle Application Server Web Services セキュリティ・ガイド』を参照してください。
- Web サービスへの信頼性の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの信頼性の確保」を参照してください。

14

J2EE Web サービス・クライアントのアセンブル

この章では、J2EE コンテナ内での Web サービス・クライアントの開発方法を説明します。バージョン 2.4 のサーブレット、2.1 の EJB または 2.0 の JSP アプリケーションなど、J2EE 1.4 互換コンテナの任意のコンポーネントを、J2EE Web サービス・クライアントとして使用できます。

この章の内容は、次のとおりです。

- J2EE Web サービス・クライアントの概要
- J2EE Web サービス・クライアントのアセンブル方法
- J2EE Web サービスのクライアント・コードの記述
- J2EE クライアントのパッケージ構造の概要

関連項目：

J2SE 環境で動作する Web サービス・クライアントのアセンブルの詳細は、第 15 章「J2SE Web サービス・クライアントのアセンブル」を参照してください。

J2EE Web サービス・クライアントの概要

J2EE プラットフォームにより、クライアント・アプリケーションが Web サービスにアクセスするための環境が提供されます。J2EE 環境では、クライアント・サイドからの Web サービスへのアクセス情報は、デプロイメント・ディスクリプタによって定義します。このアクセス情報はデプロイ時に変更できます。さらに、J2EE プラットフォームは、Web サービスへのアクセスの作成および初期化を支援します。

J2EE Web サービス・クライアントは、宣言的なセキュリティ、トランザクションおよびインスタンス管理など、J2EE プラットフォームのメリットを継承します。プラットフォームの持つこれらの特性と一緒に OracleAS Web Services の管理フレームワークを利用することで、SOAP のロギングおよび監査、WS-Reliability および WS-Security を構成できるようになります。

J2SE Web サービス・クライアントとは異なり、J2EE クライアントは OC4J コンテナ内に存在し、OC4J コンテナによって管理されます。プロキシ・コードの生成やパッケージ化は不要です。JSP、サーブレットおよび EJB に容易に埋め込んで移植できる、Web サービス・アクセス用のクライアント・アプリケーションを構築できます。コンテナ管理の永続性（CMP）、Bean 管理の永続性（BMP）およびメッセージドリブン Bean（MDB）などの EJB 変数が、Web サービス・エンドポイントをコールアウトできます。

J2EE Web サービス・クライアントのアセンブル方法

必要な情報: 次の各項には、J2EE Web サービス・クライアントのアセンブルに必要な手順が含まれています。

- 前提条件
- J2EE Web サービス・クライアントのアセンブル手順
- J2EE Web サービス・クライアント情報のデプロイメント・ディスクリプタへの追加方法

オプションの情報: 次の各項には、作成するクライアントのタイプまたはクライアントが使用する機能に依存する、オプションの情報が含まれる場合があります。

- アプリケーション・クライアント・モジュールのデプロイと実行の手順
- 同じモジュール内のクライアントからの Web サービスへのアクセス方法
- デプロイおよび実行時に対する OC4J 固有のプラットフォーム情報の追加方法
- メッセージ処理用 JAX-RPC ハンドラのデプロイメント・ディスクリプタへの追加方法

前提条件

開始する前に、次のファイルと情報を用意してください。

- サービス・エンドポイント・インタフェースおよび JAX-RPC マッピング・ファイルを生成する基となる、WSDL ファイルかその位置。
- 生成されるサービス・エンドポイント・インタフェースおよび JAX-RPC マッピング・ファイルを格納する位置。

J2EE Web サービス・クライアントのアセンブル手順

WebServicesAssembler ツールを使用して、サービス・エンドポイント・インタフェースおよび J2EE Web サービス・クライアントをアセンブルします。次に、デプロイメント・ディスクリプタを編集し、Web サービスへのアクセス情報を追加します。次の手順でこれらのタスクを詳細に説明します。

1. 「前提条件」の項で説明した WSDL および情報を、WebServicesAssembler の `genInterface` コマンドに入力します。

   次の例では、`HelloService.wsdl` を使用して、`HelloInterface` を `oracle.demo.hello` パッケージに生成します。

   コマンドライン :

   ```
   java -jar wsa.jar
        -genInterface
        -wsdl HelloService.wsdl
        -output build
        -packageName oracle.demo.hello
   ```

   Ant タスク :

   ```
   <oracle:genInterface wsdl="${etc.web1.dir}/HelloService.wsdl"
           output="build"
           packageName="oracle.demo.hello"
   />
   ```

   このコマンドおよび Ant タスクの説明 :

   - `genInterface`: 各ポート・タイプのサービス・エンドポイント・インタフェースと、WSDL に定義された複合型に対応する Java 値タイプ・クラス（Bean）を作成します。また、XML スキーマ型と Java 値タイプ・クラス間のマッピングを記述する JAX-RPC マッピング・ファイルも作成されます。18-30 ページの「genInterface」を参照してください。
   - `wsdl`: WSDL ドキュメントの絶対ファイル・パス、相対ファイル・パスまたは URL を指定します。18-64 ページの「wsdl」を参照してください。
   - `output`: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
   - `packageName`: JAX-RPC マッピング・ファイルにパッケージ名が宣言されていない場合に、生成されたクラス用に使用されるパッケージ名を指定します。18-47 ページの「packageName」を参照してください。

2. J2EE コンポーネントのデプロイメント・ディスクリプタを編集し、`<service-ref>` 要素を追加します。この要素には、Web サービスへのアクセス情報のすべてが取得されます。

   `<service-ref>` 要素およびそのサブ要素のサンプルは、14-5 ページの「J2EE Web サービス・クライアント情報のデプロイメント・ディスクリプタへの追加方法」を参照してください。

   クライアントが JAX-RPC ハンドラの形式でのメッセージ処理も行う場合は、これらのハンドラもデプロイメント・ディスクリプタに追加しておく必要があります。デプロイメント・ディスクリプタへのハンドラ情報の追加の詳細は、14-13 ページの「メッセージ処理用 JAX-RPC ハンドラのデプロイメント・ディスクリプタへの追加方法」を参照してください。

3. クライアントのデプロイ・モジュールを、EAR ファイルにアセンブルします。

   a. すべてのクライアント・ファイルをコンパイルします。

   b. デプロイメント・ディスクリプタ・ファイルを適切な位置にコピーします。

      たとえば、EJB の場合は、WSDL を `META-INF/wsdl/` にコピーし、JAX-RPC マッピング・ファイルおよび `ejb-jar.xml` や `orion-ejb-jar.xml` などのデプロイメント・ファイルは `META-INF` にコピーします。サーブレット、EJB または JSP Web サー

ビス・クライアントの各ファイルの位置は、14-19 ページの「J2EE クライアントのパッケージ構造の概要」を参照してください。

c. クライアントのデプロイ・モジュールをパッケージします。

注意： 現行のツール・セットでは J2EE Web サービス・クライアントをパッケージできません。J2EE Web サービス・クライアントは手動でパッケージする必要があります。パッケージ方法の詳細は、14-19 ページの「J2EE クライアントのパッケージ構造の概要」を参照してください。

---

4. クライアントのデプロイ・モジュールをデプロイします。

EJB、JSP またはその他の J2EE クライアントをデプロイするには、次のステップに従います。アプリケーション・クライアントをデプロイする場合は、これらのステップをスキップし、「アプリケーション・クライアント・モジュールのデプロイと実行の手順」に進みます。

a. OC4J を起動します。次は、OC4J を起動するコマンドのサンプルです。

```
java -jar oc4j.jar
```

b. クライアント・モジュールを OC4J にデプロイします。次は、デプロイメント・コマンドのサンプルです。

```
java -jar admin_client.jar deployer:oc4j:<oc4jHost>:<oc4jPort> <adminID>
<adminPassword>
         -deploy
         -file .\client\myClient.ear
         -deploymentName myClient
         -bindWebApp default-web-site
```

oc4jHost および *oc4jPort* 変数は、OC4J サーバーのホスト名およびポート番号です。*adminID* および *adminPassword* は、OC4J サーバーのユーザー名およびパスワードです。次は、`-deploy` スイッチのサブスイッチです。

- `file`: デプロイする EAR ファイルのパスおよびファイル名。
- `deploymentName`: ユーザー定義のアプリケーション・デプロイ名。OC4J 内でのアプリケーションの識別に使用します。
- `bindWebApp`: Web アプリケーションのバインド先の Web サイト。アプリケーションへのアクセスに使用する Web サイトです。

5. EJB または JSP クライアントを実行します。

アプリケーション・クライアントを実行する場合は、「アプリケーション・クライアント・モジュールのデプロイと実行の手順」を参照してください。

J2EE Web サービス・クライアント情報のデプロイメント・ディスクリプタへの追加方法

J2EE コンポーネントのデプロイメント・ディスクリプタを編集し、コンポーネントが Web サービス・エンドポイントにアクセスするための情報を追加する必要があります。

- EJB 2.1 の Web サービス・クライアントの場合は、`META-INF/ejb-jar.xml` デプロイメント・ディスクリプタを編集します。
- JSP 2.0 またはサーブレットの Web サービス・クライアントの場合は、`WEB-INF/web.xml` デプロイメント・ディスクリプタを編集します。
- アプリケーション・クライアントの場合は、`META-INF/application-client.xml` デプロイメント・ディスクリプタを編集します。

デプロイメント・ディスクリプタを編集し、`<service-ref>` 要素を追加します。この要素を追加することで、EJB、JSP またはサーブレットを、リモート Web サービスを起動可能な Web サービス・クライアントとして使用できます。`<service-ref>` 要素およびそのサブ要素には、Web サービスへのアクセス情報がすべて取得されます。この情報には、WSDL およびマッピング・ファイルの位置、サービス・インタフェース、サービス・ポート、サービス・ポートのサービス・エンドポイント・インタフェースなどがあります。`<service-ref>` 要素に格納される可能性のある情報の詳細は、`service-ref`（J2EE クライアント）スキーマを参照してください。

`http://java.sun.com/xml/ns/j2ee/j2ee_web_services_client_1_1.xsd`

例 14-1 は、`MyHelloService` Web サービス用の `web.xml` デプロイメント・ディスクリプタに追加された、サンプルの `<service-ref>` 要素を示します。この例に含まれる各 `<service-ref>` サブ要素を、表 14-1 で説明します。この `<service-ref>` のサンプルでは、スキーマで使用可能なすべての Web サービス・アクセス情報のサブセットのみが使用されている点に注意してください。

例 14-1 service-ref 要素のサンプルの内容

```
<service-ref>
    <service-ref-name>service/MyHelloServiceRef</service-ref-name>
    <service-interface>javax.xml.rpc.Service</service-interface>
    <wsdl-file>WEB-INF/wsdl/HelloService.wsdl</wsdl-file>
    <jaxrpc-mapping-file>WEB-INF/HelloService-java-wsdl-mapping.xml
    </jaxrpc-mapping-file>
    <service-qname xmlns:service-qname_ns__="http://hello.demo.oracle/">
    service-qname_ns__:HelloService</service-qname>
    <port-component-ref>
         <service-endpoint-interface>oracle.demo.hello.HelloInterface
         </service-endpoint-interface>
         <port-component-link></port-component-link>
    </port-component-ref>
</service-ref>
```

表 14-1 で、このサンプル内で使用されている <service-ref> サブ要素を説明します。

表 14-1 <service-ref> 要素のサブ要素

service-ref サブ要素	説明
<jaxrpc-mapping-file>	JAX-RPC マッピング・ファイルの完全修飾パスを指定します。
<port-component-link>（オプション）	Web サービスをクライアントと同じモジュールに実装してある場合は、この要素を追加することでサービスにアクセスできます。この要素の詳細は、「同じモジュール内のクライアントからの Web サービスへのアクセス方法」を参照してください。
<port-component-ref>	コンテナへのクライアントの依存性を宣言します。この依存性はサービス・エンドポイント・インタフェースを WSDL ポートに解決するために必要です。このサブ要素は、オプションとして、サービス・エンドポイント・インタフェースと特定のポート・コンポーネントの関連付けも行います。コンテナは、Service.getPort(Class) メソッド・コールに対してのみこれを使用します。
<service-endpoint-interface>	WSDL ポートのサービス・エンドポイント・インタフェースとなる完全修飾 Java クラスを指定します。
<service-interface>	クライアントが依存する JAX-RPC サービス・インタフェースの完全修飾クラス名を指定します。ほとんどの場合、値として javax.xml.rpc.Service を指定します。また、JAX-RPC の生成するサービス・インタフェース・クラスを指定することもできます。
<service-qname>	サービスのサービス QName を指定します。 ■ xmlns:ns: WSDL の targetNamespace 値にマッピングされます。 ■ ns: WSDL の service name 属性にマッピングされます。
<service-ref-name>	クライアントによって割り当てられる JNDI パスおよびサービス名を指定します。
<wsdl-file>	WSDL ファイルの完全修飾パスを指定します。

アプリケーション・クライアント・モジュールのデプロイと実行の手順

アプリケーション・クライアント・モジュールをデプロイおよび実行するには、次の手順に従ってください。EJB、JSP またはその他の J2EE クライアントの場合とは異なり、生成される deployment-cache.jar を格納するディレクトリを指定する必要があります。また、実行コマンドに deployment-cache.jar の位置を指定することも必要です。

1. OC4J を起動します。次は、OC4J を起動するコマンドのサンプルです。

   ```
   java -jar oc4j.jar
   ```

2. アプリケーション・クライアント・モジュールを OC4J にデプロイします。

   次は、デプロイメント・コマンドのサンプルです。

   ```
   java -jar admin_client.jar deployer:oc4j:<oc4jHost>:<oc4jPort> <adminID>
   <adminPassword>
            -deploy
            -file .¥client¥myAppClient.ear
            -deploymentName myAppClient
            -deploymentDirectory C:¥home¥myDir
   ```

   このコマンドは、deployment-cache.jar ファイルを作成し、C:¥home¥myDir に格納します。

oc4jHost、*oc4jPort*、*adminID*、*adminPassword* の各変数、および -deploy の file および deploymentName サブスイッチの詳細は、「J2EE Web サービス・クライアントのアセンブル手順」14-3 ページの**ステップ 4b** を参照してください。

deploymentDirectory サブスイッチには、OC4J による deployment-cache.jar のデプロイ先となる位置を指定します。この例では、OC4J は **C:¥home¥myDir** にこのファイルをデプロイします。このサブスイッチを指定しない場合は、OC4J はアプリケーションをディレクトリ *OC4J_HOME*/application-deployments/ にデプロイします。空の文字列（""）を指定すると、OC4J はアプリケーションをデプロイするたびに、常に EAR ファイルからデプロイ構成を読み取ります。

3. クライアントのデプロイ・モジュールを実行します。アプリケーション・クライアントの場合は、クラスパスに deployment-cache.jar の位置を含めておく必要があります。次は、実行コマンドのサンプルです。

```
java -classpath .:C:¥home¥myDir¥deployment-cache.jar:'oc4jclient.jar'
  :appclient.jar oracle.myappclient.classname
```

このサンプルでは、appclient.jar にクラス oracle.myappclient.*classname* が含まれているものと想定しています。

同じモジュール内のクライアントからの Web サービスへのアクセス方法

Oracle Application Server がクライアントと同じモジュールに実装されている Web サービスにアクセスできるようにするには、クライアントのデプロイメント・ディスクリプタの <service-ref> 句に <port-component-link> 要素を追加し、構成 system-application.xml に PortComponentLinkResolver プロパティを追加します。このタスクを、次の手順で簡単に説明します。

1. <port-component-link> 要素を、J2EE クライアントのデプロイメント・ディスクリプタの <service-ref> 句に追加します。

   このステップの詳細は、「J2EE クライアントのデプロイメント・ディスクリプタへのポート・コンポーネント・リンクの追加方法」を参照してください。

2. Oracle Application Server をシャットダウンします。

3. system-application.xml サーバー構成ファイルに、PortComponentLinkResolver プロパティを追加します。このファイルは、ディレクトリ *ORACLE_HOME*/j2ee/home/config にあります。

   次の行をこのファイルに追加します。

```
<ejb-module id="PortComponentLinkResolver"
path="../../../webservices/lib/wsserver.jar"/>
```

4. Oracle Application Server を再起動します。

J2EE クライアントのデプロイメント・ディスクリプタへのポート・コンポーネント・リンクの追加方法

Web サービスをクライアントと同じコンテナに実装してある場合に、サービスにアクセスできるようにするには、J2EE クライアントのデプロイメント・ディスクリプタ（web.xml、ejb-jar.xml または application-client.xml）の <service-ref> 句に <port-component-link> 要素を追加します。

<port-component-link> 要素は、<port-component-ref> を、サーバー・サイドのデプロイメント・ディスクリプタ内の特定ポート・コンポーネントにリンクします。<port-component-name> 要素は、サーバー・サイドのデプロイメント・ディスクリプタ、webservices.xml 内にあります。

次の各例はこの関係を示したものです。例 14-2 の webservices.xml フラグメントは、EJB InterModuleEjb を公開する Web サービスのデプロイ構成を示します。このフラグメントでは、ポート・コンポーネント名は InterPC です。例 14-3 に示すクライアント・サイドのデプロイメント・ディスクリプタでは、この名前が <service-ref> 句の <port-component-link> 要素から参照されています。この要素を使用することで、J2EE クライアントは Web サービスにアクセスできます。

これらの例では、Web サービスが J2EE Web サービス・クライアントと同じコンテナで動作しているものと想定しています。

例 14-2 webservices.xml フラグメント、ポート・コンポーネント名の識別

```
<webservices>
  <webservice-description>
    <webservice-description-name>InterModuleEjb</webservice-description-name>
    <wsdl-file>META-INF/wsdl/InterModuleService.wsdl</wsdl-file>
    <jaxrpc-mapping-file>META-INF/InterModuleService.xml</jaxrpc-mapping-file>
    <port-component>
      <port-component-name>InterPC</port-component-name>
      <wsdl-port
        xmlns:wsdl1="http://PortCompLink.org/ejb/inter">wsdl1:InterModuleSeiPort
      </wsdl-port>
      <service-endpoint-interface>oracle.demo.InterModuleSei
      </service-endpoint-interface>
      <service-impl-bean>
        <ejb-link>InterModuleEjb</ejb-link>
      </service-impl-bean>
    </port-component>
</webservices>
```

例 14-3 は、クライアント・サイドのデプロイメント・ディスクリプタのフラグメントを示します。ここでは、サーバー・サイドのデプロイメント・ディスクリプタ内の <port-component-name> 要素の値が、<service-ref> 句の <port-component-link> 要素から参照されています。この要素を使用することで、クライアントは Web サービスにアクセスできます。

<port-component-link> のポート・コンポーネント名の値には、接頭辞として PortCompLinkEjb-ejb.jar# が付けられている点に注目してください。この値は、EJB 名を、EJB 名が含まれている JAR ファイルの名前で修飾したものです。

例 14-3 クライアント・サイドのデプロイメント・ディスクリプタ内の <port-component-link> 要素

```
  <service-ref>
    <service-ref-name>service/portcomplink/inter</service-ref-name>
    <service-interface>javax.xml.rpc.Service</service-interface>
    <wsdl-file>META-INF/wsdl/InterModuleService.wsdl</wsdl-file>
    <jaxrpc-mapping-file>META-INF/InterModuleService.xml</jaxrpc-mapping-file>
    <port-component-ref>
       <service-endpoint-interface>oracle.demo.InterModuleSei
       </service-endpoint-interface>
       <port-component-link>PortCompLinkEjb-ejb.jar#InterPC</port-component-link>
```

```
    </port-component-ref>
  </service-ref>
```

デプロイおよび実行時に対する OC4J 固有のプラットフォーム情報の追加方法

<service-ref-mapping> 要素は、OC4J 固有のデプロイメント・ディスクリプタ・ファイルである orion-web.xml、orion-ejb-jar.xml または orion-application-client.xml の <orion-web-app> 要素のサブ要素として指定できます。このサブ要素によって、OC4J によって実行時およびデプロイ時に生成される独自の Web サービス参照設定を定義します。この要素を使用して、次の情報を構成できます。

- テートフル Web サービスを使用するためのクライアントの構成
- JMS トランスポート・コールを実行するためのクライアントの構成
- 対応する Web サービスに対するセキュリティ、ロギングおよび監査サービスのクオリティ（QOS）機能の構成

<service-ref-mapping> 要素は、標準のデプロイメント・ディスクリプタに含まれている <service-ref> 要素との組合せで使用します。<service-ref> 要素には、EJB、JSP またはサーブレットを、リモート Web サービスを起動するための Web サービス・クライアントとして使用するための情報を指定します。

<service-ref> 要素を web.xml、ejb-jar.xml または application-client.xml ファイルのいずれかに指定した場合には、対応する <service-ref-mapping> 要素を、orion-web.xml、orion-ejb-jar.xml または orion-application-client.xml ファイルのいずれかに指定できる点に注意してください。

<service-ref-mapping> 要素によってサポートされる機能の詳細は、orion-web、orion-ejb-jar および orion-application-client の各 XSD にインポートされている service-ref-mapping-10_0.xsd を参照してください。**service-ref-mapping-10_0.xsd** に値を設定するための、Oracle JDeveloper ウィザードのようなツールは、現時点ではサポートされていません。スキーマを参照し、XML ファイルを手動で編集する必要があります。

最も単純なケースでは、<service-ref-mapping> 要素にデプロイメント情報のみを指定します。クライアントをマネージド・クライアントとして使用しない場合は、ランタイム要素つまりサービスのクオリティ要素を追加しないでください。マネージド・クライアントは、パフォーマンスの点で高負荷です。

例 14-4 には、固有のデプロイメント・ディスクリプタで使用できる <service-ref-mapping> のサンプルが含まれます。この要素の階層がわかるように、サブ要素をすべて表示しています。この XML サンプルの後に、サブ要素について説明した表が続きます。

例 14-4 サンプルの <service-ref-mapping> セグメント

```
...
<service-ref-mapping name="service/MyJAXRPCTime">
  <service-impl-class>oracle.demo.MyTime_Impl</service-impl-class>
  <wsdl-file final-location="file:/myhome/mytime/client-wsdl/MyJAXRPCTime.wsdl">
  <wsdl-location wsdl-override-last-modified=19NOV>
  <service-qname namespaceURI="urn:oracle-ws" localpart="MyService" />
  <stub-property>
   <name>...</name>
   <value>...</value>
  </stub-property>
  <call-property>
     <name>javax.xml.rpc.service.endpoint.address</name>
     <value>http://myhost:8888/time-ejb/timeport</value>
  </call-property>
  <port-info>
    <wsdl-port>
    <service-endpoint-interface>time.TimeService</service-endpoint-interface>
    <stub-property>
      <name>another.endpoint.address</name>
```

```
      <value>http://anotherhost:8888/time-ejb/timeport</value>
    </stub-property>
    <call-property>
      <name>...</name>
      <value>...</value>
    </call-property>
    <runtime>...</runtime>
    <operations>
      <operation name="echo">
         <runtime>
           <auditing request="true" response="false" fault="false"/>
           <reliability><reject-non-reliable-messages value="false"/></reliability>
           ...
         </runtime>
      </operation>
   </operations>
  </port-info>
</service-ref-mapping>
...
```

表 14-2 は、<service-ref-mapping> のサブ要素の説明です。

表 14-2 <service-ref-mapping> 要素のサブ要素

要素名	説明
<call-property>	すべてのポートに適用可能な、コールのプロパティ値を定義します。これは、ポート名を指定せずにプロパティを指定できる便利な方法です。<call-property> の name および value サブ要素の詳細は、14-13 ページの表 14-6 を参照してください。 <port-info> 要素内にさらに <call-property> 要素を含めることができる点に注意してください。特定のポートに対して、<port-info> タグ内でコールのプロパティ値を指定する場合は、ここで設定する値がオーバーライドされます。
<port-info>	サービス参照のポートを定義します。<port-info> のサブ要素の詳細は、14-11 ページの表 14-3 を参照してください。
<service-impl-class>	Service 実装の、デプロイ時生成名を定義します。
<service-qname>	この要素はデプロイ時に導出され、Web サービスの QName を格納します。
<stub-property>	すべてのポートに適用可能な、スタブのプロパティ値を定義します。これは、ポート名を指定せずにプロパティを指定できる便利な方法です。<stub-property> の name および value サブ要素の詳細は、14-13 ページの表 14-6 を参照してください。 <stub-property> 要素を <port-info> 要素内で指定できる点にも注意してください。特定のポートに対して、<port-info> タグ内でスタブのプロパティ値を指定する場合は、ここで設定する値がオーバーライドされます。
<wsdl-file>	WSDL ファイルのデプロイ時生成名を定義します。この要素には、次の属性があります。 ■ final-location: 標準のデプロイメント・ディスクリプタの service-ref に指定されている WSDL ドキュメントのコピーを指します。

表 14-2 <service-ref-mapping> 要素のサブ要素（続き）

要素名	説明
<wsdl-location>	（オプション）WSDL ドキュメントを指す有効な URL を指定します。URL を指定すると、デプロイ時にこの URL の WSDL ドキュメントを、標準のデプロイメント・ディスクリプタで service-ref に指定された WSDL ドキュメントのかわりに使用します。<wsdl-location> のサンプル値としては、http://hostname:port/myservice/myport?WSDL および file:/home/user1/myfinalwsdl.wsdl があげられます。 この要素には、次の属性があります。 ■ wsdl-override-last-modified: このオプションの文字列値は、デプロイ時に生成され、WSDL ファイルが最後に変更された時間を表します。

表 14-3 は、<port-info> 要素のサブ要素の説明です。この要素は、サービス参照のポートの詳細を提供します。コンテナがコンテナ管理ポートの選択で使用するポートを、<service-endpoint-interface> または <wsdl-port> に指定します。両方を指定すると、<wsdl-port> の値が使用されます。<wsdl-port> や <service-endpoint-interface> を指定しない場合は、<port-info> プロパティの値が、すべての使用可能なポートに対して適用されます。

また、<port-info> 要素には、ポートおよびその操作に対して、使用可能なサービスのクオリティ機能を指定できるサブ要素も含めることができます。

表 14-3 <port-info> 要素のサブ要素

要素名	説明
<call-property>	<port-info> 要素に定義するポートに適用されるコール・プロパティ値を定義します。<call-property> の name および value サブ要素の詳細は、14-13 ページの表 14-6 を参照してください。 <service-ref-mapping> 要素内にさらに <call-property> サブ要素を含めることができる点に注意してください（14-10 ページの表 14-2 を参照）。特定のポートに対して、<port-info> タグ内でコールのプロパティ値を指定する場合は、<service-ref-mapping> に設定する <call-property> 要素の値がオーバーライドされます。
<operations>	操作ごとに要素を 1 つずつ指定した、一連の要素を指定します。個々の操作は、<operation> サブ要素に指定します。これらの各サブ要素には、参照先の Web サービスにより提供される各操作に対する、クライアント・サイドのサービスのクオリティ構成を指定します。<operations> のサブ要素の詳細は、14-12 ページの表 14-4 を参照してください。
<runtime>	参照先の Web サービスにより提供されるすべての操作に適用されるクライアント・サイドの、サービスのクオリティのランタイム情報（セキュリティまたは信頼性（あるいはその両方））を指定します。各機能の構成は、それぞれの子要素に指定します。
<service-endpoint-interface>	WSDL ポートのサービス・エンドポイント・インタフェースの完全修飾パスを指定します。コンテナは、このポートをコンテナ管理ポートの選択に使用します。

表 14-3 <port-info> 要素のサブ要素（続き）

要素名	説明
<stub-property>	<port-info> 要素に定義するポートに適用されるスタブ・プロパティ値を定義します。<stub-property> の name および value サブ要素の詳細は、14-13 ページの表 14-6 を参照してください。 <service-ref-mapping> 要素内にさらに <stub-property> サブ要素を含めることができる点に注意してください（14-10 ページの表 14-2 を参照）。特定のポートに対して、<port-info> タグ内でスタブのプロパティ値を指定する場合は、<service-ref-mapping> に設定する <stub-property> 要素の値がオーバーライドされます。
<wsdl-port>	コンテナによりコンテナ管理ポートの選択に使用される WSDL 内ポート名を指定します。 コンテナ管理ポートの選択では、コンテナがインスタンスのコールを直接管理し、クライアントは、複数の異なるインスタンスへのアクセスに使用できる汎用ポートをリクエストします。この要素には、次の属性があります。 ■ localpart: WSDL 名のローカル部分です。たとえば、authenticateHeader などです。 ■ namespaceURI: WSDL の名前空間の URI です。たとえば、http://oracle.j2ee.ws/Header などです。

表 14-4 は、<operations> 要素の <operation> サブ要素の説明です。

表 14-4 <operations> 要素のサブ要素

要素名	説明
<operation>	参照先の Web サービスにより提供される特定の操作に対する、クライアント・サイドのサービスのクオリティ構成を指定します。この構成の指定は、この要素の <runtime> サブ要素で行います。<runtime> サブ要素の詳細は、表 14-5 を参照してください。 この <operation> 要素には、次の属性があります。 ■ inputName: WSDL 内の操作の入力名を指定します。これは、name 属性を使用して操作を一意に特定できない場合にのみ必要です。 ■ name: 指定しているサービスのクオリティ構成を、この操作に関連付けます。属性の値は、WSDL 内の操作名と一致させる必要があります。 ■ outputName: WSDL 内の操作の出力名を指定します。これは、name および input 属性を使用して、操作を一意に特定できない場合にのみ必要です。

表 14-5 は、<operation> 要素の <runtime> サブ要素の説明です。

表 14-5 <operation> 要素のサブ要素

要素名	説明
<runtime>	ポート内の個々の操作に対する、クライアント・サイドのサービスのクオリティ構成を指定します。サービスのクオリティ機能（セキュリティ、信頼性または監査（あるいはこれらの任意の組合せ））の各構成は、それぞれの子要素に指定します。

表 14-6 は、<stub-property> および <call-property> 要素の、name および value サブ要素の説明です。

表 14-6 <stub-property> および <call-property> 要素のサブ要素

要素名	説明
<name>	JAX-RPC Call または Stub 実装でサポートされている任意のプロパティの名前を定義します。javax.xml.rpc.Call および javax.xml.rpc.Stub の有効なプロパティについては、Javadoc ツールの出力を参照してください。
<value>	Call オブジェクトまたは Stub オブジェクトを Web サービス・クライアントに戻す前にそのオブジェクトに設定する JAX-RPC プロパティ値を定義します。

メッセージ処理用 JAX-RPC ハンドラのデプロイメント・ディスクリプタへの追加方法

J2EE Web サービス・クライアントは、JAX-RPC ハンドラを使用することで、Web サービス・エンドポイントに関する追加のメッセージ処理機能を実現できます。たとえば、ハンドラを使用して SOAP メッセージを処理できます。

ハンドラ情報は、J2EE Web サービス・クライアントのデプロイメント・ディスクリプタ内で、<service-ref> のサブ要素として入力する必要があります。<handler> 要素にハンドラ情報をカプセル化します。

関連項目：

クライアント・サイド・ハンドラ、およびそのデプロイメント・ディスクリプタへの登録方法の詳細は、16-5 ページの「クライアント・サイドの JAX-RPC ハンドラ」を参照してください。

J2EE Web サービスのクライアント・コードの記述

この項の内容は、次のとおりです。

- J2EE Web サービス・クライアントの記述手順
- ステートフル Web サービス用 J2EE Web サービス・クライアントの構成方法
- JMS トランスポート・コール用の J2EE Web サービス・クライアントの構成方法
- HTTP 1.1 向けのチャンク・データ転送の有効化方法
- SOAP メッセージの文字コードの設定方法

J2EE Web サービス・クライアントの記述手順

この項では、J2EE コンポーネントが Web サービスにアクセスできるようにする共通コードのいくつかについて説明します。実行時には、すべての J2EE Web サービス・クライアントが標準 JNDI ルックアップを使用し、Web サービスを検索します。次に、サーブレット、EJB または JSP で使用できるように JNDI ルックアップをコーディングする際の一般的な手順を示します。

1. 初期 JNDI コンテキストの作成

   ```
   Context ic = new InitialContext();
   ```

   OC4J コンテナにより、初期コンテキストのプロパティが設定されます。

2. 初期コンテキストから lookup メソッドを使用し、サービスを検索します。例 14-5 では、comp/env/service/MyHelloServiceRef を使用してサービス参照を戻しています。この JNDI コールにより、service オブジェクトへの参照が戻されます。

   ```
   Service service = (Service) ic.lookup("java:comp/env/service/MyHelloServiceRef");
   ```

   クライアントは、常に JNDI ルックアップを使用してサービス実装にアクセスします。このルックアップにより、コンテナ管理のサービス参照が戻されます。これにより、コンテナは、クライアントに介入し、ロギング、セキュリティおよび管理などの追加サービス機能をクライアントに提供できます。

3. コンテナ管理サービス・オブジェクトに対して getPort メソッドを使用し、サービス・ポートのハンドルを取得します。戻り値をインタフェース・タイプにキャストします。

   ```
   HelloInterface helloPort = (HelloInterface) service.getPort(portQName,
   oracle.demo.hello.HelloInterface.class);
   ```

   このステップは QName が定義済であることが前提ですので、注意してください。次に例を示します。

   ```
   QName portQName = new QName("http://hello.demo.oracle/", "HelloInterfacePort");
   ```

   クライアントは、getPort のかわりに service オブジェクトを使用し、Call オブジェクトのハンドルを取得して、DII によって Web サービスをコールすることもできます。

   ```
   Call call = service.createCall(new QName("http://hello.demo.oracle/",
   "HelloInterfacePort");
   ```

4. リモート・オブジェクトのメソッドをコールします。

   ```
   resultFromService = helloPort.sayHello(name);
   ```

例 14-5 は、サーブレットまたは JSP の Web サービス・クライアントが Web サービスをルックアップするためのコードの例を示しています。

例 14-5 Web サービスをルックアップするサーブレット /JSP のコード

```
public String consumeService (String name)
{
   ......
   Context ic = new InitialContext();
   Service service = (Service)ic.lookup("java:comp/env/service/MyHelloServiceRef");
   // declare the qualified name of the port, as specified in the wsdl
   QName portQName= new QName("http://hello.demo.oracle/","HelloInterfacePort");
   //get a handle on that port : Service.getPort(portQName,SEI class)
   HelloInterface helloPort =
(HelloInterface) service.getPort(portQName,oracle.demo.hello.HelloInterface.class);
   //invoke the operation : sayHello()
   resultFromService = helloPort.sayHello(name);
   .....
}
```

ステートフル Web サービス用 J2EE Web サービス・クライアントの構成方法

J2EE Web サービス・クライアントは、構成ファイルまたはプログラムを使用して、ステートフル Web サービスを使用するように構成できます。

- デプロイメント・ディスクリプタ・ファイルを使用した J2EE クライアントの構成方法
- プログラムによる J2EE クライアントの構成方法

関連項目：

ステートフル Web サービスの詳細は、7-10 ページの「ステートフル Web サービスとしての Java クラスの公開」を参照してください。

デプロイメント・ディスクリプタ・ファイルを使用した J2EE クライアントの構成方法

ステートフル Web サービスを使用するように J2EE クライアントを構成するには、適切な Oracle 固有デプロイメント・ディスクリプタ（orion-web.xml、orion-ejb-jar.xml または orion-application-client.xml）の <service-ref-mapping> 句を編集します。

<service-ref-mapping> 句に <stub-property> 要素を追加します。具体的には、その <name> サブ要素に javax.xml.rpc.session.maintain プロパティを設定し、<value> サブ要素に true を指定します。

J2EE 標準プロパティ javax.xml.rpc.session.maintain の値を使用して、特定のサービス・エンドポイントでクライアントがセッションに参加するかどうかを設定します。このプロパティを true に設定することは、クライアントにセッションを持たせることを意味します。

例 14-6 は、ステートフル Web サービス用の Web サービス・クライアント構成を示します。<stub-property> の定義では、クライアントが、CycleCounterInterface サービス・エンドポイントで特定されるポートでセッションに参加できるようになっています。<stub-property> 要素とその設定である javax.xml.rpc.session.maintain を、太字で強調してあります。

例 14-6 ステートフル Web サービスに参加するクライアントの構成

```
     <service-ref-mapping name="service/CycleCounter">
         <port-info>

<service-endpoint-interface>test.oracle.stateful.CycleCounterInterface</service-endpoin
t-interface>
          <!-- set the javax.xml.rpc.session.maintain property to true for a stateful
client -->
                <stub-property>
                     <name>javax.xml.rpc.session.maintain</name>
                     <value>true</value>
                </stub-property>
                <stub-property>
                     <name>javax.xml.rpc.service.endpoint.address</name>

<value>http://%J2EE_HOST%:%HTTP_PORT%/testsfWS-session/testsfWS-session</value>
                 </stub-property>
           </port-info>
     </service-ref-mapping>
```

プログラムによる J2EE クライアントの構成方法

J2EE クライアントをプログラムで構成し、ステートフル Web サービスを使用できます。このような構成を行うには、スタブ、DII コールまたはエンドポイント・クライアント・インスタンスにおいて、SESSION_MAINTAIN_PROPERTY ランタイム・プロパティ（javax.xml.rpc.session.maintain）を true に設定して、クライアントがセッションに参加できるようにします。

たとえば、このプロパティの値を、javax.xml.rpc.Stub の生成される実装 _port 内で設定できます。

```
((Stub)__port)._setProperty(Stub.SESSION_MAINTAIN_PROPERTY, Boolean.valueOf
(maintainSession));
```

このプロパティを直接設定するかわりに、OracleAS Web Services に用意されている、setMaintainSession(boolean) メソッドがあるラッパー・クラスを使用することもできます。このメソッドを true に設定すると、セッションが持たれます。このようなプロパティの設定は、クライアント内でラッパーによって行います。たとえば、クライアント・コードに次のように入力します。

```
HttpSoap11Client c = new HttpSoap11Client(); // client wrapper class
c.setMaintainSession(true);
```

JMS トランスポート・コール用の J2EE Web サービス・クライアントの構成方法

J2EE クライアントを静的に構成し、JMS トランスポート・コールを実行できます。このような構成を行うには、Web サービス用の適切な Oracle 固有 J2EE クライアント用デプロイメント・ディスクリプタ・ファイル（orion-web.xml、orion-ejb-jar.xml または orion-application-client.xml）に、<service-ref-mapping> 句を追加します。この句の中で、次の各アイテムに対して、<stub-property> 要素を name および value 属性によって構成します。

- ReplyTo キュー：<stub-property> 要素を、name サブ要素を ReplyToQueueName API（oracle.webservices.transport.ReplyToQueueName）に、value サブ要素を ReplyTo キューの JNDI 名に設定して、入力します。
- ReplyTo ファクトリ名：<stub-property> 要素を、name サブ要素を ReplyToFactoryName API（oracle.webservices.transport.ReplyToFactoryName）に、value サブ要素を ReplyTo ファクトリの JNDI 名に設定して、入力します。
- サービス・エンドポイント・アドレス：<stub-property> 要素を、name サブ要素をサービス・エンドポイント・アドレス API（javax.xml.rpc.service.endpoint.address）に、value サブ要素をサービス・エンドポイント・インタフェース・ファイルに設定して入力します。

例 14-7 は、サンプルの構成を示します。

例 14-7 JMS トランスポート・コール用の J2EE クライアント構成

```
<service-ref-mapping name="service/MyJMSService">
     <stub-property>
        <name>oracle.webservices.transport.ReplyToQueueName</name>
        <value>jms/receiverQueue</value>
     </stub-property>
     <stub-property>
        <name>oracle.webservices.transport.ReplyToFactoryName</name>
        <value>jms/receiverQueueConnectionFactory</value>
     </stub-property>
     <stub-property>
        <name>javax.xml.rpc.service.endpoint.address</name>
        <value>/bank/soap12bank</value>
     </stub-property>
</service-ref-mapping>
```

HTTP 1.1 向けのチャンク・データ転送の有効化方法

プロトコルが HTTP 1.1 の場合、OracleAS Web Services では、メッセージのチャンク転送エンコーディングが可能です。チャンク・データ転送は、J2SE スタブ、J2EE スタブおよび DII Web サービス・クライアントから起動できます。

チャンクを使用すると、ペイロードが小さなピースに分割されるので、パフォーマンスが向上します。これらのピースは、ケーブル中を、大きなペイロードよりも高速に送信できます。チャンク転送エンコーディングには、受信者が、自分がメッセージ全体を受信できたかどうかを確認するために必要なすべての情報が含まれます。チャンク転送エンコーディングは、トランスポート・レベルで実行されるため、Web サービス・コールの起動元またはサーバーによって検出および処理されることはありません。

チャンクの有効化およびチャンク・サイズの設定を行うには、`oracle.webservices.ClientConstants` クラスの次のプロパティを、`Stub` または `Call` オブジェクトに設定します。

- `DO_NOT_CHUNK`: このプロパティを設定しないか、または `true` に設定すると、チャンクはオフになります（デフォルト）。このプロパティを `false` に設定すると、チャンクが有効になります。
- `CHUNK_SIZE`: チャンク・サイズをバイト単位で設定します。このプロパティを設定しない場合、デフォルトでチャンク・サイズは 4096 バイトになります。

例 14-8 は、クライアント・プロキシ・スタブ・コード内でのチャンクの設定およびチャンク・サイズのプロパティの設定を示しています。

例 14-8 データのチャンク・サイズを設定するスタブ・コード

```
import oracle.webservices.ClientConstants;
...
((OracleStub)port)._setProperty(ClientConstants.DO_NOT_CHUNK, true);
((OracleStub)port)._setProperty(ClientConstants.CHUNK_SIZE, 1024);
...
```

例 14-9 は、DII クライアント・コード内で `DO_NOT_CHUNK` および `CHUNK_SIZE` プロパティを使用してチャンク・サイズを 1024 バイトに設定するための方法を示しています。

例 14-9 データのチャンク・サイズを設定する DII クライアント・コード

```
import oracle.webservices.ClientConstants;
...
ServiceFactory factory = ServiceFactory.newInstance();
Service service = factory.createService(new
QName("http://whitemesa.net/wsdl/rpc-lit-test", "tns" ) );
QName stringType = new QName( "http://www.w3.org/2001/XMLSchema", "string");
Call call = service.createCall();
...
call.setProperty(ClientConstants.DO_NOT_CHUNK, false);
call.setProperty(ClientConstants.CHUNK_SIZE, 1024);
...
```

SOAP メッセージの文字コードの設定方法

デフォルトでは、OracleAS Web Services でアセンブルされた J2EE クライアント（静的スタブまたは DII）は、UTF-8 でエンコードされた SOAP エンベロープを持つリクエスト・メッセージを送信します。この動作をオーバーライドする場合は、次の Oracle 固有のプロパティを設定します。

```
oracle.webservices.ClientConstants.CHARACTER_SET_ENCODING
```

このプロパティは、`javax.xml.rpc.Stub` または `javax.xml.rpc.Call` オブジェクトに対して、`setProperty` メソッドを使用して適用できます。

`CHARACTER_SET_ENCODING` プロパティの値は、`java.lang.String` または `java.nio.charset.Charset` 型のいずれかを使用できます。サポートされる文字コード・セットは、背後で動作する Java Virtual Machine（JVM）に依存します。JVM でサポートされる文字コードのリストを取得するには、`Charset.availableCharsets` メソッドを使用します。`Charset.availableCharsets` メソッドの詳細は、`java.nio.charset.Charset` クラスの Javadoc ツール出力を参照してください。

また、このプロパティは、J2SE Web サービス・クライアント用にも使用できます。

例 14-10 は、SOAP エンベロープで使用する文字コードとして `Shift_JIS` を設定したスタブ・クライアント・コードを示します。

例 14-10 スタブ・クライアントにおける、SOAP エンベロープに対する文字コード Shift_JIS の設定

```
import oracle.webservices.ClientConstants;
...
((OracleStub)port)._setProperty(ClientConstants.CHARACTER_SET_ENCODING, "Shift_JIS");
...
```

例 14-11 は、SOAP エンベロープで使用する文字コードとして `Shift_JIS` を設定した DII クライアント・コードを示します。

例 14-11 DII クライアントにおける、SOAP エンベロープに対する文字コード Shift_JIS の設定

```
import oracle.webservices.ClientConstants;
...
ServiceFactory factory = ServiceFactory.newInstance();
Service service = factory.createService(new URL("path to wsdl"),
new QName("service namespace", "service name" ) );
Call call = service.createCall();
call.setProperty(ClientConstants.CHARACTER_SET_ENCODING, Charset.forName("Shift_JIS"));
...
```

J2EE クライアントのパッケージ構造の概要

Oracle JDeveloper は、Web アプリケーションおよび EJB クライアント・ファイルに対して標準パッケージ構造を作成します。この項では、クライアント EAR ファイルのコンテンツのカスタマイズが必要な場合のために、この構造を説明します。

- サーブレットまたは Web アプリケーション・クライアントのパッケージ構造の概要
- EJB クライアントのパッケージ構造の概要

サーブレットまたは Web アプリケーション・クライアントのパッケージ構造の概要

この項では、サーブレットまたは Web アプリケーション・クライアントのパッケージ化について説明します。デプロイメント・ディスクリプタ内のいくつかの要素の値には、EAR ファイル内でのファイル名および格納位置が反映されます。EAR ファイルの内容を変更した場合は、デプロイメント・ディスクリプタの内容の変更が必要になります。

- サーブレットまたは Web アプリケーション・クライアントのパッケージの構造
- サーブレット用または Web アプリケーション・クライアント用のデプロイメント・ディスクリプタ間の関係

サーブレットまたは Web アプリケーション・クライアントのパッケージの構造

サーブレットまたは Web アプリケーション・クライアントは、*<ear_file_name>*.ear という名前の EAR ファイルにパッケージ化されます。EAR ファイルでは、トップレベルにマニフェスト・ファイルおよび application.xml ファイルのための META-INF ディレクトリ、サーブレットまたは Web アプリケーション・ファイル、JAX-RPC マッピング・ファイル、WSDL ファイルおよびデプロイメント・ディスクリプタのための *<war_file_name>*.war ファイルがあります。例 14-12 は、EAR ファイルの標準パッケージの構造を示します。

例 14-12 サーブレットまたは Web アプリケーション・クライアントの EAR ファイルの構造

```
./META-INF
    ./MANIFEST.MF
    ./application.xml
./<war file>.war
    ./WEB-INF/
            /orion-web.xml
            /web.xml
            /wsdl/<wsdl file name>.wsdl
            /<mapping file>.xml
            /classes
                 /class files
            /lib
                 /.jar files
    ./*.jsp or html files
```

サーブレット用または Web アプリケーション・クライアント用のデプロイメント・ディスクリプタ間の関係

この項では、J2EE 標準デプロイメント・ディスクリプタ web.xml、サーブレットまたは Web アプリケーションのための OC4J デプロイメント・ディスクリプタ orion-web.xml、およびクライアント EAR ファイルのパッケージの構造の間の関係について説明します。クライアント EAR ファイルの構造または内容を編集すると、デプロイメント・ディスクリプタの内容も編集が必要になるため、これらの関係は重要です。

クライアント情報は、web.xml の <service-ref> 要素に指定します。この要素には、サーブレットまたは JSP の内部からルックアップおよび使用できる Web サービスの情報を指定します。たとえば、次のものの位置を指定します。WSDL（<wsdl-file>）、JAX-RPC マッピング・ファイル（<jaxrpc-mapping-file>）、JNDI ルックアップで使用するサービス・インタフェース（<service-ref-name>）、サービス・インタフェース・クラス（<service-interface>）およびサービス・エンドポイント・インタフェース（<service-endpoint-interface>）。web.xml の <service-ref-name> は、orion-web.xml の <service-ref-mapping> 要素の属性としても指定される点に注意してください。EAR の前述したアイテムの名前および位置を変更する場合は、デプロイメント・ディスクリプタも対応するように変更する必要があります。

関連項目：

<service-ref> 要素および <service-ref-mapping> 要素とそれぞれのサブ要素については、14-5 ページの「J2EE Web サービス・クライアント情報のデプロイメント・ディスクリプタへの追加方法」を参照してください。

例 14-13 は、サーブレットまたは Web アプリケーション・クライアントのための web.xml の内容を示しています。<service-ref> 要素を太字で強調してあります。

例 14-13 サーブレットまたは Web アプリケーション・クライアントのための web.xml の内容

```
<web-app>
  <servlet>
    <servlet-name>consumer</servlet-name>
    <servlet-class>oracle.ServiceConsumerServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>consumer</servlet-name>
    <url-pattern>/consumer</url-pattern>
  </servlet-mapping>
  <service-ref>
    <service-ref-name>service/MyHelloServiceRef</service-ref-name>
    <service-interface>javax.xml.rpc.Service</service-interface>
    <wsdl-file>WEB-INF/wsdl/HelloService.wsdl</wsdl-file>
 <jaxrpc-mapping-file>WEB-INF/HelloService-java-wsdl-mapping.xml</jaxrpc-mapping-file>
    <service-qname
xmlns:service-qname_ns__="http://hello.demo.oracle/">service-qname_ns__:HelloService</s
ervice-qname>
        <port-component-ref>

<service-endpoint-interface>oracle.demo.hello.HelloInterface</service-endpoint-interfac
e>
        </port-component-ref>
  </service-ref>
</web-app>
```

例 14-14 は、Web アプリケーションおよびサーブレットのための、OC4J 固有の orion-web.xml デプロイメント・ディスクリプタの内容を示しています。<service-ref-mapping> 要素を太字で強調してあります。

例 14-14 クライアント・サイドのサーブレットまたは Web アプリケーションのための orion-web.xml の内容

```
<orion-web-app
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://xmlns.oracle.com/oracleas/schema/orion-web-10_0.x
sd">
        <service-ref-mapping name="service/MyHelloServiceRef">
                        <!-- stub property applicable across all ports -->
                        <stub-property>

<name>javax.xml.rpc.service.endpoint.address</name>

<value>http://localhost:8888/hello/HelloService</value>
                        </stub-property>
        </service-ref-mapping>
</orion-web-app>
```

EJB クライアントのパッケージ構造の概要

この項では、EJB クライアントのパッケージ化について説明します。デプロイメント・ディスクリプタ内のいくつかの要素の値には、EAR ファイル内でのファイル名および格納位置が反映されます。EAR ファイルの内容を変更した場合は、デプロイメント・ディスクリプタの内容の変更が必要になります。

- EJB アプリケーション・クライアント用のパッケージの構造
- EJB アプリケーション・クライアント用のデプロイメント・ディスクリプタ間の関係

EJB アプリケーション・クライアント用のパッケージの構造

EJB クライアントは、*<ear_file_name>*.ear という名前の EAR ファイルにパッケージ化されます。EAR ファイルでは、トップレベルにマニフェスト・ファイルおよび application.xml ファイルのための META-INF ディレクトリ、EJB クラス・ファイル、および *<ejb_jar_file_name>*.jar ファイルがあります。JAR ファイルには、JAR マニフェスト・ファイル、JAX-RPC マッピング・ファイル、WSDL ファイルおよびデプロイメント・ディスクリプタがあります。例 14-15 に、EJB クライアントの EAR ファイルのパッケージの構造を示します。

例 14-15 クライアント・サイド EJB アプリケーションの EAR ファイルのパッケージの構造

```
./META-INF
     ./MANIFEST.MF
     ./application.xml
./<ejb jar file name>.jar
     ./class files
     ./META-INF/
          /MANIFEST.MF
          /ejb-jar.xml
          /orion-ejb-jar.xml
          /wsdl/<wsdl file name>.wsdl
          /<mapping file>.xml
```

EJB アプリケーション・クライアント用のデプロイメント・ディスクリプタ間の関係

この項では、J2EE 標準デプロイメント・ディスクリプタ ejb-jar.xml、サーブレットまたは Web アプリケーションのための OC4J デプロイメント・ディスクリプタ orion-ejb-jar.xml、および EJB クライアント EAR ファイルのパッケージの構造の間の関係について説明します。クライアント EAR ファイルの構造または内容を編集すると、デプロイメント・ディスクリプタの内容も編集が必要になるため、これらの関係は重要です。

クライアント情報は、ejb-jar.xml の <service-ref> 要素に指定します。この要素には、Web サービス・クライアントとして使用できるサーブレットまたは Web アプリケーションの情報を指定します。たとえば、次のものの位置を指定します。WSDL（<wsdl-file>）、JAX-RPC マッピング・ファイル（<jaxrpc-mapping-file>）、JNDI ルックアップで使用するサービス・インタフェース（<service-ref-name>）、サービス・インタフェース・クラス（<service-interface>）およびサービス・エンドポイント・インタフェース（<service-endpoint-interface>）。ejb-jar.xml の <service-ref-name> は、orion-ejb-jar.xml の <service-ref-mapping> 要素の属性としても指定される点に注意してください。EAR の前述したアイテムの名前および位置を変更する場合は、デプロイメント・ディスクリプタも対応するように変更する必要があります。

関連項目：

<service-ref> 要素および <service-ref-mapping> 要素とそれぞれのサブ要素については、14-5 ページの「J2EE Web サービス・クライアント情報のデプロイメント・ディスクリプタへの追加方法」を参照してください。

例 14-16 は、EJB クライアント用の ejb-jar.xml の内容を示しています。<service-ref> 要素を太字で強調してあります。

例 14-16 クライアント・サイド EJB アプリケーション用の ejb-jar.xml の内容

```
<ejb-jar>
  <display-name>serviceConsumerEJB</display-name>
  <enterprise-beans>
    <session>
        <ejb-name>ServiceConsumer</ejb-name>
        <home>oracle.ServiceConsumerHome</home>
        <remote>oracle.ServiceConsumerRemote</remote>
        <ejb-class>oracle.ServiceConsumerBean</ejb-class>
        <session-type>Stateless</session-type>
        <transaction-type>Container</transaction-type>
        <service-ref>
                <service-ref-name>service/MyHelloService</service-ref-name>
                <service-interface>javax.xml.rpc.Service</service-interface>
                <wsdl-file>META-INF/wsdl/HelloService.wsdl</wsdl-file>

<jaxrpc-mapping-file>META-INF/HelloService-java-wsdl-mapping.xml</jaxrpc-mapping-file>

<service-qnamexmlns:ns="http://hello.demo.oracle/">ns:HelloService</service-qname>
                <port-component-ref>

<service-endpoint-interface>oracle.demo.hello.HelloInterface</service-endpoint-interfac
e>
                </port-component-ref>
        </service-ref>
    </session>
  </enterprise-beans>
</ejb-jar>
```

例 14-17 は、Web アプリケーションおよびサーブレットのための、OC4J 固有の orion-ejb-jar.xml デプロイメント・ディスクリプタの内容を示しています。<service-ref-mapping> 要素を太字で強調してあります。

例 14-17 クライアント・サイド EJB アプリケーション用の orion-ejb-jar.xml の内容

```
<orion-ejb-jar>
    <enterprise-beans>
        <session-deployment name="ServiceConsumer">
            <service-ref-mapping name="service/MyHelloService">
                <stub-property>
                    <name>javax.xml.rpc.service.endpoint.address</name>
                    <value>http://localhost:8888/hello/HelloService</value>
                </stub-property>
            </service-ref-mapping>
        </session-deployment>
    </enterprise-beans>
</orion-ejb-jar>
```

制限事項

F-10 ページの「J2EE Web サービス・クライアントのアセンブル」を参照してください。

追加情報

詳細は、次を参照してください。

- WSDL からの Web サービスのアセンブル方法は、第 6 章「WSDL からの Web サービスのアセンブル」を参照してください。
- ステートフルな Web サービスのアセンブル方法は、第 7 章「Java クラスを使用した Web サービスのアセンブル」を参照してください。
- EJB からの Web サービスのアセンブル方法は、第 8 章「EJB を使用した Web サービスのアセンブル」を参照してください。
- JMS キューまたは JMS トピックからの Web サービスのアセンブル方法は、第 9 章「JMS 宛先を使用した Web サービスのアセンブル」を参照してください。
- データベース・リソースからの Web サービスのアセンブル方法は、第 10 章「データベース Web サービスのアセンブル」を参照してください。
- J2SE 5.0 注釈を使用した Web サービスのアセンブル方法は、第 11 章「注釈を使用した Web サービスのアセンブル」を参照してください。
- J2SE クライアントの構築方法は、第 15 章「J2SE Web サービス・クライアントのアセンブル」を参照してください。
- WebServicesAssembler ツールを使用した Web サービスのアセンブル方法は、第 18 章「WebServicesAssembler の使用方法」を参照してください。
- Web サービスのパッケージ化およびデプロイ方法は、第 19 章「Web サービスのパッケージ化およびデプロイ」を参照してください。
- クライアントのアセンブルに必要な jar ファイルの詳細は、付録 A「Web サービス・クライアントの API および JAR」を参照してください。
- Web サービスの相互運用性の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「相互運用可能な Web サービスの実現」を参照してください。
- Web サービス・クライアントにおけるサービスのクオリティ機能の使用方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの管理」を参照してください。
- Web サービスへのセキュリティの追加方法は、『Oracle Application Server Web Services セキュリティ・ガイド』を参照してください。

- トランスポート・レベルで保護されている Web サービスにアクセスするクライアントの記述方法は、『Oracle Application Server Web Services セキュリティ・ガイド』の「EJB をベースとする Web サービスに対するトランスポート・レベル・セキュリティの追加」および「トランスポート・レベルで保護された Web サービスへのアクセス」を参照してください。
- Web サービスへの信頼性の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの信頼性の確保」を参照してください。
- Web サービスへの監査およびロギング構成の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「監査メッセージおよびロギング・メッセージ」を参照してください。
- 標準外のデータ型の処理方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 値タイプのカスタム・シリアライズ」を参照してください。
- JAX-RPC マッピング・ファイルの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「JAX-RPC マッピング・ファイル記述子」を参照してください。
- OracleAS Web Services でサポートされているデータ型の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 型の XML および WSDL 型へのマッピング」を参照してください。
- Web サービス開発用の Oracle JDeveloper ツールのサポートの詳細は、Oracle JDeveloper のオンライン・ヘルプを参照してください。

15

J2SE Web サービス・クライアントのアセンブル

この章では、J2SE プラットフォーム向けの Web サービス・クライアントの開発に関する情報を提供します。この章の内容は、次のとおりです。

- J2SE Web サービス・クライアントの概要
- J2SE Web サービス・クライアントのアセンブル方法
- Web サービス・クライアント・アプリケーションの記述
- Dynamic Invocation Interface を使用した Web サービスの起動方法
- J2SE Web サービス・クライアントのアセンブルに対するツールのサポート

J2SE Web サービス・クライアントの概要

J2SE クライアントは、J2EE クライアントとは異なり、サービスのルックアップ、ならびに Web サービスにアクセスするクラスのインスタンスの作成およびメンテナンスなどの、基礎となる作業を行います。開発者は、コンテナに依存できないため、独自のサービスの作成および管理、ならびに Web サービスのアクセスに必要なすべてのランタイム環境の可用性の確保を自身で行う必要があります。

次の各項では、静的スタブ・クライアントおよび Dynamic Invocation Interface（DII）クライアントについて説明します。

- 静的スタブ・クライアントの概要
- JAX-RPC Dynamic Invocation Interface の概要

静的スタブ・クライアントの概要

WebServicesAssembler コマンドの `genProxy` は、指定された WSDL ドキュメントから静的スタブを生成します。生成されるスタブは、`javax.xml.rpc.Stub` インタフェースおよびサービス・エンドポイント・インタフェースを実装したものです。また、生成されるスタブは、HTTP トランスポートおよび SOAP プロトコルに明示的にバインドされます。生成されるスタブをインスタンス化し、そのメソッドを直接起動することで、リクエストを関連する Web サービスに送信できます。

さらに、WebServicesAssembler は、クライアントが静的スタブによって Web サービスと対話できるようにするクライアント・ユーティリティ・クラスを生成します。このユーティリティ・クライアント・クラスの名前は *<WSDL_port_name>*`Client.java` です。このクラスは、スタブのインスタンスの作成に必要なすべてのステップを処理します。ユーティリティ・クライアントは、インスタンス化して、リモート・サービスの操作を起動するために使用できます。

注意： クライアント・ユーティリティ・クラス・ファイルは、WebServicesAssembler を実行するたびに再生成されます。独自のコードは別のファイルに格納しておくことを強くお薦めします。そうしないと、変更した内容が失われます。

JAX-RPC Dynamic Invocation Interface の概要

JAX-RPC Dynamic Invocation Interface を使用すると、サービス名またはリモート・メソッドのシグネチャを実行時の前に確認できない場合でも、リモート Web サービス操作を起動できます。

DII は、OC4J での `javax.xml.rpc.Call` インタフェースの実装によってサポートされています。`javax.xml.rpc.Service` クラスは、オーバーロードされた `Service.createCall()` メソッドを使用することで、`Call` インスタンスのファクトリとして動作します。`Call` インタフェースの作成後、このインタフェースに含まれるゲッターおよびセッターを使用して、ポート・タイプ、操作名、サービス・エンドポイント・アドレス、およびリモート・メソッドの実行に必要なその他の属性を構成します。

関連項目：

DII クライアントを使用して Web サービスを起動する例は、15-10 ページの「Dynamic Invocation Interface を使用した Web サービスの起動方法」を参照してください。

J2SE Web サービス・クライアントのアセンブル方法

この項の内容は、次のとおりです。

前提条件

開始する前に、次のファイルと情報を用意してください。

- クライアント生成に使用する WSDL への URI を指定します。この章では、15-4 ページの「サンプルの WSDL ファイル」に記載の WSDL ファイルを使用します。
- アーチファクトが生成される宛先位置を決定します。
- クライアント・ファイルのパッケージ名を決定します。

静的スタブを使用した J2SE Web サービス・クライアントのアセンブル手順

WebServicesAssembler で静的スタブを使用して、J2SE Web サービス・クライアントを作成できます。静的スタブを作成する手順は、次のとおりです。

1. WSDL の URI、出力ディレクトリ名、パッケージ名および「前提条件」の項に記載したその他の情報およびファイルを、WebServicesAssembler の `genProxy` コマンドに入力します。次のコマンドは、クライアント・プロキシを生成し、`build/src/client` に格納します。クライアント・アプリケーションはこのスタブを使用して、リモート・サービスでの操作を起動します。

   コマンドライン :

   ```
   java -jar wsa.jar -genProxy
                     -output build/src/client/
                     -wsdl http://localhost:8888/hello/HelloService?WSDL
                     -packageName oracle.demo.hello
   ```

   Ant タスク :

   ```
   <oracle:genProxy
              wsdl="http://localhost:8888/hello/HelloService?WSDL"
              output="build/src/client"
              packageName="oracle.demo.hello"/>
   ```

   このコマンドおよび Ant タスクの説明 :

   - `genProxy`: J2SE Web サービス・クライアントから使用可能な静的プロキシ・スタブを作成します。18-32 ページの「genProxy」を参照してください。
   - `wsdl`: WSDL ドキュメントの絶対ファイル・パス、相対ファイル・パスまたは URL を指定します。18-64 ページの「wsdl」を参照してください。
   - `output`: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
   - `packageName`: JAX-RPC マッピング・ファイルにパッケージ名が宣言されていない場合に、生成されたクラス用に使用されるパッケージ名を指定します。18-47 ページの「packageName」を参照してください。

   `genProxy` の必須およびオプションの引数の詳細は、18-32 ページの「genProxy」を参照してください。

2. genProxy によって作成されたクライアント・ユーティリティ・クラス・ファイルを、アプリケーション・クライアントとして使用するか、またはテンプレートとして使用して独自のクライアント・コードを記述します。クライアント・ユーティリティ・クラス・ファイルは、genProxy によって作成されるいくつかのファイルの 1 つです。

   また、クライアント・ユーティリティ・クラス・ファイルは、エンドポイントをテストするためにも使用できます。クライアント・ユーティリティ・クラス・ファイルの作成例は、15-7 ページの例 15-2「HelloInterfacePortClient.java のリスト」を参照してください。このファイルの詳細は、15-6 ページの「Web サービス・クライアント・アプリケーションの記述」を参照してください。

3. クライアント・ファイルをコンパイルし、クラスパスに置きます。

   クライアントをコンパイルする前に、クラスパス上にある適切な JAR をリストします。クライアントのクラスパスで使用可能なすべての JAR ファイルのリストについては、表 A-2「クライアント・サイド・プロキシを使用するクライアントのクラスパス・コンポーネント」を参照してください。個々の JAR をリストするかわりに、クライアントのクラスパスにクライアント側の JAR として wsclient_extended.jar を含めることができます。この JAR ファイルには、Web サービス・クライアントをコンパイルおよび実行するために必要なすべてのクラスが含まれます。これらのクラスは、表 A-2 にリストされている各 JAR ファイルに含まれているものです。wsclient_extended.jar およびクライアントのクラスパスの詳細は、A-3 ページの「Web サービス・プロキシのクラスパスの設定」を参照してください。

4. J2SE クライアントをコマンドラインから実行します。

サンプルの WSDL ファイル

例 15-1 は、クライアント生成に使用する HelloService.wsdl のリストの一部です。この WSDL から生成されるファイルの 1 つが、例 15-2 のクライアント・ユーティリティ・クラス・ファイルです。

リスト断片には、クライアント・ユーティリティ・クラス・ファイルの生成に使用される、WSDL ファイル内のいくつかのエントリが記載されています。次に例を示します。

- <port name> 属性の値。これを使用して、クライアント・ユーティリティ・クラス・ファイルの名前 HelloInterfacePortClient.java が導出されます。
- <portType> 要素の下にある操作名 sayHello。これは、クライアント・ユーティリティ・クラス・ファイルのメソッドになります。
- sayHello リクエストを定義する complexType。これによって、sayHello のパラメータおよびデータ型が定義されます。

```
<complexType name="sayHello">
    <sequence>
          <element name="name" nillable="true" type="string"/>
    </sequence>
</complexType>
```

これらの各要素は、HelloService.wsdl のリスト断片では太字で示されています。

例 15-1 クライアント・ユーティリティ・クラス・ファイルで使用される要素を含む WSDL フラグメント

```
<definitions
    ...
  >
    <types>
        <schema targetNamespace="http://hello.demo.oracle/" ...
xmlns="http://www.w3.org/2001/XMLSchema"
           ...
            <complexType name="sayHello">
                <sequence>
                    <element name="name" nillable="true" type="string"/>
                </sequence>
            </complexType>
            <complexType name="sayHelloResponse">
                <sequence>
                    <element name="result" nillable="true" type="string"/>
                </sequence>
            </complexType>
            <element name="sayHelloElement" type="tns:sayHello"/>
            <element name="sayHelloResponseElement" type="tns:sayHelloResponse"/>
        </schema>
    </types>
    <message name="HelloInterface_sayHelloResponse">
        <part name="parameters" element="tns:sayHelloResponseElement"/>
    </message>
    <message name="HelloInterface_sayHello">
        <part name="parameters" element="tns:sayHelloElement"/>
    </message>
    <portType name="HelloInterface">
        <operation name="sayHello">
            <input message="tns:HelloInterface_sayHello"/>
            <output message="tns:HelloInterface_sayHelloResponse"/>
        </operation>
    </portType>
    <binding name="HelloInterfacePortBinding" type="tns:HelloInterface">
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="sayHello">
            <soap:operation soapAction="http://hello.demo.oracle/:sayHello"/>
            <input>
                <soap:body use="literal" parts="parameters"/>
            </input>
            <output>
                <soap:body use="literal" parts="parameters"/>
            </output>
        </operation>
    </binding>
    <service name="HelloService">
        <port name="HelloInterfacePort" binding="tns:HelloInterfacePortBinding">
            <soap:address location="HelloService"/>
        </port>
    </service>
</definitions>
```

Web サービス・クライアント・アプリケーションの記述

この項の内容は、次のとおりです。

クライアント・ユーティリティ・クラス・ファイルの概要

genProxy コマンドにより、Web サービス・メソッドの起動に使用できるクライアント・ユーティリティ・クラス・ファイルが生成されます。このファイルを、アプリケーション・クライアントとして使用するか、またはテンプレートとして使用して独自のアプリケーション・クライアント・コードを記述できます。

注意： クライアント・ユーティリティ・クラス・ファイルは、genProxy を実行するたびに再生成されます。テストするためにこのファイルにコードを追加した場合、プロキシを再生成すると、その変更が失われます。本番用コードとして使用するクライアント・コードは、このユーティリティ・クラスの外部に保管する必要があります。

---

genProxy コマンドは、クライアント・ユーティリティ・クラス・ファイル名を導出する際、ポート名に接尾辞 Client を追加します。たとえば、ポート名が HelloInterfacePort の場合は、genProxy により、ファイル HelloInterfacePortClient.java が生成されます。

関連項目：

genProxy コマンドによって生成されたクライアント・ユーティリティ・クラス・ファイルの例は、15-7 ページの例 15-2「HelloInterfacePortClient.java のリスト」を参照してください。

クライアント・ユーティリティ・クラスは、Web サービス実装のプロキシとしての役割を持ちます。クライアント・サイド・プロキシ・コードにより、SOAP リクエストの構成、パラメータのマーシャリングおよびアンマーシャリングが実行されます。プロキシ・クラスを使用すると、Web サービスへのアクセスや Web サービスのレスポンス処理に関する、SOAP リクエストの作成作業およびデータのマーシャリング作業が軽減されます。

クライアント・ユーティリティ・クラス・ファイルの最も重要な部分は、javax.xml.rpc.Service オブジェクトを作成し、サービスで使用可能な操作を取得するファクトリ・コードです。Service オブジェクトは、生成されるスタブ・クラスのインスタンスとして動作します。

クライアント・ユーティリティ・クラス・ファイルの次のコード行に注目してください。

```
public HelloInterfaceClient() throws Exception {
    ServiceFactory factory = ServiceFactory.newInstance();
    _port = ((HelloService)factory.loadService
                (HelloService.class)).getHelloInterfacePort();
}
```

独自のアプリケーション・クライアントを記述する場合は、このコードを追加して、サービス・ファクトリを作成し、ポートの実装を取得する必要があります。

サービス・ファクトリの作成手順

次の手順は、このコードの説明です。

1. `javax.xml.rpc.ServiceFactory` の新しいインスタンスを作成するか、または既存のインスタンスを使用します。

   ```
   ServiceFactory factory = ServiceFactory.newInstance();
   ```

2. `loadService` メソッドを使用して、特定のサービス・エンドポイント・インタフェースに対してサービスをロードします。これにより、`Service` 型のオブジェクトが戻されます。さらに、このオブジェクトにより、リクエストしたサービス・エンドポイント・インタフェースが実装されます。

   ```
   (HelloService)factory.loadService(HelloService.class)
   ```

   この例では、戻された `Service` が、サービス・エンドポイント・インタフェース `HelloService` にキャストされています。

3. `get...()` メソッドを使用して、必要なポートを取得します。省略記号（"..."）は、WSDL の `port name` 要素の値を表します。

   ```
   HelloService.getHelloInterfacePort();
   ```

   この例では、メソッド名は `getHelloInterfacePort()` です。この名前の中の `HelloInterfacePort` は、WSDL 内の `port name` の値です。メソッドにより、`HelloInterfacePort` の Java 実装が戻されます。

例 15-2 は、クライアント・ユーティリティ・クラス・ファイル `HelloInterfacePortClient.java` を示します。これは、例 15-1 の `HelloService.wsdl` から `genProxy` コマンドで生成されたものです。

`Service` オブジェクトを作成し、サービスで使用可能な操作を取得するコード行は、太字で示されています。生成されたコードを使用して、`SESSION_MAINTAIN_PROPERTY` システム・プロパティを設定し、Web サービスがステートフルであることをクライアントに警告できることに注意してください。クライアントがサーバー・サイドのステートフル Web サービスと連携して動作するとき、セッションは維持されます。セッションを維持することで、サービスに対する以降のリクエストが高速になります。

例 15-2 HelloInterfacePortClient.java のリスト

```
import oracle.webservices.transport.ClientTransport;
import oracle.webservices.OracleStub;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.Stub;

public class HelloInterfacePortClient {
    private HelloInterface _port;

    public HelloInterfacePortClient() throws Exception {
        ServiceFactory factory = ServiceFactory.newInstance();
        _port =((HelloService)factory.loadService(
           HelloService.class)).getHelloInterfacePort();
    }

    /**
     * @param args
     */
    public static void main(String[] args) {
        try {
            HelloInterfacePortClient myPort = new HelloInterfacePortClient();
            System.out.println("calling " + myPort.getEndpoint());
            // Add your own code here

        } catch (Exception ex) {
            ex.printStackTrace();
        }
```

```
    }

    /**
     * delegate all operations to the underlying implementation class.
     */
    // sayHello
    public String sayHello(String name) throws java.rmi.RemoteException {
        return _port.sayHello(name);
    }

    /**
     * used to access the JAX-RPC level APIs
     * returns the interface of the port instance
     */
    public oracle.demo.hello.HelloInterface getPort() {
        return _port;
    }

    public String getEndpoint() {
        return (String) ((Stub)
_port)._getProperty(Stub.ENDPOINT_ADDRESS_PROPERTY);
    }

    public void setEndpoint(String endpoint) {
        ((Stub) _port)._setProperty(Stub.ENDPOINT_ADDRESS_PROPERTY,
endpoint);
    }

    public String getPassword() {
        return (String) ((Stub) _port)._getProperty(Stub.PASSWORD_PROPERTY);
    }

    public void setPassword(String password) {
        ((Stub) _port)._setProperty(Stub.PASSWORD_PROPERTY, password);
    }

    public String getUsername() {
        return (String) ((Stub) _port)._getProperty(Stub.USERNAME_PROPERTY);
    }

    public void setUsername(String username) {
        ((Stub) _port)._setProperty(Stub.USERNAME_PROPERTY, username);
    }

    public void setMaintainSession(boolean maintainSession) {
        ((Stub) _port)._setProperty(Stub.SESSION_MAINTAIN_PROPERTY, new
Boolean(maintainSession));
    }

    public boolean getMaintainSession() {
        return ((Boolean) ((Stub)
_port)._getProperty(Stub.SESSION_MAINTAIN_PROPERTY)).booleanValue();
    }

   /**
     * returns the transport context
     */
    public ClientTransport getClientTransport() {
        return ((OracleStub) _port).getClientTransport();
    }
}
```

HTTP 1.1 向けのチャンク・データ転送の有効化方法

プロトコルが HTTP 1.1 の場合、OracleAS Web Services では、メッセージのチャンク転送エンコーディングが可能です。チャンク・データ転送は、J2SE スタブ、J2EE スタブおよび DII Web サービス・クライアントから起動できます。

関連項目：

この機能を有効化する方法の詳細は、14-17 ページの「HTTP 1.1 向けのチャンク・データ転送の有効化方法」を参照してください。

J2SE クライアント上での SOAP メッセージの文字コード設定方法

デフォルトでは、Oracle Application Server Web Services でアセンブルされたクライアントは、UTF-8 の文字でエンコードされた SOAP エンベロープを持つリクエスト・メッセージを送信します。この動作をオーバーライドする場合は、次の Oracle 固有のプロパティを設定します。

`oracle.webservices.ClientConstants.CHARACTER_SET_ENCODING`

このプロパティは、J2SE クライアントまたは J2EE クライアントの文字コードを設定するために使用します。

関連項目：

`CHARACTER_SET_ENCODING` プロパティの使用方法の詳細は、14-18 ページの「SOAP メッセージの文字コードの設定方法」を参照してください。

Web サービス・クライアント・スタブでの Cookie の設定方法

クライアント・スタブは、HTTP リクエストで使用される Cookie の設定に使用できます。`Cookie` クラスは HTTP Cookie を表し、Oracle Applications Server 10*g* の `HTTPClient` パッケージにあります。`Cookie` に次のコンストラクタがあります。

`Cookie (java.lang.String` *`name`*`, java.lang.String` *`value`*`, java.lang.String` *`domain`*`, java.lang.String` *`path`*`, java.util.Date` *`expires`*`, boolean` *`secure`*`)`

`expires` 以外のすべてのパラメータが、`Cookie` コンストラクタの必須パラメータです。

Cookie に対して指定するドメインは、次のいずれかである必要があります。

- 完全なホスト名
- Cookie を複数のホストで共有する場合は、ドメインをピリオドで始める必要があります。次に例を示します。

  `.oracle.com`

次に説明するのは、HTTP リクエストで Cookie を設定する方法の一般的な手順です。

1. `java.util.Map` オブジェクトを作成して、Cookie を組み込みます。
2. Oracle Applications Server 10*g* `HTTPClient` パッケージに付属する `Cookie` クラスを使用して、1 つ以上の Cookie を作成します。
3. Cookie を `Map` オブジェクトにロードします。
4. プロパティ `oracle.webservices.ClientConstants.COOKIE_MAP` を設定します。

   プロパティの値は、`HTTPClient.Cookie` 型のアイテムおよびキーが含まれる `java.util.Map` オブジェクトです。
5. `javax.xml.rpc.session.SESSION_MAINTAIN_PROPERTY` ランタイム・プロパティ・セットを `true` に設定します。

   このプロパティは、Web サービスがステートフルであることをクライアントに警告しています。このプロパティを `true` 以外に設定すると、Cookie が無視されます。

例 15-3 は、2 つの Cookie の値を設定するスタブ・コードを示しています。cookieMap 変数は、java.util.Map 型として宣言され、その値を HashMap から取得します。Cookie コンストラクタは、cookie および cookie2 の 2 つの Cookie の定義に使用されます。各 cookieMap.put 行で、Cookie が HashMap に追加されます。ClientConstants.COOKIE_MAP が cookieMap に設定されており、また、Stub.SESSION_MAINTAIN_PROPERTY が指定されて true に設定されています。

例 15-3 クライアント・スタブの Cookie 設定

```
import HTTPClient.Cookie;
   Map cookieMap = new HashMap();

   Cookie cookie = new Cookie("name", "value", ".oracle.com", "/",  null, false);
   Cookie cookie2 = new Cookie("name2", "value2", ".oracle.com", "/", null, false);
   cookieMap.put(cookie, cookie);
   cookieMap.put(cookie2, cookie2);

   ((Stub) port)._setProperty(ClientConstants.COOKIE_MAP, cookieMap);
   ((Stub) port)._setProperty(Stub.SESSION_MAINTAIN_PROPERTY, Boolean.TRUE);
...
```

Dynamic Invocation Interface を使用した Web サービスの起動方法

Dynamic Invocation Interface（DII）を使用して Web サービスを起動するには、いくつかのステップが必要です。各ステップでは、通常いくつかの選択が必要です。この項の終わりにある各例に、各ステップで行われた選択が示されます。

DII を使用して Web サービスを起動する方法は、次のような全般的なステップで構成されます。

1. コール・オブジェクトを作成します。
2. パラメータを登録します。
3. Web サービスを起動します。

コール・オブジェクトは、WSDL の有無を問わず作成できます。WSDL がないか、コールの動的作成に WSDL を使用しない場合は、「基本コール」の手順に従います。コールを構成するための WSDL がある場合は、「構成コール」の手順に従います。

基本コール

基本コールでは、WSDL がなくても、コール・オブジェクトが動的に作成されます。基本コールの構成方法の詳細は、次の手順を参照してください。

1. WSDL なしでコール・オブジェクトを動的に作成します。たとえば、次の例を参照してください。
   - 例 15-4「パラメータ登録と Java バインディングのある基本コール」
   - 例 15-7「SOAPElement あり、パラメータ登録なしの基本コール」
   - 例 15-9「document-literal 起動および SOAPElement あり、パラメータ登録なしの基本コール」
2. パラメータを登録します。
   - ケース 1: SOAP リクエストを自身で 1 つの SOAPElement として構成し、レスポンスを別の SOAPElement として受信しようとするケース。このようなケースでは、パラメータ型や戻り型の登録は不要です。このケースの例は、次を参照してください。
     - 例 15-7「SOAPElement あり、パラメータ登録なしの基本コール」
     - 例 15-9「document-literal 起動および SOAPElement あり、パラメータ登録なしの基本コール」

- ケース 2: Web サービス起動で使用されるパラメータと戻り値（部品）を基本コールで明示的に登録しようとするケース。この部品には、部品名、その XML 型名および Java 型名ならびにそのパラメータ・モードがあります。このようなケースでは、個々のパラメータを Java オブジェクト・インスタンスとして指定できます。このケースの例は、次を参照してください。
  - 例 15-4「パラメータ登録と Java バインディングのある基本コール」

3. Web サービスを起動します。Web サービスの起動は、`SOAPElement` を使用しても Java バインディングを使用しても行えます。
   - ケース 1: `SOAPElement` を使用するケース。document-literal の起動を使用する場合は、通常、メッセージ用に `SOAPElement` を構成し、`invoke()` メソッドに渡します。この起動では、`OracleCall` からパブリックの Oracle 固有 API が使用されます。このケースの例は、次を参照してください。
     - 例 15-7「SOAPElement あり、パラメータ登録なしの基本コール」
     - 例 15-9「document-literal 起動および SOAPElement あり、パラメータ登録なしの基本コール」
   - ケース 2: Java バインディングを使用するケース。rpc-literal または rpc-encoded の起動を使用する場合は、通常、Java オブジェクトが含まれる配列を `invoke()` メソッドに指定し、予期する戻り型に戻りオブジェクトをキャストします。このケースの例は、次を参照してください。
     - 例 15-4「パラメータ登録と Java バインディングのある基本コール」
     - 例 15-5「Java バインディングあり、パラメータ登録なしの構成コール」
     - 例 15-6「ラッパー・パラメータ登録と Java バインディングのある構成コール」
     - 例 15-8「WSDL、複合型戻り値 / パラメータ登録および Java バインディングのある構成コール」
     - 例 15-10「rpc-encoded 起動、複合型パラメータ登録および Java バインディングのある構成コール」

構成コール

構成コールの場合、コール・オブジェクトは WSDL から構成されます。構成コールの構成方法の詳細は、次の手順を参照してください。

1. コール・オブジェクトの構成用に WSDL を用意します。たとえば、次の例を参照してください。
   - 例 15-5「Java バインディングあり、パラメータ登録なしの構成コール」
   - 例 15-6「ラッパー・パラメータ登録と Java バインディングのある構成コール」
   - 例 15-8「WSDL、複合型戻り値 / パラメータ登録および Java バインディングのある構成コール」
   - 例 15-10「rpc-encoded 起動、複合型パラメータ登録および Java バインディングのある構成コール」
2. パラメータを登録します。構成コールでパラメータの登録が必要になるのは、次のようなケースです。
   - ケース 1: プリミティブ Java 型（またはプリミティブ Java 型のオブジェクト変数）にマップされない複合型またはその他の型を使用しようとするケース。このケースの例は、次を参照してください。
     - 例 15-8「WSDL、複合型戻り値 / パラメータ登録および Java バインディングのある構成コール」
     - 例 15-10「rpc-encoded 起動、複合型パラメータ登録および Java バインディングのある構成コール」

- ケース 2: document-literal ラップのスタイルを使用し、基本コールの例 15-7 および例 15-9 に示されているような SOAPElement は作成しないケース。このようなケースでは、パラメータ名をラッパーの QName にする必要があります。このケースの例は、次を参照してください。
  - 例 15-6「ラッパー・パラメータ登録と Java バインディングのある構成コール」
- ケース 3: ケース 1 およびケース 2 が該当しない場合は、パラメータまたは戻り値を登録する必要はありません。このケースの例は、次を参照してください。
  - 例 15-5「Java バインディングあり、パラメータ登録なしの構成コール」

3. Web サービスを起動します。Web サービスの起動は、SOAPElement を使用しても Java バインディングを使用しても行えます。
   - ケース 1: SOAPElement を使用するケース。document-literal の起動を使用する場合は、通常、メッセージ用に SOAPElement を構成し、invoke() メソッドに渡します。この起動では、OracleCall からパブリックの Oracle 固有 API が使用されます。このケースの例は、次を参照してください。
     - 例 15-7「SOAPElement あり、パラメータ登録なしの基本コール」
     - 例 15-9「document-literal 起動および SOAPElement あり、パラメータ登録なしの基本コール」
   - ケース 2: Java バインディングを使用するケース。rpc-literal または rpc-encoded の起動を使用する場合は、通常、Java オブジェクトが含まれる配列を invoke() メソッドに指定し、予期する戻り型に戻りオブジェクトをキャストします。このケースの例は、次を参照してください。
     - 例 15-4「パラメータ登録と Java バインディングのある基本コール」
     - 例 15-5「Java バインディングあり、パラメータ登録なしの構成コール」
     - 例 15-6「ラッパー・パラメータ登録と Java バインディングのある構成コール」
     - 例 15-8「WSDL、複合型戻り値 / パラメータ登録および Java バインディングのある構成コール」
     - 例 15-10「rpc-encoded 起動、複合型パラメータ登録および Java バインディングのある構成コール」

DII を使用する Web サービス・クライアントの例

この項では、基本コールまたは構成コールを使用して Web サービスを起動する様々なクライアントの例を示します。

次のコード Snippet は、後続のコード例で使用される import 文です。

```
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.Service;
import javax.xml.rpc.Call;
import javax.xml.rpc.ParameterMode;
import javax.xml.namespace.QName;
import javax.xml.soap.SOAPFactory;
import javax.xml.soap.SOAPElement;
import java.net.URL;
import oracle.webservices.OracleCall;
import oracle.xml.parser.v2.XMLElement;
```

例 15-4 パラメータ登録と Java バインディングのある基本コール

```
// (1) Creation of call object without WSDL.
String endpoint = "http://localhost:8888/echo/DiiDocEchoService";
ServiceFactory sf = ServiceFactory.newInstance();
Service service = sf.createService(new QName("http://echo.demo.oracle/", "tns"));
Call call = service.createCall();

// (2) Configuration of call and registration of parameters.
call.setTargetEndpointAddress(endpoint);
call.setProperty(Call.SOAPACTION_USE_PROPERTY, new Boolean(true));
call.setProperty(Call.SOAPACTION_URI_PROPERTY, "");
call.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY,
"http://schemas.xmlsoap.org/soap/encoding/");
call.setProperty(Call.OPERATION_STYLE_PROPERTY, "rpc");
QName QNAME_TYPE_STRING = new QName("http://www.w3.org/2001/XMLSchema", "string");
call.addParameter("s", QNAME_TYPE_STRING, ParameterMode.IN);
call.setReturnType(QNAME_TYPE_STRING);

// (3) Invocation.
System.out.println("Response is " + call.invoke(new Object[]{"hello"}));
```

例 15-5 Java バインディングあり、パラメータ登録なしの構成コール

```
/// (1) Creation of call object using WSDL.
String namespace = "http://www.xmethods.net/sd/CurrencyExchangeService.wsdl";
URL wsdl=new URL(namespace);
ServiceFactory factory = ServiceFactory.newInstance();
QName serviceName = new QName(namespace, "CurrencyExchangeService");
Service service = factory.createService(wsdl, serviceName);
QName portName = new QName(namespace, "CurrencyExchangePort");
Call call = service.createCall(portName);

// (2) Registration of parameters.
//     -> taken from the WSDL

// (3) Configuration of operation and invocation.
QName operationName = new QName("urn:xmethods-CurrencyExchange", "getRate");
call.setOperationName(operationName);
Float rate = (Float) call.invoke(new Object[]{"usa", "canada"});
System.out.println("getRate: " + rate);
```

例 15-6 ラッパー・パラメータ登録と Java バインディングのある構成コール

```
// (1) Creation of call object using WSDL.
String namespace = "http://server.hello/jaxws";
ServiceFactory factory = ServiceFactory.newInstance();
QName serviceName = new QName(namespace, "HelloImplService");
URL wsdl=new URL(namespace+"?WSDL");
Service service = factory.createService(wsdl, serviceName);
QName portName = new QName(namespace, "HelloImpl");
Call call = service.createCall(portName);

// (2) Registration of SayHello and SayHelloResponse wrapper classes
//     These must be available in the classpath.
String TYPE_NAMESPACE_VALUE = "http://server.hello/jaxws";
QName reqQname = new QName(TYPE_NAMESPACE_VALUE,"sayHelloElement");
QName respQName = new QName(TYPE_NAMESPACE_VALUE,"sayHelloResponseElement");
call.addParameter("name", reqQname, SayHello.class, ParameterMode.IN );
call.setReturnType(respQName, SayHelloResponse.class );
```

```
// (3) Invocation
SayHello input = new SayHello("Duke");
Object[] params = new Object[] { input };
SayHelloResponse result = (SayHelloResponse) call.invoke( params );
String response = result.getResult();
```

例 15-7 SOAPElement あり、パラメータ登録なしの基本コール

```
// (1) Creation of call object without WSDL
ServiceFactory sf = ServiceFactory.newInstance();
Service service = sf.createService(new QName("http://echo.demo.oracle/", "tns"));
Call call = service.createCall();
call.setTargetEndpointAddress("http://localhost:8888/echo/DiiDocEchoService");

// (2) No registration of parameters

// (3a) Direct creation of payload as SOAPElement
SOAPFactory soapfactory = SOAPFactory.newInstance();
SOAPElement m1 = soapfactory.createElement("echoStringElement", "tns",
"http://echo.demo.oracle/");
SOAPElement m2 = soapfactory.createElement("s", "tns", "http://echo.demo.oracle/");
m2.addTextNode("Bob");
m1.addChildElement(m2);
System.out.println("Request is: ");
((XMLElement) m1).print(System.out);

// (3b) Invocation
SOAPElement resp = (SOAPElement) ((OracleCall) call).invoke(m1);
System.out.println("Response is: ");
((XMLElement) resp).print(System.out);
```

例 15-8 WSDL、複合型戻り値 / パラメータ登録および Java バインディングのある構成コール

```
// (0) Preparing a complex argument value
Integer  req_I = new Integer( Integer.MAX_VALUE );
String   req_s = "testDocLitBindingAnonymAll & <body>";
Integer  req_inner_I = new Integer( Integer.MIN_VALUE );
String   req_inner_s = "<inner> & <body>";
int[]    req_inner_i = {0,Integer.MAX_VALUE, Integer.MIN_VALUE};
InnerSequence req_inner = new InnerSequence();
req_inner.setVarInteger( req_inner_I );
req_inner.setVarString ( req_inner_s );
req_inner.setVarInt    ( req_inner_i );
EchoAnonymAllElement req = new EchoAnonymAllElement();
req.setVarInteger   ( req_I );
req.setVarString    ( req_s );
req.setInnerSequence( req_inner);

// (1) Creation of call object using the WSDL
String TARGET_NS = "http://soapinterop.org/DocLitBinding";
String TYPE_NS = "http://soapinterop.org/xsd";
String XSD_NS = "http://www.w3.org/2001/XMLSchema";
QName SERVICE_NAME = new QName( TARGET_NS, "DocLitBindingService" );
QName PORT_NAME = new QName( TARGET_NS, "DocLitBindingPort");
String wsdlUrl = "http://"+getProperty("HOST")+
                 ":"+getProperty("HTTP_PORT")+
                 "/doclit_binding/doclit_binding";

QName operation = new QName(TARGET_NS, "echoAnonymAll");
ServiceFactory factory = ServiceFactory.newInstance();
Service srv = factory.createService( new URL(wsdlUrl + "?WSDL"), SERVICE_NAME);
Call call = srv.createCall( PORT_NAME, operation );
```

```
// (2) Registration of complex return parameter
call.setReturnType(new QName(TYPE_NS, "EchoAnonymAllElement"),
EchoAnonymAllElement.class);

// (3) Invocation
EchoAnonymAllElement res = (EchoAnonymAllElement) call.invoke( new Object[]{req} );
System.out.println( "AnonymAll body  : " +res.getVarString() );
System.out.println( "AnonymAll inner : " +res.getInnerSequence() );
```

例 15-9 document-literal 起動および SOAPElement あり、パラメータ登録なしの基本コール

```
// (1) Creation of Basic Call
ServiceFactory sf = ServiceFactory.newInstance();
Service service = sf.createService(new QName("http://echo.demo.oracle/", "tns"));
String endpoint="http://localhost:8888/test/echo";
Call call = service.createCall();
call.setTargetEndpointAddress(endpoint);

// (2) No parameter registration

// (3) Invocation using SOAPElement
SOAPFactory soapfactory = SOAPFactory.newInstance();
SOAPElement m1 = soapfactory.createElement("echoStringElement", "tns",
"http://echo.demo.oracle/");
SOAPElement m2 = soapfactory.createElement("s", "tns", "http://echo.demo.oracle/");
m2.addTextNode("Bob");
m1.addChildElement(m2);
System.out.println("Request is: ");
((XMLElement) m1).print(System.out);
SOAPElement resp = (SOAPElement) ((OracleCall) call).invoke(m1);
System.out.println("Response is: ");
((XMLElement) resp).print(System.out);
```

例 15-10 rpc-encoded 起動、複合型パラメータ登録および Java バインディングのある構成コール

```
// (1) Creation of ConfiguredCall using WSDL
ServiceFactory sf = ServiceFactory.newInstance();
String endpoint="http://localhost:8888/test/echo";
Service service = sf.createService(new java.net.URL(endpoint + "?WSDL"), new
QName("http://echo.demo.oracle/", "DiiRpcEchoService"));
Call call = service.createCall(new QName("http://echo.demo.oracle/", "HttpSoap11"), new
QName("http://echo.demo.oracle/", "echoStrings"));
call.setProperty(Call.SOAPACTION_USE_PROPERTY, new Boolean(true));
call.setProperty(Call.SOAPACTION_URI_PROPERTY, "");
call.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY,
"http://schemas.xmlsoap.org/soap/encoding/");
call.setProperty(Call.OPERATION_STYLE_PROPERTY, "rpc");

// (2) Registration of complex input and return arguments
QName stringArray = new QName("http://echo.demo.oracle/", "stringArray");
call.addParameter("s", stringArray, String[].class, ParameterMode.IN);
call.setReturnType(stringArray, String[].class);

// (3) Invocation
String[] request = new String[]{"bugs", "little_pieces", "candy"};
String[] resp = (String[]) call.invoke(new Object[]{request});
System.out.println("Response is: ");
for (int i = 0; i < resp.length; i++) {
   System.out.print(resp[i] + " ");
}
System.out.println();
```

J2SE Web サービス・クライアントのアセンブルに対するツールのサポート

Oracle JDeveloper で、Web サービスを使用するクライアント・アプリケーションを作成できます。Oracle JDeveloper では、OC4J J2SE Web サービス・クライアントがサポートされます。これは、Web サービス WSDL 記述からの Java スタブの作成を可能にするという形で行われます。生成したスタブを使用して、既存の Web サービスにアクセスできます。詳細は、Oracle JDeveloper のオンライン・ヘルプを参照してください。

追加情報

詳細は、次を参照してください。

- WSDL からの Web サービスのアセンブル方法は、第 6 章「WSDL からの Web サービスのアセンブル」を参照してください。
- ステートフルな Web サービスのアセンブル方法は、第 7 章「Java クラスを使用した Web サービスのアセンブル」を参照してください。
- EJB からの Web サービスのアセンブル方法は、第 8 章「EJB を使用した Web サービスのアセンブル」を参照してください。
- JMS キューまたは JMS トピックからの Web サービスのアセンブル方法は、第 9 章「JMS 宛先を使用した Web サービスのアセンブル」を参照してください。
- データベース・リソースからの Web サービスのアセンブル方法は、第 10 章「データベース Web サービスのアセンブル」を参照してください。
- J2SE 5.0 注釈を使用した Web サービスのアセンブル方法は、第 11 章「注釈を使用した Web サービスのアセンブル」を参照してください。
- J2EE クライアントの構築方法は、第 14 章「J2EE Web サービス・クライアントのアセンブル」を参照してください。
- データ・チャンクによるパフォーマンスの向上の詳細は、14-17 ページの「HTTP 1.1 向けのチャンク・データ転送の有効化方法」を参照してください。
- WebServicesAssembler ツールを使用した Web サービスのアセンブル方法は、第 18 章「WebServicesAssembler の使用方法」を参照してください。
- Web サービスのパッケージ化およびデプロイ方法は、第 19 章「Web サービスのパッケージ化およびデプロイ」を参照してください。
- クライアントのアセンブルに必要な JAR ファイルの詳細は、付録 A「Web サービス・クライアントの API および JAR」を参照してください。
- Web サービスの相互運用性の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「相互運用可能な Web サービスの実現」を参照してください。
- Web サービス・クライアントにおけるサービスのクオリティ機能の使用方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの管理」を参照してください。
- Web サービスへのセキュリティの追加方法は、『Oracle Application Server Web Services セキュリティ・ガイド』を参照してください。
- トランスポート・レベルで保護されている Web サービスにアクセスするクライアントの記述方法は、『Oracle Application Server Web Services セキュリティ・ガイド』の「EJB をベースとする Web サービスに対するトランスポート・レベル・セキュリティの追加」および「トランスポート・レベルで保護された Web サービスへのアクセス」を参照してください。
- Web サービスへの信頼性の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの信頼性の確保」を参照してください。

- Web サービスへの監査およびロギング構成の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「監査メッセージおよびロギング・メッセージ」を参照してください。
- 標準外のデータ型の処理方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 値タイプのカスタム・シリアライズ」を参照してください。
- JAX-RPC マッピング・ファイルの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「JAX-RPC マッピング・ファイル記述子」を参照してください。
- OracleAS Web Services でサポートされているデータ型の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 型の XML および WSDL 型へのマッピング」を参照してください。
- Web サービス開発用の Oracle JDeveloper ツールのサポートの詳細は、Oracle JDeveloper のオンライン・ヘルプを参照してください。

16

JAX-RPC ハンドラの使用方法

この章では、JAX-RPC メッセージ・ハンドラの操作の概要について説明します。

- メッセージ・ハンドラの概要
- JAX-RPC ハンドラの記述方法
- Ant タスクでサーバー・サイド・ハンドラを構成および登録する方法
- サーバー・サイド・ハンドラを登録するための webservices.xml の編集方法
- クライアント・サイドの JAX-RPC ハンドラ

メッセージ・ハンドラの概要

SOAP メッセージ・ハンドラは、Web サービスとやりとりするメッセージを処理するために使用できます。ハンドラには、クライアント・ハンドラとサーバー・ハンドラの 2 種類があります。

- クライアント・サイド・ハンドラは、クライアント・アプリケーションから送信されるメッセージ（リクエスト）と、それに応答してサービスからクライアントに戻されるメッセージ（レスポンス）を捕捉できます。
- サーバー・サイド・ハンドラは、Web サービスが受信するメッセージ（リクエスト）と、それに応答してサービスから戻されるメッセージ（レスポンス）を捕捉できます。

これらのハンドラには SOAP エンベロープ全体へのアクセス権があるため、これらは一般的に SOAP ヘッダーの処理に使用されます。ハンドラのその他の一般的な用途は、次のとおりです。

- ロギング
- 監査
- 暗号化 / 復号化

これらの機能のほとんどは OracleAS Web Services の管理インフラストラクチャによって用意されます。ユーザー記述のハンドラを用意する必要が生じる場合は多くありません。

Web サービスや Web サービス・クライアントに応じて、ハンドラの数はゼロにもなれば 1 つ以上にもなります。複数のハンドラがまとまると、ハンドラ・チェーンを形成します。ハンドラ・チェーンは、JAX-RPC ランタイム実装によって管理されます。このランタイム実装のデフォルトの動作は、チェーン内の各ハンドラを順番にコールすることです。ただし、この処理モデルは `javax.xml.rpc.handler.Handler` インタフェースの実装に基づいて変更されます。たとえば、`handleRequest` メッセージで `false` が戻されると、ランタイムがチェーン内の次のハンドラの処理に進まずに停止します。例外がスローされた場合も同じ結果になります。

関連資料：

JAX-RPC 1.1 仕様では、ハンドラとハンドラ・モデルについてのさらに詳しい情報が提供されています。

`http://java.sun.com/webservices/jaxrpc/index.jsp`

JAX-RPC ハンドラの記述方法

JAX-RPC ハンドラを記述するには、`javax.xml.rpc.handler.Handler` インタフェースを実装します。

```
package javax.xml.rpc.handler;
public interface Handler{
    public boolean handleRequest(javax.xml.rpc.handler.MessageContext context);
    public boolean handleResponse(javax.xml.rpc.handler.MessageContext context);
    public boolean handleFault(javax.xml.rpc.handler.MessageContext context);
    public void destroy();
    public void init(javax.xml.rpc.handler.HandlerInfo config);
    public javax.xml.namespace.QName[] getHeaders();
}
```

`Handler` インタフェースを実装するかわりに、`javax.xml.rpc.handler.GenericHandler` クラスを拡張してもかまいません。このクラスには Handler インタフェースのすべてのメソッドのデフォルト実装が用意されているため、ユーザー記述のハンドラ実装でこれらのメソッドを再定義する必要はありません。

関連項目：

次の Web サイトの API では、`Handler` インタフェースおよび `GenericHandler` クラスについてのさらに詳しい情報が提供されています。

`http://java.sun.com/j2ee/1.4/docs/api/javax/xml/rpc/handler/package-summary.html`

Ant タスクでサーバー・サイド・ハンドラを構成および登録する方法

ハンドラは最終的には、Web サービスのデプロイメント・ディスクリプタ（webservices.xml）に構成および登録されます。ただし、このファイルを自分で編集する必要はなく、開発時にハンドラ・クラス（Handler インタフェースを実装するクラス）を指定することにより適切な構成を WebServicesAssembler に生成させることができます。

注意： WebServicesAssembler では、JAX-RPC メッセージ・ハンドラを構成するための Ant タスクが提供されます。これらのタスクのリストについては、16-4 ページの「ハンドラを構成して登録できる Ant タスク」を参照してください。

ハンドラは、WebServicesAssembler コマンドラインを使用して構成することはできません。

たとえば、例 17-1 の WSDL で記述された Web サービスにサーバー・ハンドラを追加する場合は、次の Ant タスクを使用できます。サーバー・ハンドラは太字になっています。

```
<oracle:topDownAssemble appName="hello-service"
                    wsdl="Hello.wsdl"
                    input="./classes"
                    output="build"
                    ear="dist/hello-service.ear"
                    packageName="oracle.demo"
                    >
                     <oracle:porttype className="oracle.demo.HelloImpl" />
                     <oracle:handler name="ServerHandler"
                                handlerClass="oracle.demo.ServerHelloHandler"/>
</oracle:topDownAssemble>
```

この例では、hello-service Web サービスに対して、サーバー・ハンドラ oracle.demo.ServerHelloHandler が構成されています。任意の数のハンドラを追加するには、<handler> 要素を追加して、それぞれに一意の名前を使用します。ハンドラは、ハンドラ要素をリストするのと同じ順序で、チェーンに追加されます。この例はトップダウン方式の開発のものですが、同じ <handler> 要素を使用してその他の Ant タスクにハンドラを追加することもできます。

クライアント・サイド・ハンドラを追加する処理もほとんど同じです。

関連項目：

- ハンドラの構成の詳細は、18-81 ページの「Ant タスクでのハンドラの構成方法」を参照してください。
- クライアント・サイド・ハンドラの構成の詳細は、16-5 ページの「クライアント・サイドの JAX-RPC ハンドラ」を参照してください。

ハンドラを構成して登録できる Ant タスク

次に、<handler> 要素を含めることができる Ant タスクのリストを示します。これらのコマンドの詳細は、第 18 章「WebServicesAssembler の使用方法」を参照してください。

- aqAssemble
- assemble
- corbaAssemble
- dbJavaAssemble
- ejbAssemble
- genDDs
- genProxy（クライアント・サイドの生成専用）
- jmsAssemble
- plsqlAssemble
- sqlAssemble
- topDownAssemble

サーバー・サイド・ハンドラを登録するための webservices.xml の編集方法

Web サービスの生成時に Ant タスクを使用してハンドラを追加する場合は、適切な構成が webservices.xml ファイルに自動的に追加されます。このファイルに他の情報を追加する必要はありません。

一方、コマンドラインを使用する場合、または Web サービスのデプロイメント・ディスクリプタを手動で作成する場合は、webservices.xml ファイルを編集する必要があります。<port-component> 要素に <handler> サブ要素を追加することで、ファイルにハンドラを追加できます。例 16-1 は、複数のハンドラを含む <port-component> 要素を示しています。

例 16-1 webservices.xml 内の JAX-RPC ハンドラのサンプル

```
<port-component>
    ...
<handler>
    <handler-name>First Handler</handler-name>
    <handler-class>oracle.xx.AccountTransactionHandler</handler-class>
    <init-param>
       <param-name>test</param-name>
       <param-value>testValue</param-value>
    </init-param>
</handler>
<handler>
    <handler-name>Second Handler</handler-name>
    <handler-class>oracle.xx.NewAccountHandler</handler-class>
</handler>
   ...
</port-component>
```

表 16-1 は、サーバー・サイドのハンドラを指定する <handler> サブ要素を示します。

表 16-1 サーバー・サイドのハンドラのための <handler> サブ要素

サブ要素	説明
<handler-class>	ハンドラのクラスの完全修飾名。このクラスは javax.xml.rpc.handler.Handler を実装する必要があります。
<handler-name>	ハンドラを識別する一意の名前。
<init-param>	*param-name* と *param-value* のペアが 1 つ含まれるサブ要素。 *param-name* と *param-value* のペアは、ハンドラの init メソッドに渡される単一のパラメータとその値を示します。<handler> 要素内で使用可能な init-param サブ要素の数には制限がありません。

webservices.xml の内容の詳細は、次の Web アドレスにあるそのスキーマを参照してください。

http://java.sun.com/xml/ns/j2ee/j2ee_web_services_1_1.xsd

クライアント・サイドの JAX-RPC ハンドラ

Web サービス・クライアント上で、JAX-RPC ハンドラは、クライアント・アプリケーションから送信されたメッセージと、それに応答してサービスから戻されたメッセージを捕捉および処理できます。たとえば、これらのハンドラは SOAP メッセージを処理できます。次の各項では、Web サービス・クライアント上で使用するハンドラの登録方法について説明します。

- J2EE Web サービス・クライアント用 JAX-RPC ハンドラの登録方法
- J2SE Web サービス・クライアント用 JAX-RPC ハンドラの登録方法

J2EE Web サービス・クライアント用 JAX-RPC ハンドラの登録方法

J2EE Web サービス・クライアントの場合、JAX-RPC ハンドラ情報は、各 J2EE クライアントのデプロイメント・ディスクリプタの <service-ref> 要素内に表示されます。<service-ref> 要素は、サービスの J2EE クライアント関連情報をすべて取得します。この情報には、WSDL およびマッピング・ファイルの場所、サービス・インタフェース、サービスが実行されるポートおよび関連するサービス・エンドポイント・インタフェースなどがあります。

サーバー・サイドのハンドラとは異なり、クライアント・サイドのハンドラは、ポート・コンポーネント参照（<port-component>）ではなくサービス参照（<service-ref>）に関連付けられます。クライアント・サイドのハンドラには、起動されるサービスのポートに自身を関連付ける構成可能な <port-name> パラメータがあります。サービス・エンドポイント（WSDL ポート）の起動時には、<port-name> の値により、実行対象のハンドラが決まります。

J2EE Web サービス・クライアントにハンドラを登録するには、クライアントのデプロイメント・ディスクリプタの <service-ref> セクションにハンドラ情報を入力します。次のリストは、Web サービス・クライアントとして動作可能な各 J2EE コンポーネントに対する J2EE デプロイメント・ディスクリプタを示しています。

- WEB-INF/web.xml（JSP またはサーブレット用）
- META-INF/application-client.xml（アプリケーション・クライアント用）
- META-INF/ejb-jar.xml（EJB 用）

<service-ref> 要素の内容の詳細は、次の Web サイトで入手できる service-ref（J2EE クライアント）のスキーマを参照してください。

http://java.sun.com/xml/ns/j2ee/j2ee_web_services_client_1_1.xsd

J2EE Web サービス・クライアントでの handler 要素の使用方法

<handler> 要素は、J2EE Web サービス・クライアント用にハンドラ情報をカプセル化します。使用可能なサブ要素の詳細は、表 16-2 を参照してください。

表 16-2 J2EE Web サービス・クライアントのハンドラのための <handler> サブ要素

サブ要素	説明
<handler-class>	ハンドラのクラスの完全修飾名。このクラスは javax.xml.rpc.handler.Handler を実装する必要があります。
<handler-name>	ハンドラを識別する一意の名前。
<init-param>	*param-name* と *param-value* のペアが 1 つ含まれるサブ要素。 *param-name* と *param-value* のペアは、ハンドラの init メソッドに渡される単一のパラメータとその値を示します。<handler> 要素内で使用可能な init-param サブ要素の数には制限がありません。
<port-name>	ハンドラが動作するポートの名前。

<service-ref> セクションの最後にある任意のポート・コンポーネント情報の後ろにハンドラ情報を入力します。例 16-2 は、2 つのハンドラが定義された <service-ref> 要素を示します。この例では、First Handler はクラス oracle.xx.AccountTransactionHandler に関連付けられており、portA で動作します。Second Handler はクラス oracle.xx.NewAccountHandler に関連付けられており、portB で動作します。First Handler は PortA が起動されているときのみ動作し、Second Handler は PortB が起動されているときのみ動作します。

例 16-2 J2EE クライアントの JAX-RPC ハンドラのサンプル

```
<service-ref>
    <service-ref-name>service/MyHelloServiceRef</service-ref-name>
    ....
    <port-component-ref>
        ....
    </port-component-ref>
 <handler>
         <handler-name>First Handler</handler-name>
         <handler-class>oracle.xx.AccountTransactionHandler</handler-class>
         <port-name>portA</port-name>
 </handler>
 <handler>
         <handler-name>Second Handler </handler-name>
         <handler-class>oracle.xx.NewAccountHandler</handler-class>
         <port-name>portB</port-name>
      </handler>
</service-ref>
```

J2SE Web サービス・クライアント用 JAX-RPC ハンドラの登録方法

J2SE クライアントのクライアント・サイドの JAX-RPC ハンドラは、WebServicesAssembler の genProxy Ant タスクを使用して登録できます。例 16-3 では、J2SE クライアントはハンドラ oracle.demo.ClientHelloHandler を使用できるようになります。J2EE Web サービス・クライアントとは異なり、J2SE クライアントではデプロイメント・ディスクリプタは使用されません。

例 16-3 J2SE Web サービス・クライアントのためのハンドラの登録

```
<oracle:genProxy
            wsdl="http://localhost:8888/hello-service/hello-service?WSDL"
            output="build/src/client"
            packageName="oracle.demo">
            <oracle:handler
                  name="ClientHelloHandler"
                  handlerClass="oracle.demo.ClientHelloHandler" />
      </oracle:genProxy>
```

制限事項

F-10 ページの「JAX-RPC ハンドラの概要」を参照してください。

追加情報

詳細は、次を参照してください。

- メッセージを直接に処理する方法は、次の Web アドレスにある SAAJ API を参照してください。

  http://java.sun.com/webservices/saaj/index.jsp

- SOAP 1.2 メッセージを処理できるようにするための、SAAJ API に対する OracleAS Web Services の拡張機能の詳細は、5-12 ページの「SOAP メッセージの操作」を参照してください。
- WSDL からの Web サービスのアセンブル方法は、第 6 章「WSDL からの Web サービスのアセンブル」を参照してください。
- ステートフルな Web サービスのアセンブル方法は、第 7 章「Java クラスを使用した Web サービスのアセンブル」を参照してください。
- EJB からの Web サービスのアセンブル方法は、第 8 章「EJB を使用した Web サービスのアセンブル」を参照してください。
- JMS キューまたは JMS トピックからの Web サービスのアセンブル方法は、第 9 章「JMS 宛先を使用した Web サービスのアセンブル」を参照してください。
- データベース・リソースからの Web サービスのアセンブル方法は、第 10 章「データベース Web サービスのアセンブル」を参照してください。
- J2SE 5.0 注釈を使用した Web サービスのアセンブル方法は、第 11 章「注釈を使用した Web サービスのアセンブル」を参照してください。

17

SOAP ヘッダーの処理

この章では、SOAP ヘッダーを処理する方法について説明します。

- パラメータ・マッピングによる SOAP ヘッダーの処理方法
- ハンドラを使用した SOAP ヘッダーの処理方法
- ServiceLifecycle インタフェースを使用した SOAP ヘッダーの処理方法

パラメータ・マッピングによる SOAP ヘッダーの処理方法

WebServicesAssembler ツールを使用すると、WSDL ファイルの wsdl:binding 要素で定義されている SOAP ヘッダー・ブロックを、生成されるサービス・エンドポイント・インタフェース（SEI）のメソッド・パラメータにマップできます。これにより、SOAP ヘッダー・ブロックには、サービス・エンドポイント・インタフェースを実装するメソッド内から直接アクセスできるようになります。

例 17-1 に、SOAP ヘッダーを明示的に定義する単純な WSDL を示します。

例 17-1 SOAP ヘッダーを明示的に定義する単純な WSDL

```
<definition xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://test.com"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
   <types/>
   <message name="HelloHeader">
      <part name="header" type="xs:string"/>
   </message>
   <message name="HelloMessage">
      <part name="body" type="xs:string"/>
   </message>
   message name="HelloMessageResponse"/>
   <portType name="HelloPortType">
      <operation name="sayHello">
         <input message="tns:HelloMessage"/>
         <output message="tns:HelloMessageResponse/>
         </operation>
      </portType>
      <binding name="HelloBinding" type="tns:HelloPortType">
         <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
         <operation name="sayHello">
            <input>
               <soap:body use="literal" namespace="http://test.com"/>
               <!--the SOAP header must be defined here -->
               <soap:header message=""tns:HelloHeader" part="header" use="literal"/>
         </input>
         <output>
            <soap:body use="literal" namespace="http://test.com"/>
         </output>
      </operation>
   </binding>
   <service name="HelloService">
      <port name="HelloPort" binding="tns:HelloBinding">
         <soap:address location="http://localhost:8888/hello-service/hello-service"/>
      </port>
   </service>
</definition>
```

WebServicesAssembler ツールは、パラメータに SOAP ヘッダーをマップするためのブール型引数 mapHeadersToParameters を提供します。この引数のデフォルトは true のため、SOAP ヘッダーを抑止する場合を除いて明示的に指定する必要はありません。

Ant タスクまたは WebServicesAssembler ツールの使用によるパラメータ・マッピングにより、サービス・エンドポイント・インタフェースを生成できます。次の Ant タスクのサンプルでは、SOAP ヘッダーを、`hello-service` Web サービスのパラメータにマップしています。

```
<oracle:topDownAssemble appName="hello-service"
                    wsdl="Hello.wsdl"
                    input="./classes"
                    output="build"
                    ear="dist/hello-service.ear"
                    packageName="oracle.demo "
                    mapHeadersToParameters="true"
                    >
                    <oracle:porttype className="oracle.demo.HelloImpl" />
</oracle:topDownAssemble>
```

次の例は、前述の例の WebServicesAssembler コマンドライン・バージョンです。

```
java -jar wsa.jar -topDownAssemble
                  -wsdl Hello.wsdl
                  -output build
                  -ear dist/hello-services.ear
                  -mapHeadersToParameters true
                  -packageName oracle.demo
```

このコマンドおよび Ant タスクの説明 :

- `topDownAssemble`: WSDL 記述をベースにした Web サービスで必要となるクラスとデプロイメント・ディスクリプタを作成します。これらのファイルは、EAR ファイル、WAR ファイル、ディレクトリのいずれにも格納できます。18-21 ページの topDownAssemble を参照してください。
- `wsdl`: WSDL ドキュメントの絶対ファイル・パス、相対ファイル・パスまたは URL を指定します。18-64 ページの「wsdl」を参照してください。
- `output`: 生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。18-46 ページの「output」を参照してください。
- `ear`: 生成される EAR ファイルの名前と位置を指定します。18-42 ページの「ear」を参照してください。
- `mapHeadersToParameters`: 生成される Java コード内の各メソッド用のパラメータに、WSDL で定義されている SOAP ヘッダーをマップするかどうかを指定します。18-69 ページの「mapHeadersToParameters」を参照してください。
- `packageName`: JAX-RPC マッピング・ファイルにパッケージ名が宣言されていない場合に、生成されたクラス用に使用されるパッケージ名を指定します。18-47 ページの「packageName」を参照してください。

関連項目 :

この引数の詳細は、18-69 ページの「mapHeadersToParameters」を参照してください。

パラメータにヘッダーをマップできる Ant タスクとコマンド

次の WebServicesAssembler コマンドおよび Ant タスクを使用すると、`mapHeaderstoParameters` 引数をコールできます。これらのコマンドの詳細は、第 18 章「WebServicesAssembler の使用方法」を参照してください。

- genInterface
- genProxy
- topDownAssemble

ハンドラを使用した SOAP ヘッダーの処理方法

JAX-RPC ハンドラを使用すると、明示的と暗黙的の両方の SOAP ヘッダーを処理できます。明示的 SOAP ヘッダーとは、WSDL ドキュメントに定義されている SOAP ヘッダーのことです。例 17-1 に、SOAP ヘッダーが定義されている単純な WSDL を示します。暗黙的 SOAP ヘッダーとは、必ずしも特定の WSDL ドキュメントで定義されているとはかぎらないが、SOAP エンベロープに含まれているものです。

ハンドラでは、`javax.xml.rpc.handler.Handler` インタフェースで定義されたメソッドを使用して SOAP ヘッダーにアクセスできます。これらのメソッドは次のとおりです。

```
boolean handleRequest(MessageContext context);
boolean handleResponse(MessageContext context);
boolean handleFault(MessageContext context);
```

これらのメッセージの各 `context` 引数を使用して、SOAP エンベロープ内の SOAP ヘッダーを表示できます。次に、SOAP リクエストのヘッダーを表示するハンドラ実装の例を示します。

```
boolean handleRequest(MessageContext context){
      javax.xml.rpc.handler.soap.SOAPMessageContext smc =
(javax.xml.rpc.handler.soap.SOAPMessageContext)context;
      javax.xml.soap.SOAPHeader sh = smc.getSOAPMessage.getSOAPHeader();
      //the SOAPHeader will contain a list of SOAPHeaderElements (header blocks)
      Iterator it = sh.examineAllHeaderElements();
      //iterate through all the SOAP header elements and print their names
      while(it.hasNext()){
            javax.xml.soap.SOAPHeaderElement elem = (SOAPHeaderElement)it.next();
            System.out.println(elem.getElementName().getQualifiedName());
      }
      return true;
}
```

関連資料：

- `Handler` インタフェースの詳細は、次の Web アドレスにある Java API for XML-Based RPC（JAX-RPC 1.1）の仕様を参照してください。

  `http://java.sun.com/webservices/jaxrpc/index.jsp`

- メッセージの直接処理の詳細は、次の Web アドレスにある SOAP with Attachments API for Java （SAAJ）を参照してください。

  `http://java.sun.com/webservices/saaj/index.jsp`

ServiceLifecycle インタフェースを使用した SOAP ヘッダーの処理方法

サービス・エンドポイントのライフ・サイクルは、javax.xml.rpc.server.ServiceLifecycle インタフェースを実装することで管理できます。このインタフェースには次のメソッドがあります。

```
void init(Object context);
void destroy();
```

ランタイム・システムは、init メソッドを起動し、context オブジェクトを渡します。context オブジェクトには、ServletEndpointContext オブジェクトが含まれています。このオブジェクトから SOAPMessageContext を抽出し、起動のたびにヘッダーを処理できます。

例 17-2 は、javax.xml.rpc.server.ServiceLifecycle インタフェースを使用して SOAP ヘッダーにアクセスする方法を示しています。

例 17-2 ServiceLifecycle を使用して SOAP ヘッダーにアクセスする方法

```
public class HelloImpl implements HelloPortType,ServiceLifecycle{
      private Object m_context;

      public void sayHello(String body){
            javax.xml.rpc.server.ServletEndpointContext sec =
(ServletEndpointContext)m_context;
            javax.xml.rpc.handler.soap.SOAPMessageContext mc =
(SOAPMessageContext)sec.getMessageContext();
            javax.xml.soap.SOAPHeader sh = mc.getSOAPMessage().getSOAPHeader();
             // from here you can process all the header
             // blocks in the SOAP header.
}

      //this will be called by the runtime system.
      public void init(Object context){
      m_context = context;
      }
      public void destroy(){
      }
}
```

このインタフェースを実装することで、明示的および暗黙的の両方の SOAP ヘッダーを処理できます。ただし、暗黙的ヘッダーを処理する場合にとっての方が利便性は高くなります。

HTTP レスポンスおよびリクエストのヘッダーの取得方法

次の各項では、静的スタブ・クライアントおよび Dynamic Invocation Interface（DII）クライアントが HTTP のレスポンスおよびリクエストからヘッダーを取得する方法を説明します。

- スタブ・クライアントの ServiceLifecycle インタフェースによるヘッダーの取得方法
- DII クライアントの OracleCall インタフェースによるヘッダーの取得方法
- WSIF サービスの DII クライアントによるメッセージ・ヘッダーの取得方法

スタブ・クライアントの ServiceLifecycle インタフェースによるヘッダーの取得方法

J2SE および J2EE の静的スタブ・クライアントは、`oracle.webservices.ServerConstants` クラスの `HTTP_SERVLET_REQUEST` プロパティおよび `HTTP_SERVLET_RESPONSE` プロパティを使用して、HTTP メッセージのヘッダーにアクセスできます。サービス実装クラスでは、Web サービスのコール元が HTTP トランスポートを使用するときに、このプロパティを使用することで HTTP サーブレットのリクエストまたはレスポンスを取得できます。

これらのプロパティを使用するには、サービス実装では `javax.xml.rpc.server.ServiceLifecyle` を実装し、`init` メソッドに渡された `Object` を保存する必要があります。このオブジェクトは `javax.xml.rpc.server.ServletEndpointContext` のインスタンスです。

サービス実装クラスのメソッドが起動されると、`ServletEndpointContext.getMessageContext` メソッドにより `javax.xml.rpc.handler.MessageContext` が戻されます。`MessageContext.getProperty` メソッドは、`HTTP_SERVLET_REQUEST` または `HTTP_SERVLET_RESPONSE` をプロパティ名として使用できます。

`HTTP_SERVLET_REQUEST` が要求されたプロパティである場合、戻されるオブジェクトは、`javax.servlet.http.HttpServletRequest` のインスタンスです。`HTTP_SERVLET_RESPONSE` が要求されたプロパティである場合、戻されるオブジェクトは、`javax.servlet.http.HttpServletResponse` のインスタンスです。

例 17-3 に、HTTP リクエスト・ヘッダーを取得して IP アドレスを取得する方法を示します。この例では、`HelloImpl` クラスにより `ServiceLifecycle` インタフェースが実装されています。この場合、`init` メソッドに渡された `context` オブジェクトは、`ServletEndpointContext` オブジェクトにキャストされています。`destroy` メソッドにより、サービスのライフ・サイクルが破棄されています。`getIPAddress` メソッドの実装では、`getMessageContext` メソッドにより、`ServletEndpointContext` オブジェクトからメッセージ・コンテキストが引き出されます。`getProperty` メソッドでは、`HTTP_SERVLET_REQUEST` プロパティを使用し、`HttpServletRequest` オブジェクトとしてリクエストを戻します。`getRemoteAddr` メソッドでは、IP アドレスが戻されます。

例 17-3 HTTP ヘッダーの取得

```
public class HelloImpl implements ServiceLifecycle  {
   ServletEndpointContext m_context;
   public void init(Object context) throws ServiceException {
       m_context = (ServletEndpointContext)context;
   }

   public void destroy() {
    }

   public String getIPAddress(){
      HttpServletRequest request = (HttpServletRequest)m_context.
          getMessageContext().getProperty(ServerConstants.HTTP_SERVLET_REQUEST);
      return request.getRemoteAddr();
   }

}
```

DII クライアントの OracleCall インタフェースによるヘッダーの取得方法

Dynamic Invocation Interface（DII）クライアントは、`oracle.webservices.OracleCall` インタフェースのメソッドを使用して、HTTP リクエストおよびレスポンスのヘッダーを取得できます。レスポンス・ヘッダーを取得するには、`getResponseHeaders()` メソッドを使用します。このメソッドは `Vector` を戻します。`Vector` 内の要素が、SOAP レスポンス・ヘッダーです。

リクエスト・ヘッダーを取得するには、`getHeaders()` メソッドを使用します。このメソッドも `Vector` を戻し、その要素が SOAP レスポンス・ヘッダーです。通常、ヘッダーは、`addHeaders()` メソッドで `Vector` に追加されます。

例 17-4 は、レスポンス・ヘッダーを取得するための DII クライアント・コードです。

例 17-4 DII クライアント・コードでのレスポンス・ヘッダーの取得

```
...
OracleCall call = (OracleCall)port.getCall();
Vector headers = call.getResponseHeaders();
if (Vector != null) {
    for(int i = 0;i < headers.size();i++) {
        Element header = (Element) headers.get(i);
        // do something with the header Element
        ...
    }
}
...
```

WSIF サービスの DII クライアントによるメッセージ・ヘッダーの取得方法

Web Services Invocation Framework（WSIF）の DII クライアントがメッセージ・ヘッダーを取得する方法の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の WSIF サービスの DII クライアントによるメッセージ・ヘッダーの取得方法に関する項を参照してください。

制限事項

F-10 ページの「SOAP ヘッダーの処理」を参照してください。

追加情報

詳細は、次を参照してください。

- WSDL からの Web サービスのアセンブル方法は、第 6 章「WSDL からの Web サービスのアセンブル」を参照してください。
- ステートフルな Web サービスのアセンブル方法は、第 7 章「Java クラスを使用した Web サービスのアセンブル」を参照してください。
- EJB からの Web サービスのアセンブル方法は、第 8 章「EJB を使用した Web サービスのアセンブル」を参照してください。
- JMS キューまたは JMS トピックからの Web サービスのアセンブル方法は、第 9 章「JMS 宛先を使用した Web サービスのアセンブル」を参照してください。
- データベース・リソースからの Web サービスのアセンブル方法は、第 10 章「データベース Web サービスのアセンブル」を参照してください。
- J2SE 5.0 注釈を使用した Web サービスのアセンブル方法は、第 11 章「注釈を使用した Web サービスのアセンブル」を参照してください。

18

WebServicesAssembler の使用方法

この章では、WebServicesAssembler ツールに備わっている機能について説明します。

WebServicesAssembler ツールの概要

WebServicesAssembler ツールを使用すると、Oracle Application Server Web Services をアセンブルできます。WebServicesAssembler ツールを使用すると、トップダウン方式とボトムアップ方式のどちらで Web サービスを作成するかにかかわらず、サービスの開発およびデプロイに必要なアーチファクトを生成できます。また、WebServicesAssembler は、WSDL をベースにした Web サービス・クライアント・オブジェクトの作成用に呼び出すこともできます。

トップダウン方式での Web サービスの生成のサポート

トップダウン方式の場合は、WebServicesAssembler に WSDL を提供し、これによりサービス・エンドポイント・インタフェースを作成します。次に、Java クラスなどの必要なアーキテクチャ用のサービスを実装していきます。

関連項目：

WebServicesAssembler を使用したトップダウン方式の Web サービス・アセンブリの例は、第 6 章「WSDL からの Web サービスのアセンブル」を参照してください。

ボトムアップ方式での Web サービスの生成のサポート

ボトムアップ方式の場合は、既存のビジネス・ロジックをベースにします。これには、Java クラス、Enterprise JavaBeans（EJB）、CORBA オブジェクト、JMS キュー、または PL/SQL プロシージャなどのデータベース・アーチファクトなどがあります。WebServicesAssembler は、これらのアーチファクトを使用して、WSDL、マッピング・ファイルおよび必要なデプロイメント・ディスクリプタをアセンブルします。

関連項目：

WebServicesAssembler を使用したボトムアップ方式での Web サービスのアセンブリの例は、次の各章を参照してください。

- 第 7 章「Java クラスを使用した Web サービスのアセンブル」
- 第 8 章「EJB を使用した Web サービスのアセンブル」
- 第 9 章「JMS 宛先を使用した Web サービスのアセンブル」
- 第 10 章「データベース Web サービスのアセンブル」

XML スキーマ・ベースの Web サービスの生成のサポート

スキーマ・ベースの場合は、XML スキーマをベースにして JaveBean を生成します。JaveBean が生成された後、Bean を引数として使用するインタフェースを記述し、ボトムアップ方式で WSDL、マッピング・ファイルおよびデプロイメント・ディスクリプタを生成します。

JAX-B または Toplink を使用して XML スキーマから Bean を生成することもできますが、WebServicesAssembler または Ant タスクを使用することもできます。

関連資料：

Ant タスクまたは WebServicesAssembler を使用してスキーマから Web サービスを生成する方法の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「スキーマ・ドリブン方式の Web サービス開発におけるカスタム・シリアライズの使用」を参照してください。

デプロイのサポート

OC4J コンテナではデプロイを処理しますが、WebServicesAssembler ツールでは、確実に、生成されるアプリケーション・アーカイブが適切にデプロイ用に準備されるようにします。WebServicesAssembler では、すべての関連デプロイメント・ディスクリプタの生成が処理され、アプリケーションによって必要とされる固有の構成が Oracle 固有のデプロイ・ファイルにマップされます。Java クラスまたは EJB 2.1 をベースにしたアプリケーションは、様々なコンテナにデプロイ可能です。このような Web サービスは J2EE 標準のデプロイ可能形式であり、業界標準である JAX-RPC、Enterprise Web Services 1.1、Web Services-Interoperability（WS-I）などの仕様に準拠しています。

コマンドラインからの起動と Ant タスクのサポート

WebServicesAssembler ツールは、コマンドラインで、または Ant タスクによって起動できます。WebServicesAssembler を使用することで、Web サービスのアセンブル方法の選択が拡がります。アセンブリ・プロセスをいくつかのステップに分割し、Web サービスの作成方法をより綿密に制御できます。たとえば、次のタスクを実行できます。

- 別のメカニズムを使用したデプロイメント・ディスクリプタの作成

  このために、WebServicesAssembler には、デプロイメント・ディスクリプタからのサービスを追加および削除するためのコマンドライン引数が用意されています。すなわち、WebServicesAssembler では生成されない情報が含まれた、手動でコーディングした一連のデプロイメント・ディスクリプタをベースにできる、ということです。WebServicesAssembler を使用して、このディスクリプタに情報を追加できます。
- アーチファクトのコンパイル時期と使用されるクラスパスの制御
- アーチファクトがアーカイブにパッケージされる時期の制御
- アーカイブ・コンテンツの制御
- 非プラットフォーム依存ビルド・ファイルの作成

コマンドライン構文

WebServicesAssembler ツールでは、いくつかのコマンドがサポートされています。WebServicesAssembler を起動する際、コマンドライン上に指定できるコマンドは 1 つのみです。

一般的なコマンドラインの構文は次のとおりです。

```
java -jar [OC4J_HOME]/webservices/lib/wsa.jar -[command] -[argument name][argument value]...
```

この例において、*OC4J_HOME* は OC4J がインストールされた場所で、`wsa.jar` は WebServicesAssembler JAR ファイルの名前です。コマンドラインの構文規則は次のとおりです。

- コマンドラインでは最初にコマンドを指定します。
- コマンドと引数は前にダッシュ「-」を付けます。ただし、これは必須ではありません。
- 引数名とその値は空白で区切ります。
- Ant タスクで使用されるときは、すべての引数名で大 / 小文字が区別されます。コマンドラインで使用されるときは、大 / 小文字は区別されません。
- 引数値内に空白が必要な場合は、その値をエスケープします。

  たとえば、Windows と Linux では、空白は二重引用符で囲みます。その他のオペレーティング・システムでは、Java 実行可能ファイルにパラメータとして空白を渡すために別の方法が必要な場合があります。
- コマンドの後に使用する引数は、すべて任意の順序で指定できます。
- 通常、引数はコマンドラインで 1 回のみ使用できます。このルールに対する例外は、個々の引数の説明を参照してください。

WebServicesAssembler に向けた Ant の設定

Ant タスクから WebServicesAssembler コマンドをコールするには、Ant のインストールに若干の変更や追加が必要なことがあります。

関連項目：

Ant タスクをコールするために必要な変更および追加の詳細は、4-3 ページの「WebServicesAssembler 用の Ant の設定方法」を参照してください。

WebServicesAssembler コマンド

この項では、WebServicesAssembler ツールで使用可能なコマンドについて説明します。コマンドは、備えている機能別に次の各カテゴリに分類されます。

- Web サービス・アセンブリ・コマンド : Web サービスのアセンブル。このコマンドでは、WAR、EAR、EJB JAR など、デプロイ可能なアーカイブを作成するために必要なすべてのファイルが作成されます。
- WSDL 管理コマンド : WSDL に対するアクションの実行。アクションには、ボトムアップ開発用 WSDL の生成、その内容と位置の管理、WebServicesAssembler による処理の可否の設定などがあります。
- Java 生成コマンド : WSDL、プロキシ / スタブまたは JAX-RPC 値タイプ・クラスから Java インタフェースを作成するためのコードの生成。
- デプロイメント・ディスクリプタ生成コマンド : EAR、WAR または EJB JAR 用のデプロイメント・ディスクリプタの生成。
- メンテナンス・コマンド : WebServicesAssembler コマンドの概略とツールのバージョン番号の出力。

Web サービス・アセンブリ・コマンド

Web サービスのアセンブルに使用できるコマンドは、次のとおりです。

- aqAssemble: データベースのアドバンスト・キューから Web サービスをアセンブルします。
- assemble: Java クラスおよび注釈付き Java クラスから Web サービスをアセンブルします。J2SE 5.0 Web サービス注釈の詳細は、第 11 章「注釈を使用した Web サービスのアセンブル」を参照してください。
- corbaAssemble: CORBA サーバント・オブジェクトから Web サービス・エンドポイントをアセンブルします。
- dbJavaAssemble: データベースの Java クラスから Web サービスをアセンブルします。
- ejbAssemble: EJB を Web サービスとしてアセンブルします。
- jmsAssemble: JMS エンドポイント Web サービスをアセンブルします。
- plsqlAssemble: PL/SQL パッケージから Web サービスをアセンブルします。
- sqlAssemble: SQL 問合せおよび DML 文から Web サービスをアセンブルします。
- topDownAssemble: WSDL 仕様から Web サービスのクラスとデプロイメント・ディスクリプタをアセンブルします。

これらのコマンドに共通の機能と動作は、次のとおりです。

- `*Assemble` の各コマンドでは、デプロイ可能アーカイブに必要なすべてのファイルが作成されます。
- `*Assemble` の各コマンドでは、`topDownAssemble` を除いて、Java クラスの生成後、`assemble` コマンドがコールされます。

- *Assemble のコマンドによって作成されたファイルは、output 引数によって指定されたディレクトリの下のステージング・ディレクトリ構造に格納されます。

  *Assemble コマンドによって作成されるステージング・ディレクトリ構造の図は、図 18-1 を参照してください。

- ear および war ディレクトリの名前と内容は、ear および war 引数に指定した値によって決まります。

  これらの引数の詳細は、18-42 ページの「ear」および 18-49 ページの「war」を参照してください。

- *Assemble のコマンドは、ejbAssemble を除いて、EAR および WAR ファイルを出力します。また、オプションでディレクトリに出力できます。このディレクトリには、OC4J インスタンスにデプロイできる WAR の内容が格納されます。

- クラスパスの内容はアーカイブにコピーされません。*Assemble のいずれかのコマンドに classpath または input 引数を指定した場合は、クラスパス内に含まれるクラスがサーバー上に実際にあるかどうかを確認してください。

- EAR または WAR ファイルを *Assemble コマンドによってアーカイブする前に、EAR または WAR ファイルにファイルを追加できます。

  このトピックの詳細は、18-84 ページの「アーカイブにファイルを追加する方法」を参照してください。

図 18-1 に、*Assemble コマンドによって作成されるステージング・ディレクトリ構造を示します。各コマンドにより、指定された出力ディレクトリの下に、ear、src および war の 3 つのサブディレクトリが作成されます。

- ear サブディレクトリには、生成された EAR ファイルの内容が格納されます。さらに、このサブディレクトリには、application.xml ファイルが含まれる META-INF サブディレクトリがあります。

- src サブディレクトリには、生成されたソース・ファイルが格納されます。さらに、このサブディレクトリには、Java プロキシ・ファイルが含まれる proxy サブディレクトリがあります。Web サービスが WSDL からアセンブルされている場合は（トップダウン方式）、src には、生成されたサービス・エンドポイント・インタフェースと値タイプ・クラスのスケルトンが入ったサブディレクトリも含まれます。

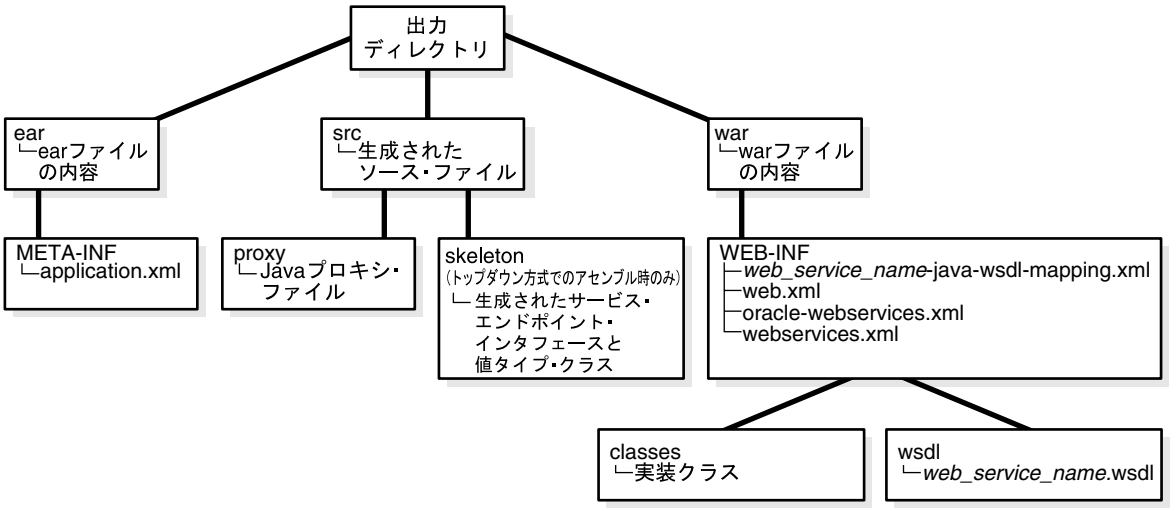
- war サブディレクトリには、生成される WAR ファイルの内容が格納されます。さらに、このサブディレクトリの下には、WEB-INF サブディレクトリがあります。WEB-INF には、マッピング・ファイルと、標準のデプロイメント・ディスクリプタおよび Oracle 固有のデプロイメント・ディスクリプタが格納されます。このようなファイルには、*web-service_name*_java_wsdl_mapping.xml、web.xml、oracle-webservices.xml および webservices.xml ファイルがあります。

  さらに、WEB-INF には、classes および wsdl サブディレクトリがあります。classes サブディレクトリには実装クラスが格納されます。wsdl サブディレクトリには Web サービスの WSDL ファイルが格納されます。

注意： デフォルトでは、WebServicesAssembler によって、war および ear ステージング・ディレクトリからファイルは削除されません。複数の WebServicesAssembler の起動に同じ出力ディレクトリを使用すると、WAR および EAR のアーカイブ内に余分な不要のファイルが含まれる場合があります。このような動作を避けるには、次のいずれかを実行します。

- WebServicesAssembler の起動ごとに異なる出力ディレクトリを指定する
- WebServicesAssembler の起動のコール間に出力ディレクトリの内容を削除する

図 18-1 *Assemble コマンドによって作成されるステージング・ディレクトリ構造



aqAssemble

aqAssemble コマンドは、データベースのアドバンスト・キューから Web サービスを生成する場合に使用します。このコマンドを使用するには、データベースに接続する必要があります。

aqAssemble コマンドでは、アドバンスト・キュー用 WSIF バインディングを WSDL に追加することもできます。データベース・リソースを Web サービスとして公開しているときに、WSIF バインディングを WSDL に追加するには、wsifDbBinding 引数を使用します。リソースの Oracle JPublisher 生成 Java クラスの className とデータベース接続を指定する必要があります。

関連資料 :

- aqAssemble コマンドを使用してアドバンスト・キューを Web サービスとして公開する方法の詳細は、10-23 ページの「Oracle Streams AQ から Web サービスをアセンブルする方法」を参照してください。
- データベースに接続できる引数の詳細は、18-75 ページの「データベース接続の確立方法」を参照してください。
- データベース・リソース用のバインディングの WSDL への追加については、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「データベース・リソースへの WSIF エンドポイントの構成」を参照してください。

コマンドラインの例 :

```
java -jar wsa.jar -aqAssemble
                  -dbUser scott/tiger
                  -sql ToyQueue
                  -dbConnection jdbc:oracle:thin:@dsunrde22:1521:sqlj
                  -dataSource jdbc/OracleManagedDS
                  -appName query
```

Ant タスクの例 :

```
<oracle:aqAssemble
     dbUser="scott/tiger"
     sql="ToyQueue"
     dbConnection="jdbc:oracle:thin:@dsunrde22:1521:sqlj"
     dataSource="jdbc/OracleManagedDS"
     appName="query"
/>
```

必須引数 :

`aqAssemble` コマンドを使用する際には、次の引数の指定が必須です。

- **データベース・アセンブリ引数 :** sql

  データベースへの接続も必要です。それには、次の引数の組合せのいずれかを使用します。

  - dataSource、dbConnection および dbUser
  - dataSource

すべての引数 :

次のリストは、`aqAssemble` コマンドで使用できるすべての引数を示しています。

- **データベース・アセンブリ引数 :** aqConnectionFactoryLocation、aqConnectionLocation、dataSource、dbConnection、dbUser、jpubProp、sql（必須）、sqlTimeout、sysUser、useDataSource、wsifDbBinding、wsifDbPort

  これらの引数の詳細は、18-52 ページの「データベース・アセンブリ引数」を参照してください。

- **デプロイメント・ディスクリプタ引数 :** appendToExistingDDs、context、ddFileName、uri

  これらの引数の詳細は、18-61 ページの「デプロイメント・ディスクリプタ引数」を参照してください。

- **アセンブリ用の一般引数 :** appName、debug、ear、emptySoapAction、help、interfaceName、mappingFileName、output、packageName、portName、restSupport、schema、serviceName、useDimeEncoding、war

  これらの引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

- **Java 生成引数 :** valueTypeClassName

  この引数の詳細は、18-68 ページの「Java 生成引数」を参照してください。

- **メッセージ書式引数 :** style、use

  これらの引数の詳細は、18-67 ページの「メッセージ書式引数」を参照してください。

- **セッション引数 :** timeout

  この引数の詳細は、18-50 ページの「セッション引数」を参照してください。

- **WSDL アクセス引数 :** httpNonProxyHosts、httpProxyHost、httpProxyPort

  これらの引数の詳細は、18-63 ページの「WSDL アクセス引数」を参照してください。

- **WSDL 管理引数 :** createOneWayOperations、genQos、qualifiedElementForm、soapVersion、targetNamespace、typeNamespace

  これらの引数の詳細は、18-64 ページの「WSDL 管理引数」を参照してください。

Ant タスクのサポート :

- `<proxy>` タグ。詳細は、18-77 ページの「Ant タスクでのプロキシ生成の構成方法」を参照してください。
- `<port>` タグ。`<port>` タグでは、`aqAssemble` は、引数 bindingName、`name`（`portName` と同じ）、portName、sendConnectionFactoryLocation、sendQueueLocation、soapVersion および uri を使用できます。詳細は、18-78 ページの「Ant タスクでのポートの構成方法」を参照してください。
- `<handler>` タグ。詳細は、18-81 ページの「Ant タスクでのハンドラの構成方法」を参照してください。

assemble

assemble コマンドは、Web サービスをボトムアップ方式で生成する場合に使用します。このコマンドにより、デプロイ可能なアーカイブを作成するために必要なすべてのファイルが作成されます。このようなファイルには、固有のデプロイメント・ディスクリプタ oracle-webservices.xml があります。このコマンドは、トランスポート・メカニズムが HTTP または JMS のどちらでも、ステートレス Web サービスを生成できます。

Java 実装クラスを検索するには、input または classpath 引数を指定する必要があります。これらの引数のいずれかを指定した場合は、クラスパス内に含まれるクラスがサーバー上に実際にあるかどうかを確認してください。クラスパスの内容はアーカイブにコピーされないためです。

WSIF バインディングの WSDL へのアセンブル

assemble コマンドでは、Web サービスを生成する以外に、WSDL への WSIF バインディングの追加も行えます。Java クラスを Web サービスとして公開しているときに、WSIF バインディングを WSDL に追加するには、wsifJavaBinding 引数を使用します。また、className 引数に Java クラスを指定する必要もあります。

関連資料：

- 詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「複数の Java ポートに対する WSIF エンドポイントの構成」を参照してください。
- WSIF の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービス起動フレームワークの使用方法」を参照してください。

J2SE 5.0 注釈付き Web サービスのアセンブル

assemble コマンドを使用して J2SE 5.0 注釈を含む Java クラスから Web サービスをアセンブルする場合は、className 引数を使用して実装クラスを指定します。@WebService 注釈がクラス宣言に存在する必要があります。

J2SE 5.0 注釈にはクラス名が必要です。注釈がインタフェースと実装クラスのいずれかに付けられる可能性があるためです。注釈がインタフェースのみに付けられている場合は、WebServicesAssembler は、インタフェースが参照する実装クラスを介して注釈を取得できます。実装クラスにも注釈が付いている場合は、この注釈が WebServicesAssembler によって処理されます。

次の例では、J2SE 5.0 注釈が付いている Java クラスから Web サービスを生成するために assemble コマンドが使用されています。className 引数には、com.mycompany.HelloImpl クラスが指定されています。

```
java -jar wsa.jar assemble -appName myService -className com.mycompany.HelloImpl
-output wsdl
```

WebServicesAssembler にインタフェースの注釈のみを処理させる場合は、serviceEndpointInterface プロパティ付きの @WebService 注釈を実装クラス・ファイルに入力します。J2SE 5.0 注釈では、残りすべての注釈もこのサービス・エンドポイント・インタフェース・クラスに付けられているものと予期します。たとえば、実装クラス・ファイルに次の注釈を入力すると、WebServicesAssembler では demo.myInterface インタフェースにある注釈のみが処理されます。

```
@WebService(serviceEndpointInterface="demo.myInterface")
```

関連項目：

J2SE 5.0 注釈を使用した Web サービスのアセンブルの詳細は、第 11 章「注釈を使用した Web サービスのアセンブル」を参照してください。

次のコマンドラインと Ant タスクの例では、assemble コマンドを使用してボトムアップ方式で Web サービスを生成する方法を示しています。このコマンドでは、build/myService.ear という名前の EAR ファイルが作成されます。

コマンドラインの例 :

```
java -jar wsa.jar -assemble
                  -input myservice.jar
                  -className com.mycompany.HelloImpl
                  -interfaceName com.myCompany.myService.Hello
                  -output build
                  -appName myService
```

Ant タスクの例 :

```
<oracle:assemble appName="myService"
          output="build"
          input="myservice.jar
          >
          <oracle:porttype
            interfaceName="com.myCompany.myService.Hello"
            className="com.mycompany.HelloImpl"/>
</oracle:assemble>
```

必須引数 :

assemble コマンドを使用する際には、次の引数の指定が必須です。

- **アセンブリ用の一般引数 :** className

すべての引数 :

次のリストは、assemble コマンドで使用できるすべての引数を示しています。

- **デプロイメント・ディスクリプタ引数 :** appendToExistingDDs、context、ddFileName、uri

  これらの引数の詳細は、18-61 ページの「デプロイメント・ディスクリプタ引数」を参照してください。

- **アセンブリ用の一般引数 :** appName、bindingName、className（必須）、classpath、debug、ear、emptySoapAction、help、input、interfaceFileName、interfaceName、mappingFileName、output、portName、portTypeName、restSupport、schema、serviceName、strictJaxrpcValidation、useDimeEncoding、war

  これらの引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

- **Java 生成引数 :** valueTypeClassName、wsifJavaBinding

  これらの引数の詳細は、18-68 ページの「Java 生成引数」を参照してください。

- **JMS アセンブリ引数 :** sendConnectionFactoryLocation、sendQueueLocation

  これらの引数の詳細は、18-57 ページの「JMS アセンブリ引数」を参照してください。

- **メッセージ書式引数 :** mtomSupport、style、use

  これらの引数の詳細は、18-67 ページの「メッセージ書式引数」を参照してください。

- **セッション引数 :** callScope、recoverable、session、timeout

  これらの引数の詳細は、18-50 ページの「セッション引数」を参照してください。

- **WSDL アクセス引数 :** httpNonProxyHosts、httpProxyHost、httpProxyPort

  これらの引数の詳細は、18-63 ページの「WSDL アクセス引数」を参照してください。

- **WSDL 管理引数 :** createOneWayOperations、genQos、qualifiedElementForm、soapVersion、targetNamespace、typeNamespace

  これらの引数の詳細は、18-64 ページの「WSDL 管理引数」を参照してください。

Ant タスクのサポート :

- <proxy> タグ。詳細は、18-77 ページの「Ant タスクでのプロキシ生成の構成方法」を参照してください。
- <port> タグ。<port> タグでは、assemble は、引数 bindingName、name（portName と同じ）、portName、sendConnectionFactoryLocation、sendQueueLocation、soapVersion および uri を使用できます。詳細は、18-78 ページの「Ant タスクでのポートの構成方法」を参照してください。
- <porttype> タグ。詳細は、18-80 ページの「Ant タスクでのポート・タイプの構成方法」を参照してください。
- <handler> タグ。詳細は、18-81 ページの「Ant タスクでのハンドラの構成方法」を参照してください。
- 複数ポート用の WSIF バインディングを指定するタグ。詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「複数の Java ポートに対する WSIF エンドポイントの構成」を参照してください。

corbaAssemble

corbaAssemble コマンドは、CORBA サーバント・オブジェクトを Web サービスとして公開する場合に使用します。このコマンドは、CORBA IDL ファイルと CORBA ネーミング・プロパティを入力として取ります。このコマンドの出力は、デプロイ可能なアーカイブを作成するために必要なすべてのファイルです。

WebServicesAssembler 内から、その環境内から見つかる IDL-to-Java コンパイラ（idlj）が起動されます。JDK bin ディレクトリが、path 環境変数の構成要素として設定されている必要があります。WebServicesAssembler では、IDL-to-Java コンパイラを使用して IDL ファイルが Java クラスにコンパイルされます。

コマンドラインの例 :

```
java -jar wsa.jar -corbaAssemble
                  -idlInterfaceName oraclecorba.Hello
                  -corbanameURL corbaname::corba.orbd.host:1050#oracle.corba/Hello
                  -idlFile ./Hello.idl
                  -uri /corba_hello
                  -output dist
                  -context corba_hello
                  -targetNamespace http://oracle.j2ee.ws/corba/Hello
                  -typeNamespace http://oracle.j2ee.ws/corba/Hello/types
                  -serviceName Corba_hello
                  -appName corba_hello
                  -style rpc
                  -use literal
```

Ant タスクの例 :

```
<oracle:corbaAssemble idlInterfaceName="oraclecorba.Hello"
                corbanameURL="corbaname::corba.orbd.host:1050#oracle.corba/Hello"
                idlFile="./Hello.idl"
                output="dist"
                context="corba_hello"
                targetNamespace="http://oracle.j2ee.ws/corba/Hello"
                typeNamespace="http://oracle.j2ee.ws/corba/Hello/types"
                serviceName="Corba_hello"
                appName="corba_hello"
                style="rpc"
                use="literal"
                >
                <oracle:port  uri="/corba_hello" />
    </oracle:corbaAssemble>
```

必須引数 :

`corbaAssemble` コマンドを使用する際には、次の引数の指定が必須です。

- **CORBA アセンブリ引数 :** idlInterfaceName

すべての引数 :

次のリストは、`corbaAssemble` コマンドで使用できるすべての引数を示しています。

- **CORBA アセンブリ引数 :** corbanameURL、corbaObjectPath、idlFile、idlInterfaceName（必須）、idljPath、ORBInitialHost、ORBInitialPort、ORBInitRef

  これらの引数の詳細は、18-51 ページの「CORBA アセンブリ引数」を参照してください。

- **デプロイメント・ディスクリプタ引数 :** appendToExistingDDs、context、ddFileName、uri

  これらの引数の詳細は、18-61 ページの「デプロイメント・ディスクリプタ引数」を参照してください。

- **アセンブリ用の一般引数 :** appName、bindingName、className、classpath、debug、ear、emptySoapAction、help、mappingFileName、output、packageName、portName、portTypeName、restSupport、schema、serviceName、useDimeEncoding、war

  これらの引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

- **JMS アセンブリ引数 :** sendConnectionFactoryLocation、sendQueueLocation

  これらの引数の詳細は、18-57 ページの「JMS アセンブリ引数」を参照してください。

- **メッセージ書式引数 :** style、use

  これらの引数の詳細は、18-67 ページの「メッセージ書式引数」を参照してください。

- **WSDL アクセス引数 :** httpNonProxyHosts、httpProxyHost、httpProxyPort

  これらの引数の詳細は、18-63 ページの「WSDL アクセス引数」を参照してください。

- **WSDL 管理引数 :** createOneWayOperations、genQos、qualifiedElementForm、soapVersion、targetNamespace、typeNamespace

  これらの引数の詳細は、18-64 ページの「WSDL 管理引数」を参照してください。

Ant タスクのサポート :

- `<proxy>` タグ。詳細は、18-77 ページの「Ant タスクでのプロキシ生成の構成方法」を参照してください。
- `<port>` タグ。`<port>` タグでは、`corbaAssemble` は、引数 bindingName、`name`（`portName` と同じ）、portName、sendConnectionFactoryLocation、sendQueueLocation、soapVersion および uri を使用できます。詳細は、18-78 ページの「Ant タスクでのポートの構成方法」を参照してください。
- `<handler>` タグ。詳細は、18-81 ページの「Ant タスクでのハンドラの構成方法」を参照してください。

dbJavaAssemble

dbJavaAssemble コマンドは、Oracle データベースにある Java VM の Java クラスから Web サービスを生成する場合に使用します。このコマンドを使用するには、データベースに接続する必要があります。

dbJavaAssemble コマンドでは、Java VM の Java クラス用 WSIF バインディングを WSDL に追加することもできます。データベース・リソースを Web サービスとして公開しているときに、WSIF バインディングを WSDL に追加するには、wsifDbBinding 引数を使用します。リソースの Oracle JPublisher 生成 Java クラスの className とデータベース接続を指定する必要があります。

関連資料 :

- dbJavaAssemble コマンドを使用して Oracle データベースにある Java VM の Java クラスから Web サービスを生成する方法の詳細は、10-32 ページの「サーバー・サイド Java クラスを Web サービスとしてアセンブルする方法」を参照してください。
- データベースに接続できる引数の詳細は、18-75 ページの「データベース接続の確立方法」を参照してください。
- データベース・リソース用のバインディングの WSDL への追加については、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「データベース・リソースへの WSIF エンドポイントの構成」を参照してください。

コマンドラインの例 :

```
java -jar wsa.jar -dbJavaAssemble
                  -dbJavaClassName oracle.sqlj.checker.JdbcVersion
                  -dbUser scott/tiger
                  -dbConnection jdbc:oracle:thin:@dsunrde22:1521:sqlj
                  -dataSource jdbc/OracleManagedDS
                  -appName javacallin
```

Ant タスクの例 :

```
<oracle:dbJavaAssemble
     dbUser="scott/tiger"
     dbJavaClassName="oracle.sqlj.checker.JdbcVersion"
     dbConnection="jdbc:oracle:thin:@dsunrde22:1521:sqlj"
     dataSource="jdbc/OracleManagedDS"
     appName="javacallin"
/>
```

必須引数 :

dbJavaAssemble コマンドを使用する際には、次の引数の指定が必須です。

- **データベース・アセンブリ引数 :** dbJavaClassName

  dbJavaClassName 引数は、Web サービスとして公開するデータベース内の Java クラスを指定します。

  データベースへの接続も必要です。それには、次の引数の組合せのいずれかを使用します。

  - dataSource、dbConnection および dbUser
  - dataSource

すべての引数 :

次のリストは、`dbJavaAssemble` コマンドで使用できるすべての引数を示しています。

- **データベース・アセンブリ引数 :** dataSource、dbConnection、dbJavaClassName（必須）、dbUser, jpubProp、sysUser、useDataSource, wsifDbBinding、wsifDbPort

  これらの引数の詳細は、18-52 ページの「データベース・アセンブリ引数」を参照してください。

- **デプロイメント・ディスクリプタ引数 :** appendToExistingDDs、context、ddFileName、uri

  これらの引数の詳細は、18-61 ページの「デプロイメント・ディスクリプタ引数」を参照してください。

- **アセンブリ用の一般引数 :** appName、className、classpath、debug、ear、emptySoapAction、help、interfaceName、mappingFileName、output、packageName、portName、restSupport、schema、serviceName、useDimeEncoding、war

  これらの引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

- **Java 生成引数 :** valueTypeClassName

  この引数の詳細は、18-68 ページの「Java 生成引数」を参照してください。

- **メッセージ書式引数 :** style、use

  これらの引数の詳細は、18-67 ページの「メッセージ書式引数」を参照してください。

- **セッション引数 :** timeout

  この引数の詳細は、18-50 ページの「セッション引数」を参照してください。

- **WSDL アクセス引数 :** httpNonProxyHosts、httpProxyHost、httpProxyPort

  これらの引数の詳細は、18-63 ページの「WSDL アクセス引数」を参照してください。

- **WSDL 管理引数 :** createOneWayOperations、genQos、qualifiedElementForm、soapVersion、targetNamespace、typeNamespace

  これらの引数の詳細は、18-64 ページの「WSDL 管理引数」を参照してください。

Ant タスクのサポート :

- `<proxy>` タグ。詳細は、18-77 ページの「Ant タスクでのプロキシ生成の構成方法」を参照してください。
- `<port>` タグ。`<port>` タグでは、`dbJavaAssemble` は、引数 bindingName、portName（または `name`）、sendConnectionFactoryLocation、sendQueueLocation、soapVersion および uri を使用できます。詳細は、18-78 ページの「Ant タスクでのポートの構成方法」を参照してください。
- `<handler>` タグ。詳細は、18-81 ページの「Ant タスクでのハンドラの構成方法」を参照してください。

ejbAssemble

ejbAssemble コマンドは、EJB を Web サービスとして公開できる EAR または EJB JAR を作成する場合に使用します。バージョン 2.1 EJB の、有効な JAR を入力として指定する必要があります。システムでは、WSDL と固有の oracle-webservices.xml デプロイメント・ディスクリプタが作成されます。

デフォルトでは、このコマンドにより、直接デプロイできるバージョン 2.1 EJB ファイルが入った EAR ファイルが作成されます。oracle-webservices.xml ファイルには、Web サービスとしての EJB へのアクセスに使用できるコンテキストと URL パターンが指定されています。

EJB を EAR ファイルとしてデプロイしない場合は、かわりに EJB JAR ファイルを作成できます。この場合、たとえば、EJB を JAR ファイルとして J2EE コンテナにデプロイすることも、Ant などのその他の J2EE アプリケーション・デプロイ・ツールを使用することもできます。EJB を JAR ファイルとして保存するには、ear 引数にディレクトリを指定します。

WSIF バインディングの WSDL へのアセンブル

ejbAssemble コマンドでは、WSIF バインディングを WSDL に追加することもできます。EJB を Web サービスとして公開しているときに、WSIF バインディングを追加するには、wsifEjbBinding 引数を使用します。className 引数に EJB のホーム・インタフェースを、jndiName 引数にその JNDI 名を、それぞれ指定する必要があります。

注意： ejbAssemble コマンドまたは WebServicesAssembler を使用して、バージョン 3.0 の EJB から Web サービスをアセンブルすることはできません。これを行うには、J2SE 5.0 注釈を使用する必要があります。詳細は、11-12 ページの「バージョン 3.0 の EJB から Web サービスをアセンブルするために注釈を使用する手順」を参照してください。

関連資料：

- このパラメータを指定する様々な方法については、18-42 ページの「ear」を参照してください。
- WSIF の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービス起動フレームワークの使用方法」を参照してください。
- 個々のポートおよび複数のポートに対する WSIF バインディングの追加の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「EJB への WSIF エンドポイントの構成」を参照してください。
- ejbAssemble コマンドを使用してバージョン 2.1 EJB を Web サービスとして公開する方法の詳細は、第 8 章「EJB を使用した Web サービスのアセンブル」を参照してください。

次のコマンドラインと Ant タスクの例では、EAR ファイル build/myService.ear を作成しています。

コマンドラインの例：

```
java -jar wsa.jar -ejbAssemble
                  -output build
                  -input myEjb.jar
                  -ejbName myEjb
                  -appName myService
```

Ant タスクの例 :

```
<oracle:ejbAssemble output="build"
                    input="myEjb.jar"
                    ejbName="myEjb"
                    appName="myService"
                    />
```

必須引数 :

ejbAssemble コマンドを使用する際には、次の引数の指定が必須です。

- **アセンブリ用の一般引数 :** ejbName、input

すべての引数 :

次のリストは、ejbAssemble コマンドで使用できるすべての引数を示しています。

- **デプロイメント・ディスクリプタ引数 :** appendToExistingDDs、context、ddFileName、uri

  これらの引数の詳細は、18-61 ページの「デプロイメント・ディスクリプタ引数」を参照してください。

- **アセンブリ用の一般引数 :** appName、bindingName、className、classpath、debug、ear、ejbName（必須）、emptySoapAction、help、initialContextFactory、input（必須）、interfaceName、jndiName、jndiProviderURL、mappingFileName、output、portName、portTypeName、restSupport、schema、serviceName、strictJaxrpcValidation、useDimeEncoding

  これらの引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

- **Java 生成引数 :** valueTypeClassName、wsifEjbBinding

  これらの引数の詳細は、18-68 ページの「Java 生成引数」を参照してください。

- **JMS アセンブリ引数 :** sendConnectionFactoryLocation、sendQueueLocation

  これらの引数の詳細は、18-57 ページの「JMS アセンブリ引数」を参照してください。

- **メッセージ書式引数 :** style、use

  これらの引数の詳細は、18-67 ページの「メッセージ書式引数」を参照してください。

- **WSDL アクセス引数 :** httpNonProxyHosts、httpProxyHost、httpProxyPort

  これらの引数の詳細は、18-63 ページの「WSDL アクセス引数」を参照してください。

- **WSDL 管理引数 :** createOneWayOperations、genQos、qualifiedElementForm、soapVersion、targetNamespace、typeNamespace

  これらの引数の詳細は、18-64 ページの「WSDL 管理引数」を参照してください。

追加 Ant サポート :

- <proxy> タグ。詳細は、18-77 ページの「Ant タスクでのプロキシ生成の構成方法」を参照してください。
- <port> タグ。<port> タグでは、ejbAssemble は、引数 bindingName、name（portName と同じ）、portName、sendConnectionFactoryLocation、sendQueueLocation、soapVersion および uri を使用できます。詳細は、18-78 ページの「Ant タスクでのポートの構成方法」を参照してください。
- <handler> タグ。詳細は、18-81 ページの「Ant タスクでのハンドラの構成方法」を参照してください。
- 複数ポート用の WSIF バインディングを指定するタグ。詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「複数の EJB ポートに対する WSIF エンドポイントの構成」を参照してください。

jmsAssemble

jmsAssemble コマンドは、JMS 宛先（キューまたはトピック）を Web サービスとして公開する場合に使用します。JMS Web サービスには、send と receive の 2 種類の操作があります。send 操作では、JMS 宛先にメッセージが送信されます。receive 操作では、JMS 宛先からのメッセージを取得します。一部の JMS メッセージ・プロパティ（たとえば、相関 ID）は、SOAP ヘッダーとして公開できます。

関連項目：

jmsAssemble コマンドを使用して JMS 宛先を Web サービスとして公開する方法の詳細は、第 9 章「JMS 宛先を使用した Web サービスのアセンブル」を参照してください。

コマンドラインの例：

```
java -jar wsa.jar -jmsAssemble
          -sendConnectionFactoryLocation jms/ws/mdb/theQueueConnectionFactory
          -sendQueueLocation jms/ws/mdb/theQueue
          -replyToConnectionFactoryLocation jms/ws/mdb/logQueueConnectionFactory
          -replyToQueueLocation jms/ws/mdb/logQueue
          -linkReceiveWithReplyTo true
          -targetNamespace http://oracle.j2ee.ws/jms-doc
          -typeNamespace http://oracle.j2ee.ws/jms-doc/types
          -serviceName JmsService
          -appName jms_service
          -context jms_service
          -input ./demo/build/mdb_service.jar
          -uri JmsService
          -output ./demo/dist
```

Ant タスクの例：

```
<oracle:jmsAssemble
       linkReceiveWithReplyTo="true"
       targetNamespace="http://oracle.j2ee.ws/jms-doc"
       typeNamespace="http://oracle.j2ee.ws/jms-doc/types"
       serviceName="JmsService"
       appName="jms_service"
       context="jms_service"
       input="./demo/build/mdb_service.jar"
       output="./demo/dist"
       >
       <oracle:port  uri="JmsService"
          sendConnectionFactoryLocation="jms/ws/mdb/theQueueConnectionFactory"
          sendQueueLocation="jms/ws/mdb/theQueue"
          replyToConnectionFactoryLocation="jms/ws/mdb/logQueueConnectionFactory"
          replyToQueueLocation="jms/ws/mdb/logQueue"/>
</oracle:jmsAssemble>
```

必須引数：

jmsAssemble コマンドを使用する際には、次の引数の指定が必須です。

- **JMS アセンブリ引数：** replyToConnectionFactoryLocation または sendConnectionFactoryLocation

すべての引数 :

次のリストは、jmsAssemble コマンドで使用できるすべての引数を示しています。

- **デプロイメント・ディスクリプタ引数 :** appendToExistingDDs、context、ddFileName、uri

  これらの引数の詳細は、18-61 ページの「デプロイメント・ディスクリプタ引数」を参照してください。

- **アセンブリ用の一般引数 :** appName、bindingName、debug、ear、emptySoapAction、help、input、output、portName、portTypeName、serviceName、useDimeEncoding、war

  これらの引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

- **JMS アセンブリ引数 :** deliveryMode、genJmsPropertyHeader、jmsTypeHeader、linkReceiveWithReplyTo、payloadBindingClassName、priority、receiveConnectionFactoryLocation、receiveQueueLocation、receiveTimeout、receiveTopicLocation、replyToConnectionFactoryLocation、replyToQueueLocation、replyToTopicLocation、sendConnectionFactoryLocation、sendQueueLocation、sendTopicLocation、timeToLive、topicDurableSubscriptionName

  これらの引数の詳細は、18-57 ページの「JMS アセンブリ引数」を参照してください。

- **WSDL 管理引数 :** createOneWayOperations、genQos、qualifiedElementForm、soapVersion、targetNamespace、typeNamespace

  これらの引数の詳細は、18-64 ページの「WSDL 管理引数」を参照してください。

Ant タスクのサポート :

- <proxy> タグ。詳細は、18-77 ページの「Ant タスクでのプロキシ生成の構成方法」を参照してください。
- <port> タグ。<port> タグでは、jmsAssemble は、引数 bindingName、name（portName と同じ）、portName、sendConnectionFactoryLocation、sendQueueLocation、soapVersion および uri を使用できます。詳細は、18-78 ページの「Ant タスクでのポートの構成方法」を参照してください。
- <handler> タグ。詳細は、18-81 ページの「Ant タスクでのハンドラの構成方法」を参照してください。

plsqlAssemble

plsqlAssemble コマンドは、ストアド・プロシージャおよびファンクションを含む PL/SQL パッケージから Web サービスを生成する場合に使用します。このコマンドを使用するには、データベースに接続する必要があります。

関連項目 :

- データベースに接続できる引数の詳細は、18-75 ページの「データベース接続の確立方法」を参照してください。
- plsqlAssemble コマンドを使用して PL/SQL パッケージを Web サービスとして公開する方法の詳細は、10-9 ページの「PL/SQL パッケージからの Web サービスのアセンブル方法」を参照してください。

WSIF バインディングの WSDL へのアセンブル

plsqlAssemble コマンドは、PL/SQL パッケージに対する WSIF バインディングを WSDL に追加することもできます。データベース・リソースを Web サービスとして公開しているときに、WSIF バインディングを WSDL に追加するには、wsifDbBinding 引数を使用します。リソースの Oracle JPublisher 生成 Java クラスの className とデータベース接続を指定する必要があります。

関連資料 :

データベース・リソース用のバインディングの WSDL への追加については、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「データベース・リソースへの WSIF エンドポイントの構成」を参照してください。

コマンドラインの例 :

```
java -jar wsa.jar -plsqlAssemble
                  -appName query
                  -dbUser scott/tiger
                  -sql Company
                  -dbConnection jdbc:oracle:thin:@dsunrde22:1521:sqlj
                  -dataSource jdbc/OracleManagedDS
```

Ant タスクの例 :

```
<oracle:plsqlAssemble
     dbUser="scott/tiger"
     appName="query"
     sql="Company"
     dbConnection="jdbc:oracle:thin:@dsunrde22:1521:sqlj"
     dataSource="jdbc/OracleManagedDS"
/>
```

必須引数 :

plsqlAssemble コマンドを使用する際には、次の引数の指定が必須です。

- **データベース・アセンブリ引数 :** sql

  sql 引数は、Web サービスとして公開する PL/SQL パッケージの名前を指定します。

  データベースへの接続も必要です。それには、次の引数の組合せのいずれかを使用します。

  - dataSource、dbConnection および dbUser
  - dataSource

すべての引数 :

次のリストは、plsqlAssemble コマンドで使用できるすべての引数を示しています。

- **データベース・アセンブリ引数 :** dataSource、dbConnection、dbUser、jpubProp、sql、sysUser、useDataSource、wsifDbBinding、wsifDbPort

  これらの引数の詳細は、18-52 ページの「データベース・アセンブリ引数」を参照してください。

- **デプロイメント・ディスクリプタ引数 :** appendToExistingDDs、context、ddFileName、uri

  これらの引数の詳細は、18-61 ページの「デプロイメント・ディスクリプタ引数」を参照してください。

- **アセンブリ用の一般引数 :** appName、className、classpath、debug、ear、emptySoapAction、help、interfaceName、mappingFileName、output、packageName、portName、restSupport、schema、serviceName、useDimeEncoding、war

  これらの引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

- **Java 生成引数 :** valueTypeClassName

  この引数の詳細は、18-68 ページの「Java 生成引数」を参照してください。

- **メッセージ書式引数 :** style、use

  これらの引数の詳細は、18-67 ページの「メッセージ書式引数」を参照してください。

- **セッション引数 :** timeout

  この引数の詳細は、18-50 ページの「セッション引数」を参照してください。

- **WSDL アクセス引数 :** httpNonProxyHosts、httpProxyHost、httpProxyPort

  これらの引数の詳細は、18-63 ページの「WSDL アクセス引数」を参照してください。

- **WSDL 管理引数 :** createOneWayOperations、genQos、qualifiedElementForm、soapVersion、targetNamespace、typeNamespace

  これらの引数の詳細は、18-64 ページの「WSDL 管理引数」を参照してください。

Ant タスクのサポート :

- `<proxy>` タグ。詳細は、18-77 ページの「Ant タスクでのプロキシ生成の構成方法」を参照してください。
- `<port>` タグ。`<port>` タグでは、`plsqlAssemble` は、引数 bindingName、`name`（`portName` と同じ）、portName、sendConnectionFactoryLocation、sendQueueLocation、soapVersion および uri を使用できます。詳細は、18-78 ページの「Ant タスクでのポートの構成方法」を参照してください。
- `<handler>` タグ。詳細は、18-81 ページの「Ant タスクでのハンドラの構成方法」を参照してください。

sqlAssemble

`sqlAssemble` コマンドは、SQL 問合せやデータ操作言語（DML）などの SQL 文から Web サービスを生成する場合に使用します。このコマンドを使用するには、データベースに接続する必要があります。

関連項目 :

- SQL 文からの Web サービスの生成方法の完全な例は、10-15 ページの「SQL 問合せまたは DML 文から Web サービスをアセンブルする方法」を参照してください。
- データベースに接続できる引数の詳細は、18-75 ページの「データベース接続の確立方法」を参照してください。

WSDL への WSIF バインディングの追加

`sqlAssemble` コマンドでは、SQL 問合せ用 WSIF バインディングを WSDL に追加することもできます。データベース・リソースを Web サービスとして公開しているときに、WSIF バインディングを WSDL に追加するには、`wsifDbBinding` 引数を使用します。リソースの Oracle JPublisher 生成 Java クラスの `className` とデータベース接続を指定する必要があります。

関連項目 :

データベース・リソース用のバインディングの WSDL への追加については、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「データベース・リソースへの WSIF エンドポイントの構成」を参照してください。

次のコマンドラインと Ant タスクの例では、データベース接続を確立して 2 つの SQL コマンドを実行しています。

コマンドラインの例 :

```
java -jar wsa.jar -sqlAssemble
  -dbUser scott/tiger
  -sqlstatement "getEmp=select ename, sal from emp where empno=:{id NUMBER}"
  -sqlstatement "getEmpBySal=select ename, sal from emp where sal>:{mysal NUMBER}"
  -dbConnection jdbc:oracle:thin:@dsunrde22:1521:sqlj
  -dataSource jdbc/OracleManagedDS
```

Ant タスクの例 :

```
<oracle:sqlAssemble
   dbUser="scott/tiger"
   dbConnection="jdbc:oracle:thin:@dsunrde22:1521:sqlj"
   dataSource="jdbc/OracleManagedDS"
   appName="query">
       <sqlstatement value="getEmp=select ename, sal from emp where empno=:{id
NUMBER}"/>
       <sqlstatement value="getEmpBySal=select ename, sal from emp where sal>:{mysal
NUMBER}"/>
</oracle:sqlAssemble>
```

必須引数 :

`sqlAssemble` コマンドを使用する際には、次の引数の指定が必須です。

- **データベース・アセンブリ引数 :** sqlstatement

  データベースへの接続も必要です。それには、次の引数の組合せのいずれかを使用します。

  - dataSource、dbConnection および dbUser
  - dataSource

すべての引数 :

次のリストは、`sqlAssemble` コマンドで使用できるすべての引数を示しています。

- **データベース・アセンブリ引数 :** dataSource、dbConnection、dbUser、jpubProp、sql、sqlstatement、sysUser、useDataSource、wsifDbBinding、wsifDbPort

  これらの引数の詳細は、18-52 ページの「データベース・アセンブリ引数」を参照してください。

- **デプロイメント・ディスクリプタ引数 :** appendToExistingDDs、context、ddFileName、uri

  これらの引数の詳細は、18-61 ページの「デプロイメント・ディスクリプタ引数」を参照してください。

- **アセンブリ用の一般引数 :** appName、className、classpath、debug、ear、emptySoapAction、help、interfaceName、mappingFileName、output、packageName、portName、restSupport、schema、serviceName、useDimeEncoding、war

  これらの引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

- **Java 生成引数 :** valueTypeClassName

  この引数の詳細は、18-68 ページの「Java 生成引数」を参照してください。

- **メッセージ書式引数 :** style、use

  これらの引数の詳細は、18-67 ページの「メッセージ書式引数」を参照してください。

- **セッション引数 :** timeout

  この引数の詳細は、18-50 ページの「セッション引数」を参照してください。

- **WSDL アクセス引数 :** httpNonProxyHosts、httpProxyHost、httpProxyPort

  これらの引数の詳細は、18-63 ページの「WSDL アクセス引数」を参照してください。

- **WSDL 管理引数 :** createOneWayOperations、genQos、qualifiedElementForm、soapVersion、targetNamespace、typeNamespace

  これらの引数の詳細は、18-64 ページの「WSDL 管理引数」を参照してください。

Ant タスクのサポート :

- <proxy> タグ。詳細は、18-77 ページの「Ant タスクでのプロキシ生成の構成方法」を参照してください。
- <port> タグ。<port> タグでは、sqlAssemble は、引数 bindingName、name（portName と同じ）、portName、sendConnectionFactoryLocation、sendQueueLocation、soapVersion および uri を使用できます。詳細は、18-78 ページの「Ant タスクでのポートの構成方法」を参照してください。
- <handler> タグ。詳細は、18-81 ページの「Ant タスクでのハンドラの構成方法」を参照してください。

topDownAssemble

topDownAssemble コマンドは、WSDL 記述をベースにした Web サービスで必要となるクラスとデプロイメント・ディスクリプタを作成する場合に使用します。これらのファイルは、EAR ファイル、WAR ファイル、ディレクトリのいずれにも格納できます。input または classpath 引数の値を指定して、指定した実装クラスが適切にロードされるようにする必要があります。

このコマンドは、通常 genInterface とともに使用され、トップダウン方式で Web サービスを生成します。これらのコマンドを一緒に使用して Web サービスを生成するに当たっては、これらのコマンドの unwrapParameters 引数には同じ値を指定する必要があります。

実装可能なサービス要素は 1 つのみです。WebServicesAssembler では、一度に 1 つのサービスに対してのみアーチファクトを生成できます。複数のサービスが WSDL で記述されている場合は、コマンドライン引数 serviceName を使用して、使用対象のサービスを指定できます。

生成されたクラスの名前の間に不整合がある場合は、JAX-RPC の名前衝突に関する 4.3.12 項で記述されているように、JAX-RPC のルールに従って解決されます。詳細は、18-71 ページの「名前競合の解決」を参照してください。

関連項目 :

topDownAssemble コマンドを使用して WSDL 記述をベースにして Web サービスを生成する方法の詳細は、第 6 章「WSDL からの Web サービスのアセンブル」を参照してください。

WSDL の制限事項

次のリストは、WebServicesAssembler が使用できる WSDL についての制限事項です。

- WSDL に複数のポート・タイプの参照が含まれている場合、`topDownAssemble` コマンドでは、ポート・タイプごとに `<porttype>` タグを指定する必要があります。
- サービス、ポート・タイプ、操作、バインディングまたはポートの名前など、WSDL の名前での National Language Support（「NLS」または「グローバリゼーション・サポート」とも呼ばれます）文字の使用は、サポートされていません。使用すると、Web サービス・テスト・ページでエラーが発生する場合もあります。
- トップダウン方式での Web サービスの開発では、WebServicesAssembler を使用してメッセージ書式を変更することはできません。これを行うには、WSDL を編集する必要があります。
- WSDL に複数のメッセージ書式を含めることはできません。使用しないメッセージ書式があるバインディングを参照するポートは、WSDL から削除してください。
- WebServicesAssembler は、`xsd:choice` または `xsd:group` の XML 型を含む WSDL を使用できません。これらの XML 型を含む WSDL を使用する場合は、WebServicesAssembler の `dataBinding` 引数を `false` に設定し、WSDL ファイルでのスキーマ定義にペイロードが準拠するように `SOAPElement` をコーディングする必要があります。

次のコマンドラインと Ant タスクでは、Web サービスに必要なクラスとデプロイメント・ディスクリプタを作成しています。`build/myService.ear` という EAR が作成されます。`input` 引数のターゲットには実装クラスが含まれています。これらのクラスは、生成されるアーカイブにコピーされます。

コマンドラインの例 :

```
java -jar wsa.jar -topDownAssemble
                  -output build
                  -wsdl my.wsdl
                  -input myClasses
                  -className com.mycompany.HelloImpl
                  -appName myService
```

Ant タスクの例 :

```
<oracle:topDownAssemble output="build"
                  wsdl="my.wsdl"
                  input="myClasses"
                  appName="myService">
         <porttype
           className="com.mycompany.HelloImpl"/>
         </oracle:topDownAssemble>
```

必須引数 :

`topDownAssemble` コマンドを使用する際には、次の引数の指定が必須です。

- **アセンブリ用の一般引数 :** className（input または classpath あるいはその両方）
- **WSDL 管理引数 :** wsdl

すべての引数 :

次のリストは、`topDownAssemble` コマンドで使用できるすべての引数を示しています。

- **デプロイメント・ディスクリプタ引数 :** appendToExistingDDs、context、ddFileName、uri

  これらの引数の詳細は、18-61 ページの「デプロイメント・ディスクリプタ引数」を参照してください。

- **アセンブリ用の一般引数 :** appName、classFileName、className（必須）、classpath、debug、ear、emptySoapAction、help、input、interfaceName、mappingFileName、output、packageName、portName、restSupport、searchSchema、serviceName、useDimeEncoding、war

  これらの引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

- **Java 生成引数 :** dataBinding、mapHeadersToParameters、overwriteBeans、unwrapParameters、valueTypePackagePrefix

  これらの引数の詳細は、18-68 ページの「Java 生成引数」を参照してください。

- **メッセージ書式引数 :** mtomSupport

  この引数の詳細は、18-67 ページの「メッセージ書式引数」を参照してください。

- **WSDL アクセス引数 :** fetchWsdlImports、httpNonProxyHosts、httpProxyHost、httpProxyPort、wsdl（必須）

  これらの引数の詳細は、18-63 ページの「WSDL アクセス引数」を参照してください。

- **WSDL 管理引数 :** wsdlTimeout

  この引数の詳細は、18-64 ページの「WSDL 管理引数」を参照してください。

追加 Ant サポート :

- `<proxy>` タグ。詳細は、18-77 ページの「Ant タスクでのプロキシ生成の構成方法」を参照してください。
- `<port>` タグ。`<port>` タグでは、`topDownAssemble` は、引数 `name`（`portName` と同じ）、portName および uri を使用できます。詳細は、18-78 ページの「Ant タスクでのポートの構成方法」を参照してください。
- `<porttype>` タグ。詳細は、18-80 ページの「Ant タスクでのポート・タイプの構成方法」を参照してください。
- `<handler>` タグ。詳細は、18-81 ページの「Ant タスクでのハンドラの構成方法」を参照してください。

WSDL 管理コマンド

次の各コマンドは、WSDL に対するアクションを実行します。fetchWsdl および genQosWsdl コマンドは、トップダウン方式での Web サービス開発に使用され、WSDL の内容と位置を管理します。genWsdl コマンドは、ボトムアップ方式での Web サービス開発での WSDL の生成に使用されます。analyze コマンドは、WSDL に記述されている機能が WebServicesAssembler によってサポートされているかどうかを判別するためにいつでも使用できます。

- analyze: WSDL に記述されている機能が WebServicesAssembler によってサポートされているかどうかを判別する。
- fetchWsdl: WSDL とそのインポートを出力ディレクトリにコピーする。
- genConcreteWsdl: WSDL の抽象部分からメッセージ書式を判別して、具体的な WSDL を作成する。
- genQosWsdl: サービスのクオリティに関するアサーション（機能アサーション）を WSDL に挿入する。
- genWsdl: Java インタフェースをベースにして WSDL を生成する。

analyze

analyze コマンドは、このバージョンの WebServicesAssembler によって WSDL を処理できるかどうかを確認する場合に使用します。analyze コマンドにより、指定された WSDL を使用してプロキシを生成できるかまたはトップダウン方式でのアセンブリ用のインタフェースを作成できるかどうかを判別します。また、このコマンドにより、WSDL が有効な XML を使用しているか、OC4J の JAX-RPC 要件に準拠しているかどうかをチェックします。

このコマンドは、WSDL を処理できない場合は、メッセージを戻します。

> **注意：** analyze コマンドでは、Web Services Interoperability（WS-I）仕様や WSDL の一般的な相互運用性への準拠はチェックされません。WS-I Analyzer ツールを直接または Oracle JDeveloper 内部から使用して、準拠をチェックすることが必要になる場合があります。相互運用性のチェックに使用できるツールの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「相互運用可能な Web サービスの実現」を参照してください。

次のコマンドラインと Ant タスクの例では、analyze を使用して、指定された WSDL を処理するかどうかを確認しています。

コマンドラインの例：

```
java -jar wsa.jar -analyze
                  -wsdl myservice.wsdl
```

Ant タスクの例：

```
<oracle:analyze wsdl="myservice.wsdl"
         />
```

必須引数：

analyze コマンドを使用する際には、次の引数の指定が必須です。

- **WSDL アクセス引数：** wsdl

すべての引数 :
次のリストは、analyze コマンドで使用できるすべての引数を示しています。

- **アセンブリ用の一般引数 :** debug、help

  これらの引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

- **WSDL アクセス引数 :** httpNonProxyHosts、httpProxyHost、httpProxyPort、wsdl（必須）

  この引数の詳細は、18-63 ページの「WSDL アクセス引数」を参照してください。

- **WSDL 管理引数 :** wsdlTimeout

  この引数の詳細は、18-64 ページの「WSDL 管理引数」を参照してください。

追加 Ant サポート :
なし

fetchWsdl

fetchWsdl コマンドは、トップダウン方式の Web サービスの生成において、基底（またはトップレベル）の WSDL ファイルと、Web サービスのインポート済 / 組込み済の WSDL とスキーマを、指定された出力ディレクトリにコピーする場合に使用します。

WSDL とスキーマはすべて同じディレクトリにダウンロードされます。ネーミングの競合がある場合でも、ファイル名の拡張子の前に数字を追加することで解決されます。たとえば、3 つの myschema.xsd ファイルをダウンロードする場合は、myschema.xsd、myschema1.xsd および myschema2.xsd とネーミングされます。

次のコマンドラインと Ant タスクの例では、URL に指定された基底 WSDL と、それにインポートされたその他の WSDL フラグメントとスキーマがフェッチされています。次に、その結果が wsdl ディレクトリに格納されます。

コマンドラインの例 :

```
java -jar wsa.jar -fetchWdsl
                  -wsdl http://someserver/services/aservice?WSDL
                  -output wsdl
```

Ant タスクの例 :

```
<oracle:fetchWsdl wsdl="http://someserver/services/aservice?WSDL"
          output="wsdl"
          />
```

必須引数 :
fetchWsdl コマンドを使用する際には、次の引数の指定が必須です。

- **WSDL アクセス引数 :** wsdl

すべての引数 :
次のリストは、fetchWsdl コマンドで使用できるすべての引数を示しています。

- **アセンブリ用の一般引数 :** debug、help、output

  これらの引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

- **WSDL アクセス引数 :** httpNonProxyHosts、httpProxyHost、httpProxyPort、wsdl（必須）

  これらの引数の詳細は、18-63 ページの「WSDL アクセス引数」を参照してください。

- **WSDL 管理引数 :** wsdlTimeout

  この引数の詳細は、18-64 ページの「WSDL 管理引数」を参照してください。

追加 Ant サポート :
なし

genConcreteWsdl

抽象 WSDL は Web サービスの API を定義するには十分ですが、WebServicesAssembler では、Web サービスをデプロイする場合や、Web サービスと通信できるクライアント・プロキシを生成する場合には、具体的 WSDL を必要とします。

抽象 WSDL しかない場合は、`genConcreteWsdl` コマンドを使用します。トップダウン方式の Web サービス開発では、抽象 WSDL があれば、このコマンドにより具体的 WSDL を生成できます。この処理は、このコマンド内で次のように行われます。すなわち、WSDL の `wsdl:portType` 部分を分析して Web サービスのバインディング（すなわち、`use` および `style` の値）が document/literal または RPC/literal であるかを判別します。このコマンドにより、これらの値が WSDL の `binding` 要素に書き込まれ、`output` 引数の値に設定した名前で保存されます。

次のコマンドラインと Ant タスクの例では、抽象 WSDL の `myAbstract.wsdl` を入力として取り、`outputDir` ディレクトリに具体的 WSDL `myConcrete.wsdl` を生成しています。

コマンドラインの例 :

```
java -jar wsa.jar -genConcreteWsdl
                  -output outputDir/myConcrete.wsdl
                  -wsdl myAbstract.wsdl
```

Ant タスクの例 :

```
<oracle:genConcreteWsdl output="outputDir/myConcrete.wsdl"
                 wsdl="myAbstract.wsdl" />
```

必須引数 :
`genConcreteWsdl` コマンドを使用する際には、次の引数の指定が必須です。

- **WSDL アクセス引数 :** wsdl

すべての引数 :
次のリストは、`genConcreteWsdl` コマンドで使用できるすべての引数を示しています。

- **デプロイメント・ディスクリプタ引数 :** ddFileName

  この引数の詳細は、18-61 ページの「デプロイメント・ディスクリプタ引数」を参照してください。

- **アセンブリ用の一般引数 :** debug、help、output

  これらの引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

- **WSDL アクセス引数 :** httpNonProxyHosts、httpProxyHost、httpProxyPort、importAbstractWsdl、wsdl（必須）

  これらの引数の詳細は、18-63 ページの「WSDL アクセス引数」を参照してください。

- **WSDL 管理引数 :** singleService、wsdlTimeout

  これらの引数の詳細は、18-64 ページの「WSDL 管理引数」を参照してください。

追加 Ant サポート :
なし

genQosWsdl

トップダウン方式の Web サービス開発では、genQosWsdl コマンドを使用して、セキュリティおよび信頼性の機能アサーションを、指定した WSDL に追加できます。機能アサーションとは Web サービス管理ポリシーの記述で、Web サービスのコンシューマはこれを使用してその Web サービスで有効な管理ポリシーを知ることができます。

機能アサーションは、通常デプロイメント・ディスクリプタに定義されます。ddFileName 引数を使用して機能アサーションを含むファイルを指定し、wsdl 引数を使用して機能アサーションの挿入先となる WSDL の名前を指定します。output 引数には、変更された WSDL ファイルが格納される場所を指定します。output 引数を指定しない場合は、元の WSDL が上書きされます。

> **関連資料 :**
>
> 機能アサーションを取得する方法とそれを WSDL に挿入する方法については、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「機能アサーションの使用」を参照してください。

次のコマンドラインと Ant タスクの例では、my.wsdl にアサーションを追加し、その結果を build ディレクトリに格納しています。

コマンドラインの例 :

```
java -jar wsa.jar -genQosWsdl
                  -wsdl my.wsdl
                  -ddFileName oracle-webservices.xml
                  -output build
```

Ant タスクの例 :

```
<oracle:genQosWsdl wsdl="my.wsdl"
            ddFileName="oracle-webservices.xml"
            output="build"
            />
```

必須引数 :

genQosWsdl コマンドを使用する際には、次の引数の指定が必須です。

- **デプロイメント・ディスクリプタ引数 :** ddFileName
- **WSDL 管理引数 :** wsdl

すべての引数 :

次のリストは、genQosWsdl コマンドで使用できるすべての引数を示しています。

- **デプロイメント・ディスクリプタ引数 :** ddFileName（必須）

  この引数の詳細は、18-61 ページの「デプロイメント・ディスクリプタ引数」を参照してください。

- **アセンブリ用の一般引数 :** debug、help、output

  これらの引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

- **WSDL アクセス引数 :** httpNonProxyHosts、httpProxyHost、httpProxyPort、wsdl（必須）

  これらの引数の詳細は、18-63 ページの「WSDL アクセス引数」を参照してください。

- **WSDL 管理引数 :** wsdlTimeout

  この引数の詳細は、18-64 ページの「WSDL 管理引数」を参照してください。

追加 Ant サポート :

なし

genWsdl

genWsdl コマンドは、Java インタフェースからボトムアップ方式で Web サービスをアセンブルするために WSDL と JAX-RPC マッピング・ファイルを生成する場合に使用します。このコマンドでは、interfaceName または className のいずれかの引数が必須です。これらの引数は注釈付きの Java クラスを指して、WSDL にサービス・エンドポイント・インタフェースの値を供給します。

次に、genWsdl コマンドの例を示します。interfaceName 引数には oracle.j2ee.demo.HelloIntf インタフェースが指定されています。

```
java -jar wsa.jar genWSDL -interfaceName oracle.j2ee.demo.HelloIntf -output wsdl
-classpath classes
```

WSIF バインディングのある WSDL の生成

genWsdl コマンドでは、次の引数を使用して、生成される WSDL に WSIF バインディングを追加できます。

- wsifJavaBinding 引数 : Java クラスを Web サービスとして公開しているときに、WSDL に WSIF バインディングを追加します。また、className 引数に Java クラスを指定する必要もあります。
- wsifEjbBinding 引数 : EJB を Web サービスとして公開しているときに、WSDL に WSIF バインディングを追加します。className 引数に EJB のホーム・インタフェースを、jndiName 引数にその JNDI 名を、それぞれ指定する必要があります。
- wsifDbBinding 引数 : データベース・リソースを Web サービスとして公開しているときに、WSDL に WSIF バインディングを追加します。リソースの Oracle JPublisher 生成 Java クラスの className とデータベース接続を指定する必要があります。

genWsdl コマンドを使用して、WSDL の複数のポート用に WSIF バインディングを追加することもできます。

> **関連資料 :**
>
> WSIF の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービス起動フレームワークの使用方法」を参照してください。

J2SE 5.0 注釈とともに使用できる WSDL の生成

J2SE 5.0 注釈を使用して WSDL を生成する場合は、interfaceName 引数ではなく className 引数を使用します。className 引数に実装クラスを指定し、このクラスの宣言に @WebService 注釈を指定する必要があります。

指定した className に J2SE 5.0 注釈が含まれていない場合は、interfaceName 引数を使用します。

J2SE 5.0 注釈にはクラス名が必要です。注釈がインタフェースと実装クラスのいずれかに付けられる可能性があるためです。注釈がインタフェースのみに付けられている場合は、WebServicesAssembler は、インタフェースが参照する実装クラスを介して注釈を取得できます。実装クラスにも注釈が付いている場合は、この注釈が WebServicesAssembler によって処理されます。

次の例では、genWsdl コマンドを使用して、J2SE 5.0 注釈とともに使用できる WSDL を生成しています。className 引数には、oracle.j2ee.demo.HelloImpl クラスが指定されています。

```
java -jar wsa.jar genWsdl -className oracle.j2ee.demo.HelloImpl -output wsdl -classpath
classes
```

WebServicesAssembler にインタフェースの J2SE 5.0 注釈のみを処理させる場合は、serviceEndpointInterface プロパティ付きの @WebService 注釈を実装クラス・ファイルに入力します。J2SE 5.0 注釈では、残りすべての注釈もこのサービス・エンドポイント・インタフェース・クラスに付けられているものと予期します。たとえば、実装クラス・ファイル

に次の注釈を入力すると、WebServicesAssembler では demo.myInterface インタフェースにある注釈のみが処理されます。

```
@WebService(serviceEndpointInterface="demo.myInterface")
```

次のコマンドラインと Ant タスクの例では、JAX-RPC マッピング・ファイルと、interfaceName に指定された Java インタフェースに相当する WSDL を出力しています。その結果は etc ディレクトリに格納されます。

コマンドラインの例 :

```
java -jar wsa.jar -genWsdl
                  -classpath myservice.jar
                  -output etc
                  -interfaceName com.mycompany.myservice.Hello
```

Ant タスクの例 :

```
<oracle:genWsdl output="etc"
    >
    <oracle:porttype interfaceName="com.mycompany.myservice.Hello"/>
    <oracle:classpath>
        <pathelement path="myservice.jar" />
    </oracle:classpath>
</oracle:genWsdl>
```

必須引数 :

genWsdl コマンドを使用する際には、次の引数の指定が必須です。

- **アセンブリ用の一般引数 :** classpath。J2SE 5.0 注釈を使用して WSDL を生成する場合、genWsdl には、注釈付きの Java クラスを指す interfaceName または className の指定が必須です。

すべての引数 :

次のリストは、genWsdl コマンドで使用できるすべての引数を示しています。

- **データベース・アセンブリ引数 :** dataSource、dbConnection、dbUser、wsifDbBinding、wsifDbPort

  これらの引数の詳細は、18-52 ページの「データベース・アセンブリ引数」を参照してください。

- **デプロイメント・ディスクリプタ引数 :** ddFileName

  この引数の詳細は、18-61 ページの「デプロイメント・ディスクリプタ引数」を参照してください。

- **アセンブリ用の一般引数 :** bindingName、className（J2SE 5.0 注釈に必須）、classpath（必須）、debug、emptySoapAction、help、initialContextFactory、interfaceName（J2SE 5.0 注釈に必須）、jndiName、jndiProviderURL、mappingFileName、output、portName、portTypeName、schema、serviceName、strictJaxrpcValidation

  これらの引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

- **Java 生成引数 :** valueTypeClassName、wsifEjbBinding、wsifJavaBinding

  これらの引数の詳細は、18-68 ページの「Java 生成引数」を参照してください。

- **JMS 引数 :** sendConnectionFactoryLocation、sendQueueLocation

  これらの引数の詳細は、18-57 ページの「JMS アセンブリ引数」を参照してください。

- **メッセージ書式引数 :** style、use

  これらの引数の詳細は、18-67 ページの「メッセージ書式引数」を参照してください。

- **WSDL 管理引数 :** createOneWayOperations、genQos、qualifiedElementForm、soapVersion、targetNamespace、typeNamespace

  これらの引数の詳細は、18-64 ページの「WSDL 管理引数」を参照してください。

追加 Ant サポート :

- <port> タグ。<port> タグでは、genWsdl は、引数 bindingName、name（portName と同じ）、portName、sendConnectionFactoryLocation、sendQueueLocation、soapVersion および uri を使用できます。詳細は、18-78 ページの「Ant タスクでのポートの構成方法」を参照してください。
- <porttype> タグ。詳細は、18-80 ページの「Ant タスクでのポート・タイプの構成方法」を参照してください。
- 複数ポート用の WSIF バインディングを指定するタグ。詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「複数の Java ポートに対する WSIF エンドポイントの構成」、「複数の EJB ポートに対する WSIF エンドポイントの構成」および「複数のデータベース・リソース・ポートに対する WSIF エンドポイントの構成」を参照してください。

Java 生成コマンド

次のコマンドでは、Java インタフェース、プロキシ / スタブまたは JAX-RPC 値タイプ・クラスを作成するためのコードが生成されます。

- genInterface: WSDL ファイルからの Java インタフェースの生成
- genProxy: WSDL ファイルからのプロキシ / スタブの生成
- genValueTypes: XML スキーマからの JAX-RPC 値タイプ・クラスの生成

genInterface

トップダウン方式での Web サービス開発の場合、このコマンドにより、各ポート・タイプのサービス・エンドポイント・インタフェースと、WSDL に定義された複合型に対応する Java 値タイプ・クラス（Bean）が作成されます。また、XML スキーマ型と Java 値タイプ・クラス間のマッピングを記述する JAX-RPC マッピング・ファイルも作成されます。次に、これらのファイルを使用して、J2EE Web サービス・クライアントを構築することや、そのサーバーで実行可能な実装を作成することができます。

J2EE Web サービス・クライアントをアセンブルしていて、Oracle 固有のクライアント構成を渡す必要がある場合は、ddFileName 引数を使用します。

JAX-RPC マッピング・ファイルに加えて、genInterface コマンドは、表 18-1 にリストされているファイルを WSDL ドキュメントから生成します。リストと説明では、JAX-RPC マッピング・ファイルを使用してこれらのファイルのデフォルトの命名規則が変更されていないものと想定しています。また、生成された Java クラスの名前の間に不整合がある場合は、JAX-RPC の名前衝突に関する 4.3.12 項で記述されているように、JAX-RPC のルールに従って解決されます。詳細は、18-71 ページの「名前競合の解決」を参照してください。

注意 : これらの生成されたファイルを変更しても、WebServicesAssembler コマンドが再び起動されると上書きされる場合があるので、変更はお薦めできません。WebServicesAssembler は、次の条件のスキーマ・タイプに対して genInterface コマンドで生成されたファイルは上書きしません。

- クラス名（パッケージ名を含む）が、WebServicesAssembler コマンドに対して渡されたクラスパス引数にすでに存在する。
- かつ、コマンドライン引数 overwriteBeans が、false（デフォルト値）に設定されている。18-69 ページの「overwriteBeans」を参照してください。

表 18-1 genInterface が生成するファイル

ファイル名	説明
<*derived_name*>.java	指定されている WSDL のスキーマで定義されている型ごとに 1 つのファイルが生成されます。Java クラス・ファイルの名前 *derived_name* は、スキーマの複合型または要素の名前から導出されます。このクラスについては、JAX-RPC の、XML の構造と複合型に関する 4.2.3 項、WSDL の障害に関する 4.3.6 項、および SOAP の障害に関する 6.5 に説明があります。
<*portTypeName*>.java	サービス・エンドポイント・インタフェース・ファイルです。このファイルには、そのポート・タイプのすべての WSDL 操作に対する Java メソッドが含まれます。*portTypeName* は、WSDL ドキュメントの `<portTypeName>` 要素の値です。このファイルについては、JAX-RPC 1.1 仕様の、WSDL ポート・タイプに関する 4.3.3 項に説明があります。

注意： WSDL ファイルの要素名でマルチバイト・キャラクタが使用されている場合、`genInterface` コマンドは WSDL ファイルを正しく処理できません。コマンドはクラス・ファイルを生成しますが、このクラス・ファイルは正しくコンパイルできません。

この制限事項を回避するには、インタフェースを生成するときに、`dataBinding` 引数に `false` を設定します。詳細は、F-12 ページの「要素名にマルチバイト・キャラクタが含まれる WSDL ファイルを、genInterface、genProxy および topDownAssemble が正しく処理できない」を参照してください。

関連項目：

- `genInterface` を使用して J2EE Web サービス・クライアントを構築する方法の詳細は、第 14 章「J2EE Web サービス・クライアントのアセンブル」を参照してください。
- `genInterface` を使用してサーバー・サイド実装を作成する方法の詳細は、第 6 章「WSDL からの Web サービスのアセンブル」を参照してください。
- WebServicesAssembler でパッケージを名前空間にマッピングする方法の詳細は、18-72 ページの「ターゲット WSDL 名前空間とパッケージ名のマッピングのデフォルト・アルゴリズム」を参照してください。
- WebServicesAssembler が WSDL 内の特殊文字、たとえばピリオド（.）やダッシュ（-）をマッピングするために使用する方法の詳細は、JAX-RPC 1.1 の仕様を参照してください。

  `http://java.sun.com/webservices/jaxrpc/index.jsp`

次のコマンドラインと Ant タスクの例では、`src` ディレクトリ（`src/oracle/demo/service`）にサービス・エンドポイント・インタフェースを作成しています。

コマンドラインの例：

```
java -jar wsa.jar -genInterface
                  -output src
                  -wsdl myservice.wsdl
                  -packageName oracle.demo.service
```

Ant タスクの例 :

```
<oracle:genInterface output="src"
              wsdl="myservice.wsdl"
              packageName="oracle.demo.service"
              />
```

必須引数 :

genInterface コマンドを使用する際には、次の引数の指定が必須です。

- **WSDL アクセス引数 :** wsdl

すべての引数 :

次のリストは、genInterface コマンドで使用できるすべての引数を示しています。

- **デプロイメント・ディスクリプタ引数 :** ddFileName

  この引数の詳細は、18-61 ページの「デプロイメント・ディスクリプタ引数」を参照してください。

- **アセンブリ用の一般引数 :** classpath、debug、help、mappingFileName、packageName、output、searchSchema、serviceName

  これらの引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

- **Java 生成引数 :** dataBinding、mapHeadersToParameters、overwriteBeans、unwrapParameters、valueTypePackagePrefix

  これらの引数の詳細は、18-68 ページの「Java 生成引数」を参照してください。

- **WSDL アクセス引数 :** httpNonProxyHosts、httpProxyHost、httpProxyPort、wsdl（必須）

  これらの引数の詳細は、18-63 ページの「WSDL アクセス引数」を参照してください。

- **WSDL 管理引数 :** wsdlTimeout

  この引数の詳細は、18-64 ページの「WSDL 管理引数」を参照してください。

追加 Ant サポート :

なし

genProxy

genProxy コマンドは、J2SE Web サービス・クライアントから使用可能な静的プロキシ・スタブを作成する場合に使用します。このコマンドにより、WSDL に指定されているポートに接続するために必要な、すべての Java ファイルが作成されます。Oracle 固有のクライアント構成がある場合、それをコマンドに渡すには ddFileName 引数を使用します。

プロキシ・コードは、コンパイルしないと使用できません。

genProxy コマンドは、WSDL ドキュメントから表 18-2 のファイルを生成します。リストと説明では、JAX-RPC マッピング・ファイルを使用してこれらのファイルのデフォルトの命名規則が変更されていないものと想定しています。また、生成された Java クラスの名前の間に不整合がある場合は、JAX-RPC の名前衝突に関する 4.3.12 項で記述されているように、JAX-RPC のルールに従って解決されます。詳細は、18-71 ページの「名前競合の解決」を参照してください。

これらの生成されたファイルを変更しても、WebServicesAssembler コマンドが再び起動されると上書きされる場合があるので、変更はお薦めできません。

注意： WebServicesAssembler は、次の条件のスキーマ・タイプに対して genProxy コマンドで生成されたファイルは上書きしません。

- クラス名（パッケージ名を含む）が、WebServicesAssembler コマンドに対して渡されたクラスパス引数にすでに存在する。
- かつ、コマンドライン引数 overwriteBeans が、false（デフォルト値）に設定されている。18-69 ページの「overwriteBeans」を参照してください。

表 18-2 genProxy が生成するファイル

ファイル名	説明
<*derived_name*>.java	指定されている WSDL のスキーマで定義されている型ごとに 1 つのファイルが生成されます。Java クラス・ファイルの名前 *derived_name* は、スキーマの複合型または要素の名前から導出されます。このクラスについては、JAX-RPC の、XML の構造と複合型に関する 4.2.3 項、WSDL の障害に関する 4.3.6 項、および SOAP の障害に関する 6.5 に説明があります。
<*portName*>Client.java	ユーティリティ・クラス・ファイルは、WSDL 内のポートごとに生成されます。このファイルには、サービスのコール方法を示すコードが含まれます。サービスをテストするときは、このファイルのみを使用する必要があります。この点は、クライアントがサービスをコールするビークルとは異なります。これはコード例です。*portName* は、WSDL ドキュメントの <portName> 要素の値です。 **注意 :** JAX-RPC の仕様では、このファイルは必須ではありません。
<*portTypeName*>.java	サービス・エンドポイント・インタフェース・ファイルです。このファイルには、そのポート・タイプのすべての WSDL 操作に対する Java メソッドが含まれます。*portTypeName* は、WSDL ドキュメントの <portTypeName> 要素の値です。このファイルについては、JAX-RPC 1.1 仕様の、WSDL ポート・タイプに関する 4.3.3 項に説明があります。
<*serviceName*>.java	この名前のサービス・インタフェース・ファイルが、WSDL ドキュメント内のすべてのサービスに対して生成されます。*serviceName* は、WSDL の <serviceName> 要素の値です。このクラスについては、WSDL サービスに関する 4.3.9 項、およびサービス・インタフェースに関する 4.3.10 項に説明があります。
ランタイム・ディレクトリ / パッケージ	これらのディレクトリには、Web サービス・プロキシ・コードが内部的に使用するファイルが含まれます。これらのファイルは、将来のリリースで変更または削除される可能性があるので、直接使用しないでください。

関連項目：

- genProxy を使用してプロシキ・スタブ・コードを作成する方法の詳細は、第 15 章「J2SE Web サービス・クライアントのアセンブル」を参照してください。
- WebServicesAssembler ツールへの入力として使用できる WSDL に対する制限事項は、18-22 ページの「WSDL の制限事項」を参照してください。

次のコマンドラインと Ant タスクの例では、プロキシ・コード全体を作成して proxysrc ディレクトリに格納しています。

コマンドラインの例 :

```
java -jar wsa.jar -genProxy
                  -output proxysrc
                  -wsdl myservice.wsdl
```

Ant タスクの例 :

```
<oracle:genProxy output="proxysrc"
          wsdl="myservice.wsdl"
          />
```

必須引数 :

genProxy コマンドを使用する際には、次の引数の指定が必須です。

- **WSDL アクセス引数 :** wsdl

すべての引数 :

次のリストは、genProxy コマンドで使用できるすべての引数を示しています。

- **デプロイメント・ディスクリプタ引数 :** ddFileName

  この引数の詳細は、18-61 ページの「デプロイメント・ディスクリプタ引数」を参照してください。

- **アセンブリ用の一般引数 :** classpath、debug、help、mappingFileName、packageName、output、searchSchema、serviceName、useDimeEncoding

  これらの引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

- **Java 生成引数 :** dataBinding、mapHeadersToParameters、overwriteBeans、unwrapParameters、valueTypePackagePrefix

  これらの引数の詳細は、18-68 ページの「Java 生成引数」を参照してください。

- **JMS アセンブリ引数 :** replyToConnectionFactoryLocation、replyToQueueLocation

  これらの引数の詳細は、18-57 ページの「JMS アセンブリ引数」を参照してください。

- **メッセージ書式引数 :** mtomSupport

  この引数の詳細は、18-67 ページの「メッセージ書式引数」を参照してください。

- **プロキシ引数 :** endpointAddress、genJUnitTest

  これらの引数の詳細は、18-60 ページの「プロキシ引数」を参照してください。

- **WSDL アクセス引数 :** httpNonProxyHosts、httpProxyHost、httpProxyPort、wsdl（必須）

  これらの引数の詳細は、18-63 ページの「WSDL アクセス引数」を参照してください。

- **WSDL 管理引数 :** wsdlTimeout

  この引数の詳細は、18-64 ページの「WSDL 管理引数」を参照してください。

追加 Ant サポート :

- <handler> タグ : 詳細は、18-81 ページの「Ant タスクでのハンドラの構成方法」を参照してください。
- <port> タグ : <port> タグでは、genProxy は、引数 endpointAddress、name（portName と同じ）、portName、replyToConnectionFactoryLocation および replyToQueueLocation を使用できます。詳細は、18-78 ページの「Ant タスクでのポートの構成方法」を参照してください。

genValueTypes

genValueTypes コマンドは、指定されたスキーマから JAX-RPC 値タイプ・クラス（Bean）を作成する場合に使用します。このコマンドにより、スキーマドリブンの Web サービス開発用の Bean が作成されます。

WebServicesAssembler では、genValueTypes の各起動用に複数の値タイプ・クラスを作成できます。このコマンドでは、コマンドラインで複数の schema 引数、または Ant タスクで <schema value=""> 行がサポートされます。

このコマンドでは、Bean 以外に custom-type-mappings.xml および jaxrpc-mappings.xml ファイルも作成されます。custom-type-mappings.xml ファイルにより、カスタム・シリアライザの構成が容易になります。生成されるカスタム・タイプ・マッピング・ファイルは、サービス・サイド・スキーマに準拠しますが、クライアント・サイドでの使用のために若干変更することはできます。

jaxrpc-mappings.xml ファイルは、ボトムアップ方式 Web サービス開発用の WSDL 生成時に供給できる部分的な JAX-RPC マッピング・ファイルです。

関連資料：

- 編集済サービス・サイドのカスタム・タイプ・マッピング・ファイルの例は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「サーバー・サイドのカスタム・タイプ・マッピング・ファイルの編集」を参照してください。
- カスタム・シリアライザの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 値タイプのカスタム・シリアライズ」を参照してください。

次のコマンドラインと Ant タスクの例では、スキーマ mySchema.xsd および otherSchema.xsd に定義されたすべての複合型について値タイプ（Bean）を、また、ファイル custom-type-mappings.xml および jaxrpc-mappings.xml を作成しています。Bean とファイルは build ディレクトリに格納されます。

コマンドラインの例：

```
java -jar wsa.jar -genValueTypes
                  -schema mySchema.xsd
                  -schema otherSchema.xsd
                  -packageName com.mycompany
                  -output build
```

Ant タスクの例：

```
<oracle:genValueTypes packageName="com.mycompany" output="build">
                <oracle:schema value="otherSchema.xsd"/>
                <oracle:schema value="mySchema.xsd"/>
 </oracle:genValueTypes>
```

必須引数：

genValueTypes コマンドを使用する際には、次の引数の指定が必須です。

- **アセンブリ用の一般引数：** schema

すべての引数：

次のリストは、genValueTypes コマンドで使用できるすべての引数を示しています。

- **アセンブリ用の一般引数：** debug、help、packageName、output、schema（必須）

  これらの引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

- **WSDL アクセス引数：** httpNonProxyHosts、httpProxyHost、httpProxyPort

  これらの引数の詳細は、18-63 ページの「WSDL アクセス引数」を参照してください。

追加 Ant サポート :
なし

デプロイメント・ディスクリプタ生成コマンド

次のコマンドにより、EAR のディスクリプタの生成に使用されるデプロイメント・ディスクリプタまたはファイルが作成されます。

- genApplicationDescriptor: `application.xml` ファイルの作成
- genDDs: デプロイメント・ディスクリプタの作成

genApplicationDescriptor

`genApplicationDescriptor` コマンドは、`application.xml` ファイルを作成する場合に使用します。このファイルは EAR の生成時に使用できます。

`input` 引数には、EAR に格納されることになる WAR および EJB JAR を格納するディレクトリを指定します。生成される `application.xml` には、指定の `input` ディレクトリにある WAR および EJB JAR のそれぞれに対するタグが含まれます。

コマンドラインの例 :

```
java -jar wsa.jar -genApplicationDescriptor
                  -input src/ejb
                  -output build
```

Ant タスクの例 :

```
<oracle:genApplicationDescriptor
        input="src/ejb"
        output="build"
/>
```

必須引数 :
`genApplicationDescriptor` コマンドを使用する際には、次の引数の指定が必須です。

- **アセンブリ用の一般引数 :** input

すべての引数 :
次のリストは、`genApplicationDescriptor` コマンドで使用できるすべての引数を示しています。

- **デプロイメント・ディスクリプタ引数 :** context

  この引数の詳細は、18-61 ページの「デプロイメント・ディスクリプタ引数」を参照してください。

- **アセンブリ用の一般引数 :** debug、help、input（必須）、output

  これらの引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

追加 Ant サポート :
なし

genDDs

genDDs コマンドは、トップダウンまたはボトムアップ方式での Web サービス生成で web.xml、webservices.xml および oracle-webservices.xml デプロイメント・ディスクリプタを作成する場合に使用します。

次のコマンドラインと Ant タスクの例では、デプロイメント・ディスクリプタを作成して WEB-INF ディレクトリに格納しています。

コマンドラインの例 :

```
java -jar wsa.jar -genDDs
                  -output WEB-INF
                  -wsdl myservice.wsdl
                  -classpath myservice.jar
                  -interfaceName com.mycompany.myservice.Hello
                  -className com.mycompany.myservice.HelloImpl
```

Ant タスクの例 :

```
<oracle:genDDs output="WEB-INF"
        wsdl="myservice.wsdl"
        classpath="myservice.jar"
        >
        <oracle:porttype
           interfaceName="com.mycompany.myservice.Hello"
           className="com.mycompany.myservice.HelloImpl"/>
        </oracle:genDDs>
```

必須引数 :

genDDs コマンドを使用する際には、次の引数の指定が必須です。

- **アセンブリ用の一般引数 :** className、interfaceName
- **WSDL アクセス引数 :** wsdl

すべての引数 :

次のリストは、genDDs コマンドで使用できるすべての引数を示しています。

- **デプロイメント・ディスクリプタ引数 :** appendToExistingDDs、context、ddFileName、uri

  これらの引数の詳細は、18-61 ページの「デプロイメント・ディスクリプタ引数」を参照してください。

- **アセンブリ用の一般引数 :** className（必須）、classpath、debug、ejbName、help、interfaceName（必須）、mappingFileName、output、serviceName、strictJaxrpcValidation、useDimeEncoding

  これらの引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

- **JMS アセンブリ引数 :** sendConnectionFactoryLocation、sendQueueLocation

  これらの引数の詳細は、18-57 ページの「JMS アセンブリ引数」を参照してください。

- **WSDL アクセス引数 :** httpNonProxyHosts、httpProxyHost、httpProxyPort、wsdl（必須）

  これらの引数の詳細は、18-63 ページの「WSDL アクセス引数」を参照してください。

- **WSDL 管理引数 :** wsdlTimeout

  この引数の詳細は、18-64 ページの「WSDL 管理引数」を参照してください。

追加 Ant サポート :

- <port> タグ。<port> タグでは、genDDs は、引数 name（portName と同じ）、portName、sendConnectionFactoryLocation、sendQueueLocation および uri を使用できます。詳細は、18-78 ページの「Ant タスクでのポートの構成方法」を参照してください。
- <porttype> タグ。詳細は、18-80 ページの「Ant タスクでのポート・タイプの構成方法」を参照してください。
- <handler> タグ。詳細は、18-81 ページの「Ant タスクでのハンドラの構成方法」を参照してください。

メンテナンス・コマンド

次の各コマンドにより、WebServicesAssembler コマンドの概略やツールのバージョン番号が戻されます。

- help: WebServicesAssembler コマンドのリストが戻される。
- version: WebServicesAssembler ツールのバージョンが戻される。

help

help コマンドは、WebServicesAssembler コマンドのリストと簡単な説明を戻す場合に使用します。WebServicesAssembler でも、次のように実行された場合にヘルプが戻されます。

- コマンドや引数を指定しなかった場合。たとえば、次のようになります。

  ```
  java -jar wsa.jar
  ```

- 不正なコマンドや引数を指定した場合。たとえば、次のようになります。

  ```
  java -jar wsa.jar -myProxy -ootput proxysrc -wsdl myservice.wsdl
  ```

特定のコマンド後ろに -help を付けて実行すると、そのコマンドのヘルプを取得できます。たとえば、次のコマンドでは、genProxy コマンドとその必須およびオプションの引数に関する詳細なヘルプが戻されます。

```
java -jar wsa.jar -genProxy -help
```

help コマンドは、コマンドライン上でのみサポートされます。これに相当する Ant タスクはありません。次のコマンドラインの例では、WebServicesAssembler によってサポートされているコマンドと引数に関するヘルプ・テキストが戻されます。

コマンドラインの例 :

```
java -jar wsa.jar -help
```

Ant タスクの例 :

Ant でのサポートはありません。

必須引数 :

なし

すべての引数 :

help コマンドで使用できる引数は、次のとおりです。

- **アセンブリ用の一般引数 :** debug

  この引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

追加 Ant サポート :

なし

version

version コマンドは、WebServicesAssembler ツールのバージョン番号を取得する場合に使用します。

コマンドラインの例 :

```
java -jar wsa.jar -version
```

Ant タスクの例 :

Ant でのサポートはありません。

必須引数 :

なし

すべての引数 :

- **アセンブリ用の一般引数 :** debug、help

  これらの引数の詳細は、18-40 ページの「Web サービス・アセンブリ用の一般引数」を参照してください。

追加 Ant サポート :

なし

WebServicesAssembler の引数

この項では、各 WebServicesAssembler コマンドでコールできる引数について説明します。

Web サービス・アセンブリ用の一般引数

この項では、多くの WebServicesAssembler コマンドで使用できる一般引数について説明します。これには、ファイル関連入出力引数、WSDL 関連引数およびマッピング関連引数があります。

- appName
- bindingName
- classFileName
- className
- classpath
- debug
- ear
- ejbName
- emptySoapAction
- help
- initialContextFactory
- input
- interfaceFileName
- interfaceName
- jndiName
- jndiProviderURL
- mappingFileName
- output
- packageName
- portName
- portTypeName
- restSupport
- schema
- searchSchema
- serviceName
- strictJaxrpcValidation
- useDimeEncoding
- war

appName

`appName <String>`

アプリケーションの名前を指定します。この名前は、通常 `context` や `uri` などの他の引数のベース値として使用されます。`ear` および `war` 引数が指定されていない場合に、この引数の値が EAR および WAR ファイルのネーミングにも使用されます。

bindingName

`bindingName <String>`

生成される WSDL 内で使用されるバインディングの名前。

classFileName

`classFileName <String>`

`className` 引数で指定された実装クラスの Java ファイル名を指定します。このファイルは、アセンブリ時に必要に応じてコンパイルされます。

className

`className <String>`

Web サービスの実装クラスのクラス名（パッケージ名を含む）を指定します。

この引数を使用すると、`<servlet-class>` 要素が `web.xml` に追加されます。

EJB を Web サービスとして公開している場合、`className` 引数の値は EJB のホーム・インタフェースである必要があります。

関連項目：

`className` 引数を指定してこのコマンドを使用する方法の詳細は、18-14 ページの「ejbAssemble」を参照してください。

この引数を `wsifJavaPort` 引数とともに使用すると、WSDL の `port` コンポーネントにある `java:address` 要素に `classname` 属性が追加されます。

`wsifEjbPort` 引数とともにこの引数を使用すると、WSDL の `port` コンポーネントにある `ejb:address` 要素に `classname` 属性が追加されます。

classpath

`classpath <String>`

WebServicesAssembler に与えられるユーザー作成クラスをすべて含むクラスパスを指定します。この引数を指定する理由の 1 つは、一部の値タイプ・クラスまたは例外を作成済の場合に、WebServicesAssembler が上書きしないようにするためです。`*Assemble` コマンドなど、WAR ファイルを生成するコマンドの場合、この引数により、Web サービスが依存するが、生成された WAR に入れないクラスを指定することができます。

この引数により、生成された WAR ファイルに追加クラスがコピーされることはありません。実行時またはデプロイ時に必要となるクラスは、手動で WAR または EAR にコピーするか、あるいはサーバー構成の各オプションを使用してアプリケーション・サーバーのクラスパスで使用可能にする必要があります。

WAR ファイルにクラスを組み入れるには、`input` 引数を使用することもできます。

関連項目：

この引数の詳細は、18-45 ページの「input」を参照してください。

クラスパスに複数のパスをリストするには、UNIX オペレーティング・システムを使用している場合は、コロン（:）で区切ります。Windows オペレーティング・システムを使用している場合は、セミコロン（;）で複数のパスを区切ります。

debug

debug <true|false>

指定されたコマンドに関するすべての詳細診断メッセージを出力するかどうかを指定します。デフォルト値は false です。

次のコマンドラインと Ant タスクの例では、assemble コマンドのパフォーマンスに関する診断メッセージを表示しています。

コマンドラインの例 :

```
java -jar -wsa.jar -assemble -debug true...
```

Ant タスクの例 :

```
<oracle:assemble debug="true"
    ....
 />
```

ear

ear <*file name*>

生成される EAR ファイルの名前と位置を指定します。次の例では、dist ディレクトリに EAR ファイル myService.ear を作成しています。

```
-ear dist/myService.ear
```

次の各項目は、WebServicesAssembler によって ear 引数の値がどのように解釈されるかについての説明です。

- ear 引数の値が拡張子 .ear のあるファイル名で終わる場合（前述の例を参照）、この名前の EAR ファイルが ear 引数に指定された位置に作成されます。output 引数は、EAR ファイルの位置の決定には使用されません。
- 前述のようでない場合はすべて、名前はディレクトリ名であるとみなされます。EAR ファイルは作成されません。かわりに、EAR の内容が指定されたディレクトリに書き込まれます。
- output 引数の値を指定して ear 引数を指定しなかった場合は、EAR ファイルにはデフォルトの名前として appName 引数の値が付けられます。たとえば、output の値が build で、appName の値が myService である場合は、ear を指定しないと、EAR ファイルは build/myService.ear として作成されます。

注意 : ear と output 引数に対して同じディレクトリを指定すると、WebServicesAssembler によりエラーが戻されます。

---

ear と war 引数は、同じコマンドラインか Ant タスクで一緒に使用できます。表 18-3 では、これらの引数にファイルとディレクトリのどちらを指定したかに応じてこれらの引数の動作がどう異なってくるかについて、説明します。

表 18-3 値がファイルとディレクトリのいずれであるかに応じた ear および war 引数の動作

ear および war 引数の組合せ	動作
ear *directory1* war *directory2*	ear と war 引数にそれぞれディレクトリが指定された場合は、次の動作が行われます。 ■ .ear ファイルは作成されません。 ■ .ear ファイルの内容が *directory1* に生成されます。 ■ .war ファイルは *directory1* に作成されます。 ■ .war ファイルの内容が *directory2* に生成されます。
ear *directory1* war *file*	ear 引数にディレクトリが、war 引数にファイルが指定された場合は、次の動作が行われます。 ■ .ear ファイルは作成されません。 ■ .ear ファイルの内容が *directory1* に生成されます。 ■ .war ファイルは *directory1* に作成されます。 ■ .war ファイルが、ユーザー指定の名前（*file* パラメータの値）で作成されます。
ear *file* war *directory2*	ear 引数にファイルが、war 引数にディレクトリが指定された場合は、次の動作が行われます。 ■ .ear ファイルが、ユーザー指定の名前（*file* パラメータの値）で作成されます。 ■ .war ファイルは、.ear ファイル内に作成されます。 ■ .war ファイルの内容は *directory2* に生成されます。
ear *file1* war *file2*	ear 引数にファイルが、war 引数にファイルが指定された場合は、次の動作が行われます。 ■ .ear ファイルが、ユーザー指定の名前（*file1* パラメータの値）で作成されます。 ■ .war ファイルは、.ear ファイル内に作成されます。 ■ .war ファイルが、ユーザー指定の名前（*file2* パラメータの値）で作成されます。
ear 引数が未指定 war 引数が未指定	ear と war のどちらの引数も指定されていない場合は、次の動作が行われます。 ■ .ear ファイルは、デフォルト名 *application_name*.ear で生成されます。 ■ .war ファイルは、デフォルト名 *application_name*_web.war で生成されます。 これらの例で、*application_name* は appName 引数の値を表します。

ejbName

ejbName <*String*>

Web サービスとして公開する EJB の名前。これはクラス名ではありません。ejb-jar.xml ファイルの <ejb-name> タグに指定された EJB の一意の名前です。

EJB がバージョン 2.1 の場合、この引数を使用すると <ejb-link> 要素が webservices.xml に追加されます。

emptySoapAction

emptySoapAction <true|false>

true の場合、生成される WSDL 内の各 SOAP バインディング操作の soapAction 属性の値は、空の文字列に設定されます。false の場合（デフォルト）、ターゲットの名前空間と操作名（<*target-namespace*>/<*operation-name*>）は、soapAction 属性の値として使用されます。

OracleAS Web Services WSDL を使用してクライアント・サイド・プロキシを生成する他のベンダー製のツールでは、soapAction のデフォルト値を認識することや受け入れることができないことがあります。この引数を true に設定すると、このようなツールと相互運用できる可能性が高くなります。

help

help

指定されたコマンドのヘルプ・メッセージを表示します。help 引数は、コマンドの前後いずれにも指定できます。

次のコマンドラインと Ant タスクの例では、assemble コマンドに関するヘルプ・メッセージを表示しています。

コマンドラインの例 :

```
java -jar -wsa.jar -assemble -help ...
```

または

```
java -jar -wsa.jar -help -assemble ...
```

Ant タスクの例 :

```
<oracle:assemble debug="help"
    ....
 />
```

initialContextFactory

initialContextFactory <*String*>

初期コンテキストを提供するファクトリの名前を指定します。これはオプションの引数で、genWsdl または ejbAssemble コマンドで EJB WSIF ポートを構成するときに、wsifEjbBinding または wsifEjbPort とともにコールできます。この属性に値を指定しなかった場合は、jndi.properties ファイルにある値が使用されます。

input

input <*String*>

WEB-INF/classes にコピーされるクラスを格納する JAR またはディレクトリを指定します。この引数は、WebServicesAssembler によって使用されるクラスパスに追加されます。ejbAssemble コマンドでは、input の値には、EJB JAR ファイルまたは EJB JAR の内容を格納するディレクトリが想定されます。

この引数に JAR を指定した場合は、JAR が展開されてその内容が WEB-INF/classes ディレクトリにコピーされます。

input 引数は、コマンドラインまたは Ant タスクで複数回使用できます。Ant タスクでは複数のタグとして書き込み、value 属性を指定します。例 18-1 に、Ant タスクにおける input の複数のインスタンスを示します。

例 18-1 Ant タスクにおける input の複数のインスタンス

```
<oracle:jmsAssemble
       linkReceiveWithReplyTo="true"
       targetNamespace="http://oracle.j2ee.ws/jms-doc"
       typeNamespace="http://oracle.j2ee.ws/jms-doc/types"
       serviceName="JmsService"
       appName="jms_service"
       context="jms_service"
       input="./demo/build/mdb_service.jar"
       output="./demo/dist"
       input="first.jar"
       >
             <oracle:input value="second.jar"/>
             <oracle:input value="third.jar"/>
</oracle:jmsAssemble>
```

interfaceFileName

interfaceFileName <*String*>

サービス・エンドポイント・インタフェース（SEI）Java ソース・コード・ファイルのパスと名前を指定します。WebServicesAssembler により、指定された名前のファイルが見つからない場合、処理が停止されます。

interfaceFileName 引数を指定すると、WebServicesAssembler が生成済 WSDL 内から Java メソッドのパラメータ名を検索する際に役立ちます。

関連項目：

このトピックの詳細は、18-87 ページの「WSDL での Java メソッド・パラメータ名の表現方法」を参照してください。

interfaceName

interfaceName <*String*>

サービス・エンドポイント・インタフェース（SEI）を格納する Java クラスの名前（パッケージ名を含む）を指定します。

この引数を使用すると、<service-endpoint-interface>*String*</service-endpoint-interface> 要素が WEB-INF/webservices.xml に追加されます。この場合、*String* は interfaceName に指定された値です。

データベース・リソースから Web サービスをアセンブルするコマンド（plsqlAssemble、sqlAssemble、dbJavaAssemble または aqAssemble）でこの引数を使用すると、<service-endpoint-interface>*String*</service-endpoint-interface> 要素が、*serviceName*-java-wsdl-mapping.xml ファイルに追加されます。*String* は interfaceName に指定された値で、*serviceName* は serviceName 引数の値です。

jndiName

`jndiName <String>`

EJB の JNDI 名を指定します。

jndiProviderURL

`jndiProviderURL <String>`

JNDI プロバイダの URL を指定します。これはオプションの引数で、`genWsdl` または `ejbAssemble` コマンドで EJB WSIF ポートを構成するときに、`wsifEjbBinding` または `wsifEjbPort` とともにコールできます。この属性に値を指定しなかった場合は、`jndi.properties` ファイルにある値が使用されます。

mappingFileName

`mappingFileName <String>`

JAX-RPC マッピング・ファイルを指すファイル位置を指定します。WebServicesAssembler により、指定された名前のファイルが見つからない場合、処理が停止されます。

この引数を使用すると、`<jaxrpc-mapping-file>` 要素が `webservices.xml` に追加されます。ファイルの位置と名前は、デプロイメント・ディスクリプタに書き込まれるときに変更されることがあります。いくつかのマッピングが元のファイルで定義されていないときは、ファイルの内容は、アーカイブに格納される前に変更されることがあります。

関連項目：

JAX-RPC マッピング・ファイルの内容とその使用方法の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「JAX-RPC マッピング・ファイル記述子」を参照してください。

output

`output <String>`

生成されるファイルを格納するディレクトリを指定します。そのディレクトリが存在しない場合は、作成します。`output` 引数を指定しなかった場合、出力は現行ディレクトリに格納されます。

`genConcreteWsdl` コマンドで使用する場合を除いて、`output` 引数のターゲットは、常にディレクトリであるものと想定されます。`genConcreteWsdl` コマンドの場合は、ターゲットにはファイルが想定されます。

`output` 引数は、通常 `ear` および `war` 引数とともに使用されます。

関連項目：

これらの引数とともに使用されたときの `output` の動作の詳細は、18-42 ページの「ear」および 18-49 ページの「war」を参照してください。

注意： `output` と `ear`（または `war`）引数に対して同じディレクトリを指定すると、WebServicesAssembler によりエラーが戻されます。

---

packageName

packageName <*String*>

JAX-RPC マッピング・ファイルにパッケージ名が宣言されていない場合に、生成されたクラス用に使用されるパッケージ名を指定します。packageName が指定されておらず、マッピング・ファイルでも宣言されていない場合、パッケージ名は WSDL のターゲットの名前空間から導出されます。

関連項目：

名前空間とパッケージのマッピングの詳細は、18-72 ページの「ターゲット WSDL 名前空間とパッケージ名のマッピングのデフォルト・アルゴリズム」を参照してください。

注意： packageName 引数は、サービス・エンドポイント・インタフェースおよび WSDL と同じターゲットの名前空間を持つスキーマ型に対するパッケージ名にのみ影響します。スキーマ値タイプが異なる名前空間にある場合は、スキーマ値タイプは、WebServicesAssembler によりデフォルトで別のパッケージに生成されます。単一パッケージへのコードの生成の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「複数の名前空間のある WSDL から単一のパッケージへのコードの生成」を参照してください。

---

portName

portName <*String*>

WSDL ドキュメント内のポート名を指定します。この引数により、<port name="..."> WSDL 要素に値が移入されます。

ポート名を指定しなかった場合、デフォルトのポート名は、トランスポートおよび SOAP のバージョンかまたは WSIF タイプに基づいて設定されます。デフォルトのポート名は次のいずれかの値になります。

- HttpSoap11
- HttpSoap12
- JmsSoap11
- JmsSoap12
- WsifEjb
- WsifJava

portTypeName

portTypeName <*String*>

生成される WSDL 内で使用されるポート・タイプ名を指定します。この引数により、<portType name="..."> WSDL 要素に値が移入されます。

restSupport

`restSupport <true|false>`

Representational State Transfer（REST）のサポートをこの Web サービスに対して有効にするかどうかを指定します。デフォルト値は `false` です。

REST サービスを使用すると、HTTP GET および POST コマンドでサービス・オブジェクトを取得および変更できるようになります。また、REST Web サービスには、SOAP エンベロープではなく XML 文書を使用したメッセージ送信の機能もあります。

関連項目：

REST サポート付き Web サービスの実装方法の詳細は、第 12 章「REST Web サービスのアセンブル」を参照してください。

schema

`schema <String>`

XML スキーマ・ドキュメントへの相対パスまたは URL を指定します。この引数により、値タイプを WebServicesAssembler に生成させるのではなく、値タイプのスキーマを指定できます。この引数は、JAX-RPC マッピング・ファイルと組み合せて使用する必要があります。

関連項目：

マッピング・ファイルの指定の詳細は、18-46 ページの「mappingFileName」を参照してください。

`schema` 引数は、コマンドラインまたは Ant タスクで複数回使用できます。Ant タスクでは複数のタグとして書き込み、`value` 属性を指定します。例 18-2 に、Ant タスクにおける `schema` の複数のインスタンスを示します。

例 18-2 Ant タスクにおける schema の複数のインスタンス

```
<oracle:genValueTypes packageName="com.mycompany"
              output="build">
                   <oracle:schema value="otherSchema.xsd"/>
                   <oracle:schema value="mySchema.xsd"/>
 </oracle:genValueTypes>
```

searchSchema

`searchSchema <true|false>`

WSDL にリストされているスキーマにあるすべての型を処理するかどうかを指定します。この引数を `true` に設定すると（デフォルト）、すべての型が処理されます。これは、スキーマ内にはあるが WSDL から直接には参照されない型を使用する Web サービス実装の使用時に便利です。この引数を `false` に設定すると、WSDL から参照されない型は処理されません。

serviceName

`serviceName <String>`

サービス名を指定します。`serviceName` 引数には、生成済またはパッケージ済の WSDL ファイル（*serviceName*`.wsdl`）およびマッピング・ファイル（*serviceName*`-java-wsdl-mapping.xml`）を指定します。また、この引数は、ボトムアップ方式 Web サービス・アセンブリにおいて、`<service name="...">` WSDL 要素の値の指定も行います。

この引数をトップダウン方式での Web サービスとプロキシのアセンブリで使用すると、WSDL では、この名前のサービスが見つかるものと予期します。この場合、`<service name="...">` WSDL 要素の値は変更されません。

strictJaxrpcValidation

`strictJaxrpcValidation <true|false>`

サービス・エンドポイント・インタフェース、例外および値タイプを、すべての JAX-RPC 検証規則に関して検証するかどうかを設定します。この引数を指定しないか `true` に設定すると(デフォルト)、次の妥当性チェックが実行されます。

- サービス・エンドポイント・インタフェースで `java.rmi.Remote` が実装されているか。
- すべてのメソッドで `java.rmi.RemoteException` がスローされるか。
- すべての値タイプとプロパティが JAX-RPC 規則に準拠しているか。

これらの妥当性チェックのいずれかが失敗した場合、処理が停止し、インタフェースの問題点を記載したエラーがスローされます。

この引数を `false` に設定した場合は、サービス・エンドポイント・インタフェースでの `java.rmi.Remote` の実装や、メソッドでの `java.rmi.RemoteException` のスローが不要になります。ただし、メソッドに、JAX-RPC 規則に準拠しないパラメータや例外がある場合は、メソッドは無視され、処理されません。

この引数を false に設定すると、Bean や例外の場合に、Bean（または例外）の無効なプロパティが無視されます。有効なプロパティは処理され、WSDL に追加されます。この場合の動作では、ユーザーが予期しないワイヤ・レベルの書式が生成されることがありますので、注意してください。無効なプロパティの場合、WebServicesAssembler により警告がスローされます。

useDimeEncoding

`useDimeEncoding <true|false>`

添付ファイル付き SOAP メッセージのケーブル上でのストリーミングに DIME エンコーディングを使用するかどうかを指定します。`useDimeEncoding` を `true` に設定すると、添付ファイルには MIME のかわりに DIME エンコーディングが使用されます。デフォルト値は `false` です。

この引数を使用すると、`<use-dime-encoding>` 要素が `oracle-webservices.xml` に追加されます。

注意： `useDimeEncoding` 引数を使用して生成したメッセージは、相互運用性がなく、Oracle 以外の Web サービスまたは以前のリリースの Oracle Application Server とは互換性がありません。この引数を指定した Web サービスの生成は、パフォーマンスが重要である社内プロジェクトに有効です。

関連資料：

DIME エンコーディングと添付ファイルの使用方法の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「DIME 添付ファイルの処理」を参照してください。

war

`war <file name or directory name>`

生成する WAR の名前を指定します。ファイルを指定する場合、拡張子は `.war` である必要があります。ディレクトリを指定する場合は、WAR の内容が指定されたディレクトリに書き込まれます。次の例では、`dist` ディレクトリに WAR ファイル `myService.war` を作成しています。

```
-war dist/myService.war
```

次の各項目は、WebServicesAssembler によって war 引数の値がどのように解釈されるかについての説明です。

- war 引数の値は、拡張子 ".war" のあるファイル名で終わる場合（前述の例を参照）、この名前の WAR ファイルが war 引数によって指定された位置に作成されます。output 引数は、WAR ファイルの位置の決定には使用されません。
- 前述のようでない場合はすべて、値はディレクトリ名であるとみなされます。WAR ファイルは、指定したディレクトリに書き込まれます。
- output 引数の値を指定して war 引数を指定しないと、WAR は EAR 内に配置されます。EAR 内での WAR ファイルの名前は、*appName*.war になります。

注意： war と output 引数に対して同じディレクトリを指定すると、WebServicesAssembler によりエラーが戻されます。

関連項目：

- WebServicesAssembler による ear 引数の値の解釈方法は、18-42 ページの「ear」を参照してください。
- war 引数と ear 引数を同じコマンドラインまたは Ant タスクで一緒に使用したときの動作の詳細は、18-43 ページの表 18-3 を参照してください。

セッション引数

セッション状態の動作の制御には、次の引数を使用できます。

- callScope
- recoverable
- session
- timeout

callScope

callScope <true|false>

サーバントがコールごとに作成され、コール後はガベージ・コレクションされるかどうかを指定します。デフォルト値は false です。callScope と session の両方を true に設定すると、エラーがスローされます。

値を true に設定してこの引数を使用すると、<param name="scope">call</param> 要素が oracle-webservices.xml に追加されます。

recoverable

recoverable <true|false>

セッション状態のあるアプリケーションをリカバリ可能にするかどうかを指定します。この引数は、サービスが、セッション・スコープを持つステートフル Web サービスとして公開されているときにのみ使用できます。デフォルトである true では、セッション状態が保持されます。リカバリ可能にすると、ブールの要素 <distributable> が web.xml に追加されます。リカバリ可能とは、対話しているノードがダウンした場合に、サービスが分散環境内でリカバリできることを意味します。つまり、Web サービスの状態も分散可能である必要があります。

recoverable が false の場合は、<distributable> 要素は web.xml に追加されません。

session

session <true|false>

HTTP セッションの期間、サーバント・インスタンスを保持することを指定します。この引数が有効なのは HTTP トランスポートの場合のみです。セッション・タイムアウトは、timeout 引数によってチューニングできます。デフォルト値は false です。timeout が設定されると、session がデフォルトで true に設定されます。callScope と session の両方を true に設定すると、エラーがスローされます。

この引数を使用すると、<param name="scope">session</param> 要素が oracle-webservices.xml に追加されます。

timeout

timeout <*int*>

セッションの開始からタイムアウトまでの秒数を指定します。デフォルト値は 60 秒です。この値を 0 または負の数値に設定すると、セッションはタイムアウトしません。

この引数に値を設定すると、session 引数は自動的に true に設定されます。session 引数が true で、timeout が設定されていない場合は、セッションは 60 秒後にタイムアウトになります。

この引数を使用すると、<param name="session-timeout">*value*</param> 要素が oracle-webservices.xml に追加されます。この場合、*value* は timeout に指定された整数値です。

CORBA アセンブリ引数

corbaAssemble コマンドで使用できる引数は、次のとおりです。これらの引数を使用して、CORBA サーバント・オブジェクトにより Web サービスをアセンブルする方法を制御できます。

- corbanameURL
- corbaObjectPath
- idlFile
- idlInterfaceName
- idljPath
- ORBInitialHost
- ORBInitialPort
- ORBInitRef

corbanameURL

corbanameURL <*String*>

CORBA ネーミング・サービスを使用した、CORBA オブジェクトのロケーティング用 URL を指定します。

corbaObjectPath

corbaObjectPath <*String*>

CORBA オブジェクトへのパスを指定します。

idlFile

`idlFile <String>`

CORBA `idl` ファイルの位置を指定します。

idlInterfaceName

`idlInterfaceName <String>`

Web サービスを生成するベースとなる `idl` 内のインタフェースの名前を指定します。

idljPath

`idljPath <String>`

IDL-to-Java コンパイラ（`idlj`）を格納するディレクトリのパスが `path` に未指定の場合に、それを指定します。

ORBInitialHost

`ORBInitialHost <String>`

ORB のホスト名を指定します。

ORBInitialPort

`ORBInitialPort <String>`

ORB のポートを指定します。

ORBInitRef

`ORBInitRef <String>`

ORB の初期参照を指定します。

データベース・アセンブリ引数

これは、データベース・アーチファクトから Web サービスをアセンブルするコマンドの引数です。データベース・アーチファクトには、PL/SQL ストアド・プロシージャ、SQL 文、Oracle アドバンスト・キュー（AQ）、Oracle データベースにある Java VM の Java クラスなどがあります。

関連項目：

データベース・アーチファクトから Web サービスをアセンブルする方法の詳細は、第 10 章「データベース Web サービスのアセンブル」を参照してください。

- aqConnectionFactoryLocation
- aqConnectionLocation
- dataSource
- dbConnection
- dbJavaClassName
- dbUser
- jpubProp
- sql
- sqlstatement
- sqlTimeout

- sysUser
- useDataSource
- wsifDbBinding
- wsifDbPort

aqConnectionFactoryLocation

`aqConnectionFactory <String>`

公開される AQ に対する Oracle Streams AQ JMS キューのコネクション・ファクトリの JNDI 位置を指定します。

aqConnectionLocation

`aqConnectionLocation <String>`

公開される AQ に接続する Oracle Streams AQ JMS キュー・コネクションの JNDI の位置を指定します。

dataSource

`dataSource <String>`

実行時に Web サービスによって使用されるデータソースの JNDI 位置を指定します。

この引数を使用すると、次の `param name` 属性が `oracle-webservices.xml` の `<implementor>` 要素に追加されます。`dataSource` 引数に指定したパスが、*`dataSource value`* 変数になります。

```
<param name="databaseJndiName">dataSource value</param>
```

たとえば、Ant タスク内に `dataSource="ws/dbws/src/query/datasource"` と指定すると、`<implementor>` 要素に次のように書き込まれます。

```
<implementor type="database">
     <param name="databaseJndiName">ws/dbws/src/query/datasource</param>
</implementor>
```

dbConnection

`dbConnection <String>`

データベースの JDBC URL を指定します。この引数は、コード生成時のデータベース接続を目的として、`dbUser` 引数とともに使用されます。

dbJavaClassName

`dbJavaClassname <String>`

Web サービスとして公開するサーバー・サイド Java クラスの名前を指定します。

dbUser

`dbUser <String>`

user/*password* の形式でデータベースのスキーマとパスワードを指定します。この引数は、コード生成時のデータベース接続を目的として、`dbConnection` 引数とともに使用されます。

コマンドラインで WebServicesAssembler を起動してデータベースにアクセスする際に、`-dbUser` を使用してパスワードを指定しないと、パスワードの入力を要求されます。これは、パスワードがクリア・テキストで表示されるのを防ぐためです。

jpubProp

`jpubProp <String>`

Oracle JPublisher 変換プロセスを制御するプロパティの入ったファイルの名前を指定します。

関連資料：

- `jpubProp` 引数の使用方法の例は、10-6 ページの「数値型に関しての SQL から XML へのマッピングの変更」を参照してください。
- Oracle JPublisher の詳細は、『Oracle Database JPublisher ユーザーズ・ガイド』を参照してください。

sql

`sql <String>`

`sql` 引数の値は、ともに使用するのが `aqAssemble` であるか `plsqlAssemble` であるかに応じて意味が異なります。

- `aqAssemble`: `aqAssemble` とともに使用する場合は、`sql` の値には AQ キュー名を指定します。
- `plsqlAssemble`: `plsqlAssemble` とともに使用する場合は、`sql` の値には PL/SQL パッケージ名を指定します。

`sql` 引数は、大 / 小文字が区別される SQL 文内の名前を処理できます。たとえば、次の SQL 文での引用符の使用は、パッケージ名 `SIMple` で大 / 小文字が区別されていることを示します。

```
create package "SIMple" as
procedure foo;
end;
```

コマンドラインで `sql` 引数のターゲットとして `SIMple` パッケージ名を使用するには、次のように入力します。

```
-sql '"SIMple"'
```

Ant タスクで使用するには、次のように入力します。

```
 sql="&quot;SIMple&quot;"
```

いずれの場合も `SIMple` を囲む引用符が必要です。引用符とパッケージ名は Oracle JPublisher に渡されます。Oracle JPublisher では、引用符を使用してデータベース内の名前と対照して識別子を解決します。

sqlstatement

`sqlstatement <String>`

Web サービスとして公開する DML 文または SQL 問合せを指定します。

- SQL 文は問合せまたは DML 文である必要があります。
- SQL 文の前には、WSDL および生成される Java コードの中で使用される対応名が付けられます。

  `methodName``=<`*`statement`*`>`

  `methodName` は、生成される Web サービスに含まれる、SQL 文に対応する操作名を示します。
- SQL 文には、パラメータの名前と SQL タイプからなる埋込みホスト変数を含めることができます。このパラメータ名は、WSDL と生成された Web サービス操作の中で使用されます。パラメータは次の形式で指定します。

  `:{`*`param_name param_sql_type`*`}`

この例において、*param_name* により、生成される Web サービスにおけるパラメータとしての識別子が定義され、*param_sql_type* により、そのパラメータの SQL タイプ名が定義されています。

- sqlstatement 引数は、ストアド・プロシージャの指定には使用できません。

WebServicesAssembler Ant タスクに SQL 文を渡すときは、正しく引用符が付けられていることを確認する必要があります。sqlstatement の値には、表 18-4 に示すように、標準の XML 引用符を使用して引用符を付けることができます。

表 18-4 sqlstatement の Ant タスクで有効な引用記号

記号	引用符
&	&amp;
"	&quot;
<	&lt;
>	&gt;
'	&apos;

複数の sqlstatement 引数をコマンドラインまたは Ant タスクに指定できます。Ant タスクでは複数のタグとして書き込み、value 属性を使用します。例 18-3 に、Ant タスクにおける sqlstatement の複数のインスタンスを示します。

例 18-3 Ant タスクにおける sqlstatement の複数のインスタンス

```
<oracle:sqlAssemble
   dbUser="scott/tiger"
   dbConnection="jdbc:oracle:thin:@dsunrde22:1521:sqlj"
   dataSource="jdbc/OracleManagedDS"
   appName="query">
       <sqlstatement value="getEmp=select ename, sal from emp where empno=:{id
NUMBER}"/>
       <sqlstatement value="getEmpBySal=select ename, sal from emp where sal&gt;:{mysal
NUMBER}"/>
</oracle:sqlAssemble>
```

sqlTimeout

sqlTimeout <*int*>

データベース操作のタイムアウトを秒単位で指定します。データベース操作には、PL/SQL ストアド・ファンクションまたは PL/SQL ストアド・プロシージャの起動、SQL 問合せおよび DML 文があります。デフォルトは 0 です。つまり、サービスはタイムアウトしないということです。

sysUser

sysUser <*String*>

SYS 権限のあるユーザーの名前とパスワードを、dbSysUser/syspassword の形式で指定します。この引数を使用すると、コード生成時に PL/SQL および Java ラッパー・コードがデータベースに自動的にインストールされます。

useDataSource

```
useDataSource <true|false>
```

useDataSource 引数は、コマンドラインや Ant タスクで、wsifDbBinding または wsifDbPort 引数とともにのみ、指定できます。useDataSource をこれらのいずれの引数もなしに指定した場合は、エラーがスローされます。

useDataSource 引数により、dataSource 引数の値を WSDL の port タグで使用するかどうかを決定します。useDataSource を指定しないか、true の値に指定した場合は、コマンドラインまたは Ant タスクで使用した dataSource 値が WSDL の port タグで使用されます。useDataSource を false の値に指定した場合は、コマンドラインまたは Ant タスクで使用した dbConnection 値が WSDL の port タグで使用されます。

wsifDbBinding

```
wsifDbBinding <true|false>
```

この引数は、ボトムアップ方式 Web サービス・アセンブリで、WSDL に WSIF SQL バインディングを追加する際に使用します。WSDL には、SOAP バインディングに加えて、ネイティブの WSIF SQL バインディングが生成されます。この引数のデフォルト値は false です。

関連資料：

- wsifDbBinding 引数が WSDL に追加する拡張の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「WSDL に対する WSIF SQL 拡張」を参照してください。
- WSIF SQL バインディングの定義をさらに細かく制御したり、複数のポートを指定したりできる引数の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「複数のデータベース・リソース・ポートに対する WSIF エンドポイントの構成」を参照してください。

wsifDbPort

```
wsifDbPort <true|false>
```

この引数は、ボトムアップ方式 Web サービス・アセンブリで、複数のポートに対するデータベース・リソースの WSIF バインディングを WSDL に追加する際に使用します。使用できるのは Ant タスクでのみです。

関連項目：

wisfDbPort の使用方法の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「複数のデータベース・リソース・ポートに対する WSIF エンドポイントの構成」を参照してください。

JMS アセンブリ引数

次の引数は、JMS エンドポイント Web サービスの EAR または WAR ディレクトリをアセンブルするために、`jmsAssemble` コマンドで使用できます。

関連資料：

- JMS エンドポイント Web サービスのアセンブルの詳細は、第 9 章「JMS 宛先を使用した Web サービスのアセンブル」を参照してください。
- 次の Web サイトの JMS 仕様を参照してください。

  `http://java.sun.com/products/jms/docs.html`

次の引数の中で、`replyToConnectionFactoryLocation`、`replyToQueueLocation`、`sendConnectionFactoryLocation` および `sendQueueLocation` は、JMS トランスポートでも JMS アセンブリでも使用できます。

- deliveryMode
- genJmsPropertyHeader
- jmsTypeHeader
- linkReceiveWithReplyTo
- payloadBindingClassName
- priority
- receiveConnectionFactoryLocation
- receiveQueueLocation
- receiveTimeout
- receiveTopicLocation
- replyToConnectionFactoryLocation
- replyToQueueLocation
- replyToTopicLocation
- sendConnectionFactoryLocation
- sendQueueLocation
- sendTopicLocation
- timeToLive
- topicDurableSubscriptionName

deliveryMode

`deliveryMode <*String*>`

`JMSDeliveryMode` ヘッダー・フィールドのデフォルト値を指定します。このフィールドの値が、メッセージの送信時に指定される配信モードになります。JMS 仕様によってサポートされている `deliveryMode` の値は、`PERSISTENT` および `NON_PERSISTENT` です。

genJmsPropertyHeader

genJmsPropertyHeader <*true*|*false*>

デフォルトでは、JMS エンドポイント Web サービスは、生成される WSDL に定義される SOAP ヘッダーの、次の JMS メッセージ・プロパティを必ず公開します。

- メッセージ ID
- 相関 ID
- 返信先（名前、タイプ、ファクトリなど）

JMS 固有ヘッダーおよびそのバインディングに対応するスキーマ定義は、WSDL に含まれます。この引数を false に設定すると、ワイヤ SOAP メッセージの OracleJmsProperties が実行時に無視されます。デフォルト値は true です。

jmsTypeHeader

jmsTypeHeader <*String*>

この引数の値を使用して、JMS メッセージ用の JMSType ヘッダー・フィールドのデフォルト値を指定します。JMS メッセージは、send 操作によって JMS 宛先に伝播されるメッセージです。jmsTypeHeader の値には、たとえば My Quote Message を指定できます。デフォルト値はありません。

linkReceiveWithReplyTo

linkReceiveWithReply <true|false>

receive 操作を返信先にリンクするかどうかを決定します。この引数を true に設定した場合は、受信先 / 受信コネクション・ファクトリまたは返信先 / 返信コネクション・ファクトリのいずれかを設定する必要があります。デフォルト値は false です。

payloadBindingClassName

payloadBindingClassname <*String*>

データ・バインディングが使用されていない場合は、javax.jms.ObjectMessage インスタンスのコンテンツ・オブジェクトの完全修飾 Java クラス名です。このコンテンツは Java 値タイプではなく、XML フラグメントです。有効な値は java.lang.String と javax.xml.soap.SOAPElement のみです。デフォルト値は java.lang.String です。

priority

priority <*int*>

JMS メッセージの JMS メッセージ優先度ヘッダー・フィールドのデフォルトの整数値を指定します。JMS メッセージは、send 操作によって JMS 宛先に伝播されるメッセージです。0 ～ 9 の値を指定できます。0 が優先度最低、9 が最高になります。

receiveConnectionFactoryLocation

receiveConnectionFactoryLocation <*String*>

receive 操作に使用される JMS ConnectionFactory の JNDI 名。ConnectionFactory のタイプは、receive 宛先との整合性が必要とされます。デフォルト値はありません。

receiveQueueLocation

receiveQueueLocation <*String*>

receive 操作に使用される JMS キューの JNDI 名。receiveQueueLocation と receiveTopicLocation 引数は相互排他的で、一緒に設定することはできません。デフォルト値はありません。

receiveTimeout

`receiveTimeout <int>`

この引数の値を使用して、メッセージ取得のために `receive` メソッドがブロックされる時間のデフォルト値を指定します。`receiveTimeout` 値は秒単位で表します。デフォルト値は 0 です。つまり、ブロックは失効せず、コールは無期限にブロックされます。

receiveTopicLocation

`receiveTopicLocation <String>`

`receive` 操作に使用する JMS トピックの JNDI 名。デフォルト値はありません。`receiveQueueLocation` と `receiveTopicLocation` 引数は相互排他的で、一緒に設定することはできません。

replyToConnectionFactoryLocation

`replyToConnectionFactoryLocation <String>`

すべての `send` 操作 JMS メッセージのデフォルトの返信先として使用する JMS コネクション・ファクトリの JNDI 名を指定します。`ConnectionFactory` のタイプは、返信先との整合性が必要とされます。デフォルト値はありません。この引数は、JMS Web サービス・アセンブリや JMS トランスポートで使用できます。

JMS トランスポートでこの引数を使用する場合は、ケーブルで SOAP メッセージを送信するすべての Web サービス操作のデフォルトの返信先として使用する JMS `ConnectionFactory` の JNDI 名を指定します。

replyToQueueLocation

`replyToQueueLocation <String>`

すべての `send` 操作 JMS メッセージのデフォルトの返信先として使用する JMS キューの JNDI 名を指定します。デフォルト値はありません。`replyToQueueLocation` と `replyToTopicLocation` 引数は相互排他的で、一緒に設定することはできません。この引数は、JMS Web サービス・アセンブリや JMS トランスポートで使用できます。

JMS トランスポートでこの引数を使用する場合は、ケーブルで SOAP メッセージを送信するすべての Web サービス操作のデフォルトの返信先として使用する JMS キューの JNDI 名を指定します。

replyToTopicLocation

`replyToTopicLocation <String>`

すべての `send` 操作 JMS メッセージのデフォルトの返信先として使用する JMS トピックの JNDI 名を指定します。デフォルト値はありません。`replyToQueueLocation` と `replyToTopicLocation` 引数は相互排他的で、一緒に設定することはできません。

sendConnectionFactoryLocation

`sendConnectionFactoryLocation <String>`

JMS `send` 操作のためのコネクションの取得に使用する JMS `ConnectionFactory` の JNDI 名を指定します。`ConnectionFactory` のタイプは、`send` 宛先との整合性が必要とされます。デフォルト値はありません。この引数は、JMS Web サービス・アセンブリや JMS トランスポートで使用できます。

JMS トランスポートにこの引数を使用する場合は、ケーブルで SOAP メッセージを送信する Web サービス操作のコネクションの取得に使用する JMS `ConnectionFactory` の JNDI 名を指定します。

sendQueueLocation

`sendQueueLocation <String>`

JMS `send` 操作に使用する JMS キューの JNDI 名を指定します。デフォルト値はありません。`sendQueueLocation` と `sendTopicLocation` 引数は相互排他的で、一緒に設定することはできません。この引数は、JMS Web サービス・アセンブリや JMS トランスポートで使用できます。

JMS トランスポートでこの引数を使用する場合は、ケーブルで SOAP メッセージを送信する Web サービス操作に使用する JMS キューを指定します。

sendTopicLocation

`sendTopicLocation <String>`

`send` 操作に使用する JMS トピックの JNDI 名。デフォルト値はありません。`sendQueueLocation` と `sendTopicLocation` 引数は相互排他的で、一緒に設定することはできません。

timeToLive

`timeToLive <int>`

この引数の値には、JMS `send` 操作のデフォルトの time-to-live（TTL）値を指定します。`timeToLive` の値は秒単位で表し、相対値です。絶対値ではありません。メッセージの送信時に、その有効期限が JMS `send` メソッドに指定された TTL 値と現行グリニッジ標準時（GMT）の値の合計として算定されます。この引数を設定しないか 0 に設定した場合は、メッセージは失効しません。

topicDurableSubscriptionName

`topicDurableSubscriptionName <String>`

この引数の値を一意の ID として使用し、トピックの `receive` 操作の永続サブスクリプションを登録します。デフォルト値はありません。

関連資料 :

永続サブスクリプションの詳細は、JMS 仕様を参照してください。

`http://java.sun.com/products/jms/docs.html`

プロキシ引数

この引数を使用して、生成されるプロキシ・コードの内容を制御できます。

- endpointAddress
- genJUnitTest

endpointAddress

`endpointAddress <URL>`

Web サービスへの接続に使用するエンドポイント・アドレスの URL を指定します。この引数を使用すると、WSDL のエンドポイント・アドレスは無視されます。

genJUnitTest

`genJUnitTest <true|false>`

`genJUnitTest` を `true` に設定すると、Web サービス内の公開対象メソッドごとに Junit テストが作成され、サービス・エンドポイント・インタフェースと同じディレクトリに格納されます。作成された Junit テストはテンプレートにすぎません。メソッドに実際のテスト・コードを供給する必要があります。デフォルト値は `false` です。

デプロイメント・ディスクリプタ引数

Web サービスのデプロイの実行方法の制御には、次の引数を使用できます。

- appendToExistingDDs
- context
- ddFileName
- uri

appendToExistingDDs

```
appendToExistingDDs <true|false>
```

新しい Web サービスのエントリを既存のデプロイメント・ディスクリプタに追加するのかまたはそれらを上書きするのかを設定します。この引数によって影響されるデプロイメント・ディスクリプタは、`oracle-webservices.xml`、`web.xml` および `webservices.xml` です。

この引数を `true` に設定すると、出力位置のデプロイメント・ディスクリプタはすべて、新しいサービスに必要なエントリで更新されます。`false` の場合、出力位置に既存のデプロイメント・ディスクリプタはすべて上書きされます。デフォルト値は `false` です。

関連項目：

`appendToExistingDDs` を使用して新しい Web サービスを既存のデプロイメント・ディスクリプタに追加する方法は、18-85 ページの「EAR または WAR アーカイブへの複数の Web サービスの割当て方法」を参照してください。

注意： `appendToExistingDDs` と `ddFileName` 引数は、呼出しまたは Ant タスクの中で一緒に使用できます。`ddFileName` 引数は、管理構成を格納するファイルを指します。デプロイメント・ディスクリプタを生成するサービスに対応する、このファイル内にあるすべての管理構成は、生成される `oracle-webservices.xml` ファイルに組み込まれます。`appendToExistingDDs` を `false` に設定すると（デフォルト）、出力ディレクトリ内のデプロイメント・ディスクリプタ・ファイル（`web.xml`、`oracle-webservices.xml` または `webservices.xml`）がすべて上書きされます。`appendToExistingDDs` を `true` に設定すると、WebServicesAssembler により、最新サービスのエントリが既存のデプロイメント・ディスクリプタ・ファイルに追加されます。

WebServicesAssembler では競合を解決しようとはしません。同じサービスを 2 回デプロイメント・ディスクリプタ・ファイルに追加すると、無効なデプロイメント・ディスクリプタになります。

---

context

context <*String*>

Web アプリケーションのルート・コンテキストを指定します。context には空の文字列は指定できません。context に値を指定しなかった場合、そのデフォルト値は次のように決定されます。

- 当該コマンドで appName 引数がサポートされている場合は、context のデフォルト値は appName 引数の値になります。appName 引数がサポートされていない場合は、デフォルト値は WSDL における最初のサービス名になります。
- 当該コマンドで WSDL、たとえば genApplicationDescriptor が使用されない場合、context のデフォルト値は、-web.war または .war 拡張子を除いた WAR ファイル名になります。

バージョン 2.1 EJB の場合、context の値は、oracle-webservices.xml デプロイメント・ディスクリプタ・ファイル内の <context-root> 要素に格納されます。

Web アプリケーションの場合、context の値は、application.xml デプロイメント・ディスクリプタ・ファイル内の <context-root> 要素に格納されます。

URL はプロトコル、ホストおよびポートを基に作成します。たとえば、次のようになります。

http://localhost:8888/*host*/*port*

プロトコル http://localhost:8888/ はデプロイ時に設定され、Web サービスへの接続に使用されます。2 番目の部分である *host* は、context 引数によって指定されます。3 番目の部分である *port* は、uri 引数によって指定されます。すなわち、URL 全体は、プロトコルを context および uri 引数の値と連結することで作成されます。

関連項目：

context および uri 引数を使用して HTTP URL を作成する例については、18-63 ページの「uri」を参照してください。

ddFileName

ddFileName <*String*>

Web サービスに割り当てる設定を含んだ oracle-webservices.xml デプロイメント・ディスクリプタを指定します。この引数を使用しなかった場合は、oracle-webservices.xml デプロイメント・ディスクリプタが生成されます。

*Assemble コマンドでこの引数を使用すると、管理およびカスタム・シリアライズの情報が指定されたファイルからアーカイブ内の oracle-webservices.xml ファイルにコピーされます。

WebServicesAssembler により、指定された名前のファイルが見つからない場合、処理が停止されます。

appendToExistingDDs と ddFileName 引数は、コマンドライン呼出しまたは Ant タスクの中で一緒に使用できます。

関連項目：

- このファイルの詳細は、付録 C「oracle-webservices.xml デプロイメント・ディスクリプタ・スキーマ」を参照してください。
- この引数と ddFileName を同じコマンドラインまたは Ant タスクで使用したときの影響の詳細は、18-61 ページの「appendToExistingDDs」を参照してください。

uri

`uri <String>`

Web サービスに使用する URI を指定します。これは、Web サービスへの接続に使用する URL の 3 番目の部分です。最初の部分はプロトコルで、デプロイ時に設定されます。2 番目の部分は、`context` 引数によって指定されます。すなわち、URL 全体は、プロトコル、コンテキストおよび uri の各引数を連結することで作成されます。

`uri` のデフォルト値は `appName` 引数の値です。`uri` には空の文字列は指定できません。

この引数を使用すると、`<url-pattern>` 要素が `web.xml` に（Web アプリケーションの場合）、または `<endpoint-address-uri>` 要素が `oracle-webservices.xml`（EJB 2.1 の場合）に追加されます。

WSDL アクセス引数

WSDL のアクセスに使用できる引数は、次のとおりです。WSDL が URL であり HTTP プロキシ・サーバーを使用するネットワーク上にある場合、`Http*` で始まる引数は WSDL へのアクセスに役立ちます。

- fetchWsdlImports
- httpNonProxyHosts
- httpProxyHost
- httpProxyPort
- importAbstractWsdl
- wsdl

fetchWsdlImports

`fetchWsdlImports <true|false>`

WSDL とそれにインポートされるものすべてについてローカル・コピーを作成するかどうかを指定します。`fetchWsdlImports` を `true` に設定すると、指定した WSDL とそれにインポートされるものすべてが WAR にコピーされます。`false` の場合は、指定した WSDL のみが WAR にコピーされます。デフォルト値は `false` です。

この引数は、通常、デプロイに伴って WSDL をパッケージ化する必要があるときに使用されます。これにより、各ランタイム呼出しに対するコールアウトでリモート・スキーマが必要とされる場合に発生する可能性のあるオーバーヘッド、信頼性の欠如、セキュリティ、変更の脅威に関する問題が回避されます。Web サービスに WSDL とそのスキーマ・インポートをパッケージ化するのは、トップダウン方式でのアセンブリの場合のみです。

httpNonProxyHosts

`httpNonProxyHosts <String>`

HTTP プロキシを使用しないホストの名前を指定します。ホストが複数ある場合は、「|」で区切ります。たとえば、`localhost|127.0.0.1` のようになります。この引数は、ホストが HTTP プロキシ・サーバーを使用するネットワーク上にある URL の場合に、WSDL を取得する際に役立ちます。

httpProxyHost

`httpProxyHost <String>`

HTTP プロキシ・ホストの名前を指定します。この引数は、ホストが HTTP プロキシ・サーバーを使用するネットワーク上にある URL の場合に、WSDL を取得する際に役立ちます。

httpProxyPort

`httpProxyPort <int>`

HTTP プロキシのポート番号を指定します。この引数は、ホストが HTTP プロキシ・サーバーを使用するネットワーク上にある URL の場合に、WSDL を取得する際に役立ちます。

importAbstractWsdl

`importAbstractWsdl <true|false>`

`wsdl` 引数で指定された WSDL を、生成される WSDL にインポートするかどうかを指定します。`true` の場合、WSDL ファイルが、生成される WSDL にインポートされます。`false` の場合は、生成される WSDL の要素と元の WSDL の要素が単一の出力ファイルに書き込まれます。デフォルト値は `false` です。

wsdl

`wsdl <String>`

WSDL ドキュメントの絶対ファイル・パス、相対ファイル・パスまたは URL を指定します。たとえば、`http://host:80/services/myservice?WSDL` と `myservice.wsdl` がこの引数に対する有効な値です。

ネットワークで HTTP プロキシ・サーバーが使用されており、WSDL が URL である場合は、必要に応じて、`httpNonProxyHosts`、`httpProxyHost` および `httpProxyPort` の各引数を設定します。

WSDL 管理引数

生成される WSDL の内容の制御には、次のオプションを使用できます。

- createOneWayOperations
- genQos
- qualifiedElementForm
- singleService
- soapVersion
- targetNamespace
- typeNamespace
- wsdlTimeout

createOneWayOperations

`createOneWayOperations <true|false>`

この引数を `true` に設定すると、`void` を戻すメソッドがレスポンス・メッセージを持たなくなります。指定しないか `false` に設定すると、レスポンス・メッセージは空になります。デフォルト値は `false` です。

genQos

genQos <true|false>

サービスのクオリティ（QOS）情報を機能アサーションの形式で WSDL に追加するかどうかを決定します。機能アサーションは、oracle-webservices.xml デプロイメント・ディスクリプタ内の Web サービス管理構成から導出されます。genQos を true に設定すると、機能アサーションが WSDL 内に生成されます。また、Web サービス管理構成を格納する oracle-webservices.xml デプロイメント・ディスクリプタに、ddFilename 引数を設定することも必要です。この引数のデフォルト値は false です。

関連資料：

Web サービスのサーバー・サイドおよびクライアント・サイドへの機能アサーションの提供方法の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「機能アサーションの使用」を参照してください。

qualifiedElementForm

qualifiedElementForm <true|false>

WSDL 内にある各生成済スキーマの elementFormDefault 要素が、qualified または unqualified のどちらに設定されているかを判別します。

elementFormDefault 要素の値は、ローカルに宣言されている要素を、インスタンス・ドキュメント内ではターゲット名前空間で修飾する必要があるかどうかを示します。この要素の値が unqualified である場合は、ローカルに宣言された要素をターゲット名前空間で修飾することはできません。値が qualified である場合は、ローカルに宣言された要素をターゲット名前空間で修飾する必要があります。

elementFormDefault 要素の値を unqualified に設定するには、qualifiedElementForm 引数を false に設定します。要素の値を qualified に設定するには、引数を true に設定します。qualifiedElementForm 引数のデフォルト値は true です。

singleService

singleService <true|false>

WSDL 内に定義された各サービスに対して単一ポートを生成するのか複数ポートを生成するのかを決定します。true の場合は、WebServicesAssembler により、複数のポートに対して単一のサービス（ポート・タイプごとに 1 つのサービス）が生成されます。false の場合は、WebServicesAssembler により、複数のサービス、すなわち単一ポートごとに 1 つのサービスが生成されます。デフォルトは false です。

soapVersion

soapVersion <1.1|1.2|1.1,1.2>

WSDL ドキュメントに対する SOAP のバージョンを指定します。使用可能な値は、1.1、1.2 または 1.1,1.2 です。デフォルト値は 1.1 です。

値 1.1,1.2 は、WebServicesAssembler が 2 つのバインディングとともに 2 つのポートを作成することを意味します。1 つのポートとバインディングがバージョン 1.1 をサポートし、もう 1 つのポートとバインディングがバージョン 1.2 をサポートします。各ポートは異なる URL にバインドする必要があります。つまり、同じ URL アドレスを使用して両方のバージョンを同時にサポートすることはできません。

targetNamespace

`targetNamespace <String>`

生成される WSDL 内に使用するターゲット名前空間を指定します。

`targetNamespace` の値は、仕様に準拠している HTTP URL、準拠していない HTTP URL、または URI であってもかまいません。`targetNamespace` の値は、次のようにしてパッケージ名にマップされます。

- ターゲットの名前空間が仕様に準拠する HTTP URL の場合は、Java パッケージ名にマップするときに、要素の順序が逆になります。次に例を示します。

  `http://hello.demo.oracle.com` というターゲット名前空間は、パッケージ名 `com.oracle.demo.hello` にマップします。

- ターゲットの名前空間が仕様に準拠しない HTTP URL の場合は、要素の順序は変更されません。次に例を示します。

  `http://hello.demo.oracle` というターゲット名前空間は、パッケージ名 `hello.demo.oracle` にマップします。

- ターゲット名前空間が URN の場合は、値は、コンポーネントがアンダースコア（_）で区切られた単一の識別子にマップされます。次に例を示します。

  `urn:oracle.demo.hello` というターゲット名前空間は、識別子 `oracle_demo_hello` にマップします。

ベスト・コーディング・プラクティスのためには、次の推奨事項に従ってください。

- コードのパッケージ名が準拠する HTTP URL にマップするときは（反転した場合）、その URL をターゲット名前空間として使用します。次に例を示します。

  パッケージ名 `com.oracle.demo.hello` に対しては、`-targetNamespace http://hello.demo.oracle.com` を使用します。

- コードのパッケージ名が準拠する HTTP URL にマップしないときは（反転した場合）、パッケージ名をそのまま変更しないでターゲット名前空間として使用します。次に例を示します。

  パッケージ名 `oracle.demo.hello` に対しては、`-targetNamespace http://oracle.demo.hello` を使用します。

注意： 次のように、URN を `targetNamespace` 引数に対する値としては使用しないでください。

```
-targetNamespace urn:oracle.demo.hello
```

このようにすると、サービスおよびサービス要素が 2 つの異なる場所にマップします。サービスは `oracle_demo_hello` にマップし、サービス要素は `oracle.demo.hello` にマップします。

意図的または非意図的に、サービスとサービス要素が異なるディレクトリに生成される可能性のある他の方法については、F-12 ページの「Web サービスとメッセージ部分が異なるディレクトリに生成される」を参照してください。

---

typeNamespace

`typeNamespace <String>`

生成される WSDL 内のスキーマ型に使用する型名前空間を指定します。指定した名前は必ず使用され、破棄されません。

関連項目：

この引数とデフォルトの名前空間 / パッケージ名マッピング規則の詳細は、18-72 ページの「ターゲット WSDL 名前空間とパッケージ名のマッピングのデフォルト・アルゴリズム」を参照してください。

wsdlTimeout

`wsdlTimeout <int>`

WebServicesAssembler がリモート WSDL リソースの HTTP または HTTPS リクエストへのレスポンスを待機する上限秒数を指定します。デフォルト値は 60 秒です。

WebServicesAssembler では、デフォルトで、ホストから WSDL 定義を求める HTTP リクエストへのレスポンスを受け取るのを 1 分間待機します。この引数を使用して、デフォルトの上限待機時間を上書きできます。値 0 は、WebServicesAssembler がプラットフォームによって設定される時間の間、待機するように指定します。

この引数は、HTTP または HTTPS を使用して行われたリクエストに対してのみ適用されます。リクエストが別のプロトコルを使用して行われている場合は無視されます。

メッセージ書式引数

生成される WSDL、つまり Web サービスによって使用されるメッセージ書式の制御には、次の引数を使用できます。

- mtomSupport
- style
- use

mtomSupport

`mtomSupport <true|false>`

ボトムアップ方式またはトップダウン方式の Web サービス・アセンブリに対し、この属性は、適切な内容（`base64binary`）のメッセージを MTOM 書式の添付ファイルとしてエンコードするかどうかを指定します。この属性のデフォルト値は `false` です。

この属性を使用すると、`<mtom-support>` 要素が `oracle-webservices.xml` ファイルに追加されます。

注意： この属性は、`assemble`、`genProxy` および `topDownAssemble` の各 Ant タスクのみと一緒に使用できます。WebServicesAssembler コマンドラインの引数としては使用できません。

関連資料：

この引数の使用方法の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の MTOM エンコードの添付ファイルの処理に関する項を参照してください。

style

`style <document-bare|document-wrapped|rpc>`

ボトムアップ方式 Web サービス・アセンブリの場合、この引数により、生成される WSDL 内のメッセージ書式の `style` 属性が指定されます。可能な値は、`document-bare`、`document-wrapped` および `rpc` です。デフォルト値は `document-wrapped` です。

関連項目：

rpc と document（wrapped および bare）メッセージ・スタイルの詳細は、5-2 ページの「OracleAS Web Services のメッセージ書式」を参照してください。

use

`use <literal|encoded>`

ボトムアップ方式 Web サービス・アセンブリの場合、この引数により、生成される WSDL 内のメッセージ書式の `use` 属性が指定されます。可能な値は、`literal` および `encoded` です。デフォルト値は `literal` です。

関連項目：

メッセージの使用方法である literal と encoded の詳細は、5-2 ページの「OracleAS Web Services のメッセージ書式」を参照してください。

Java 生成引数

Java ファイルの生成方法の制御には、次のオプションを使用できます。

- dataBinding
- mapHeadersToParameters
- overwriteBeans
- unwrapParameters
- valueTypeClassName
- valueTypePackagePrefix
- wsifEjbBinding
- wsifJavaBinding

dataBinding

`dataBinding <true|false>`

`true` の場合、WebServicesAssembler では、スキーマ内の全要素に対して、JAX-RPC デフォルト・タイプ・マッピング規則に従う Java 値タイプを生成しようとします。WebServicesAssembler ではサポートしていないタイプを検出した場合、`javax.xml.soap.SOAPElement` を使用してそのタイプを表現します。`false` の場合、WebServicesAssembler では、検出したすべてのスキーマ型に対して `SOAPElement` を使用します。デフォルト値は `true` です。

注意： OracleAS Web Services は、rpc-encoded メッセージ書式と `databinding=false` の組合せはサポートしていません。この組合せは業界内ではベスト・プラクティスとみなされていないためです。

---

mapHeadersToParameters

mapHeadersToParameters <true|false>

生成される Java コード内の各メソッド用のパラメータに、WSDL で定義されている SOAP ヘッダーをマップするかどうかを指定します。デフォルト値は true です。

overwriteBeans

overwriteBeans <true|false>

クラスパスにクラスがすでに存在する場合でも、スキーマ型用に Bean を生成するかどうかを指定します。デフォルト値は false です。

unwrapParameters

unwrapParameters <true|false>

この引数は document-literal 操作に対してのみ設定でき、その他のメッセージ書式では無視されます。通常、document-literal 操作には、単一入力スキーマ型と単一出力スキーマ型があります。これらのスキーマ型はラッパーとも呼ばれます。unwrapParameters を false に設定すると、生成対象のサービス・エンドポイント・インタフェースが、入力パラメータおよび戻り型をラップするラッパーとともに生成されます。たとえば、ラップされたパラメータのあるメソッドは次のようになります。

```
public EchoResponse echo(EchoRequest p) throws RemoteException;
```

unwrapParameters を true に設定すると（デフォルト）、戻り型とレスポンス・タイプがアンラップされます。通常はこの方が使いやすくなります。特にタイプが単純な場合はそうです。たとえば、アンラップされたパラメータのあるメソッドは次のようになります。

```
public String echo(String string) throws RemoteException;
```

valueTypeClassName

valueTypeClassName <*String*>

java.util.Collection および java.util.Map に使用される JAX-RPC 値タイプの完全修飾クラス名を指定します。この引数により、サービス・エンドポイント・インタフェースによって直接参照されないこれらのクラスのスキーマを生成できます。

> **関連項目：**
>
> valueTypeClassName を使用して Map または Collection にアイテムとして非組込み値タイプを宣言する例は、5-7 ページの「Oracle 固有の型のサポート」を参照してください。

valueTypeClassName 引数は、コマンドラインまたは Ant タスクで複数回使用できます。Ant タスクでは、複数のタグとして書き込み、name 属性を使用します。例 18-4 に、valueTypeClassName の複数インスタンスを使用して tClass1、tClass2 および tClass3 タイプ・クラスのスキーマを生成する方法を示します。

例 18-4 Ant タスクにおける valueTypeClassName の複数のインスタンス

```
<oracle:assemble appName="myService"
   output="build"
   input="myservice.jar"
   style="rpc"
   use="encoded">
            <oracle:porttype
              interfaceName="com.myCompany.myService.Hello"
              className="com.mycompany.HelloImpl"/>
            <oracle:valueTypeClassName name="tClass1"/>
            <oracle:valueTypeClassName name="tClass2"/>
            <oracle:valueTypeClassName name="tClass3"/>
</oracle:assemble>
```

valueTypePackagePrefix

`valueTypePackagePrefix <String>`

WSDL から生成するすべての値タイプのパッケージ名の接頭辞を指定します。この引数により、値タイプ・パッケージ名をサービス別にグループ化できます。

すべての値タイプ・クラスが、指定されたパッケージ名で開始されるようになります。値タイプのパッケージ名は、デフォルトで、スキーマの複合型の名前空間に基づいて付けられます。この引数を使用すると、名前空間から導出されたパッケージ名に接頭辞を付けられます。

たとえば、`valueTypePackagePrefix` を `myapp` に設定し、型のターゲット名前空間が `http://ws-i.org/` である場合、パッケージ名は `myapp.org.ws-i` になります。

WSDL 名前空間の Java パッケージへのマッピングをより詳細に制御するには、JAX-RPC マッピング・ファイルを使用します。名前空間のパッケージへのマッピングが JAX-RPC マッピング・ファイルに定義されている場合は、`valueTypePackagePrefix` 引数は無視されます。

関連項目：

- `valueTypePackagePrefix` 引数の使用方法の詳細は、18-75 ページの「ルート・パッケージ名の指定方法」を参照してください。
- WSDL 名前空間と Java パッケージ名のマッピングの詳細は、18-72 ページの「ターゲット WSDL 名前空間とパッケージ名のマッピングのデフォルト・アルゴリズム」を参照してください。

wsifEjbBinding

`wsifEjbBinding <true|false>`

この引数は、ボトムアップ方式 Web サービス・アセンブリで、WSDL に WSIF EJB バインディングを追加する際に使用します。`true` の場合は、EJB ホーム・インタフェース `className` および `jndiName` を指定する必要もあります。WSDL には、SOAP バインディングに加えて、ネイティブの WSIF EJB バインディングが生成されます。デフォルト値は `false` です。

関連項目：

- `wsifEjbBinding` 引数が WSDL に追加する拡張の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「WSDL に対する WSIF EJB 拡張」を参照してください。
- WSIF EJB バインディングの定義をさらに細かく制御したり、複数のポートを指定したりできる引数の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「複数の EJB ポートに対する WSIF エンドポイントの構成」を参照してください。

`wsifEjbBinding`、`wsifJavaBinding` および `wsifDbBinding` 引数は、相互排他的です。1 つのコマンドラインまたは Ant タスクでこれらの引数が 2 つ以上使用されると、例外がスローされます。

wsifJavaBinding

`wsifJavaBinding <true|false>`

この引数は、ボトムアップ方式 Web サービス・アセンブリで、WSDL に WSIF Java バインディングを追加する際に使用します。`true` の場合は、Java 実装クラスの `className` を指定する必要もあります。WSDL には、SOAP バインディングに加えて、ネイティブの WSIF Java バインディングが生成されます。デフォルト値は `false` です。

関連項目：

- この引数が WSDL に追加する拡張の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「WSDL に対する WSIF Java 拡張」を参照してください。
- WSIF Java バインディングの定義をさらに細かく制御したり、複数のポートを指定したりできる引数の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「複数の Java ポートに対する WSIF エンドポイントの構成」を参照してください。

`wsifEjbBinding`、`wsifJavaBinding` および `wsifDbBinding` 引数は、相互排他的です。1 つのコマンドラインまたは Ant タスクでこれらの引数が 2 つ以上使用されると、例外がスローされます。

名前競合の解決

WSDL 1.1 仕様では、様々な要素定義が、WSDL ドキュメントの特定の名前空間内で同じ名前を持つことが許されています。このため、WSDL から Java へのマッピングの際に、名前の競合が発生する可能性があります。

JAX-RPC 仕様の 4.3.12 項では、マッピングされた Java 識別子に接尾辞を追加することで名前の競合を解決するルールの概要が指定されています。次のコマンドは、Java ファイルを生成するときにこのルールを利用しています。

- genInterface
- genProxy
- topDownAssemble

参考までに、JAX-RPC 仕様の内容を表 18-5 に示します。ここでは、WSDL から Java へのマッピングで名前の競合を防ぐためのルールと接尾辞が指定されています。名前の競合がない場合は、これらの接尾辞を使用する必要はありません。

表 18-5 名前競合のルール

WSDL/XML からマッピングされた Java の定義	マッピングされた Java 識別子に付加される接尾辞	例
`xsd:simpleType` または `xsd:complexType` の名前から名前が導出される、WSDL/XML に基づく Java クラスから Java 型へのマッピング	*_Type*	XML: `<xsd:complexType name="shared">` Java: `Shared_Type.java`

Java 列挙クラス（JAX-RPC 仕様の 4.2.4 項を参照）	*_Enumeration*	XML:
		`<xsd:simpleType name="shared">`
		`<xsd:restriction base="xsd:string">`
		`<xsd:enumeration value="foo"/>`
		`</xsd:restriction>`
		`</xsd:simpleType>`
		Java: `Shared_Enumeration.java`

`xsd:element` の名前から名前が導出される、WSDL/XML に基づく Java クラスから Java 型へのマッピング	*_Element*	XML: `<xsd:element name="shared">` Java: `Shared_Element.java`
サービス・エンドポイント・インタフェース	*_PortType*	XML: `<wsdl:portType name="shared">` `<wsdl:binding name="shared"...>` Java: service endpoint interface: `Shared_PortType.java`
生成されたサービス・インタフェース	*_Service*	XML: `<wsdl: service name="shared" ... >` Java: `Shared_Service.java`
例外クラス	*_Exception*	Java: `Shared_Exception.java`

ターゲット WSDL 名前空間とパッケージ名のマッピングのデフォルト・アルゴリズム

次の各項では、WebServicesAssembler により名前空間とパッケージ名のマッピングに使用されるデフォルト・アルゴリズムについて説明します。

- Java パッケージ名と WSDL 名前空間のマッピングのアルゴリズム
- WSDL 名前空間の Java パッケージ名へのマッピングのアルゴリズム
- 名前空間の指定方法
- ルート・パッケージ名の指定方法

Java パッケージ名と WSDL 名前空間のマッピングのアルゴリズム

WebServicesAssembler では、パッケージ名からデフォルトの型名前空間が構成されます。パッケージ名が標準の最上位のドメイン名または国際標準化機構（ISO）の国コードで始まる場合は、WebServicesAssembler によりパッケージ名が前後逆になり、HTTP URL 内に配置されます。そうでない場合は、パッケージ名がそのまま HTTP URL 内に配置されます。

注意： 標準の最上位のドメイン名は、Internet Corporation For Assigned Names and Numbers（ICANN）（`http://www.icann.org/tlds/`）によって策定されています。

ISO の国コードは、国際標準化機構（`http://www.iso.org/iso/en/ISOOnline.frontpage`）によって策定されています。

たとえば、パッケージ `com.oracle.mytypes` の型名前空間は、`http://mytypes.oracle.com/` になります。パッケージ `examples.chapter1` に対する型名前空間は、`http://examples.chapter1/` になります。

次の各項では、WebServicesAssembler のマッピング・アルゴリズムが Java のアーチファクトと型を WSDL アーチファクトと XML スキーマ型にマッピングする手順について説明します。

- Java アーチファクトの WSDL アーチファクトへのマッピング
- Java 型の XML スキーマ型へのマッピング

Java アーチファクトの WSDL アーチファクトへのマッピング

次の手順により、Java アーチファクトが WSDL アーチファクトにマッピングされます。

1. 生成された WSDL 内で使用されているターゲット名前空間を、WebServicesAssembler 引数の `targetNamespace` の値によって取得します。この引数の詳細は、18-66 ページの「targetNamespace」を参照してください。
2. JAX-RPC マッピング・ファイルで、Java `<package-mapping>` 要素の値を参照します。
3. サービス・エンドポイント・インタフェースの Java パッケージ名から、WSDL 名前空間の名前を導出します。

Java 型の XML スキーマ型へのマッピング

次の手順により、Java 型が XML スキーマ型にマッピングされます。

1. JAX-RPC マッピング・ファイルで、Java `<package-mapping>` 要素の値を参照します。
2. 生成された WSDL にあるスキーマ型の型名前空間を、WebServicesAssembler 引数の `typeNamespace` の値によって取得します。この引数の詳細は、18-67 ページの「typeNamespace」を参照してください。
3. 値タイプの Java パッケージから WSDL 名前空間を導出します。
4. WSDL に定義されている `targetNamespace` 属性の値を使用します。

WSDL 名前空間の Java パッケージ名へのマッピングのアルゴリズム

WSDL 名前空間の Java パッケージ名へのマッピングは、Java Architecture for XML Data Binding （JAXB）仕様のバージョン 2.0 のアルゴリズムに基づいています。この仕様については、次の Web サイトを参照してください。

`http://www.jcp.org/en/jsr/detail?id=222/`

この仕様に定義されているアルゴリズムに対する例外は、次のとおりです。

- JAXB 仕様のアルゴリズムでは、URI のスキーム部分を、それが `http` または `urn` である場合にのみ削除するよう規定しています。この実装では、削除するスキームのセットが拡張されています。このセットに含まれているのは、`http`、`https`、`ftp`、`mailto`、`file`、`nntp`、`telnet`、`ldap`、`nfs`、`urn` および `tftp` です。
- JAXB 仕様のアルゴリズムでは、URI にグローバリゼーション・サポート文字がある場合に取るアクションは規定されていません。この実装では、非 ASCII 文字は u*xxxx* の形式でエンコードされます。ここで、*xxxx* は 4 桁の UTF8 エンコーディングです。
- JAXB 仕様では、最上位のドメイン名が標準の国コードでない場合または最上位のドメイン名でない場合は、アルゴリズムのステップ 6 はスキップするように規定されています。この仕様では、この場合にどのような処理をするのかそれ以上は明確にされていません。この実装では、URI のホスト・ストリングは、最上位のドメイン名または国コードで終わっていない場合でも、デリミタ「.」でトークン化されます。

表 18-6 に、パッケージ名の名前空間へのマッピングの例をいくつか示します。

表 18-6 名前空間のパッケージ名へのマッピングの例

名前空間	パッケージ名
http://army.mil/	mil.army
http://oracle/j2ee/ws_example/	oracle.j2ee.ws_example
http://somecompany.jp/	jp.somecompany
http://toplink.oracle.com	com.oracle.toplink
http://www.acme.com/go/espeak.xsd	com.acme.go.espeak
urn:dime/types.xsd	dime.types_xsd

次の各項により、サービス・エンドポイント・インタフェースまたは値タイプ、および WSDL 内のこれらの各関連アーチファクトが Java 名および Java 型にマッピングされます。

WSDL のサービス・エンドポイント・インタフェースおよび関連エンドポイント・アーチファクトの Java パッケージおよびクラス名へのマッピング

次の手順により、WSDL のサービス・エンドポイント・インタフェースおよび関連アーチファクトが Java 名にマッピングされます。

1. サービス・エンドポイント・インタフェースの場合は、JAX-RPC マッピング・ファイルで `<service-interface-mapping>` 要素の値を参照します。このファイルの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「JAX-RPC マッピング・ファイル記述子」を参照してください。
2. 生成されたクラスの Java パッケージ名を、WebServicesAssembler 引数の `packageName` の値によって取得します。この引数の詳細は、18-47 ページの「packageName」を参照してください。
3. JAX-RPC マッピング・ファイルで、`<package-mapping>` 要素の値を参照します。
4. WSDL 名前空間から Java パッケージ名を導出します。

WSDL の値タイプおよび関連アーチファクトの Java 名および Java 型へのマッピング

次の手順により、WSDL の値タイプおよびその関連アーチファクトが Java 名および Java 型にマッピングされます。

1. JAX-RPC マッピング・ファイルで、<java-xml-type-mapping> 要素を参照します。このファイルの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「JAX-RPC マッピング・ファイル記述子」を参照してください。
2. スキーマ型がターゲット・ドキュメント（WSDL またはスキーマ）の targetNamespace と同じ名前空間で定義されている場合は、WebServicesAssembler の packageName 引数の値を使用します。この引数の詳細は、18-47 ページの「packageName」を参照してください。
3. JAX-RPC マッピング・ファイルで、Java <package-mapping> 要素の値を参照します。
4. WSDL 名前空間と WebServicesAssembler の valueTypePackagePrefix 引数の値から Java パッケージ名を導出します（18-70 ページの「valueTypePackagePrefix」を参照）。この引数の詳細は、18-75 ページの「ルート・パッケージ名の指定方法」も参照してください。

名前空間の指定方法

名前空間を明示的に指定するには、typeNamespace 引数を使用します。指定した名前は必ず使用され、破棄されません。

ルート・パッケージ名の指定方法

valueTypePackagePrefix 引数を使用すると、指定した WSDL のスキーマにあるすべての型のルート・パッケージ名を指定できます。すべてのパッケージ名が、この値で開始されるようになります。指定された値が使用されるのは、JAX-RPC マッピング・ファイルに型名前空間とパッケージのマッピングが宣言されていない場合にかぎります。

生成されるパッケージ名が同じ値で始まる場合は、この値はパッケージ名には追加されません。これにより、com.oracle.mytypes. com.oracle.mytypes のようなパッケージが避けられます。

データベース接続の確立方法

aqAssemble、dbJavaAssemble、plsqlAssemble および sqlAssemble コマンドでは、Web サービスの生成にデータベース接続が必要です。Web サービスのアセンブリ時および実行時に、データベースへの接続が行われることがあります。Web サービスのアセンブリ時では、WebServicesAssembler により、PL/SQL パッケージや AQ キューなどのデータベース・エンティティに関する情報を取得するためにデータベースへの接続が行われます。Web サービスの実行時では、ユーザー・アプリケーションにより、データソース JNDI 位置が使用され、データベース操作用に JDBC 接続が取得されます。この情報は、WebServicesAssembler により、確実に Web サービス・ランタイム・コードに提供されます。

次の引数を使用すると、Web サービスに接続情報を提供できます。

- dbUser
- dbConnection
- dataSource

アセンブリ時および実行時にデータベースにアクセスするには、コマンドラインまたは Ant タスクで dataSource 引数か dbConnection と dbUser の組合せを使用します。

- dbConnection と dbUser 引数は、通常、一緒に使用します。これらにより、アセンブリ時および実行時のデータベースへのアクセス用に、WebServicesAssembler に対して JDBC URL とデータベースのスキーマおよびパスワードが提供されます。
- dataSource 引数により、WebServicesAssembler によるアセンブリ時および実行時のデータベースへのアクセスを可能にする JNDI 位置が提供されます。
- これらの 3 つの引数がすべて Ant タスクまたはコマンドラインで使用されている場合、dbConnection と dbUser の値はアセンブリ時のデータベース・アクセスに、dataSource は実行時のデータベース・アクセスに使用されます。

WebServicesAssembler の追加 Ant サポート

この項では、Ant タスクを通じて WebServicesAssembler で使用可能なその他の機能について説明します。この機能は、WebServicesAssembler コマンドラインでは使用できません。

- Ant における引数の複数インスタンスの使用方法
- Ant タスクでのプロキシ生成の構成方法
- Ant タスクでのポートの構成方法
- Ant タスクでのポート・タイプの構成方法
- Ant タスクでのハンドラの構成方法
- アーカイブにファイルを追加する方法
- WebServicesAssembler ビルドの制御方法
- MTOM エンコード添付ファイルのサポートを Web サービスにアセンブルする方法

Ant における引数の複数インスタンスの使用方法

WebServicesAssembler コマンドの次の引数は、コマンドラインまたは Ant タスクで複数回宣言できます。

- input
- schema
- sqlstatement
- valueTypeClassName

Ant タスクでは、これらの引数の複数のインスタンスを別個のタグとしてリストする必要があります。別個のタグとしてリストする際、input、schema および sqlstatement 引数には value 属性が必要になります。valueTypeClassName 引数では name 属性が必要となります。

たとえば、次の assemble コマンドでは、input 引数の複数インスタンスを使用して、first.jar、second.jar および third.jar が含まれる EAR ファイルが生成されます。

```
<oracle:assemble
          appName="myService"
          output="build">
             <oracle:PORTTYPE
               interfaceName="com.myCompany.myService.Hello"
               className="com.mycompany.HelloImpl">
             </oracle:porttype>
             <oracle:input value="first.jar"/>
             <oracle:input value="second.jar"/>
             <oracle:input value="third.jar"/>
</oracle:assemble>
```

次の jmsAssemble コマンドでは、valueTypeClassName 引数の複数インスタンスを使用して、JAX-RPC 値タイプのクラス名を指定しています。これらのクラス名は、java.util.Collection および java.util.Map の中のアイテムにできます。

```
<oracle:jmsAssemble
       linkReceiveWithReplyTo="true"
       targetNamespace="http://oracle.j2ee.ws/jms-doc"
       typeNamespace="http://oracle.j2ee.ws/jms-doc/types"
       serviceName="JmsService"
       appName="jms_service"
       context="jms_service"
       input="./demo/build/mdb_service.jar"
       output="./demo/dist"
       >
         <oracle:valueTypeClassName name="tClass1"/>
         <oracle:valueTypeClassName name="tClass2"/>
         <oracle:valueTypeClassName name="tClass3"/>
</oracle:jmsAssemble>
```

Ant タスクでのプロキシ生成の構成方法

proxy サブタグにより、サーバー・コードのアセンブリと同時にクライアント・プロキシを生成できます。WSDL の名前が必ずしも事前に認識されないボトムアップ方式アセンブリ時には、これが最も便利です。Ant タスク内で proxy サブタグを使用できるのは、次のコマンドに対してです。

- aqAssemble
- assemble
- corbaAssemble
- dbJavaAssemble
- ejbAssemble
- jmsAssemble
- plsqlAssemble
- sqlAssemble
- topDownAssemble

proxy サブタスクには、genProxy WebServicesAssembler コマンドに類似した機能があります。サポートされている genProxy 引数は、wsdl を除いてどれも、proxy サブタグで属性として使用できます。

output 引数は属性としてサポートされていますが、genProxy で使用される場合とは異なり、必須ではありません。output を proxy サブタグに指定しなくても、親タグで指定された output の値が、src/proxy ディレクトリ・パスの前に追加されて使用されます。たとえば、親タグで output="build" が設定されている場合、proxy サブタグの出力は build/src/proxy に格納されます。

次の引数を proxy サブタグの属性として使用できます。

- classpath
- dataBinding
- ddFileName
- endpointAddress
- genJUnitTest
- mapHeadersToParameters
- mappingFileName

- output
- overwriteBeans
- packageName
- replyToConnectionFactoryLocation
- replyToQueueLocation
- searchSchema
- unwrapParameters
- useDimeEncoding
- valueTypePackagePrefix

例 18-5 に、packageName 属性を付けた <proxy> サブタグの使用例を示します。この例では、assemble を親タグとして使用しています。

例 18-5 Ant タスクでの proxy サブタグの使用方法

```
<oracle:assemble...>   (or any command that supports the proxy tag)
  <oracle:proxy packageName="myproxy"/>
</oracle:assemble>
```

プロキシへのハンドラ情報およびポート情報の生成方法

<handler> および <port> タグを使用して、メッセージ処理情報およびポート固有情報をプロキシに追加できます。

- <handler>: リクエストを、リモート・ホストに送信する前かまたはクライアントがレスポンスを処理する前に調べて必要に応じて変更するため、<proxy> のサブタグとして <handler> タグを構成できます。<handler> タグの詳細は、18-81 ページの「Ant タスクでのハンドラの構成方法」を参照してください。
- <port>: 特定ポートに対してプロキシを生成するため、<proxy> のサブタグとして <port> タグを構成できます。<port> タグの詳細は、18-78 ページの「Ant タスクでのポートの構成方法」を参照してください。

Ant タスクでのポートの構成方法

ポートは、Web サービスをホストするエンドポイントのネットワーク・アドレスを識別します。一部の WebServiceAssembler コマンドでは、Ant タスクで port タグを子として指定できます。これにより、ポートごとに異なる構成を持つことができるようになります。たとえば、2 種類のトランスポートや 2 種類の SOAP バージョンを割当てできます。

次の Ant タスクには、port タグを指定できます。

- aqAssemble
- assemble
- corbaAssemble
- dbJavaAssemble
- ejbAssemble
- jmsAssemble
- plsqlAssemble
- sqlAssemble
- topDownAssemble
- genWsdl
- genDDs

port タグでは次の引数を使用できます（注意 : Ant タスクのコンテキストでは、引数は「属性」と呼ばれます）。

- bindingName
- endpointAddress
- portName（または name）
- replyToConnectionFactoryLocation
- replyToQueueLocation
- sendConnectionFactoryLocation
- sendQueueLocation
- soapVersion
- uri

port タグでは、これらの引数以外に、name 引数も指定できます。この引数には、ポート名のローカル部分を指定します。

port タグ内では、各コマンドのオプションの引数は指定してもしなくてもどちらでもかまいません。各コマンドでは、様々な引数サブセットがサポートされています。特定のコマンドで使用できる引数のリストについては、そのコマンドの「追加 Ant サポート」の項を参照してください。

例 18-6 に、port タグを使用して様々なポートに JMS および HTTP トランスポートを指定する方法を示します。HTTP トランスポート用の port 宣言では name 属性を使用していることに注意してください。name 属性は、Ant タスクで port を 1 つのみ指定する場合にはオプションです。Ant タスクで複数の port タグを指定する場合は、name 属性が必要です。

例 18-6 様々なポートへの各種トランスポートの割当て

```
<oracle:port
      uri="/echo"
      sendQueue="jms/senderQueue"
      sendConnectionFactoryLocation="jms/senderQueueConnectionFactory"
      replyToConnectionFactoryLocation="jms/receiverQueueConnectionFactory"
      replyToQueue="jms/receiverQueue"/>
<oracle:port uri="echo2" name="EchoHttpPort"/>
```

例 18-7 に、port タグを使用して様々なポートに各種 SOAP メッセージ・バージョンを指定する方法を示します。同じ Ant タスクに複数のポートを指定する場合は、各 port タグで name 属性が必須になります。

例 18-7 様々なポートへの各種 SOAP メッセージ・バージョンの割当て

```
<oracle:assemble
   ...
      <oracle:port uri="soap11" soapVersion="1.1" name="httpSoap11Port" />
      <oracle:port uri="soap12" soapVersion="1.2" name="httpSoap12Port" />
   ...
/>
```

Ant タスクでのポート・タイプの構成方法

一部の Ant タスクでは、子タスクとして porttype タグを指定できます。これにより、Web サービスへのインタフェースを複数の種類、構成できます。これらの WebServicesAssembler コマンドの Ant タスクでは、porttype を、子タスクとして指定できます。

- assemble
- genDDs
- genWsdl
- topDownAssemble

porttype タグでは、Web サービスのインタフェースを指定するための interfaceName 引数が必須になります。複数の <porttype> タグを指定する場合、各タグには <port> サブタグが含まれている必要があります。

関連項目：

port タグの詳細は、18-78 ページの「Ant タスクでのポートの構成方法」を参照してください。

assemble、topDownAssemble、genDDs および genWsdl コマンドで porttype を構成するときは、className 属性を指定する必要があります。assemble および topDownAssemble コマンドの場合は、className 属性ではなく classFileName 属性を指定します。

また、WSDL に複数のポート・タイプの参照が含まれている場合は、topDownAssemble コマンドでは、ポート・タイプごとに <porttype> タグを指定する必要があります。

表 18-7 に、<porttype> タグで使用できる有効な属性とサブタグをまとめます。

表 18-7 <porttype> タグの属性とサブタグ

属性とサブタグ	説明
classFileName 属性	この属性は、className が指定されていない場合に、assemble および topDownAssemble コマンドに対して指定できます。className 引数に指定した実装クラスの Java ファイル名を指定します。
className 属性	この属性の指定が必須なのは、assemble、topDownAssemble、genDDs および genWsdl コマンドです。Web サービスの実装クラスのクラス名（パッケージ名を含む）を指定します。
interfaceName 属性	interfaceName は必須属性です。Web サービスへのインタフェースを指定します。
<port> サブタグ	<port> サブタグが必須となるのは、複数の <porttype> タグを指定する場合です。Web サービスのインタフェースに適用されるポートを指定します。指定する <porttype> タグが 1 つのみの場合は、<port> タグは不要です。

例 18-8 に、Web サービスへのポート・タイプの割当て方法を示します。

例 18-8 Web サービスへのポート・タイプの割当て

```
...
<oracle:porttype
   interfaceName="my.company.MyInterface"
   className="my.company.MyImpl">
</oracle:porttype>
...
```

Ant タスクでのハンドラの構成方法

この項の内容は、次のとおりです。

WebServicesAssembler により、Ant タスクでハンドラおよびクライアント・ハンドラを構成できます。Ant タスクを使用して、Enterprise Web Services 1.1 仕様によってハンドラ用に策定されている情報をすべて構成できます。

注意： ハンドラおよびクライアント・ハンドラは、Ant タスクでのみ宣言および構成できます。コマンドラインでは宣言または構成できません。

ハンドラは、リクエスト、レスポンスまたは障害を、Web サービス・コンポーネントによる処理前に調べて、必要に応じて変更することができます。また、リクエストがコンポーネントによって処理された後でレスポンスまたは障害を調べて、必要に応じて変更することもできます。クライアント・ハンドラは、その名前が示すように、クライアント上で実行されますが、実行されるタイミングは、リモート・ホストにリクエストが送信される前、またはクライアントによってレスポンスが処理される前です。

`handler` タグにより、WebServicesAssembler Ant タスクでのハンドラを定義します。属性により、ハンドラの詳細を定義します。このタグと属性は、Enterprise Web Services 1.1 仕様によって策定された `<handler>` タグとそのサブ要素に対応しています。Ant タスク内でハンドラとその属性を宣言すると、対応する要素が `webservices.xml` に設定されます。

handler タグの属性と子タグ

次の各項では、`handler` タグで使用できる属性と子タグについて説明します。

class（属性）

`class="`*`class_name`*`"`

この属性により、ハンドラ実装の完全修飾クラス名を定義します。

この属性を使用すると、`webservices.xml` に `<handler-class>`*`class_name`*`</handler-class>` 要素が設定されます。

initparam（子タグ）

```
<oracle:initparam name="myName" value="myValue" />
```

この子タグにより、ハンドラが使用する初期化パラメータの名前と値のペアを定義します。name 属性には、パラメータ名を指定します。各パラメータ名は Web アプリケーションで一意である必要があります。value 属性には、対応するパラメータの値を指定します。

複数の initparam 宣言を、handler Ant タスク起動の子タスクとして定義できます。

initparam 子タグを使用すると、webservices.xml に次の <init-param> 構造が設定されます。

```
<init-param>
    <param-name>myName</param-name>
    <param-value>myValue</param-value>
</init-param>
```

name（属性）

```
name="handler_name"
```

この属性により、ハンドラの名前を定義します。名前はモジュール内で一意である必要があります。

この属性を使用すると、webservices.xml に <handler-name>*handler_name*</handler-name> 要素が設定されます。

soapheader（子タグ）

```
<oracle:soapheader value="{namespace_URI}local_part"/>
```

この子タグにより、ハンドラによって処理される SOAP ヘッダーの QName を定義します。value 属性には、QName のローカル部分および名前空間 URI を指定します。複数の soapheader 宣言を、handler Ant タスクの子タスクとして定義できます。

soapheader 子タグの値は、webservices.xml の soap-header タグに書き込まれます。たとえば、名前空間 URI が http://oracle.j2ee.ws/Header で、ローカル部分が authenticateHeader である場合は、soapheader 子タグには次の値を設定します。

```
<oracle:soapheader value="{http://oracle.j2ee.ws/Header}authenticateHeader"/>
```

この値に対して、webservices.xml ファイル内に次の表現が設定されます。

```
<soap-header
xmlns:wsa1="http://oracle.j2ee.ws/Header">wsa1:authenticateHeader</soap-header>
```

この例において、wsa1 は、指定する名前空間の一意の接頭辞です。

soaprole（属性）

```
soaprole ="Some_SoapRole"
```

この属性により、ハンドラの機能として SOAP アクターを定義します。

この属性を使用すると、webservices.xml に <soap-role>*Some_SoapRole*</soap-role> 要素が設定されます。

soapRole（子タグ）

この子タグは、soaprole 属性と同じ動作です。ハンドラの機能として SOAP アクターを定義します。ただし、属性とは異なり、soapRole 子タグを使用すると、ハンドラに対して複数の SOAP ロールを指定できます。

たとえば、次の Ant タスク・フラグメントは、soapRole サブタグを使用して、ハンドラがロールとして実行する 2 つの SOAP アクターを定義しています。

```
...
<oracle:handler soapRole="SomeSOAPRole" name="StaticStateHandlerName"
                class="oracle.j2ee.ws.tools.wsa.handler.StaticStateHandler">
   <oracle:soapRole name="another-soap-role"/>
   <oracle:soapRole name="yet-another-soap-role"/>
</oracle:handler>
...
```

ハンドラ構成のサンプル

Ant タスクで使用できるハンドラ構成を構成する方法を説明するため、例 18-9 に、StaticStateHandlerName ハンドラのサンプル構成を示します。このハンドラは myApp.handler.StaticStateHandler クラスによって実装され、2 つの初期化パラメータ SCOTT および TIGER を使用します。このハンドラでは、authenticateHeader と authenticateHeader2 の 2 種類の SOAP ヘッダーが処理されます。

例 18-9 ハンドラ構成のサンプル

```
<some tag that supports handlers>
 <oracle:handler soaprole="Some_SoapRole"
   name="StaticStateHandlerName"
   class="myApp.handler.StaticStateHandler">
        <oracle:initparam name="id" value="SCOTT"/>
        <oracle:initparam name="password" value="TIGER"/>
   <oracle:soapheader value="{http://oracle.j2ee.ws/Header}authenticateHeader"/>
   <oracle:soapheader value="{http://oracle.j2ee.ws/Header2} authenticateHeader2"/>
 </oracle:handler>
</some tag that supports handlers>
```

ハンドラを構成できる Ant タスク

次の WebServicesAssembler コマンドの Ant タスクでは、handler タグを子タスクとして指定できます。

- aqAssemble
- assemble
- corbaAssemble
- dbJavaAssemble
- ejbAssemble
- jmsAssemble
- plsqlAssemble
- sqlAssemble
- topDownAssemble
- genProxy
- genDDs（サーバー・サイド専用として handler タグを指定可能）

Ant タスクでの複数ハンドラの構成

ハンドラをサポートする任意のコマンドに対して、複数のハンドラを指定できます。ハンドラごとに異なる構成を設定してもかまいません。

例 18-10 に、複数ハンドラの構成を示します。親タグには、ハンドラをサポートする任意のタグを指定できます。

- 最初の handler タグにより、StaticStateHandlerName ハンドラが構成されます。このハンドラは myApp.handler.StaticStateHandler クラスによって実装され、2 つの初期化パラメータ SCOTT および TIGER を使用します。このハンドラでは、authenticateHeader と authenticateHeader2 の 2 種類の SOAP ヘッダーが処理されます。
- 2 番目の handler タグは、きわめて基本的な handler タグです。myapp.handler.MyOtherHandler クラスによって実装された MyOtherHandler ハンドラが構成されます。

例 18-10 複数ヘッダー構成のサンプル

```
<some tag that supports handlers>
   <handler soaprole="Some_SoapRole" name="StaticStateHandlerName"
        class="myApp.handler.StaticStateHandler">
     <initparam name="id" value="SCOTT"/>
     <initparam name="password" value="TIGER"/>
     <soapheader namespace="http://oracle.j2ee.ws/Header"
          localpart="authenticateHeader"/>
     <soapheader namespace="http://oracle.j2ee.ws/Header2"
         localpart="authenticateHeader2"/>
   </handler>
   <handler name="MyOtherHandler" class="myApp.handler.MyOtherHandler"/>
</some tag that supports handlers>
```

アーカイブにファイルを追加する方法

EAR または WAR アーカイブにファイルを追加するには、WebServicesAssembler によりステージング領域として使用されているディレクトリにそのファイルをコピーします。ステージング領域とは、WebServicesAssembler が *Assemble タスクをコールする前に jar でアーカイブを生成する場所です。ステージング領域のレイアウトの詳細は、図 18-1 を参照してください。

デフォルトのステージング領域は、WAR アーカイブでは <*output*>/war、EAR アーカイブでは <*output*>/ear です。この <*output*> 変数は、output 引数の値を意味しています。たとえば、output 引数の値が outputDir である場合は、各ファイルが outputDir/war および outputDir/ear に格納されます。

たとえば、次の Ant タスクでは、生成済の EAR にキーストア oraks.jks が格納されます。

```
<copy file="oraks.jks" todir="build/ear/META-INF/>
<oracle:assemble output="build"
                ...
 />
```

WebServicesAssembler ビルドの制御方法

WebServicesAssembler のブール型引数 `failonerror` を使用すると、エラーがあってもビルドが続行されます。`failonerror` の値が `true` の場合は、WebServicesAssembler でエラーが発生すると、ビルドは失敗します。値が `false` の場合は、ビルドは続行されます。デフォルト値は `true` です。

`failonerror` 引数は任意の WebServicesAssembler コマンドで使用できます。次の例では、assemble コマンドでエラーが発生した場合でも、処理が続行されます。

```
<oracle:assemble failonerror="false"
          ...
/>
```

MTOM エンコード添付ファイルのサポートを Web サービスにアセンブルする方法

`mtomSupport` Ant 属性を使用すると、MTOM エンコード・メッセージのサポートを Web サービスおよび Web サービス・クライアントにアセンブルできます。この属性は、Ant タスクでのみ使用でき、WebServicesAssembler コマンドラインでは使用できません。

`mtomSupport` 属性は、次の Ant タスクで使用できます。

- assemble
- genProxy
- topDownAssemble

関連項目：

`mtomSupport` 属性およびその使用方法の詳細は、次の項を参照してください。

- 18-67 ページの「mtomSupport」
- 『Oracle Application Server Web Services アドバンスト開発者ガイド』の MTOM エンコードの添付ファイルの処理に関する項

EAR または WAR アーカイブへの複数の Web サービスの割当て方法

1 つのアーカイブ（EAR または WAR）に複数の Web サービスを割り当てるには、WAR に入れる Web サービスごとに `assemble` または `topDownAssemble` タスクをコールする必要があります。各アセンブル・タスクに対して設定する引数は、次の規則に従う必要があります。

- すべての `output` 引数に対して同じ値を指定する必要があります。
- 最後の `assemble` タスクのみに `ear` 引数を指定できます。この理由は、`ear` 引数を指定すると、WAR ステージング・ディレクトリ（`/war`）の内容がアーカイブされ、EAR に格納されるためです。アーカイブが作成されると、ステージング領域内のファイルは削除されます。
- 最後のタスクを除くすべてのタスクの `war` 引数には、`output` 引数の値をディレクトリ名 `/war` の前に追加した値を指定する必要があります。これは、WebServicesAssembler のデフォルト動作では、出力ディレクトリ *`directory`*`/war` が WAR のステージング領域として使用されるためです。たとえば、`output` 引数を `dist` に設定する場合は、`war` 引数を `dist/war` に設定する必要があります。
- 最初のタスクを除くすべてのタスクにおいて、`appendToExistingDDs` を `true` に設定する必要があります。最初の `assemble` タスクで `appendToExistingDDs` を `true` に設定できるのは、出力先の *`directory`*`/war` に、WebServicesAssembler に変更させるデプロイメント・ディスクリプタ（たとえば、リソース参照付きの `web.xml`）がすでに格納されている場合のみです。

- 最後の assemble タスクで war 引数を指定するのは、WebServicesAssembler に EAR ではなく WAR を作成させる場合にかぎります。appName 引数は、すべての assemble タスクで一意である必要があります。
- uri 引数を指定する場合は、その値がすべての assemble タスクで一意である必要があります。

例 18-11 に、EAR ファイルの myApps.ear に、2 つの Web サービスである firstApp および nextApp を割り当てる Ant タスクを示します。このコード例の詳細は次のとおりです。

- 2 つの assemble コマンドの output 引数には、出力ディレクトリ・パスと同じ値 ${out.dir} が割り当てられています。
- 最初の assemble コマンドでは、出力が /war ディレクトリに格納されていますが、最後の assemble 引数では、2 つの Web サービスを格納する EAR ファイルが指定されています。
- 最後の Web サービスでは appendToExistingDDs が指定され、デプロイメント・ディスクリプタは、直前のコマンドによって生成されたディスクリプタに追加されるようになっています。

省略記号はさらに Ant コマンドがあることを示します。

例 18-11 EAR に 2 つの Web サービスを割り当てる Ant タスク

```
<oracle:assemble appName="firstApp"
          output="${out.dir}"
          war="${out.dir}/war"
            ...
          />
      <oracle:assemble appName="nextApp"
          output="${out.dir}
          appendToExistingDDs="true"
          ear="myApps.ear"
            ...
          />
```

例 18-12 に、EAR ファイルの myApps.ear に、3 つの Web サービスである firstApp、nextApp および lastApp を割り当てる Ant タスクを示します。このコード例の詳細は次のとおりです。

- 3 つの assemble コマンドの output 引数には、出力ディレクトリ・パスと同じ値 ${out.dir} が割り当てられています。
- 最初の 2 つの assemble コマンドでは、出力が /war ディレクトリに格納されていますが、最後の assemble 引数では、3 つの Web サービスを格納する EAR ファイルが指定されています。
- また、2 番目と最後の Web サービスでは appendToExistingDDs も指定されており、デプロイメント・ディスクリプタは、直前のコマンドによって生成されたディスクリプタに追加されるようになっています。

省略記号はさらに Ant コマンドがあることを示します。

例 18-12 EAR に 3 つの Web サービスを割り当てる Ant タスク

```
<oracle:assemble appName="firstApp"
          output="${out.dir}"
          war="${out.dir}/war"
            ...
          />
      <oracle:assemble appName="nextApp"
          output="${out.dir}
          appendToExistingDDs="true"
          war="${out.dir}/war"
            ...
          />
```

```
<oracle:assemble appName="lastApp"
    output="${out.dir}
    appendToExistingDDs="true"
    ear="myApps.ear"
      ...
    />
```

WAR ファイルに複数 Web サービスを割り当てる際の制限事項

WebServicesAssembler では、1 つの WAR に複数の Web サービスを追加する際の競合はチェックされません。次の競合により無効なアセンブリが生じますが、これはデプロイ時または実行時まで検出されません。

- 様々な Web サービスに、同じ名前の WSDL ファイルがある。

  様々な Web サービスの WSDL ファイルが同じ名前の場合、ある WSDL ファイルがその他の WSDL ファイルを上書きします。この競合を避けるため、様々な Web サービスにある WSDL ファイルの名前が一意になるようにします。

- 様々な Web サービスに、同じ名前のクラス・ファイルがある。

  様々な Web サービスでクラス・ファイルに同じ名前（パッケージ名を含む）が使用されている場合、クラスのいずれかによりその他のクラスが上書きされます。これらのファイルは、実際には同一である場合もあります。たとえば、同じスキーマ型の値タイプ・クラスは、2 つの異なるサービス間で共有できます。この種の競合では問題は発生しません。

  同一になりえないクラスには、サービス・エンドポイント・インタフェースと実装クラスがあります。クラス・ファイルが同一でない場合は、名前を変更するか別のパッケージに格納します。

WSDL での Java メソッド・パラメータ名の表現方法

Java クラスまたは EJB からボトムアップ方式で Web サービスを生成すると、WebServicesAssembler により WSDL が生成されます。WebServicesAssembler は、生成する WSDL 内の Java メソッドの `<element name="..." />` 属性に対して、メソッドの実パラメータの名前を使用しようとします。WSDL および SOAP メッセージでメソッドの実パラメータの名前を使用すると、パラメータとその対応する要素の判別が容易になります。

WebServicesAssembler では、次の方法、手順を使用してメソッドの実パラメータ名が取得されます。

1. インタフェース・ソース・ファイルを指定すると、そのコードが、パラメータ名がないかどうか解析されます。それには、Java インタフェース・ファイルのフルパス名を `interfaceFileName` 引数で指定する必要があります。
2. インタフェース・ソース・ファイルを指定しない場合は、インタフェースのメソッドを実装する Java クラス・ファイルがロードされ、解析されます。パラメータ名を抽出できるように、クラス・ファイルを、`javac` の `-g` オプションでコンパイルしておく必要があります。

この方法でパラメータ名を取得できなかった場合、あるいはクラスがロード不能になっているかまたは曖昧化している場合は、パラメータのデータ型と数値を組み合せた文字列（たとえば `string_1`）がデフォルトで使用されます。

次のメソッドを表す WSDL フラグメントを、例 18-13 および例 18-14 に示します。

```
public String sayHello(String name)
```

例 18-13 に示すのは、WebServicesAssembler が sayHello メソッドのパラメータ名を取得できた WSDL フラグメントです。<element name="..."/> 属性とその値が太字で強調表示されています。

例 18-13 生成されたパラメータ名のある WSDL フラグメント

```
<definitions name="HelloService" targetNamespace="http://hello.demo.oracle/">
    <types>
    <schema elementFormDefault="qualified" targetNamespace="http://hello.demo.oracle/">
    <complexType name="sayHello">
       <sequence>
          <element name="name" nillable="true" type="string"/>
       </sequence>
    </complexType>
...
```

例 18-14 に示すのは、sayHello メソッドのパラメータ名が見つからなかった WSDL フラグメントです。element name の属性には、値 string_1 が使用されます。<element name="..."/> 属性とその値が太字で強調表示されています。

例 18-14 デフォルトのパラメータ名のある WSDL フラグメント

```
<definitions name="HelloService" targetNamespace="http://hello.demo.oracle/">
    <types>
    <schema elementFormDefault="qualified" targetNamespace="http://hello.demo.oracle/">
    <complexType name="sayHello">
       <sequence>
          <element name="string_1" nillable="true" type="string"/>
       </sequence>
    </complexType>
...
```

制限事項

F-11 ページの「WebServicesAssembler の使用方法」を参照してください。

追加情報

詳細は、次を参照してください。

- WSDL からの Web サービスのアセンブル方法は、第 6 章「WSDL からの Web サービスのアセンブル」を参照してください。
- ステートフルな Web サービスのアセンブル方法は、第 7 章「Java クラスを使用した Web サービスのアセンブル」を参照してください。
- EJB からの Web サービスのアセンブル方法は、第 8 章「EJB を使用した Web サービスのアセンブル」を参照してください。
- JMS キューまたは JMS トピックからの Web サービスのアセンブル方法は、第 9 章「JMS 宛先を使用した Web サービスのアセンブル」を参照してください。
- データベース・リソースからの Web サービスのアセンブル方法は、第 10 章「データベース Web サービスのアセンブル」を参照してください。
- J2SE 5.0 注釈を使用した Web サービスのアセンブル方法は、第 11 章「注釈を使用した Web サービスのアセンブル」を参照してください。
- REST Web サービスのアセンブル方法は、第 12 章「REST Web サービスのアセンブル」を参照してください。
- J2EE クライアントの構築方法は、第 14 章「J2EE Web サービス・クライアントのアセンブル」を参照してください。

- J2SE クライアントの構築方法は、第 15 章「J2SE Web サービス・クライアントのアセンブル」を参照してください。
- Web サービスのパッケージ化およびデプロイ方法は、第 19 章「Web サービスのパッケージ化およびデプロイ」を参照してください。
- クライアントのアセンブルに必要な JAR ファイルの詳細は、付録 A「Web サービス・クライアントの API および JAR」を参照してください。
- Web サービスの相互運用性の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「相互運用可能な Web サービスの実現」を参照してください。
- Web サービス・クライアントにおけるサービスのクオリティ機能の使用方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの管理」を参照してください。
- Web サービスへのセキュリティの追加方法は、『Oracle Application Server Web Services セキュリティ・ガイド』を参照してください。
- トランスポート・レベルで保護されている Web サービスにアクセスするクライアントの記述方法は、『Oracle Application Server Web Services セキュリティ・ガイド』の「EJB をベースとする Web サービスに対するトランスポート・レベル・セキュリティの追加」および「トランスポート・レベルで保護された Web サービスへのアクセス」を参照してください。
- Web サービスへの信頼性の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの信頼性の確保」を参照してください。
- Web サービスへの監査およびロギング構成の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「監査メッセージおよびロギング・メッセージ」を参照してください。
- 標準以外のデータ型の処理の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 値タイプのカスタム・シリアライズ」を参照してください。
- JAX-RPC マッピング・ファイルの内容とその使用方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「JAX-RPC マッピング・ファイル記述子」を参照してください。
- OracleAS Web Services でサポートされているデータ型の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 型の XML および WSDL 型へのマッピング」を参照してください。
- Web サービス起動フレームワークの使用方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービス起動フレームワークの使用方法」を参照してください。
- Web サービス開発用の Oracle JDeveloper ツールのサポートの詳細は、Oracle JDeveloper のオンライン・ヘルプを参照してください。

19

Web サービスのパッケージ化およびデプロイ

この章では、Oracle Application Server Web Services によってサポートされる Web サービス・ファイルのパッケージ化とデプロイについて説明します。Web サービスは、いくつかの詳細事項を除き、その他の J2EE アプリケーションと同じ方法でパッケージ化およびデプロイされます。

この章の内容は、次のとおりです。

> **注意：** この章では、デプロイ自体については説明しません。Web モジュールおよび EJB のデプロイの詳細は、『Oracle Containers for J2EE デプロイメント・ガイド』を参照してください。

Web サービス・パッケージ化の概要

Web サービス・ファイルをアセンブルおよびパッケージ化するには、Oracle JDeveloper または WebServicesAssembler を使用します。これらのツールを使用すると、パッケージ化したアプリケーションに正しいファイルおよびデプロイメント・ディスクリプタが含まれます。Oracle JDeveloper ウィザードのオプションや WebServicesAssembler コマンドの引数を使用すると、実行時に Web サービスによって使用されるデプロイメント・ディスクリプタの値を構成できます。各ファイルは、Enterprise Web Services 1.1 の仕様に記載されているルールに従って、デプロイ可能な EAR ファイルにパッケージ化されます。

また、Web サービス・アプリケーションは手動でアセンブルおよびパッケージ化することもできます。この章では、Web サービスのパッケージの構造と内容について説明しますが、手動アセンブリの詳細は説明しません。

デプロイを行うには、コマンドラインまたは Ant タスクで `admin_client.jar` を使用するか、Oracle JDeveloper または Application Server Control ツールを使用します。

表 19-1 は、Oracle ツールのどのツールで Web サービスをパッケージ化またはデプロイできるかを要約したものです。

表 19-1 Oracle ツールによるパッケージ化およびデプロイのサポート

	WebServices-Assembler	Oracle JDeveloper	Application Server Control	admin_client.jar	Ant タスク
パッケージ化	○	○	×	×	×
デプロイ	×	○	○	○	○

最終ステップとして、デプロイした Web サービスを Universal Description, Discovery, and Integration（UDDI）レジストリにパブリッシュできます。

注意：

- UDDI については、このマニュアルでは説明しません。UDDI の使用の詳細は、次の Web サイトを参照してください。

  `http://www.uddi.org/specification.html`

- Oracle の UDDI 製品である Oracle Application Server UDDI レジストリの詳細は、次の Web サイトを参照してください。

  `http://www.oracle.com/technology/tech/webservices/htdocs/uddi/index.html`

Web サービス・アプリケーションのパッケージの構造

OC4J にデプロイされるように設計されている Web サービス・ファイルが、コンポーネント・デプロイ・モジュールに追加されます。このコンポーネント・デプロイ・モジュールには、EJB 用の JAR または Java クラス用の WAR ファイルがあります。コンポーネント・デプロイ・モジュールは、デプロイ可能な EAR に格納されます。EAR ファイル内の Web サービス・ファイルのパッケージの構造は、Enterprise Web Services 1.1 の仕様に定義されているルールに準拠しています。次の各項では、Java クラスおよび EJB をベースにした Web サービスのパッケージの構造について説明します。

- Java クラスをベースにした Web サービスのパッケージ化
- EJB をベースにした Web サービスのパッケージ化

Java クラスをベースにした Web サービスのパッケージ化

Java クラスをベースにした Web サービスの場合、EAR ファイルには WAR ファイルが格納されます。デプロイメント・ファイルおよびクラス・ファイルは、WAR において WEB-INF ディレクトリの下に格納されます。WEB-INF ディレクトリには、web.xml、webservices.xml、oracle-webservices.xml および JAX-RPC マッピング・ファイルがあります。また、このディレクトリには、WSDL 用の wsdl ディレクトリ、クラス・ファイル用の classes ディレクトリおよび JAR ファイル用の lib ディレクトリもあります。

META-INF ディレクトリには、拡張機能およびパッケージ関連データを定義する MANIFEST.MF マニフェスト・ファイル、J2EE アプリケーションのコンポーネントを指定する application.xml ファイルが含まれます。

例 19-1 は、Java クラスをベースにした Web サービスのパッケージの構造を示します。

例 19-1 Java クラスをベースにした Web サービスのパッケージの構造

```
<serviceName>.ear        contains
      META-INF/
         |--MANIFEST.MF
         |--application.xml

      <serviceName>.war  contains
          WEB-INF/
             |--web.xml
             |--webservices.xml
             |--oracle-webservices.xml
             |--<mapping file>
             |--wsdl/
                  |--<serviceName>.wsdl
             |--classes/
                  |--class files
             |--lib/
                  |--*jar files
```

EJB をベースにした Web サービスのパッケージ化

EJB をベースにした Web サービスのパッケージの構造は、Java クラスの場合と似ていますが、EAR ファイルにマニフェストおよび JAR ファイルのための META-INF ディレクトリが含まれる点が異なります。JAR ファイル内には、EJB 用のデプロイメント・ファイルおよびクラス・ファイルが含まれる別の META-INF ディレクトリがあります。JAR 内の META-INF ディレクトリには、ejb-jar.xml、webservices.xml、oracle-webservices.xml および JAX-RPC マッピング・ファイルがあります。また、このディレクトリには、WSDL およびクラス・ファイルのための wsdl ディレクトリもあります。例 19-2 は、EJB をベースにした Web サービスのパッケージの構造を示します。

例 19-2 EJB をベースにした Web サービスのパッケージの構造

```
<serviceName>.ear  contains
        META-INF/
           |--MANIFEST.MF
           |--application.xml

        <serviceName>.jar contains
              class files
              META-INF/
                  |--ejb-jar.xml
                  |--webservices.xml
                  |--oracle-webservices.xml
                  |--<mapping file>
                  |--wsdl/
                       |--<serviceName>.wsdl
```

パッケージ化されるファイルの説明

次のリストは、デプロイ用にパッケージ化されるファイルの説明です。

- `application.xml`: EAR に含まれる EJB JAR および WAR が記述されています。このファイルには、EJB および Web モジュールなど、J2EE アプリケーションのコンポーネントを指定し、アプリケーションの追加構成も指定できます。このディスクリプタは、アプリケーションの EAR ファイルの `/META-INF` ディレクトリに含める必要があります。`application.xml` ファイルは、次の Web サイトにある `application_1_4.xsd` スキーマによって定義されます。

  `http://java.sun.com/xml/ns/j2ee/application_1_4.xsd`

- `ejb-jar.xml`: EJB コンポーネントのデプロイメント・ディスクリプタ。このファイルは、JAR 内の Enterprise JavaBeans に固有の構造上の特性および依存性を定義し、Bean が EJB コンテナと対話するための方法を EJB コンテナに指定します。

  `webservices.xml` の内容と `ejb-jar.xml` の内容の間には、Web サービスとして公開された EJB を識別するための関係が存在します。この関係については、図 19-1 を参照してください。

  `ejb-jar.xml` ファイルは、次の Web サイトにあるスキーマによって定義されます。

  `http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd`

- *<mapping file>*`.xml`（*serviceName*`_java-wsdl-mapping.xml` など）: Java インタフェース、メソッドおよびパラメータを WSDL にマップする JAX-RPC マッピング・ファイル。このファイルの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「JAX-RPC マッピング・ファイル記述子」を参照してください。

- `oracle-webservices.xml`: OracleAS Web Services 上で動作する Web サービス・アプリケーションに固有のデプロイメント・プロパティを定義するデプロイメント・ディスクリプタ。このファイルの内容の詳細は、付録 C「oracle-webservices.xml デプロイメント・ディスクリプタ・スキーマ」を参照してください。

- `web.xml`: このデプロイメント・ディスクリプタは、Java Servlet 2.4 の仕様によって定義されます。このデプロイメント・ディスクリプタを使用すると、J2EE に準拠する任意のアプリケーション・サーバー上に Web アプリケーションをデプロイできます。`web.xml` ファイルの詳細は、次の Web サイトにある Java Servlet 2.4 の仕様を参照してください。

  `http://jcp.org/aboutJava/communityprocess/final/jsr154/index.html`

  `webservices.xml` の内容と `web.xml` の内容の間には、Web サービスとして公開されたサーブレットを識別するための関係が存在します。この関係については、図 19-2 を参照してください。

  `web.xml` ファイルは、次の Web サイトにある `web-app_2_4.xsd` スキーマによって定義されます。

  `http://java.sun.com/xml/ns/j2ee/`

- `webservices.xml`: コンポーネント・デプロイ・モジュール内にパッケージ化された Web サービス・アプリケーションの標準構成ファイル。このファイルは、Web サービス・エンドポイント、関連構成ファイル、WSDL 情報、JAX-RPC マッピング・データを定義します。このファイルは、WSDL ファイル（`<wsdl-file>`）の位置、マッピング・ファイル（`<jaxrpc-mapping-file>`）、WSDL のポートに対応する `<port-component>`、Java サービス・エンドポイント・インタフェース（`<service-endpoint-interface>`）、WSDL の Java 表現、サーブレット名（`<servlet-link>`）または EJB 名（`<ejb-link>`）を示します。

  `webservices.xml`、`ejb-jar.xml`、`oracle-webservices.xml` および `web.xml` ファイルの各内容の間には、関係が存在しています。これらの関係は、Web サービスとして公開されるコンポーネントを識別し、ファイル間でメタデータをやりとりするためのものです。これらの関係の詳細は、「デプロイメント・ディスクリプタ・ファイル間の関係」を参照してください。

webservices.xml ファイルは、次の Web サイトにある j2ee_web_services_1_1.xsd スキーマによって定義されます。

http://java.sun.com/xml/ns/j2ee/

- WSDL ファイル : Web サービスのインタフェースと、サービスを起動するためのメッセージの書式を記述します。Web サービスが含まれるアーカイブに Web Services Description Language（WSDL）ドキュメントが含まれていない場合、OracleAS Web Services は、デプロイ時に WSDL ドキュメントを生成します。WSDL は、次の Web サイトにある仕様によって定義されます。

  http://www.w3.org/TR/wsdl

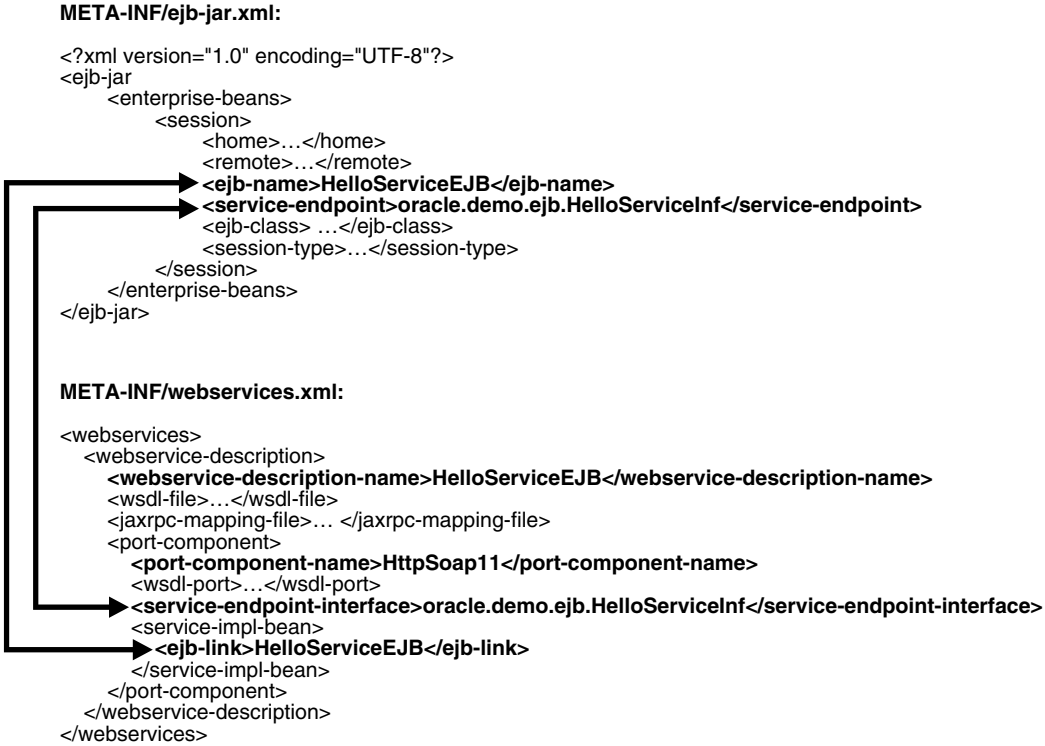
デプロイメント・ディスクリプタ・ファイル間の関係

この項では、webservices.xml と、ejb-jar.xml、oracle-webservices.xml および web.xml ファイルの関係について説明します。

webservices.xml と ejb-jar.xml の関係

図 19-1 は、EJB をベースにした Web サービス用の ejb-jar.xml の内容と webservices.xml の内容の関係を示します。webservices.xml の <ejb-link> 要素の値は、ejb-jar.xml の <ejb-name> の値と同じです。このマッピングにより、Web サービスとして公開される EJB、つまり、HTTP（SOAP over HTTP）を介して公開される EJB が識別されます。webservices.xml の <service-endpoint-interface> 要素の値は、ejb-jar.xml の <service-endpoint> の値と同じです。これは、EJB に関する追加の一意制約として機能します。

図 19-1 webservices.xml と ejb-jar.xml の関係



webservices.xml と oracle-webservices.xml の関係

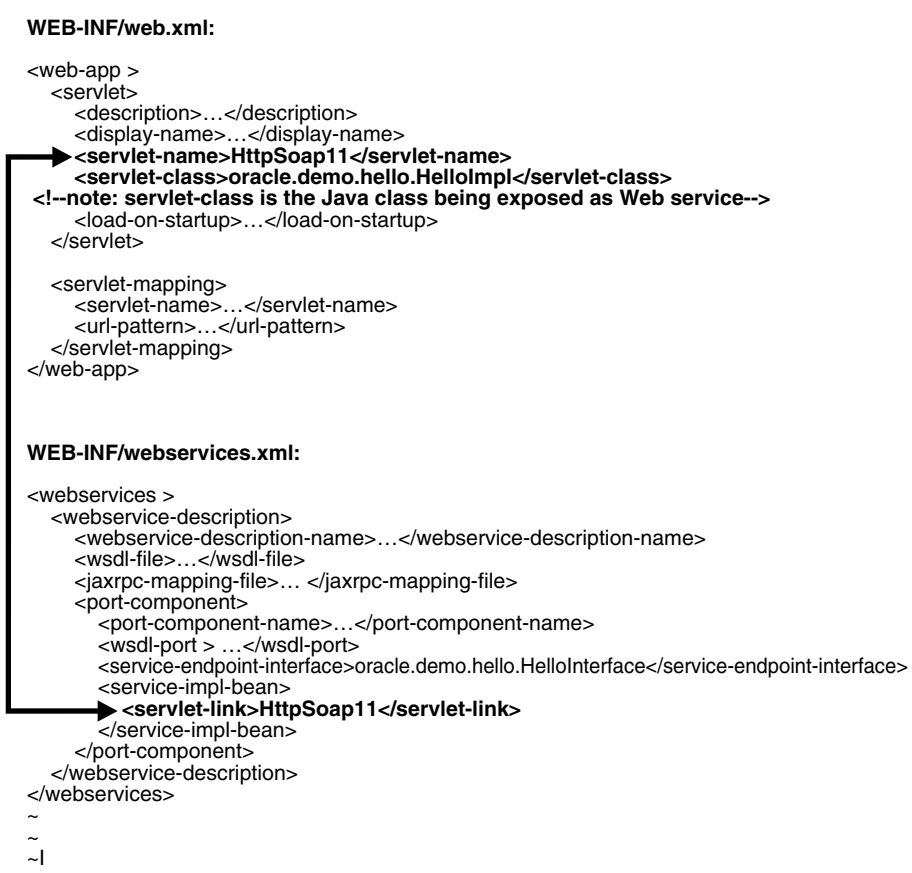
図 19-2 は、J2EE 標準の webservices.xml デプロイメント・ディスクリプタの内容と oracle-webservices.xml 固有のデプロイメント・ディスクリプタの関係を示します。この関係により、ファイル間でのメタデータのマッピングが可能になります。webservices.xml の <port-component-name> 要素の値は、oracle-webservices.xml の <port-component name="..."> の name 属性と同じ値です。webservices.xml の <webservice-description-name> 要素の値は、oracle-webservices.xml の <webservice-description name="..."> の name 属性と同じ値です。

図 19-2 webservices.xml と oracle-webservices.xml の関係

webservices.xml と web.xml の関係

図 19-3 は、webservices.xml の内容と web.xml の内容の関係を示しています。この関係により、Web サービスとして公開されるサーブレットが識別されます。webservices.xml の <servlet-link> 要素の値は、web.xml の <servlet-name> と同じ値です。これにより、Web サービス実装をサーブレットとして公開する、つまり、HTTP（SOAP over HTTP）を介して公開することが可能になります。これは一般化されている方法であるため、JAX-RPC および Enterprise Web Services 1.1 の仕様を実装する J2EE コンテナ間でのアプリケーション・パッケージの移植を可能にします。これらの標準に準拠するアプリケーションは、Enterprise Web Services 1.1 の仕様に準拠する任意のコンテナにデプロイし、任意の Web サービス・クライアントを使用して起動できます。

図 19-3 webservices.xml と web.xml の関係



パッケージ化用のツールのサポート

この項では、OracleAS Web Services に用意されている、Web サービス・ファイルをパッケージ化するためのツールのサポートについて説明します。パッケージ化は、WebServicesAssembler または Oracle JDeveloper によって行えます。

- WebServicesAssembler によるパッケージ化のサポート
- Oracle JDeveloper によるパッケージ化のサポート

WebServicesAssembler によるパッケージ化のサポート

この項では、WebServicesAssembler によるパッケージ化のサポートについて説明します。Web サービスをアセンブルするための各コマンドは、Web サービスをデプロイ可能な EAR にパッケージ化するのにも使用できます。ファイルのアセンブル時には、これらのコマンドの多くで、デプロイメント・ディスクリプタの作成も行われます。デプロイメント・ディスクリプタには、コンポーネントをデプロイするために必要な宣言データや、コンポーネントをアプリケーションに構成する方法を記述したアセンブリ方法が含まれます。

- パッケージ化を実行する WebServicesAssembler のコマンド
- デプロイメント・ディスクリプタの管理

パッケージ化を実行する WebServicesAssembler のコマンド

WebServicesAssembler には、Web サービスに必要なすべてのファイルをアセンブルするためのコマンドがいくつか用意されています。

- aqAssemble
- assemble
- corbaAssemble
- dbJavaAssemble
- ejbAssemble
- jmsAssemble
- plsqlAssemble
- sqlAssemble

WebServicesAssembler では、すべての関連デプロイメント・ディスクリプタの生成が処理され、アプリケーションによって必要とされる固有の構成が Oracle 固有のデプロイ・ファイルにマップされます。また、WebServicesAssembler は、関連するすべてのファイルをアプリケーション・サーバーにデプロイできるようにパッケージ化します。

これらのコマンドには、生成されたファイルを様々な方法でパッケージ化または保存するための引数があります。`ear` 引数を使用すると、ファイルがデプロイ可能な EAR ファイルとして保存されます。`war` 引数を使用すると、ファイルは WAR ファイルとして保存されます。また、ファイルは、WAR の内容が含まれるディレクトリに、アーカイブせずに保存されます。

関連項目：

- この項でリストされているコマンドの詳細は、18-4 ページの「Web サービス・アセンブリ・コマンド」を参照してください。
- これらの引数の詳細は、18-42 ページの「ear」および 18-49 ページの「war」を参照してください。

デプロイメント・ディスクリプタの管理

多くの WebServicesAssembler コマンドでは、Web サービスのアセンブル時にデプロイメント・ディスクリプタが作成されます。WebServicesAssembler ではデプロイは実行されませんが、WebServicesAssembler コマンドの引数を使用すると、デプロイメント・ディスクリプタの値を設定できます。

- デプロイメント・ディスクリプタを作成するコマンド
- デプロイメント・ディスクリプタの内容に影響する引数

デプロイメント・ディスクリプタを作成するコマンド いくつかの WebServicesAssembler コマンドでは、出力の一部として `application.xml`、`web.xml`、`webservices.xml` および `oracle-webservices.xml` デプロイメント・ディスクリプタが生成されます。これらのファイルを生成するコマンドは、次のとおりです。

- aqAssemble
- assemble
- corbaAssemble
- dbJavaAssemble
- ejbAssemble
- jmsAssemble
- plsqlAssemble
- sqlAssemble
- topDownAssemble
- genApplicationDescriptor（`application.xml` のみを生成）
- genDDs

デプロイメント・ディスクリプタの内容に影響する引数 この項では、デプロイメント・ディスクリプタ・ファイルのリスト、およびその内容に影響を与える可能性のある WebServicesAssembler の引数を示します。

application.xml:

- context: Web アプリケーションで使用すると、`<context-root>` 要素が追加されます。

oracle-webservices.xml:

- callScope: `<param name="scope">call</param>` 要素が追加されます。
- context: EJB 2.1 で使用すると、`<context-root>` 要素が追加されます。
- dataSource: 次の `param name` 属性が `<implementor>` 要素に追加されます。次の例において、*dataSource value* は、`dataSource` 引数に指定した値です。

  `<param name="databaseJndiName">`*dataSource value*`</param>`

- ddFileName: `*Assemble` コマンドで使用すると、管理およびカスタム・シリアライズの情報が指定したファイルからアーカイブ内の `oracle-webservices.xml` ファイルにコピーされます。
- mtomSupport: ブール型の `<mtom-support>` サブ要素が、`<port-component>` 要素に追加されます。
- restSupport: ブール型の `<rest-support>` サブ要素が `<port-component>` 要素および `<provider-port>` 要素に追加されます。
- session: `<param name="scope">session</param>` 要素が追加されます。
- timeout: `<param name="session-timeout">`*integer*`</param>` 要素が追加されます。

- uri: EJB 2.1 で使用すると、<endpoint-address-uri> 要素が追加されます。
- useDimeEncoding: <use-dime-encoding> 要素が追加されます。

***serviceName*-java-wsdl-mapping.xml:**

このファイル名で、***serviceName*** は、serviceName 引数で指定されている Web サービスの名前を示します。

- interfaceName: データベース・リソースから Web サービスをアセンブルするコマンド（plsqlAssemble、sqlAssemble、dbJavaAssemble または aqAssemble）で使用すると、<service-endpoint-interface> 要素が追加されます。

web.xml:

- className: <servlet-class> 要素が追加されます。
- recoverable: ブール型の <distributable> 要素が追加されます。
- uri: Web アプリケーションで使用すると、<url-pattern> 要素が追加されます。

webservices.xml:

- ejbName: <ejb-link> 要素が webservices.xml に追加されます（EJB 2.1 の場合のみ）。
- interfaceName: <service-endpoint-interface> 要素が追加されます。
- mappingFileName: <jaxrpc-mapping-file> 要素が追加されます。ファイルの位置と名前は、デプロイメント・ディスクリプタに格納されるときに変更されることがあります。いくつかのマッピングが元のファイルで定義されていないときは、ファイルの内容は、アーカイブに格納される前に変更されることがあります。
- <handler> タグ : handler タグのデータが webservices.xml に追加されます。これらのタグは Ant タスクでのみ使用できます。

Oracle JDeveloper によるパッケージ化のサポート

Oracle JDeveloper には、Web サービス・アプリケーションをデプロイ用にパッケージ化するためのウィザードが用意されています。Oracle JDeveloper によるサポートの詳細は、Oracle JDeveloper オンライン・ヘルプの次のトピックを参照してください。

- J2EE アーカイブ形式の概要

  Oracle JDeveloper でサポートされているアーカイブ・タイプとその関連モジュール・タイプおよび内容がリストされています。

- デプロイ用のアプリケーションの構成

  デプロイメント・ディスクリプタ、クライアント・アプリケーション、EJB およびアプレットの構成とパッケージ化に関するトピックへのリンクが記載されています。

- デプロイ用の EJB の構成

  EJB JAR ファイルのデプロイメント・プロファイルの作成や ejb-jar.xml デプロイメント・ディスクリプタの追加の手順を示します。

- デプロイ用のクライアント・アプリケーションの構成

  クライアント JAR ファイルのデプロイメント・プロファイルの作成や application-client.xml デプロイメント・ディスクリプタ・ファイルの作成の手順を示します。

- デプロイ用のアプレットの構成

  WAR ファイルのデプロイメント・プロファイルの作成や web.xml デプロイメント・ディスクリプタの追加の手順を示します。

Web サービス・デプロイの概要

デプロイとは、アプリケーションが実行されるサーバーにアプリケーション・ファイルを転送するプロセスのことです。Web モジュールおよび EJB のデプロイの詳細は、『Oracle Containers for J2EE デプロイメント・ガイド』の次の章を参照してください。

- OracleAS Web Services への WAR ファイルのデプロイのガイドラインは、「Web モジュールのデプロイ」を参照してください。
- EJB アーカイブのデプロイのガイドラインは、「Enterprise JavaBeans のデプロイ」を参照してください。
- コマンドラインからのアプリケーションのデプロイのガイドラインは、「admin_client.jar によるアプリケーションのデプロイ」を参照してください。
- Ant タスクからのアプリケーションのデプロイのガイドラインは、「OC4J Ant タスクによるデプロイ」を参照してください。

Web サービスは、Oracle JDeveloper または Application Server Control を使用してデプロイできます。また、Web サービスのデプロイには、Ant タスクを使用することや、コマンドラインで `admin_client.jar` を使用することもできます。

関連項目：

これらのツールによるデプロイの詳細は、「デプロイ用のツールのサポート」を参照してください。

デプロイ用のツールのサポート

この項では、OracleAS Web Services に用意されているデプロイ用ツールのサポートについて説明します。デプロイの実行には、コマンドライン、Oracle JDeveloper および Application Server Control が使用できます。

デプロイ用のコマンドラインのサポート

OC4J に用意されている `admin_client.jar` コマンドライン・ユーティリティを使用して、EAR ファイルにパッケージ化された Web サービスをデプロイできます。アプリケーションのデプロイ・プロセスをスクリプトで記述する場合は、このユーティリティを使用できます。ただし、WAR ファイルにパッケージ化された Web モジュールなど、スタンドアロンのモジュールは、`admin_client.jar` ではサポートされていません。

関連資料：

このツールを使用したアプリケーションのデプロイの詳細は、『Oracle Containers for J2EE デプロイメント・ガイド』の「admin_client.jar によるアプリケーションのデプロイ」を参照してください。

admin_client.jar を使用したデプロイメントのサンプル

Web サービス用のファイルが含まれる EAR ファイルは、他の J2EE アプリケーションと同じ方法でデプロイできます。次は、デプロイメント・コマンドのサンプルです。

```
java -jar <oc4jHome>/j2ee/home/admin_client.jar deployer:oc4j:<oc4jHost>:<oc4jOrmiPort>
<adminId> <adminPassword>
            -deploy
            -file dist/hello.ear
            -deploymentName hello
            -bindWebApp default-web-site
```

次のリストは、このコード例のパラメータの説明です。

- *<oc4jHome>*: OC4J インストールが含まれているディレクトリ。
- *<oc4jHost>*:*<oc4jOrmiPort>*: EAR ファイルまたは J2EE アプリケーションをデプロイする OC4J サーバーのホスト名およびポート。
- *<adminId>*: OC4J のインスタンスのユーザー名。この値は OC4J のインストール時にユーザーが割り当てます。
- *<adminPassword>*: OC4J のインスタンス用のパスワード。この値は OC4J のインストール時にユーザーが割り当てます。
- `default-web-site`: アプリケーションのバインド先の Web サイト。通常、これは `default-web-site` になります。Web サイトを構成するには、*<oc4jHome>*`/j2ee/home/config` の `server.xml` ファイルを参照してください。

デプロイ用の Ant タスクのサポート

OracleAS Web Services には、OC4J インスタンスに対する J2EE アプリケーションのデプロイおよびアンデプロイ用として一連の Ant タスクが用意されています。Ant タスクの詳細や、タスクをアプリケーションのビルド・プロセスに組み込むためのガイドラインについては、『Oracle Containers for J2EE デプロイメント・ガイド』の「OC4J Ant タスクによるデプロイ」を参照してください。この章の内容は、次のとおりです。

- OC4J Ant タスクを環境に組み込む手順については、ビルド環境への OC4J Ant タスクの組込みに関する項を参照してください。
- 次の Ant タスクを起動する方法については、「OC4J Ant タスクの起動」を参照してください。
  - `bindAllWebApps`: `bindAllWebApps` タスクは、以前にデプロイされている EAR 内の Web モジュールを、指定された Web サイトにバインドします。
  - `bindWebApp`: `bindWebApp` タスクは、アプリケーションへのアクセスに使用する Web サイトに、アプリケーションをバインドします。
  - `deploy`: `deploy` タスクは、アーカイブにパッケージ化された J2EE アプリケーションまたはモジュールをデプロイします。
  - `deployerUri`: `deployerUri` タスクは、タスクに対するターゲットの OC4J インスタンス（1 つまたは複数）を指定します。
  - `modifySharedLibrary`: `modifySharedLibrary` タスクを使用すると、Oracle Application Server クラスタ内の 1 つの OC4J インスタンスまたはインスタンスのグループにインストールされている既存の共有ライブラリを変更できます。
  - `publishSharedLibrary`: `publishSharedLibrary` タスクは、Oracle Application Server クラスタ内の 1 つの OC4J インスタンスまたはインスタンスのグループに共有ライブラリをインストールします。
  - `removeSharedLibrary`: `removeSharedLibrary` タスクは、Oracle Application Server クラスタ内の 1 つの OC4J インスタンスまたはインスタンスのグループから共有ライブラリを削除します。

- `start`、`stop`: `start` および `stop` タスクは、特定の OC4J インスタンスでのデプロイ操作の一部として、アプリケーションおよびその子アプリケーションを起動、停止または再起動します。
- `undeploy`: `undeploy` タスクは、指定されたアプリケーションまたはモジュールを OC4J インスタンスから削除します。また、このタスクは、Web サイトからアプリケーションを自動的にアンバインドします。
- `updateEjbModule`: `updateEjbModule` タスクを使用すると、Oracle Application Server クラスタ内の 1 つの OC4J インスタンスまたはインスタンスのグループで稼働しているアプリケーション内の EJB モジュールの、増分デプロイまたは部分的なデプロイを行うことができます。

Oracle JDeveloper によるデプロイのサポート

次のリストは、Web サービスのデプロイに役に立つ Oracle JDeveloper のオンライン・ヘルプ内のトピックです。これらのトピックの詳細は、Oracle JDeveloper のオンライン・ヘルプを参照してください。

- 単純な JAR デプロイ

  実行可能 JAR ファイル、またはファイル・システム上の JAR ファイルにアプリケーションをデプロイする手順について説明します。

- 埋込み OC4J への Web サービスのデプロイ

  ローカル・サーバー上で動作する OC4J の埋込みインスタンスまたは Oracle Application Server に Web サービスをデプロイする手順について説明します。

- 外部 OC4J への Web サービスのデプロイ

  リモート・サーバー上で動作する OC4J の外部インスタンスまたは Oracle Application Server に Web サービスをデプロイする手順について説明します。

- セキュアな Oracle Application Server Web Services のデプロイ

  セキュリティを使用する J2EE 1.4 Web サービスをデプロイする手順について説明します。この手順では、デプロイ用の Web サービスにキーストアをバンドルする方法について説明します。

Application Server Control によるデプロイのサポート

次のリストは、Web サービスのデプロイに役に立つ Application Server Control のオンライン・ヘルプ内のトピックです。これらのトピックの詳細は、Application Server Control のオンライン・ヘルプを参照してください。

- 「デプロイ : デプロイ設定」ページ

  Application Server Control コンソールに用意されている「デプロイ・プラン」ページには、デプロイメント時に OracleAS Web Services デプロイメント・ディスクリプタ（`oracle-webservices.xml`）に値を設定する機能があります。

- 「デプロイ : アプリケーション属性」ページ

  Application Sever Control を使用して構成できるデプロイメント属性について説明します。

- アプリケーションのデプロイ

  アプリケーションをデプロイする手順について説明します。

- アプリケーションの再デプロイおよびアンデプロイ

  アプリケーションをデプロイまたはアンデプロイする手順について説明します。

制限事項

F-13 ページの「Web サービスのパッケージ化およびデプロイ」を参照してください。

追加情報

詳細は、次を参照してください。

- WSDL からの Web サービスのアセンブル方法は、第 6 章「WSDL からの Web サービスのアセンブル」を参照してください。
- ステートフルな Web サービスのアセンブル方法は、第 7 章「Java クラスを使用した Web サービスのアセンブル」を参照してください。
- EJB からの Web サービスのアセンブル方法は、第 8 章「EJB を使用した Web サービスのアセンブル」を参照してください。
- JMS キューまたは JMS トピックからの Web サービスのアセンブル方法は、第 9 章「JMS 宛先を使用した Web サービスのアセンブル」を参照してください。
- データベース・リソースからの Web サービスのアセンブル方法は、第 10 章「データベース Web サービスのアセンブル」を参照してください。
- J2SE 5.0 注釈を使用した Web サービスのアセンブル方法は、第 11 章「注釈を使用した Web サービスのアセンブル」を参照してください。
- J2EE クライアントの構築方法は、第 14 章「J2EE Web サービス・クライアントのアセンブル」を参照してください。
- J2SE クライアントの構築方法は、第 15 章「J2SE Web サービス・クライアントのアセンブル」を参照してください。
- WebServicesAssembler ツールを使用した Web サービスのアセンブル方法は、第 18 章「WebServicesAssembler の使用方法」を参照してください。
- JAX-RPC マッピング・ファイルの内容とその使用方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「JAX-RPC マッピング・ファイル記述子」を参照してください。
- `wsmgmt.xml` 管理ポリシー・ファイルの内容については、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービス管理スキーマの概要」を参照してください。
- DIME 添付ファイルの操作方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の添付ファイルの処理に関する項を参照してください。
- EJB をベースにした Web サービスへのトランスポート・レベルのセキュリティの追加方法は、『Oracle Application Server Web Services セキュリティ・ガイド』の「トランスポート・レベル・セキュリティの Web サービスへの追加」および「トランスポート・レベルで保護された Web サービスへのアクセス」を参照してください。
- Oracle Web Services Manager ツールの詳細は、『Oracle Web Services Manager 管理者ガイド』を参照してください。
- Web サービス・クライアントにおけるサービスのクオリティ機能の使用方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの管理」を参照してください。
- Web サービスへのセキュリティの追加方法は、『Oracle Application Server Web Services セキュリティ・ガイド』を参照してください。
- Web サービスへの信頼性の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービスの信頼性の確保」を参照してください。
- Web サービスへの監査およびロギング構成の追加方法は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「監査メッセージおよびロギング・メッセージ」を参照してください。

A

Web サービス・クライアントの API および JAR

この付録では、Web サービス・クライアントの API パッケージについて詳しく説明します。また、Web サービス・クライアントを実行するために必要な JAR ファイルも示します。

この付録の内容は、次のとおりです。

Web サービスの API パッケージ

表 A-1 は、Oracle Application Server Web Services の現行リリースで使用可能なパブリック Oracle API のリストです。`provider` パッケージは、OC4J コンテナでのみ使用できます。その他の API は、コンテナまたは Web サービス・クライアントで使用できます。

関連資料：

この表の API の詳細は、次を参照してください。

- 『Oracle Application Server Web Services Java API Reference』
- Oracle Technology Network

  `http://www.oracle.com/technology/index.html`

表 A-1 クライアント API パッケージ

パッケージ名	説明
oracle.webservices	すべてのモジュールに共通する Oracle 固有の拡張クラスおよび拡張インタフェースが含まれます。
oracle.webservices.attachments	添付ファイルのストリーミングをサポートするクラスおよびインタフェースが含まれます。このパッケージの使用方法の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「メッセージ添付ファイルの処理」を参照してください。
oracle.webservices.databinding	カスタム・シリアライズ用として使用可能な `SOAPElementSerializer` インタフェースが含まれます。`databinding` パッケージに含まれている機能の使用方法の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Java 値タイプのカスタム・シリアライズ」を参照してください。
oracle.webservices.management-portability	Web サービスのプラットフォーム・ポータビリティ・レイヤーです。
oracle.webservices.provider	プロバイダ・ベースのエンドポイントを実装するために必要なインタフェースおよびクラスが含まれます。`provider` パッケージの使用方法の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービス・プロバイダの使用方法」を参照してください。
oracle.webservices.reliability	WS-Reliability を使用した信頼性の高い SOAP メッセージ交換を実装するために必要なインタフェースおよびクラスが含まれます。詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』のクライアント・サイドの信頼性の動的な構成に関する項を参照してください。
oracle.webservices.security.callback	Web サービスのセキュリティ・コールバックをサポートするクラスを提供します。
oracle.webservices.soap	SOAP 1.2 をサポートする Oracle の拡張クラスおよび拡張インタフェースが含まれます。この API の詳細は、5-13 ページの「OraSAAJ API の概要」を参照してください。
oracle.webservices.transport	トランスポートの複数のバインディングをサポートするクラスおよびインタフェースが含まれます。JMS トランスポートでこのパッケージのクラスを使用する方法の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の JMS トランスポートをサポートするクライアント・コードの記述に関する項を参照してください。

表 A-1 クライアント API パッケージ（続き）

パッケージ名	説明
oracle.webservices.wsdl	WSDL 1.1 で SOAP 1.2 操作をサポートするための Oracle 固有のインタフェースが含まれます。このインタフェースの詳細は、付録 B「WSDL 1.1 API の Oracle 実装」を参照してください。

Web サービス・プロキシのクラスパスの設定

Web サービス・クライアントを作成する場合、このクライアントを実行するには正しいクラスパスを指定する必要があります。次の各項に、クラスパスに含めることができる JAR ファイルをリストします。この項の各表では、Oracle Application Server またはスタンドアロンの OC4J がインストールされている位置を指定するために *OC4J_HOME* 環境変数を使用しています。

- wsclient_extended.jar によるクラスパスの単純化
- クライアント・サイド・プロキシを使用するクライアントのクラスパス・コンポーネント
- OC4J セキュリティ関連のクライアント JAR ファイル
- WS-Security 関連のクライアント JAR ファイル
- 信頼性関連のクライアント JAR ファイル
- JMS トランスポート関連のクライアント JAR ファイル
- データベース Web サービス関連のクライアント JAR ファイル

wsclient_extended.jar によるクラスパスの単純化

J2SE 環境の構成を単純化するために、OracleAS Web Services には、Web サービス・クライアントをコンパイルおよび実行するために必要なすべてのクラスが含まれる `wsclient_extended.jar` ファイルが用意されています。このファイルをクラスパス上にリストすることで、個々の JAR をリストしないですみます。

`wsclient_extended.jar` ファイルには、次のクラス・ファイルが含まれます。

- 表 A-2 にリストされている個々の JAR ファイル内のコア Web サービス・クライアント・クラス・ファイル
- 表 A-3 にリストされている JAR 内の OC4J セキュリティ関連のクラス・ファイル
- 表 A-4 にリストされている JAR 内の WS-Security 関連のクラス・ファイル
- 表 A-5 にリストされている JAR 内の WS-Reliability 関連のクラス・ファイル
- 表 A-6 にリストされている JAR 内の JMS トランスポート関連のクラス・ファイル

`wsclient_extended.jar` ファイルを使用してクライアントを実行する場合は、これらの JAR ファイルをクラスパスに含める必要はありません。

`wsclient_extended.jar` ファイルは、Oracle Technology Network から単独のダウンロードとして入手可能です。

`http://download.oracle.com/otn/java/oc4j/10131/wsclient_extended.zip`

また、このファイルは OC4J Companion CD からインストールすることもできます。この場合、このファイルは、*ORACLE_HOME*`/webservices/lib` ディレクトリにあります。

クライアント・サイド・プロキシを使用するクライアントのクラスパス・コンポーネント

表 A-2 は、J2SE Web サービス・クライアントのクラスパスに含めることができる JAR ファイルのリストです。

注意：

- この表にリストされている `wsclient.jar` に含まれるのは、Web サービス・クライアントのコア・クラスのみです。コンパイルおよび実行時には、Web サービス・クライアントには、XML パーサー、WSDL パーサーおよびサービスのクオリティ（QOS）用のクラスなど、他の多くのクラスが必要です。これらのすべての必須クラスは、クラスパスの設定を簡単にするために `wsclient_extended.jar` にパッケージ化されています。
- 常にすべての JAR ファイルが必要になるわけではありません。

表 A-2 クライアント・サイド・プロキシを使用するクライアントのクラスパス・コンポーネント

コンポーネント JAR	説明
ORACLE_HOME/diagnostics/lib/ojdl.jar および *ORACLE_HOME*/diagnostics/lib/ojdl2.jar	ODL には、Oracle 製品でエラー診断を発行するために使用される API や、分析用のエラー診断ログを収集する LogLoader ツールが実装されています。
ORACLE_HOME/j2ee/home/oc4j-api.jar	パブリック OC4J API が含まれます。
ORACLE_HOME/j2ee/home/oc4jclient.jar	OC4J クライアントに必要となるファイルが含まれます。
ORACLE_HOME/j2ee/home/lib/activation.jar	通常、このコンポーネントは JRE で使用できます。JRE で使用できない場合は、クラスパスに含めてください。この JAR が必要なのは、添付ファイルを処理する場合のみです。
ORACLE_HOME/j2ee/home/lib/adminclient.jar	プラットフォームの管理およびアプリケーションのデプロイを行うための J2EE API が含まれます。
OC4J_HOME/j2ee/home/lib/ejb.jar	Enterprise JavaBeans 用のクラス・ファイルが含まれます。
OC4J_HOME/j2ee/home/lib/http_client.jar	Oracle HTTP クライアントのトランスポート実装が含まれます。
OC4J_HOME/j2ee/home/lib/javax77.jar	J2EE 管理仕様（JSR 77）の API が含まれます。
OC4J_HOME/j2ee/home/lib/jax-qname-namespace.jar	QName 定義が含まれます。
OC4J_HOME/j2ee/home/lib/jmxri.xml	JMX API が含まれます。
OC4J_HOME/j2ee/home/lib/jmx_remote_api.jar	JMX リモート API が含まれます。
OC4J_HOME/j2ee/home/lib/mail.jar	JavaMail API およびすべてのサービス・プロバイダが含まれます。通常、このコンポーネントは JRE で使用できます。JRE で使用できない場合は、クラスパスに含めてください。
OC4J_HOME/j2ee/home/lib/oc4j-schemas.jar	OracleAS Web Services のパブリック・スキーマが含まれます。
OC4J_HOME/j2ee/home/lib/servlet.jar	サーブレット実装が含まれます。
OC4J_HOME/lib/dms.jar	Oracle Diagnostics および Oracle Monitoring の DMS 実装が含まれます。
OC4J_HOME/lib/xml.jar	JAXB 実装が含まれます。
OC4J_HOME/lib/xmlparserv2.jar	Oracle XML Parser の JAR が含まれます。
OC4J_HOME/lib/xsu12.jar	Oracle XDK SQL ユーティリティが含まれます。

表 A-2 クライアント・サイド・プロキシを使用するクライアントのクラスパス・コンポーネント（続き）

コンポーネント JAR	説明
OC4J_HOME/webservices/lib/commons-logging.jar	ロギング・ライブラリ・パッケージが含まれます。
OC4J_HOME/webservices/lib/jaxrpc-api.jar	`javax.xml.rpc` および `java.xml.namespace` を含む JAX-RPC API が含まれます。
OC4J_HOME/webservices/lib/orasaaj.jar	SAAJ API の Oracle 実装が含まれます。
OC4J_HOME/webservices/lib/orawsdl.jar	WSDL 仕様（JSR 110）用の Java API の Oracle 実装（OraWSDL）が含まれます。
OC4J_HOME/webservices/lib/relaxngDatatype.jar	複数の JAX* テクノロジで共用される RELAX NG データ型ライブラリが含まれます。
OC4J_HOME/webservices/lib/saaj-api.jar	添付ファイル付きメッセージを処理するための SAAJ API 1.1 が含まれます。
OC4J_HOME/webservices/lib/wsclient.jar	Web サービス・クライアントのランタイムに必要となるクラスが含まれます。
OC4J_HOME/webservices/lib/wsdl-api.jar	WSDL（JSR 110）用の Java API が含まれます。
OC4J_HOME/webservices/lib/wsif.jar	Oracle による WSIF 実装が含まれます。
OC4J_HOME/webservices/lib/xsdlib.jar	複数の JAX* テクノロジで共用される XML スキーマ型ライブラリが含まれます。

OC4J セキュリティ関連のクライアント JAR ファイル

表 A-3 は、J2SE Web サービス・クライアントで OC4J セキュリティをサポートする際にクラスパスに含める必要がある JAR ファイルのリストです。`wsclient_extended.jar` ファイルには、この表にリストされている JAR 内の OC4J セキュリティ関連のクラス・ファイルが含まれています。`wsclient_extended.jar` ファイルを使用してクライアントを実行する場合は、これらの JAR を含める必要はありません。

表 A-3 クライアント・サイド・プロキシを使用するクライアント用 OC4J セキュリティ CLASSPATH コンポーネント

コンポーネント名	説明
OC4J_HOME/j2ee/home/lib/jta.jar	JTA（Java Transaction API）仕様 API が含まれます。
OC4J_HOME/j2ee/home/lib/jaas.jar	Java Authorization and Authentication Specification（JAAS）API が含まれます。
OC4J_HOME/j2ee/home/lib/jazn.jar	JAZN（Oracle JAAS プロバイダ）実装が含まれます。
OC4J_HOME/j2ee/home/lib/jazncore.jar	JAZN（Oracle JAAS プロバイダ）実装が含まれます。

WS-Security 関連のクライアント JAR ファイル

表 A-4 は、J2SE Web サービス・クライアントで WS-Security をサポートする際にクラスパスに含める必要がある JAR ファイルのリストです。`wsclient_extended.jar` ファイルには、この表にリストされている JAR 内の WS-Security 関連のクラス・ファイルが含まれています。`wsclient_extended.jar` ファイルを使用してクライアントを実行する場合は、これらの JAR を含める必要はありません。

表 A-4 クライアント・サイド・プロキシを使用するクライアント用 WS-Security CLASSPATH コンポーネント

コンポーネント名	説明
OC4J_HOME/jlib/javax-ssl-1_1.jar	トランスポート・レベルのセキュリティのサポートが含まれます。
OC4J_HOME/jlib/jaxen.jar	Jaxen（dom4j や JDOM などの複数のモードで XPath 式を評価できる Java XPath エンジン）を定義するクラスが含まれます。
OC4J_HOME/jlib/ojpse.jar	コアである暗号化実装が含まれます。
OC4J_HOME/jlib/oraclepki.jar	Oracle orapki の keytool ユーティリティが含まれます。
OC4J_HOME/jlib/osdt_core.jar	Oracle Security Developer's Toolkit（OSDT）API が含まれます。
OC4J_HOME/jlib/osdt_cert.jar	Oracle Security Developer's Toolkit の暗号化 API が含まれます。
OC4J_HOME/jlib/osdt_saml.jar	Oracle Security Developer's Toolkit の Security Assertion Markup Language（SAML）API が含まれます。
OC4J_HOME/jlib/osdt_wss.jar	Oracle Security Developer's Toolkit の Web Services Security（WS-Security）API が含まれます。
OC4J_HOME/jlib/osdt_xmlsec.jar	Oracle Security Developer's Toolkit の XML 署名および暗号化の API が含まれます。
OC4J_HOME/webservices/lib/wssecurity.jar	WS-Security API が含まれます。

信頼性関連のクライアント JAR ファイル

表 A-5 は、J2SE Web サービス・クライアントで WS-Reliability をサポートする際にクラスパスに含める必要がある JAR ファイルのリストです。`wsclient_extended.jar` ファイルには、この表にリストされている JAR 内の WS-Reliability 関連のクラス・ファイルが含まれています。`wsclient_extended.jar` ファイルを使用してクライアントを実行する場合、この JAR を含める必要はありません。

表 A-5 クライアント・サイド・プロキシを使用するクライアント用の信頼性関連の CLASSPATH コンポーネント

コンポーネント名	説明
OC4J_HOME/webservices/lib/orawsrm.jar	Web Services Reliability（WS-Reliability）API およびその実装が含まれます。

JMS トランスポート関連のクライアント JAR ファイル

表 A-6 は、J2SE Web サービス・クライアントでトランスポート・メカニズムとして JMS をサポートする際にクラスパスに含める必要がある JAR ファイルのリストです。`wsclient_extended.jar` ファイルには、この表にリストされている JAR 内の JMS トランスポート関連のクラス・ファイルが含まれています。`wsclient_extended.jar` ファイルを使用してクライアントを実行する場合、この JAR を含める必要はありません。

表 A-6 トランスポート・メカニズムとして JMS を使用するクライアント用の CLASSPATH コンポーネント

コンポーネント名	説明
J2EE_HOME/j2ee/home/lib/jms.jar	JMS API が含まれます。

データベース Web サービス関連のクライアント JAR ファイル

表 A-7 にリストされている JAR ファイルをクラスパスに含める必要があるのは、J2SE Web サービス・クライアントが、SQL/XML 書式または `WebRowSet` 型であるパラメータまたは戻り値を使用するサービスを起動する場合です。また、ネイティブの WSIF バインディングを使用してデータベース Web サービスを起動する場合も、クライアントにこれらのライブラリを含める必要があります。

注意： この表にリストされている JAR 内のクラス・ファイルは、`wsclient_extended.jar` ファイルには含まれていません。これらのファイルで提供される機能がクライアントに必要となる場合は、これらのファイルをクラスパスに明示的に含める必要があります。

表 A-7 クライアント・サイド・プロキシを使用するクライアント用のデータベース関連の CLASSPATH コンポーネント

コンポーネント名	説明
ORACLE_HOME/rdbms/jlib/aqapi.jar	Oracle アドバンスト・キューイングの API が含まれます。
ORACLE_HOME/jdbc/lib/ocrs12.jar	Oracle による WebRowSet 実装が含まれます。
ORACLE_HOME/jdbc/lib/ojdbc14dms.jar	Oracle JDBC ドライバが含まれます。
OC4J_HOME/lib/xsu12.jar	Oracle による SQL/XML 書式実装が含まれます。

CLASSPATH コマンドのサンプル

例 A-1 は、すべての Oracle Application Server Web Services クライアント JAR ファイルを対象とした、Windows プラットフォームの `set CLASSPATH` コマンドのサンプルです。UNIX プラットフォームのクラスパスも同じ方法で設定されます。

例 A-1 Windows プラットフォームの set CLASSPATH コマンド

```
set CLASSPATH=%ORACLE_HOME%¥j2ee¥home¥oc4jclient.jar;%CLASSPATH%
set CLASSPATH=%ORACLE_HOME%¥j2ee¥home¥lib¥adminclient.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥j2ee¥home¥lib¥ejb.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥j2ee¥home¥lib¥mail.jar;%CLASSPATH%
set CLASSPATH=%ORACLE_HOME%¥j2ee¥home¥lib¥activation.jar;%CLASSPATH%
set CLASSPATH=%J2EE_HOME%¥j2ee¥home¥lib¥jms.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥j2ee¥home¥lib¥http_client.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥j2ee¥home¥lib¥jaas.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥j2ee¥home¥lib¥jazn.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥j2ee¥home¥lib¥javax77.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥j2ee¥home¥lib¥jax-qname-namespace.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥j2ee¥home¥lib¥jta.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥j2ee¥home¥lib¥jazncore.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥j2ee¥home¥lib¥servlet.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥j2ee¥home¥lib¥jmx_remote_api.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥j2ee¥home¥lib¥jmxri.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥j2ee¥home¥lib¥oc4j-schemas.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥lib¥xmlparserv2.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥xdk¥lib¥xml.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥webservices¥lib¥orawsrm.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥webservices¥lib¥xsdlib.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥webservices¥lib¥relaxngDatatype.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥lib¥dms.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥webservices¥lib¥commons-logging.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥webservices¥lib¥jaxrpc-api.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥webservices¥lib¥wsclient.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥webservices¥lib¥wsdl-api.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥webservices¥lib¥wsif.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥webservices¥lib¥wssecurity.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥webservices¥lib¥saaj-api.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥webservices¥lib¥orasaaj.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥webservices¥lib¥orawsdl.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥jlib¥osdt_wss.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥jlib¥osdt_xmlsec.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥jlib¥ojpse.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥jlib¥osdt_saml.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥jlib¥oraclepki.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥jlib¥osdt_core.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥jlib¥osdt_cert.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥jlib¥jaxen.jar;%CLASSPATH%
set CLASSPATH=%OC4J_HOME%¥jlib¥javax-ssl-1_1.jar;%CLASSPATH%
```

B

WSDL 1.1 API の Oracle 実装

この付録では、WSDL バージョン 1.1 用 Java API の Oracle 実装（OraWSDL）について説明します。この API を使用することで、メモリー内の WSDL ドキュメントを読取り、変更、書込み、作成および再編成することができます。

主要な WSDL 用 Java API は、`javax.wsdl.factory.WSDLFactory` クラスです。アプリケーションでは、この抽象クラスを使用することで、新しい定義、新しい `WSDLReaders` および新しい `WSDLWriters` を生成できる WSDL ファクトリを取得できます。

関連資料：

次の Web サイトにある WSDL バージョン 1.1 用 Java API の説明。

`http://www.jcp.org/en/jsr/detail?id=110`

OraWSDL API の概要

javax.wsdl.factory.WSDLFactory クラスの OracleAS Web Services 実装は、oracle.webservices.wsdl.WSDLFactoryImpl です。このクラスを使用して、新しい WSDL ファクトリ・インスタンスを作成します。OracleAS Web Services には、抽象 WSDLFactory クラスに必要となる XMLSchema、SOAP、HTTP および MIME をサポートする拡張機能が用意されています。

WSDL_READ_TIMEOUT プロパティは、HTTP プロトコルまたは HTTPS プロトコルでのみ使用できます。このプロパティには、リモート WSDL 定義についてのリクエストに対するレスポンスを受信するまでに WSDLReader 実装が待機する最大時間を秒単位で指定します。

次に、WSDLFactory クラスに用意されている機能を使用した例をいくつか示します。

- WSDL 操作ツールの記述
- プログラムによる WSDL の分析
- WSIF プロバイダの記述

例 B-1 に、WSDL 用 Java API を使用して WSDL ファイルを取得および操作するサンプル・コードを示します。サンプルでは次のタスクの例が示されています。

- WSDL ファクトリ・インスタンスの取得
- WSDL ファイル・リーダーの作成と標準拡張機能の登録
- WSDL 定義の読取りのタイムアウトの設定
- WSDL ファイルの読取り
- WSDL ファイルからの情報の取得

WSDL ファクトリを取得するコード行では、WSDLFactory.newInstance メソッドで Oracle WSDLFactoryImpl クラスを使用し、WSDL ファクトリ・インスタンスを作成しています。

例 B-1 WSDL 用 Java API を使用して WSDL を操作する方法

```
   ...
//--Get the WSDLFactory. You must specifiy Oracle's implementation class name
       WSDLFactory wsdlFactory =
WSDLFactory.newInstance("oracle.webservices.wsdl.WSDLFactoryImpl");
   ...
//--Create a reader and register the standard extensions
       WSDLReader wsdlReader = wsdlFactory.newWSDLReader();
       ExtensionRegistry extensionRegistry =
wsdlFactory.newPopulatedExtensionRegistry();
       wsdlReader.setExtensionRegistry(extensionRegistry);
   ...
 //--Set a sixty-second timeout for reading the WSDL definition
        System.setProperty(oracle.webservices.wsdl.WSDLFactoryImpl.WSDL_READ_TIMEOUT,
"60" );
   ...
//--Read a WSDL file, including any imports
       Definition def = wsdlReader.readWSDL("http://some.com/someservice?WSDL");
   ...
//--You can now explore the WSDL definition, for example,
       Map services = def.getServices();
       String targetNamespace = def.getTargetNamespace();
   ...
```

例 B-2 のコードで、javax.xml.rpc.server.ServletEndpointContext および javax.servlet.ServletContext クラスにあるメソッドを使用して WSDL を取得する方法を示します。取得された WSDL は、InputStream に格納されます。この技法は、WSDL に対してのみでなく、どのようなリソースに対しても使用できます。

// code to handle the input stream セクションには、例 B-1 のサンプル内にあるような、WSDL ファイルを取得および操作するための OraWSDL API コードを挿入します。

例 B-2 リソースとしての WSDL の取得

```
public class HelloImpl  implements javax.xml.rpc.server.ServiceLifecycle {
    // default constructor
    public HelloImpl() {
}
    public init(Object context){
        javax.xml.rpc.server.ServletEndpointContext ctx =
(javax.xml.rpc.server.ServletEndpointContext) context;
        javax.servlet.ServletContext servletContext= ctx.getServletContext();
        //we can read any other resource the same way.
        java.io.InputStream wsdlDocument =
servletContext.getResourceAsStream("/WEB-INF/wsdl/MyWsdl.wsdl")) ;

       // code to handle the input stream
      ...
    }

   //empty implementation
    public void destroy() {}

    // sayHello method
    public String sayHello(String name) {
        return ("Hello " + name + "!");
    }
}
```

C

oracle-webservices.xml デプロイメント・ディスクリプタ・スキーマ

oracle-webservices.xml デプロイメント・ディスクリプタは、標準の webservices.xml とともに使用します。OracleAS Web Services に固有のデプロイメント情報およびランタイム情報が含まれています。たとえば、コンテキスト URI や Web サービス・エンドポイントのアドレスなどが含まれています。このファイルの一部の要素の値は、WebServicesAssembler のコマンドライン引数および Ant タスクを使用して変更できます。

oracle-webservices.xml のすべての要素はオプションです。このファイルを指定しなくても、アプリケーションは、未指定の要素に適したデフォルト値を使用してデプロイおよび実行されます。

oracle-webservices.xml デプロイメント・ディスクリプタには、セキュリティ、信頼性、監査およびロギングに関する Web サービスの管理情報が含まれます。デプロイ時には、このファイルが解析され、そのオブジェクト表現がキャッシュされます。Web サービスの管理情報はこのファイルから抽出され、OC4J コンテナに wsmgmt.xml として保存されます。

スキーマを調べて oracle-webservices.xml を手動で作成することもできますが、通常は、WebServicesAssembler によって作成されたバージョンのファイルを使用します。このファイルを編集し、必要な機能を生成できます。スキーマは oracle-webservices-10_0.xsd ファイルで定義されており、oc4j-schemas.jar ファイルにあります。

OC4J_HOME\j2ee\home\lib\oc4j-schemas.jar

OC4J_HOME は、Oracle Containers for J2EE（OC4J）をインストールしたディレクトリです。

関連項目：

- これらの引数の一覧は、19-9 ページの「デプロイメント・ディスクリプタの内容に影響する引数」を参照してください。
- 管理構成要素の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービス管理スキーマの概要」を参照してください。

oracle-webservices.xml での XML 要素の階層

oracle-webservices.xml ファイルの要素階層の概要を次に示します。

```
<oracle-webservices>
   <web-site>
   <context-root>
   <webservice-description>
      <expose-wsdl>
      <expose-testpage>
      <resolve-relative-imports>
      <download-external-imports>
      <port-component>
         <endpoint-address-uri>
         <ejb-transport-security-constraint>
            <wsdl-url>
            <soap-port>
            <role-name>
            <transport-guarantee>
<!-- For a description of transport-level security constraints for EJBs, see  "Securing EJB-Based Web Services at the
Transport Level" on page C-15. -->
         <implementor>
            <param>
         <runtime>
            <owsm>
<!-- For a description of the Oracle Web Services Manager (<owsm>) element, see the  Oracle Web Services Manager Administrator's
Guide  -->
            <security>
<!-- For a description and listing of security elements, see the  Oracle Application Server Web Services Security Guide .  -->
            <reliability>
               <repository>
 <!-- For a description and listing of port-level reliability elements,  see "Port-Level Reliability Elements on the Server" in the
Oracle Application Server Advanced Web Services Developer's Guide .  -->
            <logging>
         <operations>
            <operation>
               <runtime>
                  <security>
<!-- For a description and listing of security elements, see the  Oracle Application Server Web Services Security Guide . -->
                   <reliability>
                     <duplication-elimination-required>
                     <guaranteed-delivery-required>
<!-- For a listing and description of operation-level reliability elements, see "Operation Level Reliability Elements on the
Server" in the  Oracle Application Server Advanced Web Services Developer's Guide .  -->
                  <auditing>
<!-- For a listing and description of operation-level auditing elements, see "Server-Side  Auditing Configuration Elements" in
the  Oracle Application Server Advanced Web Services Developer's Guide .  -->
                  <logging>
<!-- For a listing and description of operation-level logging  elements, see "Operation Level Logging Elements on the Server"
in the  Oracle Application Server Advanced Web Services Developer's Guide .  -->
   <provider-description>
      <provider-description-name>
      <wsdl-file>
      <wsdl-service-name>
      <property>
      <provider-port>
         <provider-name>
         <wsdl-port>
         <expose-testpage>
         <expose-wsdl>
         <implementation-class>
         <servlet-link>
         <max-request-size>
         <property>
```

```
            <policy>
               <runtime>
               <operations>
                  <operation>
<!-- Here you can enter a management configuration for a particular operation -->
   <ejb-transport-login-config>
      <auth-method>
      <realm-name>
<!-- For a listing and description of elements that define transport-level  security for EJBs, see  "Securing EJB-Based Web Services at the Transport Level" on page C-15. -->
```

oracle-webservices.xml の要素と属性

ここでは、oracle-webservices.xml ファイルの要素をアルファベット順に説明します。階層については、C-2 ページの「oracle-webservices.xml での XML 要素の階層」を参照してください。サンプル・ファイルについては、C-16 ページの「oracle-webservices.xml ファイルのリスト」を参照してください。

<auth-method>

親要素： <ejb-transport-login-config>

子要素： なし

必須 / オプション オプション

このサブ要素を使用して、EJB アプリケーションの認証メカニズムを構成します。ユーザーに認証制約によって保護された Web リソースにアクセスするために、構成したメカニズムを使用してユーザーは認証を受ける必要があります。このサブ要素の有効値は、BASIC、DIGEST、CLIENT-CERT またはベンダー固有のシングル・サインオン認証スキームです。C-15 ページの「トランスポート・レベルでの EJB ベースの Web サービスの保護」を参照してください。

<context-root>

親要素： <oracle-webservices>

子要素： なし

必須 / オプション バージョン 2.1 の EJB Web サービスの場合のみ

この要素の値の型は string です。デフォルト：.jar 拡張子なしの EJB アーカイブ・ファイル名。

このサブ要素には、公開される Web サービスのルート・コンテキストを指定します。このサブ要素が必要なのは、Web サービスとして公開されるバージョン 2.1 の EJB の場合のみです。context-root を指定しない場合、デフォルトでこの属性は .jar 拡張子なしの EJB アーカイブ・ファイル名に設定されます。たとえば、EJB アーカイブ・ファイル名が foo-ejb.jar である場合、コンテキスト・ルートは /foo-ejb になります。

Java クラスの Web サービスの場合、コンテキスト・ルートは application.xml 内に指定します。

<context-root> 要素の詳細は、『Oracle Containers for J2EE 構成および管理ガイド』を参照してください。

<download-external-imports>

親要素： <webservice-description>

子要素： なし

必須 / オプション オプション

このブール型要素には、相対インポートをダウンロードして絶対 URL に解決するかどうかを指定します。デフォルトは `false` です。

注意： `download-external-imports` を `true` に設定すると、`resolve-relative-imports` は自動的に `true` に設定されます。

<ejb-transport-login-config>

親要素： <oracle-webservices>

子要素： <auth-method>、<realm-name>

必須 / オプション オプション

この要素を使用して、この EJB アプリケーションで使用する必要があるトランスポート・レベルの認証方式およびレルム名を構成します。Web サービスとして公開する EJB アプリケーションの URL は、port component の `<endpoint-address-uri>` 要素によって指定します。詳細は、C-15 ページの「トランスポート・レベルでの EJB ベースの Web サービスの保護」を参照してください。

<ejb-transport-security-constraint>

親要素： <port-component>

子要素： <role-name>、<soap-port>、<transport-guarantee>、<wsdl-url>

必須 / オプション オプション

EJB をベースにした Web サービスにトランスポート・レベルのセキュリティを定義します。この要素の使用方法の詳細は、C-15 ページの「トランスポート・レベルでの EJB ベースの Web サービスの保護」を参照してください。

<endpoint-address-uri>

親要素： <port-component>

子要素： なし

必須 / オプション バージョン 2.1 の EJB に基づく Web サービスの場合のみ

EJB 2.1 の Web サービスでのみ必要となるこのサブ要素には、この EJB を Web サービスとして公開する際に使用する HTTP URL のサブコンテキストを指定します。値を指定しない場合、デフォルトで `port-component` 名に設定されます。この URI を保護するために 2 つのトランスポート・レベルの要素が用意されています。これらの要素の詳細は、「<ejb-transport-security-constraint>」および「<ejb-transport-login-config>」を参照してください。また、C-15 ページの「トランスポート・レベルでの EJB ベースの Web サービスの保護」も参照してください。

Web モジュール（Java クラスから導出された Web サービス）の場合、この情報はすでに `web.xml` ファイルに含まれており、不要です。

<expose-testpage>

親要素： <webservice-description>

子要素： なし

必須 / オプション オプション

このブール型要素には、テスト・ページを公開するかどうかを指定します。デフォルトは `true` です。

親要素： <provider-port>

子要素： なし

必須 / オプション オプション

このブール型要素には、テスト・ページを公開するかどうかを指定します。デフォルトは `true` です。

<expose-wsdl>

親要素： <webservice-description>

子要素： なし

必須 / オプション オプション

このブール型要素には、WSDL を公開するかどうかを指定します。デフォルトは `true` です。

親要素： <provider-port>

子要素： なし

必須 / オプション オプション

このブール型要素には、WSDL を公開するかどうかを指定します。デフォルトは `true` です。

<implementation-class>

親要素： <provider-port>

子要素： なし

必須 / オプション 必須

oracle.webservices.provider.Provider インタフェースを実装するクラスの名前を指定します。

<implementor>

親要素： <port-component>

子要素： <param>

必須 / オプション オプション

このサブ要素には、デプロイ時に生成される、OC4J 固有の Web サービスに関する情報が取得されます。このサブ要素には、param サブ要素があります。

表 C-1 <implementor> の属性

名前	説明
type	値 : implementorTypeAttr デフォルト : なし（必須） たとえば、データベース。

<jms-address>

親要素： <port-component>

子要素： なし

必須 / オプション オプション

JMS 宛先情報を指定するために使用されるアドレス要素。これは、JMS を介して SOAP メッセージを送信するために JMS トランスポートが指定されているときに使用されます。

表 C-2 <jms-address> の属性

名前	説明
jndiConnectionFactoryName	値 : string デフォルト : なし（必須） 使用されるコネクション・ファクトリの JNDI 名。Web サービスをボトムアップ方式で生成している場合は、この属性は sendConnectionFactoryLocation 引数によって設定されます。
jndiDestinationName	値 : string デフォルト : なし（必須） メッセージ送信先の JMS キューの JNDI 名。Web サービスをボトムアップ方式で生成している場合は、この属性は sendQueueLocation 引数によって設定されます。

<max-request-size>

親要素： <port-component>

子要素： なし

必須 / オプション オプション

この要素を使用すると、Web サービスに渡されるメッセージの最大サイズ（バイト単位）を構成できます。このバイト数を超えるメッセージを Web サービスが読み取ると、送信が失敗し、接続が閉じられます。

この要素の値は `long integer` です。正でない値を割り当てると、制限がないものとみなされます。デフォルトの `-1` は、無制限を意味します。サイズに関する制限がなくなります。

親要素： <provider-port>

子要素： なし

必須 / オプション オプション

正の値を指定すると、サービスはリクエストのサイズをその値（バイト単位）に制限します。最大長を超えるリクエストにはすべて、エラーが生成されます。デフォルトは `-1` で、無制限を示します。

<operation>

親要素： <operations>

子要素： <runtime>

必須 / オプション オプション

参照先の Web サービスにより提供される特定の操作に対する、サービスのクオリティ構成を指定します。この構成の指定は、この要素の `<runtime>` サブ要素で行います。

表 C-3 <operation> の属性

名前	説明
input	値 : string デフォルト : なし WSDL 内の操作の入力名を指定します。
name	値 : string デフォルト : なし（必須） 指定しているサービスのクオリティ構成を、この操作に関連付けます。属性の値は、WSDL 内の操作名と一致させる必要があります。
output	値 : string デフォルト : なし WSDL 内の操作の出力名を指定します。

<operations>

親要素： <policy>

子要素： <operation>

必須 / オプション オプション

操作ごとに要素を 1 つずつ指定した、一連の要素を指定します。<operation> サブ要素は、個々の操作に適用する管理ポリシーを記述します。

<oracle-webservices>

親要素： なし（ルート）

子要素： <context-root>、<ejb-transport-login-config>、<web-site>、<webservice-description>

必須 / オプション 必須

<oracle-webservices> 要素は、Web サービス用の Oracle コンテナ固有の情報を取得します。

表 C-4 <oracle-webservices> の属性

名前	説明
noNamespaceSchemaLocation	値 : URI デフォルト : なし この属性は、XML 文書が準拠するスキーマをパーサーに通知するための標準的な方法となります。

<param>

親要素： <port-component>

子要素： なし

必須 / オプション オプション

実行時のデータソースの JNDI 位置を指定します。

<policy>

親要素： <provider-port>

子要素： なし

必須 / オプション オプション

プロバイダに対する Web サービス管理ポリシーを定義します。

<port-component>

親要素： <webservice-description>

子要素： <ejb-transport-security-constraint>、<endpoint-address-uri>、<implementor>、<jms-address>、<max-request-size>、<rest-support>、<runtime>、<use-dime-encoding>

必須 / オプション オプション

この <port-component> 要素は、標準のデプロイメント・ディスクリプタ webservices.xml 内の同様の要素（ただし、-name が後ろに追加されています）にマップするための参照として使用します。これらの要素には、特定のポートに関する情報が含まれます。

表 C-5 <port-component> の属性

名前	説明
name	値 : string デフォルト : なし（必須） webservices.xml の name 要素にマップします。

<property>

親要素： <provider-description>

子要素： なし

必須 / オプション オプション

グローバルに定義されたプロパティ、つまりすべての定義済プロバイダ・ポートで使用できるプロパティを指定します。この要素には、必須の string 属性 name があります。

親要素： <provider-port>

子要素： なし

必須 / オプション オプション

ローカルに定義されたプロパティです。特定の名前のプロパティがローカルとグローバルの両方で定義されている場合は、ローカルに定義されているプロパティの方がグローバルに定義されているプロパティより優先されます。

<provider-description>

親要素： <oracle-webservices>

子要素： <property>、<provider-description-name>、<provider-port>, <wsdl-file>、<wsdl-service-name>

必須 / オプション オプション

この要素は、Web サービスをプロバイダ対応にし、プロバイダ・ポートのコレクションを識別します。Web サービス・プロバイダの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービス・プロバイダの使用方法」を参照してください。

<provider-description-name>

親要素： <provider-description>

子要素： なし

必須 / オプション 必須

この string 型要素は、プロバイダの説明の名前を指定します。

<provider-name>

親要素： <provider-port>

子要素： なし

必須 / オプション 必須

プロバイダの名前です。`<provider-port>` 内の文字列 `<provider-name>` は、デプロイメント・ディスクリプタ内で一意である必要があります。

<provider-port>

親要素： <provider-description>

子要素： <expose-testpage>、<expose-wsdl>、<implementation-class>、<max-request-size>、<policy>、<property>、<provider-name>、<servlet-link>、<wsdl-port>

必須 / オプション 必須

WSDL ポートを Web サービスのインタフェースおよび実装と関連付けます。ポートの名前をコンポーネントとして定義します。また、ポートとサーブレット・エンドポイントの関連付けも行います。`wsdl-file` を指定しないと、`provider-port` はパススルー・ゲートウェイを定義します。

<realm-name>

親要素： <ejb-transport-login-config>

子要素： なし

必須 / オプション オプション

このサブ要素には、Web サービスとして公開する EJB に対する HTTP Basic 認証内で使用するレルム名を指定します。詳細は、C-15 ページの「トランスポート・レベルでの EJB ベースの Web サービスの保護」を参照してください。

<resolve-relative-imports>

親要素： <webservice-description>

子要素： なし

必須 / オプション オプション

このブール型要素は、相対インポートを絶対 URL に解決するかどうかを指定します。デフォルトは `false` です。

<rest-support>

親要素： <port-component>

子要素： なし

必須 / オプション オプション

このブール型 rest 要素は、REST スタイルの GET および POST のリクエストおよびレスポンスをこのポートでサポートするかどうかを指定します。デフォルトは false です。

<role-name>

親要素： <ejb-transport-security-constraint>

子要素： なし

必須 / オプション オプション

この string 型要素には、セキュリティ・ロールの名前を指定します。この名前は、トークンの字句ルールに準拠する必要があります。デフォルト値は定義されていません。

ここで使用する role-name は、次のどちらかに該当する必要があります。

- この EJB アプリケーションに定義されているセキュリティ・ロール要素の 1 つのロール名。
- EJB アプリケーション内のすべてのロールを意味する予約済ロール名 * である。

このサブ要素に * とロール名の両方が入力されている場合、このサブ要素はすべてのロールを意味するものとして解釈されます。ロールが定義されていない場合、ユーザーはセキュリティ制約に記述されている Web アプリケーションの部分にアクセスできません。ロール名は大文字と小文字が区別されて照合されます。

この要素の使用方法の詳細は、C-15 ページの「トランスポート・レベルでの EJB ベースの Web サービスの保護」を参照してください。

<runtime>

親要素： <port-component>

子要素： なし

必須 / オプション オプション

実行時に読み取られる Web サービス管理情報の先頭を示しています。詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の「Web サービス管理スキーマの概要」を参照してください。

親要素： <policy>

子要素： なし

必須 / オプション オプション

参照先の Web サービスにより提供されるすべての操作に適用されるサーバー・サイドの、サービスのクオリティのランタイム情報（セキュリティ、信頼性、監査およびロギング）を指定します。各機能の構成は、それぞれの子要素に指定します。

<servlet-link>

親要素： <provider-port>

子要素： なし

必須 / オプション 必須

provider-port の値を、/WEB-INF/web.xml ファイルで定義されているサーブレット・エンドポイントと関連付けます。

<soap-port>

親要素： <ejb-transport-security-constraint>

子要素： なし

必須 / オプション オプション

この要素を指定した場合は、セキュリティ制約を SOAP ポートのみに適用する必要があることを示します。デフォルトは定義されていません。この要素の使用方法の詳細は、C-15 ページの「トランスポート・レベルでの EJB ベースの Web サービスの保護」を参照してください。

<transport-guarantee>

親要素： <ejb-transport-security-constraint>

子要素： なし

必須 / オプション オプション

この要素には、クライアント / サーバー間で転送されるデータへのアクセスに関する制約を指定します。このサブ要素には、次のいずれかの値を指定できます。

- NONE: アプリケーションにおいてトランスポート保証を不要にします。
- INTEGRAL: アプリケーションにおいて、クライアントとサーバーの間で転送されるデータが転送中に変更できないようにします。
- CONFIDENTIAL: アプリケーションにおいて、他の実体が転送内容を確認できないようにデータを転送するようにします。

多くの場合、INTEGRAL または CONFIDENTIAL フラグを指定するときは SSL を使用する必要があります。

この要素の使用方法の詳細は、C-15 ページの「トランスポート・レベルでの EJB ベースの Web サービスの保護」を参照してください。

<use-dime-encoding>

親要素： <port-component>

子要素： なし

必須 / オプション オプション

`true` に設定すると、このサービスから戻される添付ファイル付き SOAP レスポンスは DIME でエンコードされます。`false`（デフォルト）の場合、添付ファイル付き SOAP レスポンスは MIME エンコーディングで戻されます。DIME でエンコードされた添付ファイルの詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の DIME 添付ファイルの処理に関する項を参照してください。

<web-site>

親要素： <oracle-webservices>

子要素： なし

必須 オプション

デフォルト : サービス・エンドポイント・インタフェースがインストールされている位置

このオプションの要素には、更新する WSDL ポート位置内で代用するポート名およびホスト名を指定します。

たとえば、`port-component-link` による解決を使用して Web サービスにアクセスする場合や、`wsdl-publish-location` によって指定された位置に WSDL を公開する場合に、`oracle-webservices.xml` ファイルにこのサブ要素を入力する必要があることがあります。

この要素を使用しない場合、WSDL を取得するために使用する HTTP のホストおよびポート値が代用されます。

表 C-6 <web-site> の属性

名前	説明
host	値 : string デフォルト : なし（必須） 更新する WSDL ポート位置内で代用するホストの名前です。
port	値 : string デフォルト : なし（必須） 更新する WSDL ポート位置内で代用するポートの名前です。

<webservice-description>

親要素： <oracle-webservices>

子要素： <download-external-imports>、<expose-testpage>、<expose-wsdl>、<port-component>、<resolve-relative-imports>、<wsdl-file>、<wsdl-publish-location>

必須 / オプション オプション

この要素は、標準のデプロイメント・ディスクリプタ `webservices.xml` の `<webservice-description>` 要素を拡張するものです。

表 C-7 <webservice-description> の属性

名前	説明
name	値 : string デフォルト : なし（必須） `webservices.xml` の `name` 要素にマップします。

<wsdl-file>

親要素： <provider-description>

子要素： なし

必須 / オプション オプション

この string 型要素は、プロバイダに対する関連 WSDL の位置を指定します。

親要素： <webservice-description>

子要素： なし

必須 / オプション オプション

`wsdl-file` 型のこの要素は、デプロイ時に生成され、エンドユーザーは指定できません。最終的に更新される WSDL の位置を指定します。

表 C-8 <wsdl-file> の属性

名前	説明
final-location	値 : anyUri デフォルト : なし（必須） 最終的に更新される WSDL の位置を指定します。

<wsdl-port>

親要素： <provider-port>

子要素： なし

必須 / オプション オプション

プロバイダと、`wsdl-file` 要素によって定義される WSDL の特定のポートを関連付けます。WSDL ファイルが定義されている場合、この要素は必要ありません。

<wsdl-publish-location>

親要素： <webservice-description>

子要素： なし

必須 / オプション オプション

この要素を使用して、最終 WSDL とその依存ファイル（インポート）を格納する位置を指定します。型は anyUri で、値は `file:/location/` という書式に設定する必要があります。デフォルトは定義されていません。

<wsdl-service-name>

親要素： <provider-description>

子要素： なし

必須 / オプション オプション

WSDL のサービス名と、プロバイダの名前を関連付けます。WSDL に複数のサービスがある場合は、この要素を指定する必要があります。WSDL ファイルのサービスが 1 つのみの場合は、この要素は必要ありません。

<wsdl-url>

親要素： <ejb-transport-security-constraint>

子要素： なし

必須 / オプション オプション

この要素を指定した場合は、セキュリティ制約を WSDL URL のみに適用する必要があることを示します。デフォルト値は定義されていません。この要素の使用方法の詳細は、C-15 ページの「トランスポート・レベルでの EJB ベースの Web サービスの保護」を参照してください。

トランスポート・レベルでの EJB ベースの Web サービスの保護

`oracle-webservices.xml` デプロイメント・ディスクリプタには、EJB をベースにした Web サービスに対してトランスポート・レベルのセキュリティを定義できるようにするために、`<ejb-transport-security-constraint>` および `<ejb-transport-login-config>` という 2 つの要素が用意されています。

関連資料：

このトピックの詳細は、『Oracle Application Server Web Services セキュリティ・ガイド』の「EJB をベースとする Web サービスに対するトランスポート・レベル・セキュリティの追加」および「トランスポート・レベルで保護された Web サービスへのアクセス」も参照してください。

`<ejb-transport-security-constraint>` 要素は、Web サービスとして公開されるバージョン 2.1 の EJB に、トランスポート・レベルのセキュリティの制約を適用するために使用します。Web サービスとして公開する EJB の URL は、port component の `<endpoint-address-uri>` 要素によって指定します。
`<ejb-transport-security-constraint>` 要素には次のサブ要素があります。

- `<role-name>`: セキュリティ・ロールの名前を指定します。詳細は、C-11 ページの「<role-name>」を参照してください。

- <soap-port>: セキュリティ制約を SOAP ポートのみに適用する必要があることを示します。詳細は、C-12 ページの「<soap-port>」を参照してください。
- <transport-guarantee>: クライアント / サーバー間で転送されるデータへのアクセスに関する制約を指定します。詳細は、C-12 ページの「<transport-guarantee>」を参照してください。
- <wsdl-url>: セキュリティ制約を WSDL URL のみに適用する必要があることを示します。詳細は、C-15 ページの「<wsdl-url>」を参照してください。

サブ要素 <wsdl-url> および <soap-port> は、セキュリティ制約を WSDL URL と SOAP ポートのどちらに適用するかを選択する際に使用する識別子です。<ejb-transport-security-constraint> に <wsdl-url> と <soap-port> の両方が存在する場合、または両方が存在しない場合、セキュリティ制約は WSDL と SOAP ポートの両方に適用されます。

<ejb-transport-login-config> 要素を使用して、この EJB アプリケーションで使用する必要があるトランスポート・レベルの認証方式およびレルム名を構成します。Web サービスとして公開する EJB アプリケーションの URL は、port component の <endpoint-address-uri> 要素によって指定します。

- <auth-method>: EJB アプリケーションの認証メカニズムを構成します。詳細は、C-3 ページの「<auth-method>」を参照してください。
- <realm-name>: Web サービスとして公開する EJB に対する HTTP Basic 認証内で使用するレルム名を指定します。詳細は、C-10 ページの「<realm-name>」を参照してください。

oracle-webservices.xml ファイルのリスト

例 C-1 は、oracle-webservices.xml デプロイメント・ディスクリプタに対するテンプレートのリストです。なお、このテンプレート・ファイルには、Web サービス管理要素も含まれています。これらの要素の詳細は、このファイル内に記載された各参照先を参照してください。

例 C-1 サンプルの oracle-webservices.xml ファイル

```
<?xml version="1.0" encoding="UTF-8"?>
<oracle-webservices xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://xmlns.oracle.com/oracleas/schema/oracle-webservices-10_0.xsd"
   deployment-version="String" deployment-time="String" schema-major-version="10"
   schema-minor-version="0">
   <web-site host="String" port="String"/>
   <context-root>String</context-root>
   <webservice-description name="String">
      <expose-wsdl>true</expose-wsdl>
      <expose-testpage>true</expose-testpage>
      <resolve-relative-imports>false</resolve-relative-imports>
      <download-external-imports>false</download-external-imports>
      <port-component name="String">
         <endpoint-address-uri>String</endpoint-address-uri>
         <ejb-transport-security-constraint>
            <wsdl-url/>
            <soap-port/>
            <role-name>Manager</role-name>
            <role-name>Administrator</role-name>
            <transport-guarantee>NONE</transport-guarantee>
         </ejb-transport-security-constraint>
<!-- For a listing and description of elements that define transport-level security constraints for EJBs, see "Securing EJB-Based Web Services at the Transport Level" on page C-15. -->

         <implementor type="database">
            <param name="String">String</param>
         </implementor>
```

```
         <runtime enabled="String">
            <owsm/>
```
<!-- For a description of the element for the Oracle Web Services Manager (<owsm>) see the *Oracle Web Services Manager Administrator's Guide* -->

```
            <security>
```
<!-- For a description and listing of security elements, see the *Oracle Application Server Web Services Security Guide*. -->

```
            </security>
            <reliability>
               <repository jndiLocation="..." name="..." type="..."/>
```
<!-- For a description and listing of port-level reliability elements, see "Port-Level Reliability Elements on the Server" in the *Oracle Application Server Advanced Web Services Developer's Guide*. -->

```
            </reliability>
            <logging/>
         </runtime>
         <operations>
            <operation name="String" input="String">
               <runtime>
                  <security>
```
<!-- For a description and listing of security elements, see the *Oracle Application Server Web Services Security Guide*. -->

```
                  </security>
                  <reliability>
                     <duplication-elimination-required/>
                     <guaranteed-delivery-required/>
```
<!-- For a listing and description of operation-level reliability elements, see "Operation Level Reliability Elements on the Server" in the *Oracle Application Server Advanced Web Services Developer's Guide*. -->

```
                  </reliability>
                  <auditing request="false" response="false" fault="false"/>
```
<!-- For a listing and description of operation-level auditing elements, see "Server-Side Auditing Configuration Elements" in the *Oracle Application Server Advanced Web Services Developer's Guide*. -->

```
                  <logging>
```
<!-- For a listing and description of operation-level logging elements, see "Operation Level Logging Elements on the Server" in the *Oracle Application Server Advanced Web Services Developer's Guide*. -->

```
                  </logging>
               </runtime>
            </operation>
         </operations>
      </port-component>
   </webservice-description>
   <ejb-transport-login-config>
      <auth-method>BASIC</auth-method>
      <realm-name>sec-ejb</realm-name>
   </ejb-transport-login-config>
```
<!-- For a listing and description of elements that define transport-level security for EJBs, see "Securing EJB-Based Web Services at the Transport Level" on page C-15. -->

```
</oracle-webservices>
```

D

service-ref-mapping スキーマ

この付録では、service-ref-mapping スキーマについて説明します。このスキーマは、service-ref-mapping-10_0.xsd ファイルで定義されており、oc4j-schemas.jar ファイルにあります。

OC4J_HOME\j2ee\home\lib\oc4j-schemas.jar

このスキーマでは、OC4J によって実行時およびデプロイ時に生成される独自の Web サービス参照設定を定義する要素が定義されています。このスキーマで定義されている要素を使用して、次の情報を構成できます。

- ステートフル Web サービスを使用するためのクライアントの構成
- JMS トランスポート・コールを実行するためのクライアントの構成
- 対応する Web サービスに対するセキュリティ、ロギングおよび監査サービスのクオリティ（QOS）機能の構成

service-ref-mapping-10_0.xsd は、orion-web、orion-ejb-jar および orion-application-client の各 XSD にインポートされます。

<service-ref-mapping> 要素は、OC4J 固有のデプロイメント・ディスクリプタ・ファイルである orion-web.xml、orion-ejb-jar.xml または orion-application-client.xml の <orion-web-app> 要素のサブ要素として指定できます。デプロイメント・ディスクリプタへの追加として service-ref-mapping-10_0.xsd に値を設定するための Oracle JDeveloper ウィザードのようなツールは、現時点ではサポートされていません。スキーマを参照し、<service-ref-mapping> 要素に対する値を適切な XML ファイルに手動で入力する必要があります。

service-ref-mapping スキーマの階層

次に示すのは、service-ref-mapping スキーマの要素階層の概要です。

```
<service-ref-mapping>
  <service-impl-class>
  <wsdl-file>
  <wsdl-location>
  <service-qname>
  <stub-property>
   <name>
   <value>
  <call-property>
   <name>
   <value>
  <port-info>
    <wsdl-port>
    <service-endpoint-interface>
    <stub-property>
      <name>
      <value>
    <call-property>
      <name>
      <value>
    <runtime>
      ...
```

Under <runtime>, you can add client-side quality of service (Web Service Management) features. This can include a reliabilty and security configuration.

```
      ...
    <operations>
      <operation>
         <runtime>
           ...
```

Under <runtime>, you can add client-side quality of service (Web Service Management) features. This can include an auditing, reliabilty, and security configuration.

```
            ...
```

service-ref-mapping スキーマの要素と属性

ここでは、service-ref-mapping スキーマの要素をアルファベット順に示します。階層については、前の「service-ref-mapping スキーマの階層」を参照してください。

ここでの要素の説明は、orion-web.xml、orion-ejb-jar.xml または orion-application-client.xml の各固有デプロイメント・ディスクリプタ・ファイルに挿入される service-ref-mapping 要素に適用されます。

<call-property>

親要素： <service-ref-mapping> および <port-info>

子要素： <name>、<value>

必須 / オプション オプション

<call-property> 要素は、<service -ref-mapping> および <port-info> の両方のサブ要素になることができます。<call-property> を <service-ref-mapping> のサブ要素として使用すると、すべてのポートに適用されるコール・プロパティ値を定義します。これは、ポート名を指定せずにプロパティを指定できる便利な方法です。

<call-property> を <port-info> のサブ要素として使用すると、<port-info> で定義されている特定のポートに適用されるコール・プロパティ値を定義します。

特定のポートに対して、<port-info> タグ内でコールのプロパティ値を指定する場合は、<service-ref-mapping> に設定する <call-property> 要素の値がオーバーライドされます。

<name>

親要素： <call-property> および <stub-property>

子要素： なし

必須 / オプション 必須

<name> 要素は、<call-property> または <stub-property> のサブ要素として使用できます。JAX-RPC Call または Stub 実装でサポートされている任意のプロパティの名前を定義します。javax.xml.rpc.Call および javax.xml.rpc.Stub の有効なプロパティについては、Javadoc ツールの出力を参照してください。

<value> 要素は、プロパティに値を割り当てます。この要素の詳細は、D-6 ページの「<value>」を参照してください。

<operation>

親要素： <operations>

子要素： <runtime>

必須 / オプション オプション

参照先の Web サービスにより提供される特定の操作に対する、クライアント・サイドのサービスのクオリティ構成を指定します。この構成の指定は、この要素の <runtime> サブ要素で行います。

表 D-1 <operation> の属性

名前	説明
inputName	値 : string デフォルト : なし WSDL 内の操作の入力名を指定します。これは、name 属性を使用して操作を一意に特定できない場合にのみ必要です。
name	値 : string デフォルト : なし（必須） 指定しているサービスのクオリティ構成を、この操作に関連付けます。属性の値は、WSDL 内の操作名と一致させる必要があります。
outputName	値 : string デフォルト : なし WSDL 内の操作の出力名を指定します。これは、name および input 属性を使用して、操作を一意に特定できない場合にのみ必要です。

<operations>

親要素 : <port-info>

子要素 : <operation>

必須 / オプション オプション

操作ごとに要素を 1 つずつ指定した、一連の要素を指定します。個々の操作は、<operation> サブ要素に指定します。これらの各サブ要素には、参照先の Web サービスにより提供される各操作に対する、クライアント・サイドのサービスのクオリティ構成を指定します。

<port-info>

親要素 : <service-ref-mapping>

子要素 : <call-property>、<operations>、<runtime>、<service-endpoint-interface>、<stub-property>、<wsdl-port>

必須 / オプション オプション

サービス参照のポートを定義します。この要素は、サービス参照のポートの詳細を提供します。コンテナがコンテナ管理ポートの選択で使用するポートを、<service-endpoint-interface> または <wsdl-port> に指定します。両方を指定すると、<wsdl-port> の値が使用されます。<wsdl-port> や <service-endpoint-interface> を指定しない場合は、<port-info> プロパティの値が、すべての使用可能なポートに対して適用されます。

また、この要素には、ポートおよびその操作に対して、使用可能なサービスのクオリティ機能を指定できるサブ要素も含めることができます。

<runtime>

親要素： <port-info> および <operation>

子要素： なし

必須 / オプション オプション

<runtime> 要素は、<port-info> 要素および <operation> 要素のサブ要素として使用できます。<runtime> 要素を <port-info> のサブ要素として使用するときは、参照先の Web サービスにより提供されるすべての操作に適用されるクライアント・サイドの、サービスのクオリティのランタイム情報（セキュリティまたは信頼性（あるいはその両方））を指定します。各機能の構成は、それぞれの子要素に指定します。

<runtime> 要素を <operation> のサブ要素として使用するときは、ポート内の個別の操作に対するクライアント・サイドのサービスのクオリティ構成を指定します。サービスのクオリティ機能（セキュリティ、信頼性または監査（あるいはこれらの任意の組合せ））の各構成は、それぞれの子要素に指定します。

<service-endpoint-interface>

親要素： <port-info>

子要素： なし

必須 / オプション オプション

WSDL ポートのサービス・エンドポイント・インタフェースの完全修飾パスを指定します。コンテナは、このポートをコンテナ管理ポートの選択に使用します。この要素の値は string です。

<service-impl-class>

親要素： <service-ref-mapping>

子要素： なし

必須 / オプション オプション

Service 実装の、デプロイ時生成名を定義します。

<service-qname>

親要素： <service-ref-mapping>

子要素： なし

必須 / オプション オプション

この要素はデプロイ時に導出され、Web サービスの QName を格納します。

<service-ref-mapping>

親要素： ルート要素

子要素： <call-property>、<port-info>、<service-impl-class>、<stub-property>、<service-qname>、<wsdl-file>、<wsdl-location>

必須 / オプション オプション

<service-ref-mapping> 要素は、標準のデプロイメント・ディスクリプタ web.xml、ejb-jar.xml および application-client.xml に含まれている <service-ref> 要素と組み合せて使用します。<service-ref> 要素には、EJB、JSP またはサーブレットを、リモート Web サービスを起動するための Web サービス・クライアントとして使用するための情報を指定します。

<service-ref> 要素を web.xml、ejb-jar.xml または application-client.xml ファイルのいずれかに指定した場合には、対応する <service-ref-mapping> 要素を、orion-web.xml、orion-ejb-jar.xml または orion-application-client.xml ファイルのいずれかに指定できる点に注意してください。

最も単純なケースでは、<service-ref-mapping> 要素にデプロイメント情報のみを指定します。クライアントをマネージド・クライアントとして使用しない場合は、ランタイム要素つまりサービスのクオリティ要素を追加しないでください。マネージド・クライアントは、パフォーマンスの点で高負荷です。

<stub-property>

親要素： <service-ref-mapping> および <port-info>

子要素： <name>、<value>

必須 / オプション オプション

<stub-property> 要素は、<service-ref-mapping> および <port-info> の両方のサブ要素になることができます。<stub-property> を <service-ref-mapping> のサブ要素として使用するときは、すべてのポートに適用されるスタブル・プロパティ値を定義します。これは、ポート名を指定せずにプロパティを指定できる便利な方法です。

<stub-property> を <port-info> のサブ要素として使用するときは、<port-info> で定義されている特定のポートに適用されるスタブ・プロパティ値を定義します。

特定のポートに対して、<port-info> タグ内でスタブのプロパティ値を指定する場合は、<service-ref-mapping> に設定する <stub-property> 要素の値がオーバーライドされます。

<value>

親要素： <call-property> および <stub-property>

子要素： なし

必須 / オプション 必須

<value> 要素は、<call-property> または <stub-property> のサブ要素として使用できます。Call オブジェクトまたは Stub オブジェクトを Web サービス・クライアントに戻す前にそのオブジェクトに設定する JAX-RPC プロパティ値を定義します。

<name> 要素は、JAX-RPC プロパティの名前を割り当てます。この要素の詳細は、D-3 ページの「<name>」を参照してください。

<wsdl-file>

親要素： <service-ref-mapping>

子要素： なし

必須 / オプション オプション

WSDL ファイルのデプロイ時生成名を定義します。

表 D-2 <wsdl-file> の属性

名前	説明
final-location	値 : `anyURI` デフォルト : なし（必須） 標準のデプロイメント・ディスクリプタの `service-ref` に指定されている WSDL ドキュメントのコピーを指します。

<wsdl-location>

親要素： <service-ref-mapping>

子要素： なし

必須 / オプション オプション

WSDL ドキュメントを指す有効な URL を指定します。URL を指定すると、デプロイ時にこの URL の WSDL ドキュメントを、標準のデプロイメント・ディスクリプタで `service-ref` に指定された WSDL ドキュメントのかわりに使用します。`<wsdl-location>` のサンプル値としては、`http://hostname:port/myservice/myport?WSDL` および `file:/home/user1/myfinalwsdl.wsdl` があげられます。

表 D-3 <wsdl-location> の属性

名前	説明
wsdl-override-last-modified	値 : `string` デフォルト : なし このオプションの文字列値は、デプロイ時に生成され、WSDL ファイルが最後に変更された時間を表します。

<wsdl-port>

親要素： <port-info>

子要素： なし

必須 / オプション オプション

コンテナによりコンテナ管理ポートの選択に使用される WSDL 内ポート名を指定します。

コンテナ管理ポートの選択では、コンテナがインスタンスのコールを直接管理し、クライアントは、複数の異なるインスタンスへのアクセスに使用できる汎用ポートをリクエストします。

表 D-4 <wsdl-port> の属性

名前	説明
localpart	値 : string デフォルト : なし（必須） WSDL 名のローカル部分です。たとえば、`authenticateHeader` などです。
namespaceURI	値 : anyURI デフォルト : なし（必須） WSDL の名前空間の URI です。たとえば、`http://oracle.j2ee.ws/Header` などです。

service-ref-mapping ファイルのリスト

固有のデプロイメント・ディスクリプタで使用できる `service-ref-mapping` のサンプルは、14-9 ページの「デプロイおよび実行時に対する OC4J 固有のプラットフォーム情報の追加方法」を参照してください。

E

エラー・メッセージ接頭辞

表 E-1 は、OracleAS Web Services の異なる機能領域で発生する可能性のあるエラー・メッセージの接頭辞です。変数 *n* は整数を表しています。現在のリリースと以前のリリースではメッセージ接頭辞が変わっていることに注意してください。

表 E-1 OracleAS Web Services の機能のエラー・メッセージ接頭辞

バージョン 10.1.3.1 の接頭辞	バージョン 10.1.3 以前の接頭辞	OracleAS Web Services の機能領域
OWS-00*nnn*	J2EE-WSM*nnnnn*	Web サービス共通スタック・メッセージ
OWS-03*nnn*	J2EE-WSC*nnnnn*	Web サービスのクライアント関連メッセージ
OWS-04*nnn*	J2EE-WSS*nnnnn*	Web サービスのサーバー・サイド・メッセージ
OWS-08*nnn*	J2EE-WSD*nnnnn*	Web サービスのデプロイ関連メッセージ
OWS-11*nnn*	J2EE-WSW*nnnnn*	WSDL 1.1 API（OraWSDL）関連メッセージの Oracle 実装
OWS-12*nnn*	J2EE-WSJ*nnnnn*	SOAP with Attachments API for Java（SAAJ）関連メッセージ
OWS-15*nnn*	J2EE-WSIF*nnnnn*	Web Services Invocation Framework（WSIF）関連メッセージ
OWS-20*nnn*	J2EE-WSE*nnnnn*	Web サービスのセキュリティ関連メッセージ
OWS-25*nnn*	J2EE-WSL*nnnnn*	Web サービスの信頼性関連メッセージ
OWS-30*nnn*	J2EE-WSA*nnnnn*	WebServicesAssembler 関連メッセージ
OWS-40*nnn*	J2EE-WSR*nnnnn*	JAX-R 関連メッセージ

F

トラブルシューティング

この付録では、Oracle Application Server Web Services の使用時に発生する可能性のある問題の解決策を提供します。この付録の項のタイトルは、『Oracle Application Server Web Services 開発者ガイド』および『Oracle Application Server Web Services アドバンスト開発者ガイド』の章タイトルに対応しています。

OracleAS Web Services のメッセージ

配列値要素をコレクション型にシリアライズまたはデシリアライズできない

rpc-encoded の Web サービスで Java コレクション型（java.util.Map、java.util.Collection またはこれらのサブクラスなど）をパラメータ型または戻り型として使用する場合、ランタイムでは、これらのコレクション・パラメータに配列値要素を正しくシリアライズまたはデシリアライズできません。

rpc-encoded のメッセージ書式の使用時に、Java 配列タイプ用にシリアライザおよびデシリアライザが登録されるようにするには、Java 値タイプを作成して各 Java 配列を表現します。

1. 使用する Java 配列タイプごとに Java 値タイプを作成します。

   注意： Java 値タイプの名前の中には、「Array」という語を使用しないようにします。「Array」は予約済のパターンです。

   次の例は、demo/StringAry.java ファイルの内容を表現しています。ラッパー・クラスの StringAry は、Java の String[] 配列を表現しています。クラス名の接尾辞に「Ary」が使用されていることに注意してください。

   ```
   package demo;
   public class StringAry
   { public StringAry() { }
     public String[] getValue() { return m_value; }
     public void setValue(String[] value) { m_value=value; }
     private String[] m_value;
   }
   ```

2. すべての値タイプ用に、適切なシリアライザおよびデシリアライザが登録されるようにします。

   それには、Web サービスのアセンブルの際に valueType 引数を使用します。次の例では、この引数に、ステップ 1 で作成された demo/StringAry.java ファイルを指定しています。

   ```
   java wsa.jar -assemble -valueType demo.StringAry ...
   ```

3. コレクション型パラメータにおける配列値要素の設定および取得用に定義した値タイプを使用します。

   たとえば、次のクラス定義があるとします。

   ```
   package demo;
   public class Service extends java.rmi.Remote
   { java.util.Map getMap(String input)throws java.rmi.RemoteException
     { ... }
   }
   ```

   次のコードを記述して、配列値要素の 1 つとして String[] 値をマップに戻すことができます。

   ```
   HashMap map = new HashMap();
   String[] str_array = new String[]{"a","b","c"};
   StringAry sa = new StringAry();
   sa.setValue(str_array);
   map.put("myArray", sa);
   return map;
   ```

多次元配列を使用する Web サービスの公開時にエラーが発生した

多次元配列を使用する Web サービスを公開しようとするとエラーが発生します。たとえば、入力または戻り引数として多次元配列を取るメソッドがある Java クラスを公開しようとすると、エラーが戻されることがあります。

この問題の解決策としては次の 2 つが考えられます。

- 配列の次元ごとの JavaBean の作成
- rpc-encoded のメッセージ書式を使用した Web サービスの公開

配列の次元ごとの JavaBean の作成 : 配列の各次元を Java 値タイプにラップして制限を回避できます。

> **注意 :** Java 値タイプの名前の中には、「Array」という語を使用しないようにします。「Array」は予約済のパターンです。

次の例では、`public static class StringAry` により、文字列の内部配列をラップしています。`public StringAry[]` により、内部配列の配列を表現しています。すなわち、`String` Java 値タイプの配列を含んでいます。このコード・サンプル内で接尾辞「Ary」が使用されていることに注意してください。

```
package demo;
public interface SampleItf extends java.rmi.Remote

 // wrap the inner array as a Java value type
{ public static class StringAry
  { public StringAry() { }
    public String[] getValue() { return m_value; }
    public void setValue(String[] value) { m_value=value; }
    private String[] m_value;
  }

  // create an  array of the inner array elements
  public StringAry[] echoString2(StringAry[] input)
        throws java.rmi.RemoteException;
}
```

次の `Sample` クラスで示されているのは、`String` Java 値タイプの `StringAry[]` 配列を公開する方法です。

```
package demo;
public class Sample implements java.rmi.Remote, SampleItf
{ public SampleItf.StringAry[] echoString2(SampleItf.StringAry[] input)
         throws java.rmi.RemoteException
  { return input; }
}
```

rpc-encoded スタイルを使用して Web サービスを公開する方法 : rpc-encoded スタイルを使用して、多次元配列を使用する Web サービスを公開できます。次に例を示します。

```
package demo;
public interface SampleItf extends java.rmi.Remote
{ public String[][] echoString2(String[][] input)
         throws java.rmi.RemoteException;
}


package demo;
public class Sample implements java.rmi.Remote, SampleItf
{ public String[][] echoString2(String[][] input)
         throws java.rmi.RemoteException
  { return input; }
}
```

レスポンスまたはリクエストの処理中にデシリアライズ・エラーが発生した

このエラーは、デシリアライズ・ステップの最中に OracleAS Web Services がスキーマ検証の実行を試みると発生する場合があります。エラーの原因は、`wsdl:types` 要素にあるスキーマ定義と、SOAP ペイロードのワイヤ書式の不一致です。

`genProxy` でアセンブルされたクライアント・プロキシがサーバーからのレスポンスをデシリアライズしようとすると、このエラーが発生する場合があります。サーバーがリクエストをデシリアライズしようとしても発生する場合があります。

次に示すのは、ワイヤと WSDL で要素の順序が異なっている場合の例です。デシリアライズ・コードがワイヤの `ParentItemId` を処理すると、スキーマからの `Quantity` 要素と `PrePick` 要素をスキップします。ワイヤで `PrePick` が検出されると、処理は失敗します。`ParentItemId` の後にはこの要素は出現できません。

ワイヤ書式:

```
...
<ListOfProduct>
 <Product>
   <ParentItemId></ParentItemId>
   <PrePick>Y</PrePick>
   <Quantity>1</Quantity>
...
```

WSDL 定義:

```
...
<xsd:complexType name="Product">
  <xsd:sequence>
    <xsd:element name="Quantity" maxOccurs="1" minOccurs="0"
                 type="xsd:string"></xsd:element>
    <xsd:element name="PrePick" maxOccurs="1" minOccurs="0"
                 type="xsdLocal11:string1"></xsd:element>
   <xsd:element name="ParentItemId" maxOccurs="1" minOccurs="0"
                type="xsdLocal11:string30"></xsd:element>
...
```

次に示すのは、この状況でスローされる可能性のあるデシリアライズ・エラーの例です。

```
unexpected element name:
expected={http://myCompany.com/Catalog/Data/CategoryProduct}Description,
actual={http://myCompany.com/Catalog/Data/CategoryProduct}PrePick.
at oracle.j2ee.ws.common.util.exception.JAXRPCExceptionBase.<init>(JAXRPCExceptio
nBase.java:93)
at oracle.j2ee.ws.common.util.exception.JAXRPCExceptionBase.<init>(JAXRPCExceptio
nBase.java:111)
at oracle.j2ee.ws.common.encoding.DeserializationException.<init>(Deserialization
Exception.java:54)
at com.myCompany.catalog.data.categoryproduct.runtime.Product_LiteralSerializer.
doDeserialize(Product_Literal Serializer.java:471).
```

このエラーを回避する方法は次のとおりです。

- ワイヤ書式と WSDL で記述されている書式が一致するように、サービスの実装を変更します。

  この方法は、サービスの実装を変更でき、トップダウン方式による Web サービスのアセンブルのシナリオで作業している場合にのみ使用できます。スキーマ定義と SOAP ペイロードの不一致は、規約を実装するときにレスポンスの書式設定を誤ったことが原因である可能性があります。SAAJ API を使用している場合、つまり `dataBinding=false` を指定して Web サービスをトップダウン方式でアセンブルしているか、JAX-RPC ハンドラ・チェーンまたはプロバイダ API を使用してカスタム・コードを作成している場合に、このシナリオが発生する可能性があります。

- スキーマ定義がワイヤ書式と一致するように WSDL を編集します。

  サービスを所有していて、XML スキーマ定義のオーサリングで誤った場合は、サーバー・サイドでこの回避策を実施できます。サービスの実装を所有していない場合、または変更できない場合は、クライアント・サイドでこの方法を実施することもできます。

- `dataBinding=false` を指定して Web サービスを生成し、Java レイヤーに対して XML を無効にします。すべてのスキーマ型は、`SOAPElement` として表されます。その後、SAAJ API を使用して RAW XML を処理するカスタム・コードを作成します。

rpc-encoded 書式とデータ・バインディングに対する制限

OracleAS Web Services は、rpc-encoded メッセージ書式と `databinding=false` の組合せはサポートしていません。この組合せは業界内ではベスト・プラクティスとみなされていないためです。

document-encoded メッセージ書式が、OracleAS Web Services によってサポートされない

`style="document"` と `use="encoded"` の組合せは、SOAP 仕様では有効でも、OracleAS Web Services など、主要な Web サービスのプラットフォームではサポートされていません。

document-literal bare メッセージ書式が 1 つの入力部分に限定されている

OracleAS Web Services では、bare の場合の入力として 1 つの部分のみがサポートされています。その他の入力パラメータはすべて SOAP ヘッダー部分にマップする必要があります。

BigDecimal 値のシリアライズにより端数処理エラーが発生することがある

`java.Math.BigDecimal` で利用できるいくつかのコンストラクタがあります。これらのコンストラクタは、次の値タイプを入力として取れます。

- 倍精度浮動小数点
- 整数および桁移動子
- 10 進数の String 型表現

`BigDecimal(double)` コンストラクタを使用する際は十分注意してください。計算に端数処理エラーが入り込むことがあります。かわりに、`int` または `String` ベースのコンストラクタを使用してください。

たとえば、値 123.45 を取る次の文を考えてみます。

```
...
double d = 1234.45;
System.out.println(d);
System.out.println(new BigDecimal(d));
...
```

これらの文により次の出力が生成されます。第 2 の値は予期した値ではありません。

```
1234.45
1234.450000000000045474735088646411895751953125
```

getChildNode ではなく getFirstChild と getNextSibling を使用して NodeList を取得する

node.getChildNode を使用して取得した NodeList に対して反復処理を行うと、パフォーマンスが低下する場合があります。このような低下は、非常に長い NodeList の場合にのみ顕著です。

node.getChildNode で取得した NodeList を使用するかわりに、現在の Oracle XDK の実装では、node.getFirstChild を使用し、node.getNextSibling でループする方法により、子ノードのリストの移動に対する最適化が提供されています。次のコード・サンプルは、この技法を示したものです。

```
Node n = ...;
if (n.hasChildNodes()) {
   for(Node nd=n.getFirstChild(); nd!=null; nd=nd.getNextSibling()){
         nd.getValue(); // do something with nd
      }
}
```

WSDL からの Web サービスのアセンブル

document-literal メッセージ書式に関する制限

document-literal メッセージ書式を使用する Web サービスをトップダウン方式でアセンブルしようとすると、WebServicesAssembler により、同じ入力メッセージを使用する操作が WSDL で 2 つ以上検出された場合、警告が戻されます。OC4J ランタイムが、どのメソッドを起動するのかを識別できなくなるためです。

たとえば、次の WSDL フラグメントでは WebServicesAssembler により警告が戻されます。このフラグメントでは、addRelationship および addRelationship3 の操作が定義されています。これらの各操作で addRelationshipRequest 入力メッセージが使用されています。

```
...
<operation name="addRelationship">
     <input name="addRelationship1Request"
message="tns:addRelationship1Request"/>
            <output name="addRelationship1Response"
message="tns:addRelationship1Response"/>
        </operation>
         <operation name="addRelationship3">
            <input name="addRelationship1Request"
message="tns:addRelationship1Request"/>
            <output name="addRelationship1Response"
message="tns:addRelationship1Response"/>
        </operation>
...
```

クライアントから addRelationship 操作を起動する場合は、実装クラス内で操作が使用される順序に応じて、addRelationship または addRelationship3 が起動されます。

スキーマ機能の制限事項

SOAPElement にマップされるスキーマ機能

次のいずれかのスキーマ機能が WSDL で検出されると、`SOAPElement` にマップされます。

- 複数の `xsd:any` 要素を含むモデル・グループ
- `xsd:choice` 要素
- 混合コンテンツ
- 置換グループ
- 複数の `xsd:any` 属性を含む型

Web サービスをトップダウン方式でアセンブルする場合、または Web サービス・プロキシをアセンブルする場合には、WebServicesAssembler は、`xsd:choice` または `xsd:group` の XML 型を含む WSDL を使用できません。これらの XML 型を含む WSDL を使用する場合は、WebServicesAssembler の `dataBinding` 引数を `false` に設定し、WSDL ファイルでのスキーマ定義にペイロードが準拠するように `SOAPElement` をコーディングする必要があります。

rpc-encoded が属性の複雑な型をサポートしない

スキーマが rpc-encoded メッセージ書式とのバインディングを含む場合、WebServicesAssembler は、属性で complexType を検出すると、サポートされない型が検出されたことを示すエラー・メッセージをスローします。

Java クラスからの Web サービスのアセンブル

ステートフル Web サービスに対する制限事項

OracleAS Web Services により、ステートフル Web サービスは、Java クラスをベースにしたサービスに対してのみサポートされます。これらのサービスは、Oracle 独自の拡張機能を含んでおり、同じセマンティクスを持つスコープを用意したサービス・プロバイダとのみ相互運用可能ですので、注意してください。

OracleAS Web Services によるステートフル Web サービスのサポートは、HTTP をベースにしています。ステートフル Web サービスは、SOAP/HTTP エンドポイントに対してのみ機能し、SOAP/JMS エンドポイントに対しては機能しません。

Java クラスからの Web サービスのアセンブル : リリース 10.1.3.1 と 10.1.2 の相違点

Oracle Web サービスのリリース 10.1.2（およびそれ以前）とリリース 10.1.3.1 には次のような相違点があります。

- リリース 10.1.2 では、`RemoteInterface` の機能拡張やメソッドによる `RemoteException` のスローは必要ありませんでした。リリース 10.1.3.1 では、J2SE 5.0 Web サービスの注釈機能を使用していない場合には、これが必要になります。
- リリース 10.1.2 では、インタフェースを提供しなくても単独でクラスを公開できました。リリース 10.1.3.1 では、J2SE 5.0 Web サービスの注釈機能を使用していない場合には、クラスを公開するにはインタフェースを提供する必要があります。

EJB からの Web サービスのアセンブル

EJB に対するトランザクション境界（デマーケーション）の設定

Web サービスとして公開する EJB では、そのトランザクション境界（デマーケーション）として TX_REQUIRED または TX_MANDATORY を設定しないでください。

JMS 宛先を使用した Web サービスのアセンブル

サポートされているメッセージ・ペイロード・タイプ

JMS エンドポイント Web サービスの場合、OracleAS Web Services では、JMS メッセージのペイロードとして、java.lang.String または javax.xml.soap.SOAPElement のインスタンスのみがサポートされています。

SOAP メッセージ・ヘッダーの JMS プロパティ

SOAP ヘッダーによって送信できる JMS プロパティ数はごく限られています。genJmsPropertyHeader 引数の値が true の場合（デフォルト）は、次の JMS プロパティを SOAP ヘッダーによって送信できます。

- メッセージ ID
- 相関 ID
- 返信先（名前、タイプ、ファクトリなど）

データベース・リソースからの Web サービスの開発

データ型の制限

- Streams は、Oracle Streams AQ Web サービスではサポートされていません。
- SQL タイプの SYS.ANYDATA は PL/SQL Web サービスではサポートされていません。
- パラメータとしての REF CURSOR は PL/SQL Web サービスではサポートされていません。
- Oracle WebRowSet および XDB RowSet として戻される REF CURSOR では、その結果内で複合型はサポートされません。
- JDBC での制限のため、PL/SQL ストアド・プロシージャでは、次の SQL タイプは OUT または INOUT パラメータとしてサポートされていません。
  - char 型（char、character、nchar など）
  - long 型（long、long raw など）

10.1.3.1 とそれより前のリリースの間でのデータベース Web サービスの相違点

リリース 9.0.4 または 10.1.2 で生成されたデータベース Web サービス用に記述された Web サービス・クライアントは、リリース 10.1.3.1 でボトムアップ方式で生成されたデータベース Web サービスで使用しようとすると、失敗します。PL/SQL の構造に変更がなくても同様です。

この理由の 1 つは、SQL コレクション型がリリース 9.0.4 および 10.1.2 内では単一の配列プロパティを持つ複合型にマップされていたためです。リリース 10.1.3.1 では、かわりに配列に直接マップされます。

Web サービス・クライアントを再生成する場合は、クライアント・コードを記述しなおす必要があります。これは、再生成されるコードでは現在、BeanWrappingArray ではなく array[] が使用されるためです。

注釈を使用した Web サービスのアセンブル

サポートされていない Web サービス・メタデータ機能

Web Services Metadata for the Java Platform 仕様には、OracleAS Web Services でサポートされていない部分があります。たとえば、Java Architecture for XML Binding（JAXB）仕様の 2.2.2 および 2.2.3 項で定義されている「Start With WSDL」モードおよび「Start With WSDL and Java」モードは、OracleAS Web Services ではサポートされていません。OracleAS Web Services でサポートされているのは「Start With Java」モードのみです。

WebServicesAssembler コマンドの `assemble` または `genWsdl` を使用して、J2SE 5.0 Web サービス注釈準拠の注釈を付けて使用する WSDL を生成する場合、これらのコマンドは、注釈を含まないファイルの処理での使用時とは別の方法で指定する必要があります。

関連項目：

これらのコマンドを使用して J2SE 5.0 Web サービス注釈準拠の注釈を付けて使用する WSDL を生成する方法の詳細は、18-8 ページの「assemble」および 18-28 ページの「genWsdl」を参照してください。

注釈付きのクラスを WebServicesAssembler assemble コマンドに対する classpath 引数でリストする必要がある

WebServicesAssembler の `assemble` コマンドを使用して、注釈付きのクラスを Web サービスにアセンブルする場合は、`input` 引数と `classpath` 引数の両方でクラスを指定する必要があります。注釈付きでないクラスの場合は `input` 引数でリストされているファイルは classpath の一部であるとみなされるので、`classpath` は本来は必要ありません。

REST Web サービスのアセンブル

REST Web サービスのサポートに関する制限事項

次のリストは、OracleAS Web Services による REST Web サービスのサポートにおける制限事項です。

- REST は、literal 操作（リクエストとレスポンスの両方がリテラル）の Web サービス・アプリケーションに対してのみサポートされます。
- HTTP GET は、（必須の）複合パラメータのない Web サービス操作に対してのみサポートされます。
- ブラウザによっては、HTTP GET URL のサイズを制限している場合があります（一般的には 2000 文字以下）。パラメータ数を制限し、パラメータの値と名前を短くすることで、URL サイズを小さくするように注意してください。
- REST Web サービスでは、簡易 XML メッセージのみが送信されます。添付ファイル付きメッセージは送信できません。
- セキュリティや信頼性など、管理機能の多くは、REST Web サービスでは利用できません。これは、通常この情報の保持に使用する SOAP ヘッダーが、サービスの REST 起動では使用できないためです。
- REST 起動は、生成されたスタブまたは DII クライアントから行うことはできません。それらのクライアントからの起動は SOAP で行われます。
- プロバイダのフレームワークでは、REST はサポートされません。
- REST での操作名には、マルチバイト・キャラクタは使用できません。

Web サービス・デプロイのテスト

Web サービスのテスト・ページには、次の制限事項があります。

- Web サービス・テスト・ページでは、WS-Security に対して基本サポートのみが提供されます。エディタでは、SOAP エンベロープにユーザー名とパスワードのみを入力できます。暗号化や署名など、その他の複雑なまたは拡張された WS-Security 機能を使用するには、SOAP リクエストを起動ページで直接編集する必要があります。
- Web サービス・テスト・ページでは、添付ファイルをアップロードできません。
- テスト・ページを使用して WSIF サービスを起動することはできません。
- 独自の拡張機能が含まれる WSDL ファイルは、Web サービス・テスト・ページでは正常に動作しません。たとえば、JMS をトランスポートとして使用するサービスは、テスト・ページを使用してテストできません。

J2EE Web サービス・クライアントのアセンブル

クライアント・アプリケーションとスレッド使用量

クライアント・アプリケーションが処理（たとえば、別個のスレッドを使用する非同期コールを有効にする場合）用に独自のスレッドを作成する場合、アプリケーション・サーバーを `-userThreads` オプションで起動する必要があります。

```
java -jar oc4j.jar -userThreads
```

`-userThreads` オプションを使用すると、ユーザー作成スレッドからのコンテキストのルックアップとクラスのロードができるようになります。

JAX-RPC ハンドラの概要

WebServicesAssembler では、JAX-RPC メッセージ・ハンドラを構成するための Ant タスクが提供されます。ハンドラは、WebServicesAssembler コマンドラインを使用して構成することはできません。

SOAP ヘッダーの処理

強い型指定と ServiceLifecycle インタフェース

`ServiceLifecycle` インタフェースを使用すると、WSDL ファイルに宣言されていない可能性のある SOAP ヘッダー・ブロックにアクセスできますが、ブロックには強い型指定が行われません。SOAP ヘッダーの XML 構造を処理するには、その理解も必要です。SOAP ヘッダー・ブロックに対して強い型指定を行うには、WebServicesAssembler の `mapHeadersToParameters` 引数を `true` に設定するようにしてください（`true` がデフォルト値です）。これが可能なのは、SOAP ヘッダーが WSDL ファイルに宣言されており、SOAP ヘッダーのタイプが JAX-RPC でサポートされているタイプの場合のみです。

WebServicesAssembler の使用方法

ファイル名が長いためデプロイできない

生成されたファイル名とディレクトリ名を組み合せた長さが一定のサイズ制限を超える場合は、デプロイが失敗し、エラーがスローされます。サイズ制限はオペレーティング・システムに応じて異なります。たとえば、Windows オペレーティング・システムではサイズ制限は 255 文字です。

名前の長さは、WebServicesAssembler とデプロイ・コードで制御されます。ファイル名は、WebServicesAssembler により、Java クラスのメソッド名または WSDL の操作名に基づいて生成されます。ディレクトリは、デプロイ・コードにより、コード生成時に EAR および WAR ファイルの名前に基づいて作成されます。

生成されるファイル名およびディレクトリ名が長くならないようにするには、次の名前の文字数を妥当な長さに制限します。

- Java クラスのメソッド名
- WSDL の操作名
- OC4J インストール位置のディレクトリ名
- WAR ファイルのファイル名
- EAR ファイルのファイル名

また、J2SE 5.0 JDK の最新バージョン（jdk-1_5_0_06 以降）にアップグレードすることによっても、この問題を回避できます。

WebServicesAssembler エラーの詳細情報の取得

コマンドラインまたは Ant タスクに `debug` 引数を指定することで、WebServicesAssembler によって戻されるエラーに関する詳細な診断情報を取得できます。

関連項目：

この引数の詳細は、18-42 ページの「debug」を参照してください。

WebServicesAssembler でファイルをコンパイルできない

WebServicesAssembler でファイルを正常にコンパイルできないと、次のようなエラーが戻されます。

```
java? java.io.IOException: CreateProcess: javac -encoding UTF-8 -classpath
```

WebServicesAssembler で Java ファイルがコンパイルできるよう、`javac` コンパイラを利用できるようにする必要があります。パスには *`JAVA_HOME`*`/bin` を含めるようにしてください。

WebServicesAssembler が必要なクラスを見つけられない

すべての J2EE 1.4 アプリケーションに共通するクラスの使用が必要な場合もあります。標準 J2EE 1.4 クラスおよび Oracle データベース・クラスはすべて自動的に組み込まれます。Ant タスクの使用時に、WebServicesAssembler では、これらのクラスが含まれた JAR を検索する必要があります。

WebServicesAssembler Ant タスクでは、`wsa.jar` を検索してこれらの補足的なクラスをロードしようとします。Ant タスクでは、次の Ant プロパティまたは環境変数をこの順序で検索します。タスクによって `wsa.jar` が見つからないか、プロパティが定義されていない場合は、タスクは次のプロパティを検索します。

1. `oc4j.home`: OC4J のルート・インストール・ディレクトリが指定されている Ant プロパティ。このプロパティは、環境変数のかわりに使用できます。
2. `OC4J_HOME`: OC4J のルート・インストール・ディレクトリが指定されている環境変数。
3. `oracle.home`: Oracle 製品のルート・インストール・ディレクトリが指定されている Ant プロパティ。このプロパティは、環境変数のかわりに使用できます。

4. ORACLE_HOME: Oracle 製品のルート・インストール・ディレクトリが指定されている環境変数。

Ant タスクは、wsa.jar を検出すると、wsa.jar の位置をルートとして、そのマニフェスト・ファイルにリストされているすべてのクラスをロードします。

Web サービスとメッセージ部分が異なるディレクトリに生成される

意図的に Web サービスとメッセージ部分を異なるディレクトリに生成することができ、また同じことが非意図的に発生する場合もあります。そのための方法の 1 つは、targetNamespace 引数を誤って設定することです。

targetNamespace の値は、仕様に準拠している HTTP URL、準拠していない HTTP URL、または URI であってもかまいません。targetNamespace の値は、次のようにしてパッケージ名にマップされます。

- ターゲットの名前空間が仕様に準拠する HTTP URL の場合は、Java パッケージ名にマップするときに、要素の順序が逆になります。次に例を示します。

  http://hello.demo.oracle.com というターゲット名前空間は、パッケージ名 com.oracle.demo.hello にマップします。

- ターゲットの名前空間が仕様に準拠しない HTTP URL の場合は、要素の順序は変更されません。次に例を示します。

  http://hello.demo.oracle というターゲット名前空間は、パッケージ名 hello.demo.oracle にマップします。

- ターゲット名前空間が URN の場合は、値は、コンポーネントがアンダースコア（_）で区切られた単一の識別子にマップされます。次に例を示します。

  urn:oracle.demo.hello というターゲット名前空間は、識別子 oracle_demo_hello にマップします。

たとえば、パッケージ・パス oracle/demo/hello で Web サービスをアセンブルしているものとします。targetNamespace 引数に http://hello.demo.oracle（非準拠 URL）を設定すると、WSDL のターゲット名前空間は http://hello.demo.oracle になりますが、メッセージ部分はまだ名前空間 oracle.demo.hello（サービス・パッケージ名から）を使用しています。この WSDL でプロキシを生成すると、サービスは hello.demo.oracle に生成され、メッセージ部分は oracle.demo.hello に生成されます。

この場合は、targetNamespace 引数の値として http://oracle.demo.hello を使用する必要があります。それにかわる方法としては、Java コードが com/oracle/demo/hello にある場合は、targetNamespace 引数の値として http://hello.demo.oracle.com を使用します。このようにすると、生成されるコードは意図した Java パッケージに格納されます。

関連項目：

- 異なる URI 値に対するこの引数の使用方法の詳細は、18-66 ページの「targetNamespace」を参照してください。
- この引数の動作の詳細は、18-47 ページの「packageName」を参照してください。

要素名にマルチバイト・キャラクタが含まれる WSDL ファイルを、genInterface、genProxy および topDownAssemble が正しく処理できない

WSDL ファイルの要素名でマルチバイト・キャラクタが使用されている場合、genInterface、genProxy および topDownAssemble コマンドは WSDL ファイルを正しく処理できません。コマンドは Java ソース・ファイルを生成しますが、このソース・ファイルをコンパイルすることはできません。

この制限事項を回避するには、インタフェースを生成するときに、dataBinding 引数に false を設定します。

Web サービスのパッケージ化およびデプロイ

J2EE クライアントのパッケージ化

現行のツール・セットでは J2EE Web サービス・クライアントをパッケージできません。手動でクライアントをパッケージする必要があります。

関連項目：

J2EE Web サービス・クライアントのパッケージ方法の詳細は、14-19 ページの「J2EE クライアントのパッケージ構造の概要」を参照してください。

WSDL に複数の HTTP ポートがある場合の、正しいエンドポイント・アドレスの取得

`oracle-webservices.xml` の `<web-site>` または `<wsdl-publish-location>` 要素に値を入力する場合、戻された WSDL に複数の HTTP ポートがあるときは、その WSDL には正しいエンドポイント・アドレスが含まれていない可能性があります。

WebServicesAssembler ツールでは、作成する `oracle-webservices.xml` ファイルには、`<web-site>` または `<wsdl-publish-location>` 要素を挿入しません。これらの要素は手動で挿入する必要があります。

クライアントとサービス・クラスを同じ EAR ファイルに生成すると、OWS-04005 エラーが戻される

クライアントと Web サービス・クラスを同じディレクトリに生成すると、WebServicesAssembler はクライアント・クラスを Web サービスの EAR ファイルにパッケージする場合があります。この EAR ファイルをサーバーにデプロイすると、OracleAS Web Services からの `OWS-04005` エラーでデプロイは失敗します。

`OWS-04005` ポートでエラーが発生しました： ...。

たとえば、Application Server Control を使用して、生成されたクライアント・クラスを含む EAR ファイルをデプロイすると、次のようなエラーが戻されます。

```
ERROR OWS-04005 An error occurred for port: {http://ws.myservice.org/}HttpSoap11: no
serializer is registered for (class
org.myservice.ws.MyWebService_sayHello_ResponseStruct,
{http://ws.myservice.org/}sayHelloResponse)
```

Web サービス・テスト・ページでこのサービスを起動しようとすると、ページは診断メッセージで応答します。たとえば、次のような応答メッセージがテスト・ページに表示されます。

```
<env:Envelope  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
<env:Header/>
<env:Body>
 <env:Fault
   xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <faultcode>env:Server</faultcode>
  <faultstring>no serializer is registered for (class
org.rajkovic.ws.MyWebService_sayHello_ResponseStruct,
{http://ws.myService.org/}sayHelloResponse)</faultstring>
  <faultactor/>
 </env:Fault>
</env:Body>
</env:Envelope>
```

この問題を回避するには、クライアント・クラスと Web サービス・クラスを異なるディレクトリに生成してください。

相互運用可能な Web サービスの実現

WebServicesAssembler によって生成される名前の先行アンダースコア

WebServicesAssembler ツールのデフォルト動作は、Java パッケージ名から名前空間を生成することです。Java パッケージ名が先行アンダースコア（「_」）で始まる場合は、生成される名前空間の URI にはアンダースコアが含まれます。.NET WSDL ツールの一部のバージョンでは、生成された名前空間が有効であっても、それらの名前空間を消費できないことがあります。

この .NET の問題を回避するには、WebServicesAssembler の引数 `targetNamespace` または `mappingFileName`（あるいはその両方）を使用して、デフォルトでパッケージによって導出される名前空間を回避します。

AXIS プラットフォームが URI レスポンスの値として 0 をデシリアライズできない

JAX-RPC の仕様によると、J2SE 1.4 以降で実行するアプリケーションでは、`xsd:anyURI` 型は `java.net.URI` クラスにマップされる必要があります。1.4 より前の環境との互換性のため、JAX-RPC の実装は、この型を `java.lang.String` にマップすることが許されています。

しかし、AXIS プラットフォームは、`xsd:anyURI` を独自のクラス `org.apache.axis.types.URI` にマップしています。このため、新しい URI をインスタンス化するために 0 を渡すと、問題が発生する場合があります。

たとえば、AXIS プラットフォームが次のコード・サンプルで値 0 を使用すると、URI パーサーはポートに「0」を割り当てて、インスタンス化のためにコンストラクタをコールします。

```
java.net.URIret = new java.net.URI("0"); // <wsdl:part
name="inUri" type="xsd:anyURI"/>
return ret;
```

このような状況では、AXIS プラットフォームの `setPort` メソッドが次のようなエラーをスローします。

```
org.apache.axis.types.URI$MalformedURIException: Port cannot be set when host is null!
```

回避策としては、AXIS プラットフォームでは、`xsd:anyURI` のかわりに `xsd:string` を使用してください。

.NET クライアントが、OracleAS Web Services からの多次元 soapenc:array レスポンスを正しくデシリアライズできない

.NET クライアントは、多次元の SOAP エンコード配列（`soapenc:array`）を、配列の配列として誤って表します。このようなクライアントが、OracleAS Web Services から `soapenc:array` を含むレスポンスを受け取ると、次のようなエラーを戻します。

```
Unhandled Exception: System.InvalidOperationException: There is an error in XML
document (2, 713). ---> System.ArgumentException: SOAP-ENC:arrayType with
multdimensional array found at <ArrayOfArrayOf_xsd_anyType  xmlns='http://tips.cf'>.
```

.NET クライアントでのこのような制限事項を回避するには、.NET との相互運用性が必要な場合は、多次元の `soapenc:array` を使用しないようにします。1 次元の配列を使用するか、またはかわりに配列の配列を使用してください。

メッセージ添付ファイルの処理

swaRef 添付ファイルによるフォルトの Web サービスへの追加

OracleAS Web Services では、SOAP フォルト・メッセージに追加できるのは `swaRef` MIME タイプ添付ファイルのみです。`SWA` タイプ添付ファイルの追加はサポートされていません。

添付ファイルによるフォルトは、Web サービスを WSDL から（トップダウン方式で）アセンブルする場合にのみ、Web サービスに追加できます。Web サービスをボトムアップ方式でアセンブルする場合は追加できません。

添付ファイル用にサポートされているメッセージ書式

rpc-literal および document-literal の Web サービスのみが、WS-I Attachments Profile 1.0 によってサポートされています。したがって、これらのタイプのサービスのみが `swaRef` MIME タイプを使用できます。

rpc-encoded メッセージ書式を指定すると、WebServicesAssembler では、`swaRef` MIME 添付ファイルを渡せる Web サービスをアセンブルできなくなります。このようなサービスをアセンブルするには、別の書式を選択する必要があります。

Web サービスの管理

Application Server Control での制限事項

Application Server Control では、`wsmgmt.xml` ファイルに指定可能な内容をすべて変更できるわけではありません。たとえば、信頼性の構成に関する部分は変更できません。

Web サービスの信頼性の確保

OracleAS Web Services の信頼性に関する制限事項

- クライアント上の信頼性プロセスは、クライアント・プロセスと同じ期間有効です。クライアント・プロセスが永久的に無効になると、再試行する必要があるメッセージがすべて無視されます。
- 非同期ポーリング機能は、ポート・レベルでのみ、および構成によってのみ有効にできます。

監査メッセージおよびロギング・メッセージ

xpath 問合せに関する制限事項

`xpath` 問合せではプリミティブ型を戻す必要があります。つまり、この問合せではテキスト・ノードまたはまたは属性値のコンテキストを戻す必要があります。

`xpath` 問合せによって戻されるプリミティブ型には、少ない文字数を設定する必要があります。たとえば、120 文字を超えないようにします。

Java 値タイプのカスタム・シリアライズ

この項では、標準外のデータ型のカスタム・シリアライズに関する制限事項について説明します。

use="literal"

このリリースでは、メッセージ書式の use 部分としては、literal のみがサポートされています。literal には、rpc-literal と document-literal があります。このリリースでは、rpc-encoded はサポートされていません。

オブジェクト・グラフ

このリリースでは rpc-encoded がサポートされていないため、シリアライズおよびデシリアライズのこの初期サポートでは、href を使用したオブジェクト・グラフのマーシャリングはできません。Java オブジェクトがリクエストのパラメータまたはレスポンスの戻り値の中に複数の参照を持つ場合は、シリアライズおよびデシリアライズを行うと、オブジェクト・グラフが保持されなくなる可能性があります。

WSDL レベルおよびサービス・レベルでの構成

SoapElementSerializer は、サービスまたは WSDL ごとに構成します。たとえば、dateTime と oracle.sql.DATE 間のマッピングを表す SoapElementSerializer 実装は、dateTime と java.util.Calendar 間のデフォルト・マッピングに取って代わるように構成できます。この構成では、マッピングのすべてのインスタンスが置換されます。各操作レベルまたは各メッセージ・レベルでの構成は、このリリースではサポートされていません。

サブツリーのシリアライズ

各カスタム・シリアライザでは XML サブツリー全体を取得し、XML 要素オブジェクト・モデル全体のシリアライズおよびデシリアライズを実行します。たとえば、2 つのトップレベル complexTypes 用に開発および構成された 2 つのカスタム・シリアライザ TypeA および TypeB があるとします。TypeA には、TypeB サブ要素があります。カスタム・シリアライザが TypeB 用に構成されていても、TypeB サブ要素が TypeA のカスタム・シリアライザ内でシリアライズされる場合は、OC4J ランタイムは TypeB のカスタム・シリアライザを自動的には起動することができません。TypeA のカスタム・シリアライザが、TypeB サブ要素も処理する必要があります。TypeA のカスタム・シリアライザから TypeB のカスタム・シリアライザをコールすることは可能ですが、これは実装計画次第です。

document-literal ラッパー

カスタム・シリアライザを使用してグローバル complexType を処理するとします。この型はグローバル要素によって参照され、document-literal 操作の単一部分を定義します。unwrapParameters 引数を使用して戻り型とレスポンス・タイプをアンラップする場合は、前述のグローバル型は、入力メッセージのボディ部分としてグローバル要素を使用する操作で無視されます。

Web サービス・トランスポートとしての JMS の使用方法

トランスポート・メカニズムとして JMS を使用する場合のメッセージの相互運用性

トランスポート・メカニズムとして JMS を有効にする WSDL 拡張機能は、Oracle 独自のものです。この Web サービスによって生成されるメッセージは、他のベンダーによって提供されるアプリケーションまたはサービスと相互運用できないことがあります。

JMS Web サービス・トランスポートからのクライアント・レスポンスの取得

クライアント・プロセスがレスポンスや後続の戻り値を受け取ることなく無効になった場合に、クライアント・プロセスがその古いレスポンスをキューから取得する機能は用意されていません。

JMS プロパティの変更

Web サービスをデプロイした後で JMS バインディングのプロパティを変更する必要がある場合は、WSDL ドキュメントを手作業で編集し、JMS アドレス（`<jms:address>`）要素と JMS プロパティ値（`<jms:propertyValue>`）要素に正しい値を入力する必要があります。WSDL ドキュメントを編集した後、サービスを再デプロイします。

`oracle-webservices.xml` ファイルの JMS 値を変更することで、問題を解決しないでください。このファイルの JMS 部分は、サービスを再デプロイしてもリロードされません。`oracle-webservices.xml` ファイルを使用してこの問題を解決する場合は、WebServicesAssembler を使用してサービスを生成しなおす必要があります。

関連資料：

JMS アドレスおよび JMS プロパティ値の WSDL 要素の詳細は、『Oracle Application Server Web Services アドバンスト開発者ガイド』の JMS トランスポートに対する WSDL 拡張に関する項を参照してください。

Web サービス起動フレームワークの使用方法

この項では、OracleAS Web Services による Web サービス起動フレームワーク（WSIF）のサポートに関する制限事項について説明します。

- データベース WSIF では、データベース・アクセスのためプロバイダに渡せるのは、データソースのみで JDBC 接続ではありません。
- データベース WSIF はステートレスです。各操作は、開始時に JDBC 接続を取得し、終了時に JDBC 接続を閉じます。自動コミットは、JDBC 接続では常に有効になっています。データベース・オーバーヘッドが削減されるようにデータソースを設定する場合は、接続プーリングを使用することをお薦めします。
- Oracle Application Server Control の Web サービスの管理および監視では、SOAP サービスが直接監視できるのみです。Java、EJB、データベース WSIF バインディングなどの WSIF バインディングを活用するサービスとの対話は監視できません。SOAP プロトコルを完全にバイパスすることで、Oracle Application Server Control によって提供される Web サービス管理インフラストラクチャもバイパスされます。

Web サービス・プロバイダの使用

Web サービス・プロバイダの操作に対して MBean が重複して発生する

動的プロバイダ・エンドポイントを使用していて、Application Server Control の操作に対して MBean が重複して発生する場合は、`oracle-webvservices.xml` ファイルで、操作に対して `inputName` 属性と `outputName` 属性が指定されていることを確認してください。

G

サード・パーティ・ライセンス

この付録には、Oracle Application Server Web Services Security に付属のすべてのサード・パーティ製品のサード・パーティ・ライセンスを記載します。

Apache

このプログラムには、Apache Software Foundation（Apache）から提供されるサード・パーティ・コードが組み込まれています。Apache のライセンス条件に基づき、Oracle は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（Apache ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはありません。

Apache ライセンス契約は、付属する次の Apache コンポーネントに適用されます。

- Apache HTTP Server
- Apache JServ
- mod_jserv
- 正規表現パッケージ・バージョン 1.3
- Apache Expression Language（commons-el.jar 内にパッケージ化されています）
- mod_mm 1.1.3
- Apache XML 署名および Apache XML 暗号 v. 1.4（Java 対応）および 1.0（C++ 対応）
- log4j 1.1.1
- BCEL v. 5
- XML-RPC v. 1.1
- Batik v. 1.5.1
- ANT 1.6.2 および 1.6.5
- Crimson v. 1.1.3
- ant.jar
- wsif.jar
- bcel.jar
- soap.jar
- Jakarta CLI 1.0
- jakarta-regexp-1.3.jar
- JSP 標準タグ・ライブラリ 1.0.6 および 1.1
- Struts 1.1
- Velocity 1.3
- svnClientAdapter
- commons-logging.jar
- commons-el.jar
- standard.jar
- jstl.jar

Apache Software ライセンス

Apache Web Server 1.3.29 のライセンス

```
/* ====================================================================
 * The Apache Software License, Version 1.1
 *
 * Copyright (c) 2000-2002 The Apache Software Foundation.  All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The end-user documentation included with the redistribution,
 *    if any, must include the following acknowledgment:
 *       "This product includes software developed by the
 *        Apache Software Foundation (http://www.apache.org/)."
 *    Alternately, this acknowledgment may appear in the software itself,
 *    if and wherever such third-party acknowledgments normally appear.
 *
 * 4. The names "Apache" and "Apache Software Foundation" must
 *    not be used to endorse or promote products derived from this
 *    software without prior written permission. For written
 *    permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 *    nor may "Apache" appear in their name, without prior written
 *    permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 * ====================================================================
 *
 * This software consists of voluntary contributions made by many
 * individuals on behalf of the Apache Software Foundation.  For more
 * information on the Apache Software Foundation, please see
 * <http://www.apache.org/>.
 *
 * Portions of this software are based upon public domain software
 * originally written at the National Center for Supercomputing
Applications,
 * University of Illinois, Urbana-Champaign.
```

Apache Web Server 2.0 のライセンス

Copyright (c) 1999-2004, The Apache Software Foundation
Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
Copyright (c) 1999-2004, The Apache Software Foundation

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted"

means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and

do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

Apache SOAP

このプログラムには、Apache Software Foundation（Apache）から提供されるサード・パーティ・コードが組み込まれています。Apache のライセンス条件に基づき、Oracle は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（Apache ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはありません。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、Apache ソフトウェアは現状のままで Oracle から提供されるものであり、いかなる種類の保証またはサポートも Oracle または Apache から提供されません。

Apache SOAP License

Apache SOAP license 2.3.1

```
Copyright (c) 1999 The Apache Software Foundation.  All rights reserved.
TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION
1. Definitions.

      "License" shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      "Licensor" shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.

      "Legal Entity" shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
      "control" means (i) the power, direct or indirect, to cause the
      direction or management of such entity, whether by contract or
      otherwise, or (ii) ownership of fifty percent (50%) or more of the
      outstanding shares, or (iii) beneficial ownership of such entity.

      "You" (or "Your") shall mean an individual or Legal Entity
      exercising permissions granted by this License.

      "Source" form shall mean the preferred form for making modifications,
      including but not limited to software source code, documentation
      source, and configuration files.

      "Object" form shall mean any form resulting from mechanical
      transformation or translation of a Source form, including but
      not limited to compiled object code, generated documentation,
      and conversions to other media types.

      "Work" shall mean the work of authorship, whether in Source or
      Object form, made available under the License, as indicated by a
      copyright notice that is included in or attached to the work
      (an example is provided in the Appendix below).

      "Derivative Works" shall mean any work, whether in Source or Object
      form, that is based on (or derived from) the Work and for which the
      editorial revisions, annotations, elaborations, or other modifications
      represent, as a whole, an original work of authorship. For the purposes
      of this License, Derivative Works shall not include works that remain
      separable from, or merely link (or bind by name) to the interfaces of,
      the Work and Derivative Works thereof.

      "Contribution" shall mean any work of authorship, including
      the original version of the Work and any modifications or additions
      to that Work or Derivative Works thereof, that is intentionally
      submitted to Licensor for inclusion in the Work by the copyright owner
      or by an individual or Legal Entity authorized to submit on behalf of
```

the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents

of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the
origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

JSR 110

このプログラムには、IBM Corporation（IBM）から提供されるサード・パーティ・コードが組み込まれています。IBM 社のライセンス条件に基づき、Oracle は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（IBM ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはありません。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、IBM ソフトウェアは現状のままで Oracle から提供されるものであり、いかなる種類の保証またはサポートも Oracle または IBM から提供されません。

```
Copyright IBM Corporation 2003 - All rights reserved

Java APIs for the WSDL specification are available at:
http://www-124.ibm.com/developerworks/projects/wsdl4j/
```

Jaxen

このプログラムには、Apache Software Foundation（Apache）および Jaxen Project（Jaxen）から提供されるサード・パーティ・コードが組み込まれています。Apache および Jaxen のライセンス条件に基づき、Oracle は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（Apache および Jaxen ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはありません。

The Jaxen License

```
Copyright (C) 2000-2002 bob mcwhirter & James Strachan. All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are
permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of
conditions, and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of
conditions, and the disclaimer that follows these conditions in the documentation
and/or other materials provided with the distribution.

The name "Jaxen" must not be used to endorse or promote products derived from this
software without prior written permission. For written permission, please contact
license@jaxen.org.

Products derived from this software may not be called "Jaxen", nor may "Jaxen" appear
in their name, without prior written permission from the Jaxen Project Management
(pm@jaxen.org).

In addition, we request (but do not require) that you include in the end-user
documentation provided with the redistribution and/or in the software itself an
acknowledgment equivalent to the following: "This product includes software developed
by the Jaxen Project (http://www.jaxen.org/)." Alternatively, the acknowledgment may be
graphical using the logos available at http://www.jaxen.org/.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE Jaxen AUTHORS OR THE PROJECT
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

This software consists of voluntary contributions made by many individuals on behalf of the Jaxen Project and was originally created by bob mcwhirter and James Strachan . For more information on the Jaxen Project, please see http://www.jaxen.org/.

SAXPath

このプログラムには、SAXPath から提供されるサード・パーティ・コードが組み込まれています。SAXPath のライセンス条件に基づき、Oracle は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（SAXPath ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはありません。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、SAXPath ソフトウェアは現状のままで Oracle から提供されるものであり、いかなる種類の保証またはサポートも Oracle または SAXPath から提供されません。

The SAXPath License

Copyright (C) 2000-2002 werken digital. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the disclaimer that follows these conditions in the documentation and/or other materials provided with the distribution.

The name "SAXPath" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact license@saxpath.org.

Products derived from this software may not be called "SAXPath", nor may "SAXPath" appear in their name, without prior written permission from the SAXPath Project Management (pm@saxpath.org).

In addition, we request (but do not require) that you include in the end-user documentation provided with the redistribution and/or in the software itself an acknowledgment equivalent to the following: "This product includes software developed by the SAXPath Project (http://www.saxpath.org/)." Alternatively, the acknowledgment may be graphical using the logos available at http://www.saxpath.org/.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE SAXPath AUTHORS OR THE PROJECT CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFIWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This software consists of voluntary contributions made by many individuals on behalf of the SAXPath Project and was originally created by bob mcwhirter and James Strachan . For more information on the SAXPath Project, please see http://www.saxpath.org/.

W3C DOM

このプログラムには、World Wide Web Consortium（W3C）から提供されるサード・パーティ・コードが組み込まれています。W3C のライセンス条件に基づき、Oracle は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（W3C ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはありません。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、W3C ソフトウェアは現状のままで Oracle から提供されるものであり、いかなる種類の保証またはサポートも Oracle または W3C から提供されません。

The W3C License

W3C® SOFTWARE NOTICE AND LICENSE
http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
This work (and included software, documentation such as READMEs, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions.

Permission to copy, modify, and distribute this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications:

The full text of this NOTICE in a location viewable to users of the redistributed or derivative work.
Any pre-existing intellectual property disclaimers, notices, or terms and conditions. If none exist, the W3C Software Short Notice should be included (hypertext is preferred, text is permitted) within the body of any redistributed or derivative code.
Notice of any changes or modifications to the files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)
THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

索引

記号

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

あ

え

お

か

き

く

こ

て

と

な

は

ひ

ふ

へ

ほ

ま

め

も

る

れ

ろ