# Oracle® Application Server Web Services

開発者ガイド

10*g* リリース 2（10.1.2）

**部品番号 : B15738-01**

2005 年 1 月

ORACLE®

Oracle Application Server Web Services 開発者ガイド, 10*g* リリース 2（10.1.2）

部品番号 : B15738-01

原本名 : Oracle Application Server Web Services Developer's Guide, 10*g* Release 2 (10.1.2)

原本部品番号 : B14027-01

原本著者 : Thomas Van Raalte

原本協力者 : Rodney Ward, Jeremy Blanchard, Marco Carrer, Anirban Chatterjee, Daxin Cheng, David Clay, Tony D'Silva, Neil Evans, Bert Feldman, Kathryn Gruenefeldt, Steven Harris, Anish Karmarkar, Prabha Krishna, Sunil Kunisetty, Wai-Kwong (Sam) Lee, Gary Moyer, Steve Muench, Giuseppe Panciera, Wei Qian, Eric Rajkovic, Venkata Ravipati, Susan Shepard, Alok Srivastava, Zhe (Alan) Wu, Joyce Yang, Chen Zhou

# 目次

## 5 ストアド・プロシージャ Web サービスの開発とデプロイ



## 6 ドキュメント・スタイルの Web サービスの開発とデプロイ



## 7 JMS Web サービスの開発とデプロイ

## 8 Web サービスを使用するクライアントの作成



## 9 Web サービスのツール



## 10 Web サービスの検出と公開

# A Oracle Application Server SOAP の使用

## B Web サービスのセキュリティ



## C OracleAS Web Services のトラブルシューティング

# はじめに

このマニュアルでは、Oracle Application Server Web Services について説明します。

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。

- Oracle Application Server にインストールされているソフトウェアの構成
- Web サービス実装プログラムの作成
- Web サービスのクライアントとして動作するプログラムの作成

このマニュアルを使用するには、Java プログラミング言語の基礎に関する実務上の知識が必要です。

# 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

JAWS（Windows のスクリーン・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。

# このマニュアルの構成

このマニュアルは次の章で構成されています。

### 第 1 章「Web サービスの概要」

この章では、Oracle Application Server Web Services の概要について説明します。

### 第 2 章「Oracle Application Server Web サービス」

この章では、Oracle Application Server Web Services の機能、アーキテクチャおよび実装について説明します。

### 第 3 章「Java クラス Web サービスの開発とデプロイ」

この章では、Java クラスとして実装される Oracle Application Server Web Services を記述する手順とデプロイする手順について説明します。

### 第 4 章「EJB Web サービスの開発とデプロイ」

この章では、ステートレス Session Enterprise JavaBeans（EJB）として実装される Oracle Application Server Web Services を記述してデプロイする手順について説明します。

### 第 5 章「ストアド・プロシージャ Web サービスの開発とデプロイ」

この章では、PL/SQL ストアド・プロシージャまたはファンクションとして実装される Oracle Application Server Web Services を記述してデプロイする手順について説明します。

### 第 6 章「ドキュメント・スタイルの Web サービスの開発とデプロイ」

この章では、Java クラスとして実装されるドキュメント・スタイルの Oracle Application Server Web Services を記述してデプロイする手順について説明します。

### 第 7 章「JMS Web サービスの開発とデプロイ」

この章では、JMS 宛先を Web サービスとして公開する Oracle Application Server Web Services を記述してデプロイする手順について説明します。

### 第 8 章「Web サービスを使用するクライアントの作成」

この章では、Oracle Application Server Web Services を使用するクライアント・アプリケーションを作成する手順について説明します。

### 第 9 章「Web サービスのツール」

この章では、Oracle Application Server Web Services のアセンブリ・ツール `WebServicesAssembler` について説明します。このツールを使用すると、Oracle Application Server Web Services をアセンブルできます。

### 第 10 章「Web サービスの検出と公開」

この章では、Universal Discovery Description and Integration（UDDI）準拠の Web サービス・レジストリについて説明します。エンタープライズ環境内のビジネス Web サービス・プロバイダは、このレジストリに Web サービスを公開し、記述できます。

### 第 11 章「J2EE アプリケーションでの Web サービスの使用」

この章では、J2EE アプリケーションで Web サービスを使用する方法について説明します。

### 第 12 章「Web サービスの高度な機能」

この章では、型未指定のリクエスト処理オプションや SOAP ヘッダーのサポートなど、Oracle Application Server Web Services のいくつかの拡張機能について説明します。

### 付録 A「Oracle Application Server SOAP の使用」

この付録では、OracleAS SOAP、Apache SOAP および OracleAS SOAP の違いについて説明します。

### 付録 B「Web サービスのセキュリティ」

この付録では、Oracle Application Server UDDI Registry も含め、Oracle Application Server Web Services に関するセキュリティのアーキテクチャと構成について説明します。

### 付録 C「OracleAS Web Services のトラブルシューティング」

この付録では、Web サービスでの問題のトラブルシューティングについて説明します。

### 用語集

用語集では、Web サービスの用語について説明します。

# 関連ドキュメント

詳細は、次のマニュアルを参照してください。

- 『Oracle Application Server 10*g* 概要』
- 『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』

リリース・ノート、インストール関連ドキュメント、ホワイト・ペーパーまたはその他の関連ドキュメントは、OTN-J（Oracle Technology Network Japan）から、無償でダウンロードできます。OTN-J を使用するには、オンラインでの登録が必要です。登録は、次の Web サイトから無償で行えます。

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

すでに OTN-J のユーザー名およびパスワードを取得している場合は、次の URL で OTN-J Web サイトのドキュメントのセクションに直接接続できます。

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

# 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 |
|---|---|
| .<br>.<br>. | 例の中で使用されている垂直の省略記号は、その例に直接関係のない情報が省略されていることを示します。 |
| ... | 文またはコマンドの中で使用されている水平の省略記号は、例に直接関係のない文またはコマンドの一部が省略されていることを示します。 |
| **太字のテキスト** | テキスト内の太字は、本文または用語集、あるいはその両方で定義されている用語を示します。 |
| [ ] | 大カッコで囲まれている句は、その中から 1 つを選択するか、または何も選択しないオプションの句を示します。 |
| $ | ドル記号は、Windows のコマンド言語のプロンプトおよび UNIX の Bourne シェルのプロンプトを示します。 |

# 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。

### オラクル社カスタマ・サポート・センター

オラクル製品サポートの購入方法、およびオラクル社カスタマ・サポート・センター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、Web サービスの概要について説明します。Oracle Application Server Web Services の機能、アーキテクチャおよび実装については、第 2 章「Oracle Application Server Web サービス」を参照してください。

この章の内容は、次のとおりです。

- Web サービス
- Web サービス標準の概要
- SOAP メッセージ交換および SOAP メッセージ・エンコーディング

# Web サービス

Web サービスは、メッセージ・プロトコル、プログラミング標準、ネットワーク登録および検出機能のセットで構成されており、Web に接続しているあらゆるデバイスから、許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します。

この項の内容は、次のとおりです。



## Web サービスの理解

**Web サービス**は、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 構造によって定義、記述および検出することができます。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。WSDL ファイルを提供することによって、Web サービスは、この機能を他のアプリケーションで使用できるようにしています。
- Web 上での他のサービスによる検索 - Web サービスは UDDI レジストリに登録できるため、アプリケーションで検索できます。
- 起動 - 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、サービスを起動できます。
- Web サービスは、リクエスト / レスポンス・スタイルまたは一方向スタイルのどちらかであり、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 サービスのメリット

ビジネス・プロセスを効率化するために Web サービスを開発および使用する企業にとって、次のようなメリットがあります。

- オープンなインターネット標準のサポート。Oracle は、Web サービス開発用の主な標準として SOAP、WSDL および UDDI をサポートしています。Oracle 製品を使用して開発された Web サービスは、Microsoft 社の .NET アーキテクチャに対して開発された Web サービスと相互運用できます。
- 単純で生産性の高い開発機能。Oracle は、J2EE アプリケーションの場合と同じプログラミング・モデルを使用して Web サービスを開発できるように、開発者に使用しやすく生産性の高い環境を提供します。
- ミッション・クリティカルなデプロイ機能。Oracle は、Web サービスと J2EE のランタイム・インフラストラクチャを統一することで、Web サービスのデプロイ用にミッション・クリティカルなプラットフォームを提供します。Oracle Application Server Web Services には最適化機能があり、Web サービスのレスポンスを高速化し、単一または複数の CPU 上で Web サービスを拡張し、フォルト・トレラントな設計とクラスタリングを通じて高可用性を提供します。

**関連項目：** 1-4 ページの「Web サービス標準の概要」

## Web サービスによる E-Business への変換

ビジネスを E-Business へと変換する動きにより、世界中の企業が自社業務の管理にインターネットを採用し始めています。しかし、インターネット・ビジネスは依然として一連のローカル・ノード、つまり、Point-to-Point 通信を使用する Web サイトとして機能しています。ビジネスのオンライン化が進むにつれて、インターネットはこのように静的にではなく汎用のビジネス・ネットワークとして使用されるようになり、サービスが自由にやりとりされ、アプリケーション間での対話やネゴシエーションが可能になります。

E-Business への変換を可能にするには、インターネットに、企業やエンタープライズ・アプリケーション間での通信を効率的にする標準ベースのインフラストラクチャが必要になります。これらの標準では、個別のビジネス・プロセスの公開およびインターネット上での記述、他のサービスからの検索および起動、予測可能なレスポンスの提供を可能にする必要があります。

Web サービスは、ビジネスの機能の仕方やエンタープライズ・アプリケーションの開発およびデプロイ方法を根本的に変化させることにより、この変換を促進します。

この E-Business への変換は、次の 2 つの分野で発生しています。

- Web サービスによるビジネスの変換
- Web サービスによるテクノロジの変換

### Web サービスによるビジネスの変換

Web サービスにより、次世代の E-Business や、カスタマ集約型の、時代の流れに敏感な企業は、次のことができます。

- 市場の拡大 - 既存および新規の顧客にインターネット経由でサービスとしてビジネス・プロセスを提供し、新たなグローバル・チャネルを開いて新たな収益のチャンスを獲得します。
- 効率の改善 - 最新情報を使用してリアルタイムで措置を講じ、企業全体およびビジネス・パートナとの間でビジネス・プロセスを効率化します。
- サプライヤおよびパートナとの関係強化 - サプライ・チェーン内で緊密に連携しているビジネス・パートナとの間に、事前に定義され契約で交渉済の組織的な関係と、動的で単発的な提携関係を生み出して維持します。

### Web サービスによるテクノロジの変換

Web サービスにより、エンタープライズ・アプリケーションで次のテクノロジ変換が可能になります。

- 開発とデプロイ - Web サービスを生産的な方法で迅速に開発し、デプロイできます。
- サービスの検索 - Web サービスによって、アプリケーションをインターネット・ポータル、エンタープライズ・ポータル、あるいはインターネット版イエロー・ページとして機能するサービス・レジストリ内で集約させ、検出できます。
- サービスの統合 - Web サービスにより、アプリケーションでは社内や社外にある他のアプリケーションを検索し、電子媒体を通じて通信できます。
- サービスの相互運用 - Web サービスにより、様々なプログラミング言語を使用し、様々なコンポーネントのパラダイムに従って開発されたアプリケーションを相互運用できます。

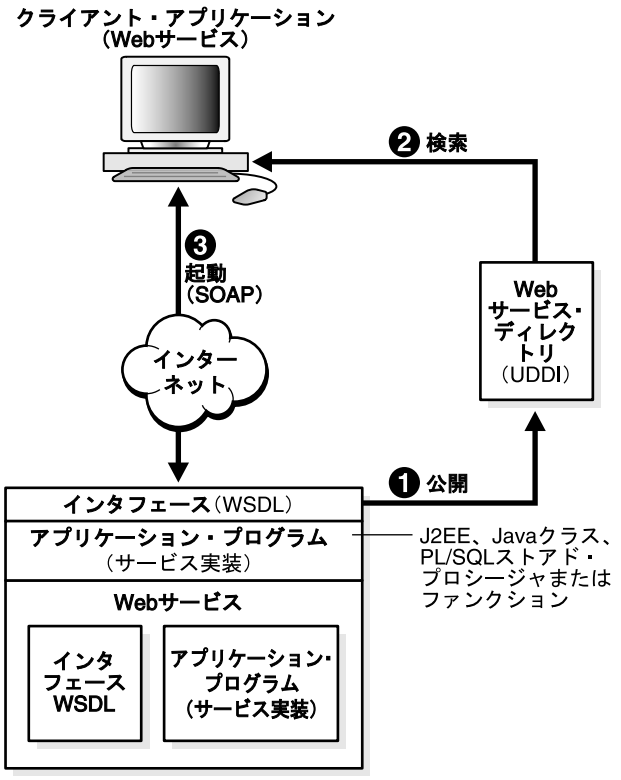
## Web サービス標準の概要

この項では、Web サービスを構成する次のインターネット標準について説明します。

- SOAP 標準
- Web Services Description Language（WSDL）
- Universal Description, Discovery, and Integration（UDDI）

図 1-1 に、これらの標準を使用する Web サービスのアーキテクチャの概念を示します。

**図 1-1 Web サービスの標準**



## SOAP 標準

**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**の情報交換では、リクエスト / レスポンス処理が可能です。この場合、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、ビジネスや他のタイプ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したり待機することができません。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
- 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）

> **関連項目：** SOAP 1.1 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（WSDL）

Web Services Description Language（WSDL）は、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 フォーマットです。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。

> **関連項目：** Web Services Description Language（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。

## Universal Description, Discovery, and Integration（UDDI）

Universal Description, Discovery, and Integration（UDDI）仕様は、電子的なイエロー・ページとして機能するオンライン電子レジストリです。各種ビジネス・エンティティが WSDL 定義経由で提供するサービスおよびビジネス・エンティティ自体を登録する情報構造を提供します。

UDDI レジストリには 2 つのタイプがあります。一方のパブリック UDDI レジストリはサービスを公開する各種ビジネスの集合ポイントとして機能し、他方のプライベート UDDI レジストリは社内で同様の役割を果たします。

> **関連項目：** Universal Description, Discovery, and Integration（UDDI）仕様の詳細は、`http://www.uddi.org` を参照してください。

# SOAP メッセージ交換および SOAP メッセージ・エンコーディング

SOAP 標準は、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を定義します。SOAP では、リモート・プロシージャ・コール（**RPC スタイル**）やメッセージ指向（**ドキュメント・スタイル**）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。SOAP メッセージでは、RPC スタイルかドキュメント・スタイルかに関係なく、SOAP メッセージの要素に指定された `encodingStyle` 属性の指定に従って、特定のエンコーディングが使用されます。この項では、これらの SOAP メッセージ機能について次の各項で説明します。

- SOAP メッセージのコンポーネント
- RPC スタイルの SOAP メッセージの操作
- ドキュメント・スタイルの SOAP メッセージの操作

## SOAP メッセージのコンポーネント

各 SOAP メッセージは、SOAP 送信者と SOAP 受信者の間の伝達手段です。各 SOAP メッセージは、2 つの副要素 `Header` と `Body` が含まれる SOAP エンベロープで構成されます。SOAP ヘッダーはオプションです。SOAP ヘッダーの子は `header blocks` と呼ばれ、各ヘッダー・ブロックは、データの論理グループを表します。SOAP ボディは、SOAP メッセージ内の必須要素です。ボディによって、SOAP メッセージで伝達されるエンドツーエンドの情報が搬送されます。どのデータをヘッダー・ブロックに置き、どのデータを SOAP ボディの要素にするかは、アプリケーションの設計時に決定する事項です。

開発者は、Oracle Application Server Web Services を使用して、RPC スタイルまたはドキュメント・スタイルのメッセージが実装でサポートされているかどうかを判断します。開発者は、その実装に適したアプリケーション・ロジックおよび WebServicesAssembler 構成ファイルを作成します。

## RPC スタイルの SOAP メッセージの操作

Oracle Application Server Web Services では、RPC スタイルとドキュメント・スタイルの 2 種類の SOAP メッセージ交換がサポートされます。RPC スタイルによる交換は、リモート・プロシージャ・コール（RPC）としてモデリングされる交換を表します。これらは、リモート・コールとその戻り値用に明確に定義されたシグネチャに交換メッセージが準拠する、特定のプログラム動作をモデリングする必要がある場合に使用されます。RPC スタイルのメッセージを使用した場合、SOAP は SOAP メッセージのボディの書式を指定します。

RPC スタイルによる情報交換では、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、レスポンス・メッセージで応答します。RPC スタイルの SOAP メッセージ交換を使用した場合、SOAP メッセージのボディの内容は、プロシージャを指定し、一連のパラメータ、または結果と追加のパラメータが付加されたレスポンスを含む構造に準拠します。ボディ内の SOAP メッセージは XML 文書ですが、SOAP 仕様で指定された制限に準拠する XML 文書です。

例 1-1 に、SOAP の RPC スタイルのリクエストを示します。このリクエストには、パラメータがいくつか指定された `ChargeReservation` メソッドが含まれます。例 1-2 に、SOAP の RPC スタイルのレスポンス・メッセージを示します。このレスポンスには、Response 文字列が追加された `ChargeReservationResponse` が含まれます。

**例 1-1 SOAP の RPC スタイルのリクエスト・メッセージ**

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:helloWorld xmlns:ns1="urn:oracle-j2ee-ws_example-StatelessExample"
           SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        <param0 xsi:type="xsd:string">Wendy</param0>
      </ns1:helloWorld>
   </SOAP-ENV:Body>
```

```
</SOAP-ENV:Envelope>
```

**例 1-2 SOAP の RPC スタイルのレスポンス・メッセージ**

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:helloWorldResponse
           xmlns:ns1="urn:oracle-j2ee-ws_example-StatelessExample"
           SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
             <return xsi:type="xsd:string">Hello World, Wendy</return>
      </ns1:helloWorldResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

## ドキュメント・スタイルの SOAP メッセージの操作

Oracle Application Server Web Services では、RPC スタイルとドキュメント・スタイルの 2 種類の SOAP メッセージ交換がサポートされます。ドキュメント・スタイルの交換（メッセージ指向の交換とも呼ばれます）は、XML 文書の交換をモデリングします。交換のパターンは、送信アプリケーションと受信アプリケーションで定義されます。ドキュメント・スタイルのメッセージの場合、SOAP メッセージのボディで送信されるドキュメントの構造について SOAP による制限はありません。SOAP メッセージのボディで送信される XML 文書の構造は、アプリケーションまたは外部で指定された XML Schema によって決まります。

メッセージ指向の情報交換では、ビジネスや他のタイプ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したり待機することができません。メッセージ指向の情報交換は、ドキュメント・スタイルの SOAP メッセージ交換とも呼ばれます。ドキュメント・スタイルのメッセージは、XML 文書の交換をモデリングします。交換パターンのセマンティクスは、送信アプリケーションと受信アプリケーションで定義されます。

例 1-3 に、ドキュメント・スタイルの SOAP メッセージのサンプルを示します。このメッセージは、クライアントから Oracle Application Server Web Services のドキュメント・スタイルのサービスに送信されます。クライアントは、名前、emp_id、部署および連絡先情報などの要素が格納された従業員レコードを含む XML 文書を送信します。電話番号のリストを生成するためにこの XML 文書を処理する Web サービスは、名前と電話番号の要素のみを含む XML 文書を提供できます。

**例 1-3 ドキュメント・スタイルの SOAP メッセージ**

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
<organisation>
  <employee>
   <name>Bob</name>
   <emp_id>1234</emp_id>
   <department>hr</department>
   <contact>
      <phone>827 644 5674</phone>
      <email>bob@organisation.com</email>
   </contact>
  </employee>
  <employee>
   <name>Susan</name>
   <emp_id>2434</emp_id>
   <department>it</department>
   <contact>
```

```
      <phone>827 644 5674</phone>
      <email>Susan@organisation.com</email>
   </contact>
  </employee>
</organisation>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**例 1-4 Web サービスによって処理されたドキュメント・スタイルの SOAP メッセージ**

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
   <employee>
      <name>Bob</name>
        <phone>827 644 5674</phone>
      <name>Susan</name>
        <phone>827 644 5674</phone>
   </employee>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

# 2

# Oracle Application Server Web サービス

この章では、Oracle Application Server Web Services の機能、アーキテクチャおよび実装について説明します。

この章の内容は、次のとおりです。

# Oracle Application Server OC4J（J2EE）ベースの Web サービスおよび OracleAS SOAP ベースの Web サービス

Oracle Application Server では、2 つの異なる Web サービス・オプションがサポートされます。一方は Oracle Application Server Containers for J2EE（OC4J）に組み込まれた J2EE ベースの Web サービス環境で、他方は Oracle Application Server SOAP と呼ばれる Apache SOAP ベースの Web サービス環境です。

この章では、OC4J（J2EE）の Web サービス環境について説明します。この環境では、J2EE 構造を使用したサービスの開発とデプロイを容易にし、Oracle Application Server Web Services の機能を次のリリースの J2EE（J2EE 1.4）に含まれる新たな Web サービス標準に移行します。Oracle Application Server Web Services 環境には、様々な開発およびデプロイ機能が組み込まれており、Oracle Application Server の拡張機能と統合されています。

Apache SOAP（Oracle Application Server SOAP）に対する Oracle Application Server サポートについては、付録 A「Oracle Application Server SOAP の使用」を参照してください。Apache SOAP の実装は最も初期の SOAP 実装の 1 つであり、既存の Web サービス・アプリケーションをサポートしているため、Oracle Application Server には Apache SOAP のサポート機能が組み込まれています。

---

**注意：** Web サービスの開発には、Oracle Application Server OC4J（J2EE）の Web サービス環境を使用することをお薦めします。Apache SOAP（Oracle Application Server SOAP）の実装は、現在はメンテナンス・モードになっています。

---

## Oracle Application Server Web Services の標準

Oracle Application Server Web Services でサポートされる Web サービスの標準は、次のとおりです。

- SOAP 1.1。次の項目を含みます。
  - RPC/ エンコード
  - ドキュメント / リテラル
- WSDL 1.1
- UDDI 2.0

**関連項目：** 1-4 ページの「Web サービス標準の概要」

# Oracle Application Server Web Services の機能

Oracle Application Server には、Web サービスを開発してデプロイできるように、拡張ランタイム機能と包括的なサポートが用意されています。Oracle Application Server Infrastructure には、次のサポートが組み込まれています。

- エンドツーエンドの Web サービスの開発
- Web サービスのデプロイと管理
- Oracle JDeveloper と Web サービスの使用
- Web サービスの保護
- Web サービスの集約

## エンドツーエンドの Web サービスの開発

Oracle Application Server Web Services には、Web サービス開発用に次の包括的なサポートが用意されています。

- 開発環境 - Oracle Application Server Web Services では、アプリケーション開発者は J2EE コンポーネントを使用して Web サービスを実装できます。また、Java クラスまたは PL/SQL ストアド・プロシージャを使用することもできます。Web サービスは、J2EE アプリケーションのランタイムおよびライフ・サイクル管理要素をすべて継承します。
- 開発ツールとウィザード - Oracle Application Server Web Services 開発者は、他の Oracle Application Server Containers for J2EE（OC4J）アプリケーションと同じコマンドライン・ユーティリティ・セットを使用して、Web サービスを作成、パッケージングおよびデプロイできます。
- WSDL を自動的に生成 - Oracle Application Server Web Services では、WSDL とクライアント・サイド・プロキシ・スタブを生成できます。この処理は、Web サービス・アセンブリ・ツールを使用して Web サービスをアセンブルするときに発生します。また、デプロイ済の Web サービスの場合は、WSDL またはクライアント・サイド・プロキシ・スタブが初めてリクエストされた時点で発生します（2 度目以降のリクエスト時には、以前に生成された WSDL またはクライアント・サイド・プロキシ・スタブが送信されます）。
- 登録、公開および検出 - Oracle Application Server Web Services には、業界標準に準拠した UDDI レジストリが用意されており、Web サービスを公開および検出できます。Oracle UDDI レジストリでは、プライベートとパブリックの UDDI レジストリが両方ともサポートされ、他の UDDI ノードと情報を同期化できます。
- 開発の容易さ - Oracle Application Server Web Services を使用すると、開発者は新たな概念セットを習得する必要がありません。Web サービスの開発、デプロイおよび管理には、J2EE アプリケーションの場合と同じプログラミングの概念およびツールを使用します。
- ビジネス・ロジックの再利用 - アプリケーション開発者は、J2EE アプリケーションを、変更を加えることなく新規の Web サービス・クライアントに透過的に公開できます。J2EE で開発された既存のビジネス・ロジックには、既存の J2EE/EJB クライアントまたは新規の Web サービス・クライアントから透過的にアクセスできます。
- 共通のランタイム・サービス - Oracle Application Server には、J2EE アプリケーションと Web サービスのための共通のランタイムおよびブローカ環境があります。そのため、Web サービスはトランザクション管理、メッセージ機能、ネーミング、ロギングおよびセキュリティなど、J2EE コンテナで使用可能な各種サービスを透過的に継承します。

## Web サービスのデプロイと管理

Oracle Enterprise Manager と Web サービス・アセンブリ・ツールを使用すると、Oracle Application Server Web Services をデプロイおよび管理できます。この 2 つのツールには、次の Web サービス・サポート機能が用意されています。

- パッケージングとアセンブリ - Web サービス・アセンブリ・ツールを使用すると、Web サービスをアセンブルし、J2EE の .ear ファイルを生成できます。
- デプロイ - Oracle Enterprise Manager には、Web サービスを Oracle Application Server にデプロイするための包括的な機能セットが用意されています。Oracle Enterprise Manager には、Web サービスを Oracle Application Server にデプロイするための、一元的で一貫性のあるアプリケーションのデプロイ・ウィザードがあります。このウィザードでは J2EE の .ear ファイルを指定でき、一連の手順を通じてデプロイ対象アプリケーションの情報を取得し、そのアプリケーションをデプロイできます。
- Web サービスの登録 - アプリケーションのデプロイ・ウィザードを使用できるのは、Web サービスをデプロイする場合のみです。この手順により、UDDI レジストリに Web サービスを登録する機能にアクセスできます。
- UDDI レジストリの参照 - Oracle の UDDI レジストリには、UDDI 標準に準拠した事前定義の階層カテゴライズ・スキームが用意されています。Oracle Enterprise Manager では、これらのカテゴリをドリルダウンして、必要なカテゴリに登録されている特定の Web サービスを参照できます。
- モニターと管理 - Web サービスのデプロイ後の Oracle Enterprise Manager は、Web サービスをアンインストールする機能と、Web サービスのパフォーマンス（レスポンス時間とスループット）とステータス（アップタイム、CPU およびメモリー使用量）をモニターする機能を提供します。また、特定の Oracle Application Server インスタンスにデプロイされている Web サービスをすべて識別してリストする機能も用意されています。

## Oracle JDeveloper と Web サービスの使用

Oracle JDeveloper IDE では、Oracle Application Server Web Services がサポートされます。Oracle JDeveloper は、業界最先端の Java および XML IDE であり、高い生産性、およびエンドツーエンドの J2EE および統合された Web サービス標準への準拠を提供します。

Oracle JDeveloper では、次の機能によって Oracle Application Server Web Services がサポートされます。

- 開発者は、Web サービスの WSDL 記述から Java スタブを作成し、既存の Web サービスをプログラムで使用できます。
- 開発者は、Java または EJB クラスから新規 Web サービスを作成できます。必要なデプロイメント・ディスクリプタ、web.xml および WSDL ファイルは自動的に生成されます。
- WSDL ファイルのスキーマ・ドリブン編集機能を提供します。
- Web サービス J2EE の .ear ファイルに対する有効な J2EE デプロイのサポート機能があり、OC4J に自動的にデプロイできます。

## Web サービスの保護

Oracle Enterprise Manager では、OC4J で動作する J2EE サーブレットが保護されるのと同じ方法で、Oracle Application Server Web Services が保護されます。これにより、次のように包括的なセキュリティ機能セットが提供されます。

- Web サービスの暗号化、認証および認可のための完全で業界標準に基づくセキュリティ・アーキテクチャ
- ユーザーが単一のパスワードで複数の Web サービスにアクセスできるシングル・サインオン
- Web サービスのセキュリティを一元管理するための単一の管理ポイント

## Web サービスの集約

OracleAS Portal には、組織内の Oracle Application Server Web Services を Portal に集約する機能が用意されています。また、OracleAS Portal フレームワークのポートレットを Web サービスとして公開できます。

# Oracle Application Server Web Services のアーキテクチャ

Oracle Application Server Containers for J2EE（OC4J）は、アプリケーションをコンポーネントとして作成するための基盤を提供し、Oracle Application Server Web Services をサポートします。Oracle Application Server Web Services では、RPC スタイルおよびドキュメント・スタイルの両方の Web サービスがサポートされます。

Oracle Application Server Web Services でサポートされる RPC Web サービスは、次のとおりです。

- Java クラス
- ステートレス Session Enterprise JavaBeans（EJB）
- ステートレス PL/SQL ストアド・プロシージャまたはファンクション

Oracle Application Server Web Services でサポートされるドキュメント・スタイルの Web サービスは、次のとおりです。

- Java クラス・ドキュメント・スタイルの Web サービス
- JMS ドキュメント・スタイルの Web サービス

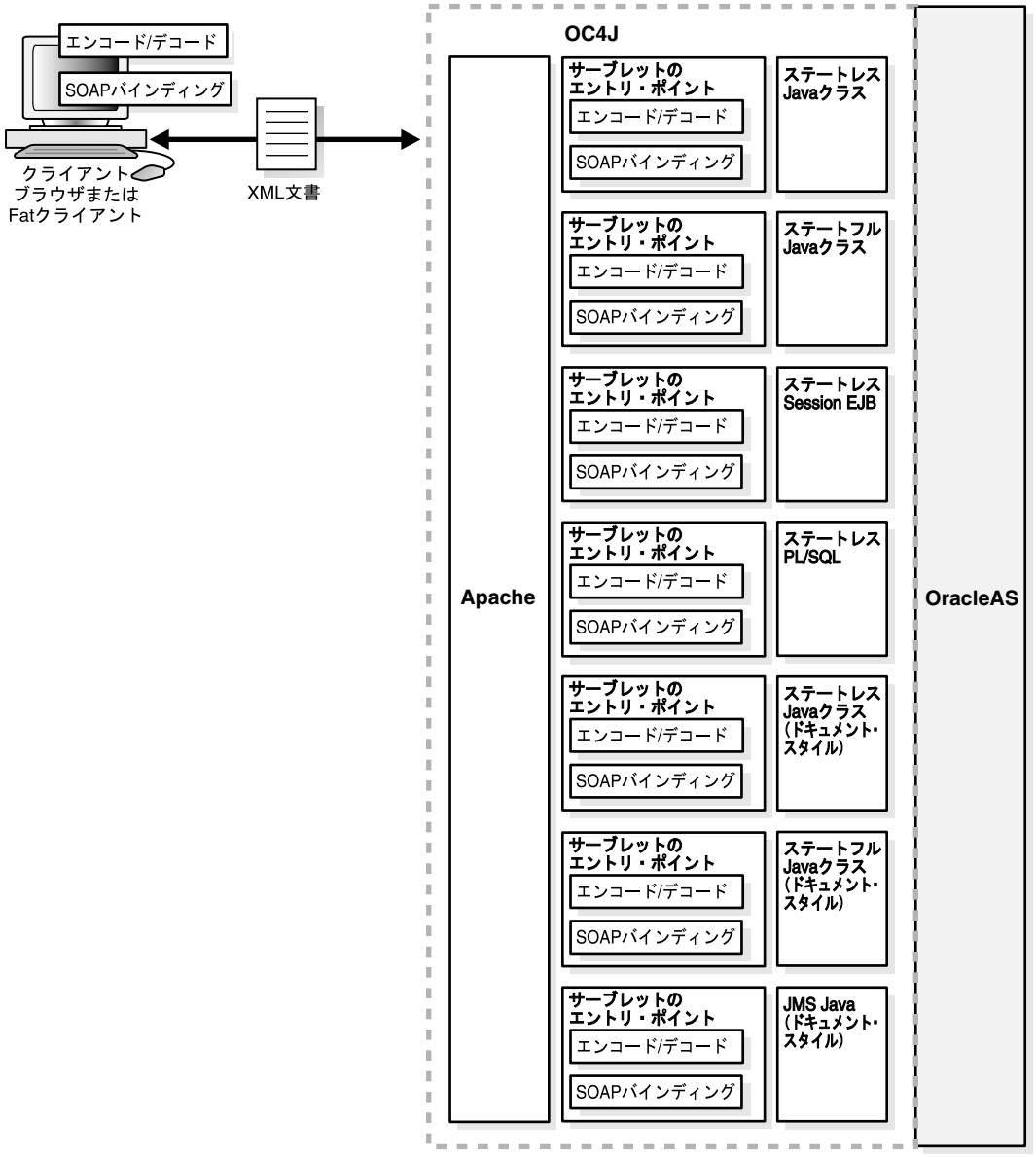
Oracle Application Server Web Services では、実装タイプごとに J2EE 標準に準拠する様々なサーブレットを使用して、Web サービス実装へのエントリ・ポイントが提供されます。図 2-1 に、サーブレットのエントリ・ポイントを含む Web サービスのランタイム・アーキテクチャを示します。

Oracle Application Server Web Services のランタイム・アーキテクチャについて説明する内容は、次のとおりです。

- Web サービス用のサーブレットのエントリ・ポイント
- Web サービス用のパッケージング・オプションとデプロイ・オプション
- Web サービス用のサーバー・スケルトン・コードの生成

> **関連項目：** RPC スタイルおよびドキュメント・スタイルの Web サービスの詳細は、1-5 ページの「SOAP 標準」を参照してください。

**図 2-1 Web サービスのランタイム・アーキテクチャ（RPC スタイルおよびドキュメント・スタイルとサーブレットのエントリ・ポイント）**



## Web サービス用のサーブレットのエントリ・ポイント

Oracle Application Server Web Services を使用するには、J2EE の .ear ファイルを Oracle Application Server にデプロイする必要があります。J2EE の .ear ファイルには、Web サービス・サーブレット構成と、Web サービスの実装が含まれています。Oracle Application Server Web Services には、サポートされている実装タイプごとに 1 つずつサーブレット・クラスがあります。実行時には、Oracle Application Server はサーブレット・クラスを使用してユーザー指定の Web サービス実装にアクセスします。

Oracle Application Server Web Services のサーブレット・クラスでは、次の Web サービス実装タイプがサポートされます。

- Java クラス（ステートレス）- Web サービスを実装するオブジェクトは、任意の Java クラスです。Web サービスはステートレスです。
- Java クラス（ステートフル）- Web サービスを実装するオブジェクトは、任意の Java クラスです。Web サービスはステートフルであるとみなされます。サーブレット `HttpSession` では、同じクライアントからのリクエスト間でオブジェクトの状態が保持されます。
- ステートレス Session EJB - ステートレス Session EJB を Web サービスとして公開できます。Web サービスはステートレスであるとみなされます。
- PL/SQL ストアド・プロシージャまたはファンクション - Web サービスを実装するオブジェクトは、PL/SQL ストアド・プロシージャまたはファンクションにアクセスする Java クラスです。Web サービスはステートレスであるとみなされます。Oracle JPublisher ツールにより、PL/SQL ストアド・プロシージャまたはファンクション用の Java アクセス・クラスが生成されます。
- Java クラスのドキュメント・スタイルの Web サービス（ステートレス）- Web サービスを実装するオブジェクトは、サポートされるメソッド・シグネチャを使用する Java クラスです。Web サービスはステートレスです。
- Java クラスのドキュメント・スタイルの Web サービス（ステートフル）- Web サービスを実装するオブジェクトは、サポートされるメソッド・シグネチャを使用する Java クラスです。Web サービスはステートフルであるとみなされます。サーブレット `HttpSession` では、同じクライアントからのリクエスト間でオブジェクトの状態が保持されます。
- Java JMS Web サービス - JMS 宛先との間でメッセージの送受信をサポートします。JMS Web サービスを使用すると、メッセージを処理または生成する MDB を組み込むことができます。

デプロイされた Web サービスは、サーブレット・クラスの一意のインスタンスにより管理されます。サーブレット・クラスは、Oracle Application Server Web Services のランタイム・サポートの一部として実装されます。Web サービスをアクセス可能にするには、Web サービスの実装を、対応する Web サービス・サーブレットとともにデプロイします。

---

**注意：** Apache SOAP 2.3.1 に準拠する Oracle Application Server SOAP を使用する場合、システム全体のすべての Web サービスに対し、単一のサーブレット・エントリ・ポイントのインスタンスは 1 つのみです。ただし、Oracle Application Server Web Services の場合はアーキテクチャが異なり、各 Web サービスが一意のサーブレット・インスタンスによりサポートされます。

---

パラメータとして値を取るか、クライアントに値を戻す Oracle Application Server Web Services での RPC スタイルの Web サービス実装の場合は、渡される型を制限する必要があります。この制限により、渡される型を XML と Java オブジェクト間（および Java オブジェクトと XML 間）で変換できます。表 2-1 に、Oracle Application Server Web Services との間の受渡しでサポートされる型を示します。

Oracle Application Server Web Services でのドキュメント・スタイルの Web サービス実装の場合は、Web サービスを実装する Java メソッドのシグネチャが制限されます。これらの Web サービスの間で受け渡しや送信が可能なのは、`org.w3c.dom.Element` のみです。

**注意：** 前述の制限とは、Web サービスを実装するメソッドで、`org.w3c.dom.Element` タイプをパラメータとして他のタイプと混在させることができないことを意味します。

**表 2-1 Web サービスでサポートされるデータ型（RPC のパラメータと戻り値の場合）**

| プリミティブ型 | オブジェクト型 |
|---|---|
| `Boolean` | `java.lang.Boolean` |
| `byte` | `java.lang.Byte` |
| `double` | `java.lang.Double` |
| `float` | `java.lang.Float` |
| `int` | `java.lang.Integer` |
| `long` | `java.lang.Long` |
| `short` | `java.lang.Short` |
| `string` | `java.lang.String` |
| | `java.util.Date` |
| | `java.util.Map` |
| | `org.w3c.dom.Element` |
| | `org.w3c.dom.Document` |
| | `org.w3c.dom.DocumentFragment` |
| | JavaBeans（その JavaBean のプロパティの型がこの表に含まれている型、または他のサポートされる JavaBeans） |
| | この表に示す型の単一次元配列 |

## Web サービス用のパッケージング・オプションとデプロイ・オプション

Oracle Application Server Web Services はサーブレットとしてアクセスされるため、Web サービスをアセンブルする必要があります。`WebServicesAssembler` ツールにより、J2EE の .war ファイルのコンポーネントである `web.xml` ファイルが構成され、必要なリソースと実装およびサポート・クラスが組み込まれ、J2EE の .ear ファイルが Web サービス用に準備されます。

アセンブリ・ツールを使用して Web サービスを作成するために、Web サービス実装を含む Jar ファイル、.war ファイル、ebj.jar または .ear ファイルを使用することが可能です。これにより、アセンブリ・ツールでは XML 構成ファイルに指定されている構成情報を使用して、Web サービスが作成されます。

**関連項目：**

- 第 3 章「Java クラス Web サービスの開発とデプロイ」
- 第 4 章「EJB Web サービスの開発とデプロイ」
- 第 5 章「ストアド・プロシージャ Web サービスの開発とデプロイ」
- 第 6 章「ドキュメント・スタイルの Web サービスの開発とデプロイ」

## Web サービス用のサーバー・スケルトン・コードの生成

Oracle Application Server Web Services がサービス・リクエストを初めて受信すると、サーブレットのエントリ・ポイントにより自動的に次の操作が実行されます（この説明は、処理方法の異なる JMS Web サービスには該当しません）。

- クラス・ローディングが検証されます。Web サービスの実装に必要なすべてのクラスが、J2EE のクラス・ローディング標準に準拠している必要があります。
- データ型が検証されます。すべての Java クラスまたは EJB が、表 2-1 のように、サポートされるパラメータと戻り型の制限に準拠している必要があります。
- サーバー・スケルトン・コードが生成されます。サーバー・スケルトン・コードが生成されるのは、Web サービスへの初回アクセス時または .ear ファイルの再デプロイ時のみです（アプリケーションが再デプロイされると、サーバー・スケルトン・コードと他の Web サービスのサポート・ファイルが再生成されます）。生成されたコードは、サーブレット・コンテキストに関連する一時ディレクトリに格納されます。サーバー・スケルトン・コードにより、EJB のライフ・サイクルが制御され（ステートレス Session EJB 実装の場合）、パラメータと戻り型のマーシャリングが処理され（SOAP RPC ベースの Web サービスの場合）、サービスを実装する実際の Java クラスまたは EJB メソッドにディスパッチされます。

  サーバー・スケルトン・クラスの生成後は、サービスに対する後続のリクエストを受信すると、サーバー・スケルトンはマーシャリングを直接処理し、サービスを実装するメソッドを起動します（PL/SQL ストアド・プロシージャまたはファンクションを使用して実装された Web サービスの場合、サーバー・スケルトンは、PL/SQL ストアド・プロシージャまたはファンクションを含むデータベースにアクセスする Java クラスを起動します）。

  ドキュメント・スタイルの Web サービスの場合は、サーバー・スケルトンからサービスを実装するメソッドに DOM 要素が渡されます。

# Web サービス用の WSDL とクライアント・プロキシ・スタブ

Oracle Application Server Web Services には、WSDL ファイルを生成するツールが用意されています。WSDL ファイルは、アセンブリ時に Web サービスとともにパッケージ化できます（WSDL ファイルをパッケージ化しない場合は、実行時に生成できます）。また、このツールでは、WSDL ファイルの指定によるクライアント・サイド・プロキシ・スタブの生成もサポートされます。

Oracle Application Server Web Services の WSDL サポートには、複数の要素があります。まず、RPC スタイルの Web サービスは相互運用可能な XML データ表現に基づいており、通常、任意の Java オブジェクトを XML にマップしません。Oracle Application Server Web Services では、Java 型のセットに対応する XML 型のセットがサポートされます（サポートされる Java 型のリストは表 2-1 を参照してください）。

次に、Oracle Application Server Web Services を使用すると、アプリケーション開発者は Web サービス用の WSDL インタフェースを静的に生成できます。また、Oracle Application Server Web Services ランタイムは、WSDL とクライアント・サイド・プロキシ・スタブが Web サービスのデプロイ時に提供されない場合、これらを生成することも可能です。これらのファイルは、サーバー・サイドでランタイムが生成し、Web サービス・クライアントからリクエストされた時点で配信できます。

Oracle Application Server には、Java クラスまたは J2EE アプリケーションについて、WSDL を静的に生成するためのクライアント・サイド・ツールも用意されています。同様に、Web サービス・アセンブリ・ツールでは、生成された WSDL ファイルまたは既知の WSDL エンド・ポイントについて、クライアント・サイド・プロキシを生成できます。

**関連項目：**

- 8-6 ページの「WebServicesAssembler を使用したクライアント・サイド・プロキシの生成」
- 9-4 ページの「WSDL ファイルとクライアント・サイド・プロキシの生成」

## WSDL ベースの Web サービス・クライアントの概要

Web サービスを使用して、クライアント・アプリケーションは Web サービスを起動する SOAP リクエストを送信し、サービスからの SOAP レスポンスを処理します。クライアント・アプリケーションの開発を容易にするために、Oracle Application Server Web Services ランタイムでは Web サービスを記述する WSDL を生成できます。WSDL を使用すると、開発者は開発ツールを使用して、Web サービスを起動するアプリケーションを作成できます。

**関連項目：**

- 2-4 ページの「Oracle JDeveloper と Web サービスの使用」
- 第 8 章「Web サービスを使用するクライアントの作成」

## クライアント・サイド・プロキシ・スタブ・ベースの Web サービス・クライアントの概要

Web サービスを使用して、クライアント・アプリケーションは Web サービスを起動する SOAP リクエストを送信し、サービスからの SOAP レスポンスを処理します。クライアント・サイド・アプリケーションの開発を容易にするために、Oracle Application Server Web Services ではクライアント・サイド・プロキシ・スタブを生成できます。クライアント・サイド・プロキシ・スタブにより、SOAP リクエストの構成と SOAP レスポンスの分解の詳細が隠されます。生成されたクライアント・サイド・プロキシ・スタブでは、リクエストとレスポンスの同期起動モデルがサポートされます。また、生成されたスタブにより、Java クライアント・アプリケーションを簡単に記述して Web サービス（SOAP）・リクエストを発行し、レスポンスを処理できます。

**関連項目：** 第 8 章「Web サービスを使用するクライアントの作成」

# Web サービス・ホーム・ページ

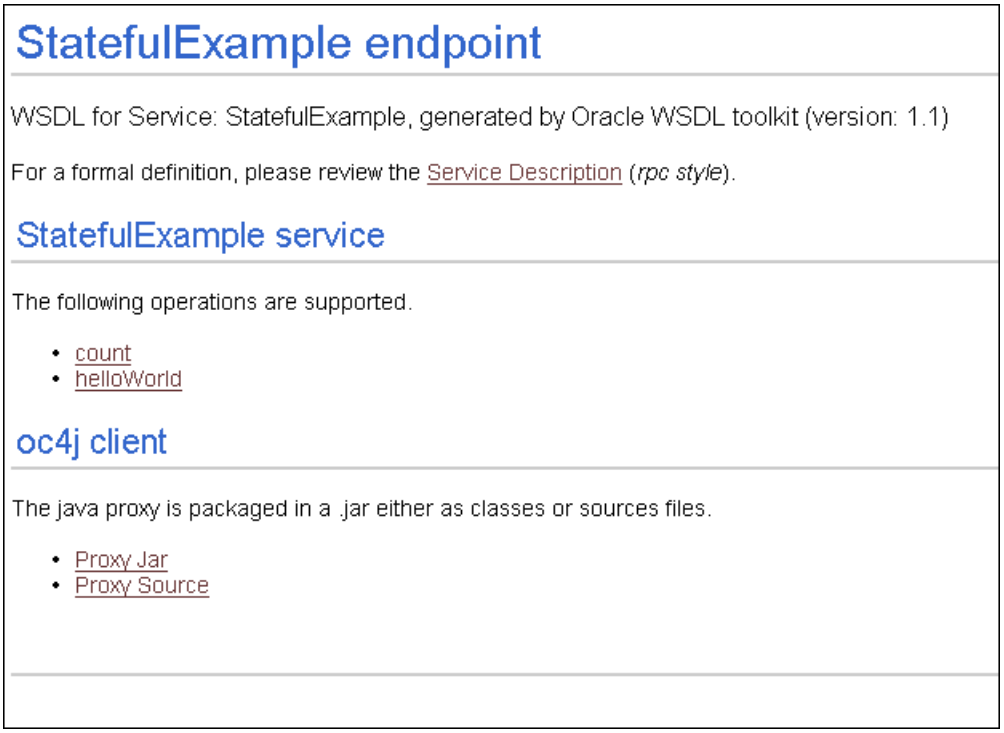
Oracle Application Server Web Services では、デプロイされた Web サービスごとに Web サービス・ホーム・ページが提供されます。

Web サービス・ホーム・ページで提供される要素は、次のとおりです。

- WSDL ファイルへのリンク - Web サービス用の WSDL ファイルを取得するには、「Service Description」リンクを選択してファイルを保存します。
- サポートされる操作ごとの Web サービス・テスト・ページへのリンク - 使用可能な Web サービスの操作をテストするには、その操作のパラメータ値があれば入力し、「Invoke」ボタンを選択します。
- Web サービスのクライアント・サイド・プロキシ Jar およびクライアント・サイド・プロキシ・ソースへのリンク - クライアント・サイド・プロキシ Jar またはクライアント・サイド・プロキシ・ソースを取得するには、「Proxy Jar」または「Proxy Source」のうち適切なリンクを選択し、ファイルを保存します。

図 2-2 に、Web サービス・ホーム・ページのサンプルを示します。

**図 2-2 Web サービス・ホーム・ページ**




# Universal Description, Discovery, and Integration（UDDI）レジストリ

Universal Description, Discovery, and Integration（UDDI）仕様は、4 層の階層形式の XML Schema で構成されており、Web サービスに関する情報を公開、検証および起動するための基本的な情報モデルを提供します。UDDI XML Schema では、次の 4 種類の情報が定義されます。

- ビジネス・エンティティ - UDDI エントリのトップレベルの XML 要素は、名称、産業または製品カテゴリ、地理的所在地およびオプションのカテゴリと連絡先の情報など、企業のサービス情報を検索しているパートナに必要な情報の初期セットを獲得します。これには、産業別、製品別または地域別にビジネスを検索するためのイエロー・ページ分類のサポートが含まれます。
- ビジネス・サービス - businessService 構造には、関連する一連の Web サービスがグループ化されているため、ビジネス・プロセスやサービス・カテゴリへの関連付けができます。ビジネス・プロセスの一例には、出荷、経路指定、入出庫および最終配送サービスなど、複数の Web サービスを含むロジスティクス / 搬送プロセスがあります。Web サービスをカテゴリやビジネス・プロセスに関連するグループ単位で編成することで、UDDI では Web サービスを効率的に検索して検出できます。
- バインディング情報 - 各 businessService には、バインディング・テンプレートと呼ばれる XML 要素内で獲得された 1 つ以上の技術的な Web サービス記述があります。バインディング・テンプレートには、特定の Web サービスを起動またはバインディングするために必要なアプリケーション・プログラムに関連する情報が含まれています。この情報には、Web サービスの URL アドレスや、ホスティングされるサービス、ルーティングおよびロード・バランシング機能を記述する他の情報が含まれます。
- 準拠情報 - bindingTemplate にはサービスの起動に必要な情報が含まれていますが、特定の Web サービスへの接続位置を知るのみでは不十分な場合があります。たとえば、ビジネス・パートナの Web サービスに発注書を送信するには、起動側サービスはサービスの位置や URL のみでなく、発注書の送信フォーマット、適切なプロトコル、必要なセキュリティ、発注書の送信後に戻されるレスポンスの書式を知る必要があります。Web サービス

を起動する前に、起動する特定サービスが特定の動作やプログラミング・インタフェースに準拠しているかどうかを判断すると役立ちます。したがって、各 bindingTemplate 要素には、tModel と呼ばれる要素が含まれています。クライアントは tModel に含まれている情報を使用して、特定の Web サービスが準拠実装かどうかを判断できます。

## Oracle Enterprise Manager の Web サービス登録機能

Web サービスを Oracle Application Server にデプロイすると、Oracle Enterprise Manager を使用して特定の Web サービスを登録し、その WSDL を UDDI レジストリに公開し、公開されている Web サービスを検出できます。

**関連項目：** 第 10 章「Web サービスの検出と公開」

# 3

# Java クラス Web サービスの開発とデプロイ

この章では、Java クラスとして実装される Oracle Application Server Web Services を記述する手順とデプロイする手順について説明します。

この章の内容は、次のとおりです。

# Oracle Application Server Web Services と Java クラスの使用

この章では、Java クラスを使用して実装される Web サービスを記述するためのサンプル・コードを例として、ステートフル Java とステートレス Java による Web サービスの記述方法の違いについて説明します。

Oracle Application Server には、Web サービスを実装する Java クラスにアクセスするサーブレットが用意されています。このサーブレットでは、Web サービス・クライアントにより生成されたリクエストが処理され、Web サービスを実装する Java メソッドが実行され、結果が Web サービス・クライアントに戻されます。

**関連項目：**

- 第 2 章「Oracle Application Server Web サービス」
- 第 4 章「EJB Web サービスの開発とデプロイ」
- 第 5 章「ストアド・プロシージャ Web サービスの開発とデプロイ」
- 第 8 章「Web サービスを使用するクライアントの作成」

# Java クラス・ベースの Web サービスの記述

Java クラス・ベースの Web サービスを記述するには、1 つ以上のメソッドを含む Java クラスを作成する必要があります。Web サービス・クライアントがサービス・リクエストを発行すると、Oracle Application Server Web Services により Web サービス・サーブレットが起動され、このサーブレットによってサービス・リクエストを実装するメソッドが実行されます。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。

この項では、文字列「Hello World」を戻すステートフル Java Web サービスとステートレス Java Web サービスの記述方法について説明します。ステートフル・サービスは、サービスへのメソッド・コール数の実行中のカウントを示す整数も戻します。この Java Web サービスはクライアント・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。

サンプル・コードは、次の OTN の Web サイトで提供されています。

```
http://otn.oracle.com/tech/java/oc4j/demos/1012/index.html
```

Web サービスの `demo.zip` ファイルを解凍すると、Java クラス・ベースの Web サービスが、UNIX の場合は `/webservices/demo/basic/java_services` の下のディレクトリ、Windows の場合は `¥webservices¥demo¥basic¥java_services` の下のディレクトリに格納されます。

# ステートレス Java Web サービスとステートフル Java Web サービスの記述

Oracle Application Server Web Services では、Web サービスとして動作する Java クラスのステートフル実装とステートレス実装が次のようにサポートされます。

- ステートフル Java 実装の場合、Oracle Application Server Web Services では、単一の Java インスタンスで個々のクライアントからの Web サービス・リクエストが処理されます。
- ステートレス Java 実装の場合、Oracle Application Server 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。

---

**注意：** Web サービス開発者は、ステートフル Web サービスとステートレス Web サービスのどちらを実装するかを、設計段階で決定する必要があります。この 2 種類の Web サービスは、パッケージング時の処理方法が多少異なります。この違いについては、3-7 ページの「Java クラス・ベースの Web サービスの準備とデプロイ」を参照してください。

---

# サンプル Java クラス実装の作成

Java Web サービスの開発手順は、次のとおりです。

- Web サービス用のメソッドを含む Java クラスの定義
- 明示的なメソッド公開用のインタフェースの定義
- WSDL ファイルの記述（オプション）

## Web サービス用のメソッドを含む Java クラスの定義

Web サービスとしてデプロイするメソッドを含む Java クラスを記述または提供して、Java Web サービスを作成します。`java_services` サンプル・ディレクトリに用意されているサンプルでは、.ear ファイル `ws_example.ear` に Web サービスのソース、クラスおよび構成ファイルが含まれています。この .ear ファイルを展開すると、クラス `StatefulExampleImpl` によりステートフル Java サービスが提供され、`StatelessExampleImpl` によりステートレス Java サービスが提供されます。

Java Web サービスを記述する場合に、Java サービスをパッケージングする必要があるときは、Java `package` 仕様に従ってパッケージ名を指定します。`StatefulExampleImpl.java` の 1 行目で、パッケージ名を次のように指定します。

```
package oracle.j2ee.ws_example;
```

ステートレスのサンプル Web サービスは、パブリック・クラス `StatelessExampleImpl` で実装されます。このクラスでは、パブリック・メソッド `helloWorld()` が定義されています。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。

例 3-1 に、`StatelessExampleImpl` を示します。

ステートフルのサンプル Web サービスは、パブリック・クラス `StatefulExampleImpl` で実装されます。このクラスによりカウントが初期化され、2 つのパブリック・メソッド `count()` および `helloWorld()` が定義されます。

例 3-2 に、`StatefulExampleImpl` を示します。

**例 3-1 ステートレス Web サービス用の Java メソッドを含むパブリック・クラスの定義**

```
package oracle.j2ee.ws_example;

public class StatelessExampleImpl {
    public StatelessExampleImpl() {
  }
  public String helloWorld(String param) {
    return "Hello World, " + param;
```

```
  }
}
```

**例 3-2 ステートフル Web サービス用の Java メソッドを含むパブリック・クラスの定義**

```
package oracle.j2ee.ws_example;

public class StatefulExampleImpl {
  int count = 0;
  public StatefulExampleImpl() {
  }
  public int count() {
    return count++;
  }
  public String helloWorld(String param) {
    return "Hello World, " + param;
  }
}
```

Web サービス用の Java クラス実装には、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ります。例 3-1 にパブリック・コンストラクタ `StatelessExampleImpl()`、例 3-2 に `StatefulExampleImpl()` を示します。

Java クラスとして実装された Web サービスの実行中にエラーが発生した場合、その Java クラスは例外をスローします。例外がスローされると、Web サービス・サーブレットは 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 サービスについてサーブレット・エラーのログを表示するには、標準の J2EE および OC4J 管理機能を使用します。

Web サービスの実装メソッドを含む Java クラスを作成する場合、メソッドのパラメータと戻り値にはサポート対象の型を使用するか、インタフェース・クラスを使用して、公開するメソッドをサポート対象の型のみを使用するメソッドに限定する必要があります。表 3-1 に、Web サービスを実装する Java メソッドのパラメータと戻り値について、サポートされている型を示します。

---

**注意：** パラメータと戻り値についてサポートされている型のリストは、表 3-1 を参照してください。

---

SOAP のリクエスト・ヘッダー・エントリを処理する必要がある場合は、Java Web サービスを実装するために追加の手順を実行する必要があります。

**関連項目：** 12-4 ページの「SOAP ヘッダーのサポート」

## 明示的なメソッド公開用のインタフェースの定義

Oracle Application Server Web Services では、パブリック・インタフェースを提供し、Web サービスとして公開するメソッドを制限できます。Web サービスで公開するメソッドを制限するには、公開するメソッドのシグネチャをリストするパブリック・インタフェースを組み込みます。例 3-3 にクラス `StatelessExampleImpl` 内のメソッドへのインタフェース、例 3-4 にクラス `StatefulExampleImpl` 内のメソッドへのインタフェースを示します。

**例 3-3 パブリック・インタフェースを使用したステートレス Web サービスのメソッドの公開**

```
package oracle.j2ee.ws_example;

public interface StatelessExample {
  String helloWorld(String param);
}
```

**例 3-4 パブリック・インタフェースを使用したステートフル Web サービスのメソッドの公開**

```
package oracle.j2ee.ws_example;

public interface StatefulExample {
   int count();
   String helloWorld(String param);
}
```

Web サービスにインタフェース・クラスが含まれていない場合、Web サービスのデプロイでは Java クラスに定義されているパブリック・メソッドがすべて公開されます。例 3-3 の `StatelessExample` や例 3-4 の `StatefulExample` のようなインタフェースを使用すると、インタフェースにリストされているメソッドのみが公開されます。

---

**注意：** インタフェースを使用すると、Java クラスが準備され、Web サービスとしてデプロイされた時点で、指定したメソッド・シグネチャを持つメソッドのみが公開されます。

---

Web サービス・インタフェースの用途は、次のとおりです。

1. メソッドの公開を、あるクラスのパブリック・メソッドのサブセットに限定するため。
2. Web サービスとして公開されるメソッドのセットを拡張し、あるクラスのスーパークラスのメソッドを含めるため。
3. メソッドの公開を、あるクラスのパブリック・メソッドのサブセットに限定するため。この場合、サブセットには、パラメータや戻り値にサポート対象の型を使用するメソッドのみが含まれます。表 3-1 に、Web サービスを実装する Java メソッドのパラメータと戻り値についてサポートされている型を示します。

**関連項目：** 3-6 ページの「Java Web サービスについてサポートされるデータ型の使用」

## WSDL ファイルの記述（オプション）

`WebServicesAssembler` では、Web サービス開発者が WSDL ファイルとクライアント・サイド・プロキシ・ファイルを生成できるように、`<wsdl-gen>` および `<proxy-gen>` タグがサポートされます。これらのタグを使用して、WSDL ファイルとクライアント・サイド・プロキシを生成するかどうかを制御できます。また、これらのタグを使用すると、生成された WSDL ファイルまたは記述する WSDL ファイルを、Web サービスの J2EE の .ear とともにパッケージするように指定できます。

クライアント・サイド開発者が Web サービスを使用するアプリケーションを作成するには、デプロイされた Web サービスから取得した WSDL ファイルを使用する方法と、WSDL から生成されるクライアント・サイド・プロキシを使用する方法があります。

**関連項目：** 9-4 ページの「WSDL ファイルとクライアント・サイド・プロキシの生成」

## Java Web サービスについてサポートされるデータ型の使用

表 3-1 に、Oracle Application Server Web Services のパラメータと戻り値についてサポートされているデータ型を示します。

**表 3-1 Web サービスでサポートされるデータ型**

| プリミティブ型 | オブジェクト型 |
|---|---|
| Boolean | java.lang.Boolean |
| byte | java.lang.Byte |
| double | java.lang.Double |
| float | java.lang.Float |
| int | java.lang.Integer |
| long | java.lang.Long |
| short | java.lang.Short |
| string | java.lang.String |
| | java.util.Date |
| | java.util.Map |
| | org.w3c.dom.Element |
| | org.w3c.dom.Document |
| | org.w3c.dom.DocumentFragment |
| | JavaBeans（その JavaBean のプロパティの型がこの表に含まれている型、または他のサポートされる JavaBeans） |
| | この表に示す型の単一次元配列 |

Oracle Application Server Web Services でのドキュメント・スタイルの Web サービス実装の場合は、Web サービスを実装する Java メソッドのシグネチャが制限されます。これらの Web サービスの間で受け渡しや送信が可能なのは、org.w3c.dom.Element のみです。

**注意：** 前述の制限とは、Web サービスを実装するメソッドで、org.w3c.dom.Element タイプをパラメータとして他のタイプと混在させることができないことを意味します。

**注意：** Oracle Application Server Web Services では、Element[]（org.w3c.dom.Element の配列）はサポートされません。

Web サービスに使用する Bean は、次の制限に準拠する Java クラスです。

- 引数を取らないコンストラクタが必須です。
- 必要なすべての状態をプロパティを通じて公開する必要があります。
- setX または getX メソッドなどのプロパティのアクセッサは、呼び出される順番に依存しないようにする必要があります。

Oracle Application Server Web Services では、Bean が表 3-1 に示すプロパティ型またはサポートされている他の JavaBean のみで構成されているかぎり、Bean を J2EE Web サービスのメソッドに戻したり、引数として渡すことができます。

JavaBeans を Oracle Application Server Web Services へのパラメータとして使用する場合、クライアント・サイド・コードでは、ダウンロードしたクライアント・サイド・プロキシに含まれている、生成された Bean を使用する必要があります。これは、生成されたクライアント・サイド・プロキシ・コードでは、SOAP 構造の名前空間と完全修飾 Bean クラス名の間で変換が行われることで、SOAP 構造と JavaBean の間での変換が行われるためです。指定した名前の Bean が指定したパッケージに存在しない場合、生成されたクライアント・コードは失敗します。

ただし、クライアントで Web Services Description Language（WSDL）を使用して、クライアント・サイド・プロキシではなく Oracle Application Server Web Services のコールを形成する場合、特別な要件はありません。生成された WSDL ドキュメントでは、SOAP 構造が標準的な方法で記述されます。WSDL ドキュメントから直接動作する Oracle JDeveloper などのアプリケーション開発環境では、JavaBeans をパラメータとして使用し、Oracle Application Server Web Services を適切にコールできます。

---

**注意：** Web サービスのプロキシ・クラスと WSDL が生成されると、サーバー・サイド・サービス実装に含まれるすべての Java プリミティブ型は、プロキシ・コードまたは WSDL 内のオブジェクト型にマップされます。たとえば、Web サービス実装に Java プリミティブ型 `int` のパラメータが含まれている場合、プロキシ内の対応するパラメータは `java.lang.Integer` 型です。このマッピングは、どのプリミティブ型の場合にも発生します。

---

**関連項目：** 第 8 章「Web サービスを使用するクライアントの作成」

# Java クラス・ベースの Web サービスの準備とデプロイ

Java クラスを Web サービスとしてデプロイするには、Oracle Application Server Web Services サーブレットのデプロイメント・ディスクリプタと Java 実装を提供する Java クラスを含む J2EE の .ear ファイルをアセンブルする必要があります。この項では、Oracle Application Server Web Services のツールである `WebServicesAssembler` の使用方法について説明します。`WebServicesAssembler` では、Java クラスの Web サービスを記述する XML 構成ファイルを使用して、Oracle Application Server Web Services でデプロイできる J2EE の .ear ファイルが生成されます。

この項の内容は、次のとおりです。

- Java クラスの Web サービスをアセンブルするための構成ファイルの作成
- Java クラスの Web サービスを準備するための WebServicesAssembler の実行

# Java クラスの Web サービスをアセンブルするための構成ファイルの作成

Oracle Application Server Web Services のアセンブリ・ツール `WebServicesAssembler` を使用すると、Oracle Application Server Web Services をアセンブルできます。この項では、Java クラスの Web サービスで使用する構成ファイルの作成方法について説明します。

次のタグを追加して、`WebServicesAssembler` 構成ファイルを作成します。

- Web サービスのトップレベル・タグの追加
- Java ステートレス・サービス・タグの追加
- Java ステートフル・サービス・タグの追加
- WSDL およびクライアント・サイド・プロキシ生成タグの追加

## Web サービスのトップレベル・タグの追加

表 3-2 に、`WebServicesAssembler` 構成ファイルのトップレベル・タグを示します。これらのタグを追加して、Java ステートレス Web サービスまたは Java ステートフル Web サービスのトップレベル記述情報を提供します。これらのタグは、構成ファイル内で `<web-service>` タグ内に挿入します。

トップレベル・タグなど、`config.xml` ファイルの詳細は、例 3-5 を参照してください。

**表 3-2 トップレベルの `WebServicesAssembler` 構成タグ**

| タグ | 説明 |
|---|---|
| `<context>`<br>*context*<br>`</context>` | Web サービスのコンテキスト・ルートを指定します。<br>このタグは必須です。 |
| `<datasource-JNDI-name>`<br>*datasource*<br>`</datasource-JNDI-name>` | Web サービスに関連するデータソースを指定します。 |
| `<description>`<br>*description*<br>`</description>` | Web サービスの簡単な説明を提供します。<br>このタグはオプションです。 |
| `<destination-path>`<br>*dest_path*<br>`</destination-path>` | 生成される J2EE の .ear ファイルの出力名を指定します。*dest_path* には、出力ファイルのフルパスを指定します。<br>このタグは必須です。 |
| `<display-name>`<br>*disp_name*<br>`</display-name>` | Web サービスの表示名を指定します。<br>このタグはオプションです。 |
| `<option name="source-path"`<br>`[contextroot="`*path1*`"] >`<br>*path2*<br>`<option>` | 指定したファイルを出力 .ear ファイルに含めます。このオプションを使用して Java リソースを指定するか、J2EE の出力 .ear ファイルのソース・ファイルとして使用される既存の .war、.ear または ejb-jar ファイルの名前を指定します。<br>.war ファイルが入力として提供されると、オプションの contextroot によって、.war ファイルの root-context が指定されます。<br>*path1* には、.war の context-root を指定します。<br>*path2* には、組み込むファイルへのパスを指定します。<br>次に例を示します。<br>`<option name="source-path" contextroot="/test">/myTestArea/ws/src/statefull.war</option>` |

**表 3-2 トップレベルの WebServicesAssembler 構成タグ（続き）**

| タグ | 説明 |
|---|---|
| <stateless-java-service><br>*sub-tags*<br></stateless-java-service> | このタグを使用して、ステートレス・サービスを定義する Java Web サービスを追加します。有効な *sub-tags* の詳細は、表 3-3 を参照してください。 |
| <stateful-java-service><br>*sub-tags*<br></stateful-java-service> | このタグを使用して、ステートフル・サービスを定義する Java Web サービスを追加します。有効な *sub-tags* の詳細は、表 3-3 を参照してください。 |
| <temporary-directory><br>*temp_dir*<br></temporary-directory> | アセンブラで一時ファイルを格納できるディレクトリを指定します。<br>このタグはオプションです。 |

## Java ステートレス・サービス・タグの追加

WebServicesAssembler の <stateless-java-service> タグを使用して、Java ステートレス Web サービスを準備します。このタグは、構成ファイル内で <web-service> タグ内に挿入します。このタグを追加して、ステートレス Java Web サービスの生成に必要な情報を提供します。

表 3-3 に、<stateless-java-service> のサブタグと <stateful-java-service> のサブタグを示します。表 3-3 のように、一部のサブタグは <stateful-java-service> の使用時にのみ適用されます。

<stateless-java-service> など、config.xml ファイルの詳細は、例 3-5 を参照してください。

---

**注意：** Web サービス開発者は、ステートフル Web サービスとステートレス Web サービスのどちらを実装するかを、設計段階で決定する必要があります。この 2 種類の Web サービスは、パッケージング時の処理方法が多少異なります。

---

## Java ステートフル・サービス・タグの追加

WebServicesAssembler の <stateful-java-service> タグを使用して、Java ステートフル Web サービスを準備します。このタグは、構成ファイル内で <web-service> タグ内に挿入します。このタグを追加して、ステートフル Java Web サービスの生成に必要な情報を提供します。

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。

表 3-3 に、<stateful-java-service> のサブタグを示します。

<stateful-java-service> など、config.xml ファイルの詳細は、例 3-5 を参照してください。

**表 3-3 ステートレス Java サービスとステートフル Java サービスのサブタグ**

| タグ | 説明 |
|---|---|
| <accept-untyped-request><br>*value*<br></accept-untyped-request> | *value* に true を設定すると、Web サービスで型未指定のリクエストが受入可能であることを WebServicesAssembler に通知します。value を false に設定すると、Web サービスは型未指定のリクエストを受け入れません。<br>有効な値 : true、false<br>（大 / 小文字は区別されません。TRUE および FALSE も有効です）<br>このタグはオプションです。<br>デフォルト値 : false |
| <class-name><br>*class*<br></class-name> | Web サービス実装を提供するクラスの完全修飾クラス名を指定します。<br>このタグは必須です。 |
| <interface-name><br>*interface*<br></interface-name> | Web サービスとして公開する必要のあるメソッドを、Web サービス・サーブレット生成コードに対して指定する、インタフェースの完全修飾名を指定します。<br>このタグはオプションです。 |
| <ejb-resource><br>*ejb-resource*<br></ejb-resource> | これは下位互換性タグです。<br>関連項目 : 表 3-2 のトップレベルの <option name="source-path"> タグ。<br>このタグはオプションです。 |
| <java-resource><br>*resource*<br></java-resource> | これは下位互換性タグです。<br>関連項目 : 表 3-2 のトップレベルの <option name="source-path"> タグ。<br>このタグはオプションです。 |
| <message-style><br>*rpc*<br></message-style> | メッセージ・スタイルを設定します。Java Web サービスを定義するときに <message-style> タグを組み込む場合は、値 rpc を指定する必要があります。<br>有効な値 : doc、rpc<br>このタグはオプションです。<br>デフォルト値 : rpc（<message-style> タグを指定しない場合） |
| <scope><br>*scope*<br></scope> | ステートフル・サービスのセッションの有効範囲を設定します。<br><scope> タグは、ステートフル・サービスにのみ適用されます。このタグは、<stateful-java-service> タグ内でのみ使用します。<br>このタグはオプションです。<br>有効な値 : application、session<br>デフォルト値 : session |
| <session-timeout><br>*value*<br></session-timeout> | ステートフル・セッションのセッション・タイムアウトを設定します。<br><session-timeout> タグは、ステートフル・サービスにのみ適用されます。このタグは、<stateful-java-service> タグ内でのみ使用します。<br>*value* には、セッションのタイムアウト秒数を定義する整数を指定します。セッション・タイムアウト指定のないステートフル Java セッションの場合、デフォルトのセッション・タイムアウト値は 60 秒です。<br>このタグはオプションです。 |
| <uri><br>*URI*<br></uri> | Web サービスを実装するサーブレットのサーブレット・マッピング・パターンを指定します。*URI* として指定したパスは、Web サービスの位置を指定する <context> に追加されます。<br>このタグは必須です。 |

**例 3-5 WebServicesAssembler のサンプル構成ファイル**

```
<web-service>
    <display-name>Web Services Example</display-name>
    <description>Java Web Service Example</description>
    <!-- Specifies the resulting web service archive will be stored in
         ./ws_example.ear -->
    <destination-path>./ws_example.ear</destination-path>
    <!-- Specifies the temporary directory that web service assembly
         tool can create temporary files. -->
    <temporary-directory>./tmp</temporary-directory>
    <!-- Specifies the web service will be accessed in the servlet context
         named "/webservices". -->
    <context>/webservices</context>

    <!-- Specifies the web service will be stateless -->
    <stateless-java-service>
        <interface-name>oracle.j2ee.ws_example.StatelessExample</interface-name>
        <class-name>oracle.j2ee.ws_example.StatelessExampleImpl</class-name>
        <!-- Specifies the web service will be accessed in the uri named
             "statelessTest" within the servlet context. -->
        <uri>/statelessTest</uri>
        <!-- Specifies the location of Java class files are under
             ./src -->
        <java-resource>./src</java-resource>
    </stateless-java-service>

    <stateful-java-service>
        <interface-name>oracle.j2ee.ws_example.StatefulExample</interface-name>
        <class-name>oracle.j2ee.ws_example.StatefulExampleImpl</class-name>
        <!-- Specifies the web service will be accessed in the uri named
             "statefullTest" within the servlet context. -->
        <uri>/statefulTest</uri>
        <!-- Specifies the location of Java class files are under
             ./src -->
        <java-resource>./src</java-resource>
    </stateful-java-service>
</web-service>
```

## WSDL およびクライアント・サイド・プロキシ生成タグの追加

WebServicesAssembler では、Web サービス開発者が WSDL ファイルとクライアント・サイド・プロキシ・ファイルを生成できるように、<wsdl-gen> および <proxy-gen> タグがサポートされます。これらのタグを使用して、WSDL ファイルとクライアント・サイド・プロキシを生成するかどうかを制御できます。また、これらのタグを使用すると、生成された WSDL ファイルまたは提供する WSDL ファイルを、Web サービスの J2EE の .ear とともにパッケージするように指定できます。

クライアント・サイド開発者が Web サービスを使用するアプリケーションを作成するには、デプロイされた Web サービスから取得した WSDL ファイルを使用する方法と、WSDL から生成されるクライアント・サイド・プロキシを使用する方法があります。

> **関連項目：** 9-4 ページの「WSDL ファイルとクライアント・サイド・プロキシの生成」

## Java クラスの Web サービスを準備するための WebServicesAssembler の実行

WebServicesAssembler 構成ファイルの作成後に、Web サービス用の J2EE の .ear ファイルを生成できます。J2EE の .ear ファイルには、ファイル web.xml などの Java Web サービス・サーブレット構成情報と、提供する Java クラスおよびインタフェースが含まれます。

Oracle Application Server Web Services のアセンブリ・ツール WebServicesAssembler を次のように実行します。

```
java -jar WebServicesAssembler.jar -config config_file
```

ここで、*config_file* は、<stateless-java-service> または <stateful-java-service> タグを含む構成ファイルです。

**関連項目：**

- 3-8 ページの「Java クラスの Web サービスをアセンブルするための構成ファイルの作成」
- 9-2 ページの「Web サービス・アセンブリ・ツールの実行」

## Java クラス・ベースの Web サービスのデプロイ

Java クラスと Web サービス・サーブレットのデプロイメント・ディスクリプタを含む J2EE の .ear ファイルの作成後に、.ear ファイルに格納されている標準的な J2EE アプリケーションの場合と同様に、Web サービスを（OC4J で動作するように）デプロイできます。

**関連資料：**『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』

# Web サービス用のパラメータと結果のシリアライズおよびエンコーディング

Web サービス・クライアントと Web サービス実装間では、パラメータと結果が次の手順で送信されます。

1. パラメータは、Web サービス・クライアントから送信されるときにシリアライズされ、XML でエンコードされます。
2. パラメータは、Web サービスがサーバー・サイドでリクエストを受信するときにデシリアライズされ、XML からデコードされます。
3. パラメータまたは結果は、Web サービスから Web サービス・クライアントにリクエストが戻されるときにシリアライズされ、XML でエンコードされます。
4. パラメータまたは結果は、Web サービス・クライアントがリプライを受信するときにデシリアライズし、XML からデコードする必要があります。

Oracle Application Server Web Services では、この 4 つの手順、つまりシリアライズ、エンコード、デシリアライズおよびデコードを処理するために、事前にパッケージングされた実装がサポートされます。事前にパッケージングされたメカニズムにより、4 つのシリアライズおよびエンコード手順は、Web サービス・クライアント・サイド・アプリケーションと、Web サービスを実装する Java サービス作成者の両方にとって透過的になります。Oracle Application Server Web Services では、事前にパッケージングされたメカニズムを使用して、次のエンコーディング・メカニズムがサポートされます。

- 標準 SOAP 1.1 エンコーディング。標準 SOAP 1.1 エンコーディングを使用すると、Java クラス実装をコールするサーバー・サイド Web サービス・サーブレットは、Oracle Application Server Web Services でサポートされる型に対してシリアライズとエンコーディングを内部的に処理します。標準 SOAP 1.1 のエンコーディングを使用する場合にサポートされる Web サービス・パラメータと戻り値の型については、表 3-1 を参照してください。
- リテラル XML エンコーディング。リテラル XML エンコーディングを使用すると、W3C Document Object Model（DOM）準拠の org.w3c.dom.Element としてエンコードされ

る値を、Web サービス・クライアントがパラメータとして渡すか、Java サービスが結果の戻り値として使用できます。Web サービスにパラメータとして `Element` が渡されると、サーバー・サイド Java 実装では `org.w3c.dom.Element` が処理されます。Web サービスから送信される戻り値の場合、Web サービス・クライアントは `org.w3c.dom.Element` を解析または処理します。

---

**注意：** Web サービスのパラメータ、あるいは Web サービスが生成して Web サービス・クライアントに戻る結果の場合、Oracle Application Server Web Services 実装では、特定の Web サービス（Java メソッド）について、標準 SOAP エンコーディングまたはリテラル XML エンコーディングのいずれか一方のみがサポートされます。

---

**関連項目：** 第 8 章「Web サービスを使用するクライアントの作成」

# 4

# EJB Web サービスの開発とデプロイ

この章では、ステートレス Session Enterprise JavaBeans（EJB）として実装される Oracle Application Server Web Services を記述してデプロイする手順について説明します。

この章の内容は、次のとおりです。

# Oracle Application Server Web Services とステートレス Session EJB の使用

この章では、ステートレス Session EJB を使用して実装される Web サービスを記述するためのサンプル・コードを示します。

Oracle Application Server には、Web サービスを実装する EJB にアクセスするサーブレットが用意されています。サーブレットでは、Web サービス・クライアントにより生成されたリクエストの処理、EJB のホームおよびリモート・インタフェースの検索、Web サービスを実装する EJB の実行、結果を Web サービス・クライアントに戻すなどの処理が行われます。

**関連項目：**

- 第 2 章「Oracle Application Server Web サービス」
- 第 3 章「Java クラス Web サービスの開発とデプロイ」
- 第 5 章「ストアド・プロシージャ Web サービスの開発とデプロイ」
- 第 8 章「Web サービスを使用するクライアントの作成」

# ステートレス Session EJB Web サービスの記述

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 を取得または作成する必要があります。この EJB には、Oracle Application Server で動作する Web サービス・サーブレットにより起動される 1 つ以上のメソッドを含める必要があります。これらのメソッドが起動されるのは、クライアントが Web サービス・リクエストを発行したときです。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 サービスでは通常、Web サービス・クライアントに送信するデータが生成されるか、Web サービスのメソッド・リクエストで指定されたアクションが実行されます。

この項では、文字列 Hello World をクライアントに戻す単純なステートレス Session EJB Web サービス `HelloService` の記述方法について説明します。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。

サンプル・コードは、次の OTN の Web サイトで提供されています。

`http://otn.oracle.com/tech/java/oc4j/demos/1012/index.html`

Web サービスの `demo.zip` ファイルを解凍すると、EJB ベースの Web サービスが、UNIX の場合は `/webservices/demo/basic/stateless_ejb` の下のディレクトリ、Windows の場合は `¥webservices¥demo¥basic¥stateless_ejb` の下のディレクトリに格納されます。

ステートレス Session EJB Web サービスを作成するには、リモート・インタフェース、ホーム・インタフェースおよびエンタープライズ Bean クラスを含む、標準的な J2EE のステートレス Session EJB を記述します。Oracle Application Server Web Services では、Web サービス・クライアントにより発行されたリクエストにレスポンスを戻す時に、Oracle Application Server Web Services としてデプロイされた EJB が実行されます。

ステートレス Session EJB の開発手順は、次のとおりです。

- ステートレス Session EJB のリモート・インタフェースの定義
- ステートレス Session EJB のホーム・インタフェースの定義
- ステートレス Session EJB の Bean の定義
- EJB Web サービスからの結果の戻し
- EJB Web サービスのエラー処理
- EJB Web サービス用のパラメータと結果のシリアライズおよびエンコーディング
- ステートレス Session EJB Web サービスについてサポートされているデータ型の使用
- EJB Web サービス用 WSDL ファイルの記述（オプション）

**関連項目：** 4-7 ページの「ステートレス Session EJB ベースの Web サービスの準備とデプロイ」

## ステートレス Session EJB のリモート・インタフェースの定義

HelloService EJB Web サービスを参照すると、.ear ファイル HelloService.ear で Web サービスとその構成ファイルが定義されていることがわかります。サンプル・ディレクトリ内の HelloService.java ファイルは、HelloService EJB 用のリモート・インタフェースを提供します。

例 4-1 に、サンプル・ステートレス Session EJB 用の Remote インタフェースを示します。

**例 4-1 Web サービス用ステートレス Session EJB のリモート・インタフェース**

```
package demo;

public interface HelloService extends javax.ejb.EJBObject {
java.lang.String hello(java.lang.String phrase) throws java.rmi.RemoteException;
}
```

## ステートレス Session EJB のホーム・インタフェースの定義

サンプル・ファイル HelloServiceHome.java は、HelloService EJB 用のホーム・インタフェースを提供します。

例 4-2 に、サンプル・ステートレス Session EJB 用の EJBHome インタフェースを示します。

**例 4-2 Web サービス用ステートレス Session EJB のホーム・インタフェース**

```
package demo;
/**
 * This is a Home interface for the Session Bean
 */
public interface HelloServiceHome extends javax.ejb.EJBHome {

HelloService create() throws javax.ejb.CreateException, java.rmi.RemoteException
;
}
```

## ステートレス Session EJB の Bean の定義

サンプル・ファイル HelloServiceBean.java は、HelloService EJB 用の Bean のロジックを提供します。Bean を作成して Web サービスを実装する場合は、サポートされる型のパラメータと戻り値を使用する必要があります。Web サービスを実装するステートレス Session EJB のパラメータと戻り値用にサポートされている型については、表 4-1 を参照してください。

例 4-3 に、HelloService Bean のソース・コードを示します。

**例 4-3 Web サービス用のステートレス Session EJB の Bean クラス**

```
package demo;

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

/**
 * This is a Session Bean Class.
 */
public class HelloServiceBean implements SessionBean {
    private javax.ejb.SessionContext mySessionCtx = null;

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 {}
public javax.ejb.SessionContext getSessionContext() {
    return mySessionCtx;
}
public String hello(String phrase)
{
    return "HELLO!! You just said :" + phrase;
}
public void setSessionContext(javax.ejb.SessionContext ctx) throws
java.rmi.RemoteException {
    mySessionCtx = ctx;
}
}
```

## EJB Web サービスからの結果の戻し

例 4-3 の `hello()` メソッドは `String` を戻します。Oracle Application Server Web Services のサーバー・サイド・サーブレットでは、クライアントから Web サービス・リクエストを受信すると、`hello()` メソッドをコールする Bean が実行されます。`hello()` メソッドの実行後に、サーブレットは結果を Web サービス・クライアントに戻します。

例 4-3 は、EJB Bean の作成者が、サポートされる型の値を戻すのみで、ステートレス Session EJB として実装される Web サービスを作成できることを示しています。

**関連項目：** 4-5 ページの「ステートレス Session EJB Web サービスについてサポートされているデータ型の使用」

## EJB Web サービスのエラー処理

EJB として実装された Web サービスの実行中にエラーが発生した場合、その EJB は例外をスローします。例外がスローされると、Web サービス・サーブレットは Web サービス（SOAP）障害を戻します。実装にステートレス Session EJB を使用する Web サービスについてサーブレット・エラーをロギングするには、標準の J2EE および OC4J 管理機能を使用します。

## EJB Web サービス用のパラメータと結果のシリアライズおよびエンコーディング

Web サービス・クライアントと Web サービス実装の間で送信されるパラメータと結果は、エンコードしてシリアライズする必要があります。これにより、コールおよび戻り値を、SOAP を使用して XML 文書として渡すことができます。

**関連項目：** 3-12 ページの「Web サービス用のパラメータと結果のシリアライズおよびエンコーディング」

## ステートレス Session EJB Web サービスについてサポートされているデータ型の使用

表 4-1 に、Oracle Application Server Web Services のパラメータと戻り値についてサポートされているデータ型を示します。

**表 4-1 Web サービスでサポートされるデータ型**

| プリミティブ型 | オブジェクト型 |
|---|---|
| `Boolean` | `java.lang.Boolean` |
| `byte` | `java.lang.Byte` |
| `double` | `java.lang.Double` |
| `float` | `java.lang.Float` |
| `int` | `java.lang.Integer` |
| `long` | `java.lang.Long` |
| `short` | `java.lang.Short` |
| `string` | `java.lang.String` |
| | `java.util.Date` |
| | `java.util.Map` |
| | `org.w3c.dom.Element` |
| | `org.w3c.dom.Document` |
| | `org.w3c.dom.DocumentFragment` |
| | JavaBeans（その JavaBean のプロパティの型がこの表に含まれている型、または他のサポートされる JavaBeans） |
| | この表に示す型の単一次元配列 |

**注意：** Oracle Application Server Web Services では、`Element[]`（`org.w3c.dom.Element` の配列）はサポートされません。

Oracle Application Server Web Services でのドキュメント・スタイルの Web サービス実装の場合は、Web サービスを実装する Java メソッドのシグネチャが制限されます。これらの Web サービスの間で受け渡しや送信が可能なのは、`org.w3c.dom.Element` のみです。

**注意：** 前述の制限とは、Web サービスを実装するメソッドで、`org.w3c.dom.Element` タイプをパラメータとして他のタイプと混在させることができないことを意味します。

Web サービスに使用する Bean は、次の制限に準拠する Java クラスです。

- 引数を取らないコンストラクタが必須です。
- 必要なすべての状態をプロパティを通じて公開する必要があります。
- set*X* または get*X* メソッドなどのプロパティのアクセッサは、呼び出される順番に依存しないようにする必要があります。

Oracle Application Server Web Services では、Bean が表 4-1 に示すプロパティ型またはサポートされている他の JavaBean のみで構成されているかぎり、Bean を J2EE Web サービスのメソッドに戻したり、引数として渡すことができます。

JavaBeans を Oracle Application Server Web Services へのパラメータとして使用する場合、クライアント・サイド・コードでは、ダウンロードしたクライアント・サイド・プロキシに含まれている、生成された Bean を使用する必要があります。これは、生成されたクライアント・サイド・プロキシ・コードでは、SOAP 構造の名前空間と完全修飾 Bean クラス名の間で変換が行われることで、SOAP 構造と JavaBean の間での変換が行われるためです。指定した名前の Bean が指定したパッケージに存在しない場合、生成されたクライアント・コードは失敗します。

ただし、クライアントで Web Services Description Language（WSDL）を使用して、クライアント・サイド・プロキシではなく Oracle Application Server Web Services のコールを形成する場合、特別な要件はありません。生成された WSDL ドキュメントでは、SOAP 構造が標準的な方法で記述されます。WSDL ドキュメントから直接動作する Oracle JDeveloper などのアプリケーション開発環境では、JavaBeans をパラメータとして使用し、Oracle Application Server Web Services を適切にコールできます。

---

**注意：** Web サービスのプロキシ・クラスと WSDL が生成されると、サーバー・サイド・サービス実装に含まれるすべての Java プリミティブ型は、プロキシ・コードまたは WSDL 内のオブジェクト型にマップされます。たとえば、Web サービス実装に Java プリミティブ型 `int` のパラメータが含まれている場合、プロキシ内の対応するパラメータは `java.lang.Integer` 型です。このマッピングは、どのプリミティブ型の場合にも発生します。

---

**関連項目：** 第 8 章「Web サービスを使用するクライアントの作成」

## EJB Web サービス用 WSDL ファイルの記述（オプション）

`WebServicesAssembler` では、Web サービス開発者が WSDL ファイルとクライアント・サイド・プロキシ・ファイルを生成できるように、`<wsdl-gen>` および `<proxy-gen>` タグがサポートされます。これらのタグを使用して、WSDL ファイルとクライアント・サイド・プロキシを生成するかどうかを制御できます。また、これらのタグを使用すると、生成された WSDL ファイルまたは記述する WSDL ファイルを、Web サービスの J2EE の .ear とともにパッケージするように指定できます。

クライアント・サイド開発者が Web サービスを使用するアプリケーションを作成するには、デプロイされた Web サービスから取得した WSDL ファイルを使用する方法と、WSDL から生成されるクライアント・サイド・プロキシを使用する方法があります。

**関連項目：** 9-4 ページの「WSDL ファイルとクライアント・サイド・プロキシの生成」

# ステートレス Session EJB ベースの Web サービスの準備とデプロイ

ステートレス Session EJB を Web サービスとしてデプロイするには、Oracle Application Server Web Services サーブレットのデプロイメント・ディスクリプタと Java 実装を提供する ejb.jar ファイルを含む、J2EE の .ear ファイルをアセンブルする必要があります。この項では、Oracle Application Server Web Services のツールである `WebServicesAssembler` の使用方法について説明します。`WebServicesAssembler` では、ステートレス Session EJB Web サービスを記述する XML 構成ファイルを使用して、Oracle Application Server Web Services でデプロイできる J2EE の .ear ファイルが生成されます。

この項の内容は、次のとおりです。

- ステートレス Session EJB Web サービスをアセンブルするための構成ファイルの作成
- ステートレス Session EJB Web サービスを準備するための WebServicesAssembler の実行
- EJB として実装された Web サービスのデプロイ

## ステートレス Session EJB Web サービスをアセンブルするための構成ファイルの作成

Oracle Application Server Web Services のアセンブリ・ツール `WebServicesAssembler` を使用すると、Oracle Application Server Web Services をアセンブルできます。この項では、ステートレス Session EJB Web サービスで使用する構成ファイルの作成方法について説明します。

次のタグを追加して、`WebServicesAssembler` 構成ファイルを作成します。

- Web サービスのトップレベル・タグの追加
- ステートレス Session EJB サービス・タグの追加
- WSDL およびクライアント・サイド・プロキシ生成タグの追加

### Web サービスのトップレベル・タグの追加

表 4-2 に、`WebServicesAssembler` 構成ファイルのトップレベル・タグを示します。これらのタグを追加して、Java ステートレス Web サービスまたは Java ステートフル Web サービスのトップレベル記述情報を提供します。これらのタグは、構成ファイル内で `<web-service>` タグ内に挿入します。

トップレベル・タグなど、`config.xml` ファイルの詳細は、例 4-4 を参照してください。

**表 4-2 トップレベルの `WebServicesAssembler` 構成タグ**

| タグ | 説明 |
|---|---|
| `<context>`<br>*context*<br>`</context>` | Web サービスのコンテキスト・ルートを指定します。<br>このタグは必須です。 |
| `<datasource-JNDI-name>`<br>*datasource*<br>`</datasource-JNDI-name>` | Web サービスに関連するデータソースを指定します。 |
| `<description>`<br>*description*<br>`</description>` | Web サービスの簡単な説明を提供します。<br>このタグはオプションです。 |
| `<destination-path>`<br>*dest_path*<br>`</destination-path>` | 生成される J2EE の .ear ファイルの出力名を指定します。*dest_path* には、出力ファイルのフルパスを指定します。<br>このタグは必須です。 |
| `<display-name>`<br>*disp_name*<br>`</display-name>` | Web サービスの表示名を指定します。<br>このタグはオプションです。 |

**表 4-2 トップレベルの WebServicesAssembler 構成タグ（続き）**

| タグ | 説明 |
|---|---|
| <option name="source-path"><br>*path*<br><option> | 指定したファイルを出力 .ear ファイルに含めます。このオプションを使用して Java リソースを指定するか、J2EE の出力 .ear ファイルのソース・ファイルとして使用される既存の .war、.ear または ejb-jar ファイルの名前を指定します。<br>.war ファイルが入力として提供されると、オプションの contextroot によって、.war ファイルの root-context が指定されます。<br>*path1* には、.war の context-root を指定します。<br>*path2* には、組み込むファイルへのパスを指定します。<br>次に例を示します。<br><option name="source-path" contextroot="/test">/myTestArea/ws/src/statefull.war<br></option><br>このタグはオプションです。 |
| <stateless-session-ejb-service><br>*sub-tags*<br></stateless-session-ejb-service> | このタグを使用して、ステートレス Session EJB Web サービスを追加します。有効な *sub-tags* の詳細は、表 4-3 を参照してください。 |
| <temporary-directory><br>*temp_dir*<br></temporary-directory> | アセンブラで一時ファイルを格納できるディレクトリを指定します。<br>このタグはオプションです。 |

## ステートレス Session EJB サービス・タグの追加

WebServicesAssembler <stateless-session-ejb-service> タグを使用して、ステートレス Session EJB Web サービスを準備します。このタグは、構成ファイル内で <web-service> タグ内に挿入します。このタグを追加して、ステートレス Session EJB Web サービスの生成に必要な情報を提供します。

表 4-3 に、<stateless-session-ejb-service> のサブタグを示します。

<stateless-session-ejb-service> など、config.xml ファイルの詳細は、例 4-4 を参照してください。

**表 4-3 ステートレス Session EJB Web サービスのサブタグ**

| タグ | 説明 |
|---|---|
| <accept-untyped-request><br>*value*<br></accept-untyped-request> | *value* に true を設定すると、Web サービスで型未指定のリクエストが受入可能であることを WebServicesAssembler に通知します。value を false に設定すると、Web サービスは型未指定のリクエストを受け入れません。<br>有効な値：true、false<br>（大 / 小文字は区別されません。TRUE および FALSE も有効です）<br>このタグはオプションです。<br>デフォルト値：false |
| <ejb-name><br>*name*<br></ejb-name> | ステートレス Session EJB の名前を指定します。<br>このタグは必須です。 |

**表 4-3 ステートレス Session EJB Web サービスのサブタグ（続き）**

| タグ | 説明 |
|---|---|
| <ejb-resource><br>*resource*<br></ejb-resource> | これは下位互換性タグです。<br>関連項目：表 4-2 のトップレベルの <option name="source-path"> タグ。<br>このタグはオプションです。 |
| <path><br>*path*<br></path> | これは下位互換性タグです。<br>関連項目：表 4-2 のトップレベルの <option name="source-path"> タグ。<br>このタグはオプションです。 |
| <uri><br>*URI*<br></uri> | Web サービスを実装するサーブレットのサーブレット・マッピング・パターンを指定します。*URI* として指定したパスは、Web サービスの位置を指定する <context> に追加されます。<br>このタグは必須です。 |

**例 4-4 ステートレス Session EJB の WebServicesAssembler 構成ファイルのサンプル**

```
<web-service>
    <display-name>EJB Web Services Demo</display-name>
    <destination-path>tmp/HelloService.ear</destination-path>
    <temporary-directory>tmp</temporary-directory>
    <context>/sejb_webservices</context>

    <stateless-session-ejb-service>
        <path>tmp/Hello.jar</path>
        <uri>/HelloService</uri>
        <ejb-name>HelloService</ejb-name>
    </stateless-session-ejb-service>
</web-service>
```

## WSDL およびクライアント・サイド・プロキシ生成タグの追加

WebServicesAssembler では、Web サービス開発者が WSDL ファイルとクライアント・サイド・プロキシ・ファイルを生成できるように、<wsdl-gen> および <proxy-gen> タグがサポートされます。これらのタグを使用して、WSDL ファイルとクライアント・サイド・プロキシを生成するかどうかを制御できます。また、これらのタグを使用すると、生成された WSDL ファイルまたは記述する WSDL ファイルを、Web サービスの J2EE の .ear とともにパッケージするように指定できます。

クライアント・サイド開発者が Web サービスを使用するアプリケーションを作成するには、デプロイされた Web サービスから取得した WSDL ファイルを使用する方法と、WSDL から生成されるクライアント・サイド・プロキシを使用する方法があります。

**関連項目：** 9-4 ページの「WSDL ファイルとクライアント・サイド・プロキシの生成」

## ステートレス Session EJB Web サービスを準備するための WebServicesAssembler の実行

`WebServicesAssembler` 構成ファイルの作成後に、Web サービス用の J2EE の .ear ファイルを生成できます。J2EE の .ear ファイルには、ステートレス Session EJB Web サービス・サーブレットの構成情報が含まれます。

Oracle Application Server Web Services のアセンブリ・ツール `WebServicesAssembler` を次のように実行します。

```
java -jar WebServicesAssembler.jar -config config_file
```

ここで、*config_file* は、`<stateless-session-ejb-service>` タグを含む構成ファイルです。

**関連項目：**

- 4-7 ページの「ステートレス Session EJB Web サービスをアセンブルするための構成ファイルの作成」
- 9-2 ページの「Web サービス・アセンブリ・ツールの実行」

## EJB として実装された Web サービスのデプロイ

ステートレス Session EJB を含む .ear ファイルの作成後、.ear ファイルに格納されている標準的な J2EE アプリケーションの場合と同様に、Web サービスを（OC4J で動作するように）デプロイできます。

**関連資料：**『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』

# 5

# ストアド・プロシージャ Web サービスの開発とデプロイ

この章では、ステートレス PL/SQL ストアド・プロシージャまたはファンクション（**ストアド・プロシージャ Web サービス**）として実装される Oracle Application Server Web Services を記述およびデプロイする方法について説明します。ストアド・プロシージャ Web サービスを使用すると、Oracle データベース・サーバー上で動作する PL/SQL プロシージャおよびファンクションを、Oracle Application Server Web Services で動作するサービスとしてエクスポートできます。

この章の内容は、次のとおりです。

# Oracle Application Server Web Services とストアド・プロシージャの使用

この章では、ステートレス PL/SQL ストアド・プロシージャまたはファンクションを使用して実装される Web サービスを記述するためのサンプル・コードを示します。このサンプルは、従業員を管理する会社を表す PL/SQL パッケージをベースとしています。

Oracle Application Server Web Services には、PL/SQL ストアド・プロシージャ Web サービスをサポートする Java クラスにアクセスするためのサーブレットが用意されています。このサーブレットでは、Web サービス・クライアントにより生成されたリクエストの処理、Web サービス実装用のストアド・プロシージャにアクセスする Java メソッドの実行、結果を Web サービス・クライアントに戻すなどの処理が行われます。

Oracle データベース・サーバーでは、Java や C/C++ など、PL/SQL 以外の言語で実装されたプロシージャがサポートされます。PL/SQL インタフェースを使用すると、これらのストアド・プロシージャを Web サービスとして公開できます。

**関連項目：**

- 第 2 章「Oracle Application Server Web サービス」
- 第 3 章「Java クラス Web サービスの開発とデプロイ」
- 第 6 章「ドキュメント・スタイルの Web サービスの開発とデプロイ」

# ストアド・プロシージャ Web サービスの記述

ストアド・プロシージャ Web サービスを記述するには、PL/SQL パッケージを作成し、Oracle Application Server のデータソースとして使用可能な Oracle データベース・サーバーにインストールして、ストアド・プロシージャにアクセスする 1 つ以上のメソッドを含む Java クラスを生成する必要があります。

サンプル・コードは、次の OTN の Web サイトで提供されています。

```
http://otn.oracle.com/tech/java/oc4j/demos/1012/index.html
```

Web サービスの `demo.zip` ファイルを解凍すると、ストアド・プロシージャ Web サービスのサンプルが、UNIX の場合は `/webservices/demo/basic/stored_procedure` の下のディレクトリ、Windows の場合は `¥webservices¥demo¥basic¥stored_procedure` の下のディレクトリに格納されます。

ストアド・プロシージャ Web サービスを作成するには、PL/SQL ストアド・プロシージャを記述してインストールします。そのためには、Oracle Application Server Web Services に依存しない機能を使用する必要があります。

たとえば、サンプルの `COMPANY` パッケージを使用するには、まず `create.sql` スクリプトを使用してパッケージを作成し、作成したパッケージをデータベース・サーバーにロードします。このスクリプトと、必要な他の `.sql` スクリプトは、`stored_procedure` ディレクトリにあります。これらのスクリプトでは、複数のデータベース表とサンプルの `COMPANY` パッケージが作成されます。

Oracle データベース・サーバーがローカル・システムで稼働している場合は、次のコマンドを使用してサンプル PL/SQL パッケージを作成します。

```
sqlplus scott/tiger @create
```

Oracle データベース・サーバーがローカル以外のシステムで稼働している場合は、次のコマンドを使用して接続識別子を指定し、サンプル PL/SQL パッケージを作成します。

```
sqlplus scott/tiger@db_service_name @create
```

*db_service_name* は、Oracle データベース・サーバーのネット・サービス名です。

**関連資料：**

- 5-9 ページの「Web サービスとして動作するストアド・プロシージャの制限」
- 『PL/SQL ユーザーズ・ガイドおよびリファレンス』
- 『Oracle Net Services 管理者ガイド』

# ストアド・プロシージャ Web サービスの準備

この項では、Oracle Application Server Web Services のツール `WebServicesAssembler` を使用して、ストアド・プロシージャ Web サービスとしての PL/SQL プロシージャまたはファンクションの使用をサポートする J2EE の .ear ファイルを準備する方法について説明します。

この項の内容は、次のとおりです。

- ストアド・プロシージャ Web サービスをアセンブルするための構成ファイルの作成
- ストアド・プロシージャ Web サービスを使用した WebServicesAssembler の実行
- Oracle Application Server Web Services（OC4J）でのデータソースの設定

## ストアド・プロシージャ Web サービスをアセンブルするための構成ファイルの作成

Oracle Application Server Web Services のアセンブリ・ツール `WebServicesAssembler` を使用すると、Oracle Application Server Web Services をアセンブルできます。この項では、ストアド・プロシージャ Web サービスのアセンブルに使用する構成ファイルの作成方法について説明します。Web サービス・アセンブリ・ツールでは、ストアド・プロシージャ Web サービスを記述する XML 構成ファイルを使用して、Oracle Application Server Web Services でデプロイできる J2EE の .ear ファイルが生成されます。

次のタグを追加して、`WebServicesAssembler` 構成ファイルを作成します。

- Web サービスのトップレベル・タグの追加
- ステートレス・ストアド・プロシージャ Java サービス・タグの追加
- WSDL およびクライアント・サイド・プロキシ生成タグの追加

### Web サービスのトップレベル・タグの追加

表 5-1 に、`WebServicesAssembler` 構成ファイルのトップレベル・タグを示します。これらのタグを追加して、PL/SQL ストアド・プロシージャ Web サービスを記述するトップレベルの情報を提供します。

トップレベル・タグなど、`config.xml` ファイルの詳細は、例 5-1 を参照してください。

**表 5-1 トップレベルの `WebServicesAssembler` 構成タグ**

| タグ | 説明 |
|---|---|
| `<context>`<br>*context*<br>`</context>` | Web サービスのコンテキスト・ルートを指定します。<br>このタグは必須です。 |
| `<datasource-JNDI-name>`<br>*datasource*<br>`</datasource-JNDI-name>` | Web サービスに関連するデータソースを指定します。 |
| `<description>`<br>*description*<br>`</description>` | Web サービスの簡単な説明を提供します。<br>このタグはオプションです。 |

**表 5-1 トップレベルの WebServicesAssembler 構成タグ（続き）**

| タグ | 説明 |
|---|---|
| <destination-path><br>*dest_path*<br></destination-path> | 生成される J2EE の .ear ファイルの出力名を指定します。*dest_path* には、出力ファイルのフルパスを指定します。<br>このタグは必須です。 |
| <display-name><br>*disp_name*<br></display-name> | Web サービスの表示名を指定します。<br>このタグはオプションです。 |
| <option name="source-path"><br>*path*<br><option> | 指定したファイルを出力 .ear ファイルに含めます。このオプションを使用して、Java リソースを含めます。<br>*path* には、組み込むファイルへのパスを指定します。 |
| <stateless-stored-procedure-java-service><br>*sub-tags*<br></stateless-stored-procedure-java-service> | このタグを使用して、ステートレス・ストアド・プロシージャ Web サービスを追加します。有効な *sub-tags* の詳細は、表 5-2 および表 5-4 を参照してください。 |
| <temporary-directory><br>*temp_dir*<br></temporary-directory> | アセンブラで一時ファイルを格納できるディレクトリを指定します。<br>このタグはオプションです。 |

## ステートレス・ストアド・プロシージャ Java サービス・タグの追加

WebServicesAssembler を使用してストアド・プロシージャ Web サービスを開発するには、次の 2 つの方法があります。

- Jar 生成を使用したステートレス・ストアド・プロシージャ Java サービスの追加
- 事前生成済 Jar を使用したステートレス・ストアド・プロシージャ Java サービスの追加

---

**注意：** ほとんどのストアド・プロシージャ Web サービス開発者は、Jar 生成技法を使用して Web サービスの J2EE の .ear ファイルをアセンブルします。J2EE の .ear ファイル作成に事前生成済 Jar の技法を使用するのは、Oracle JPublisher で生成されたクラスを含む事前生成済 Jar ファイルがある場合のみです。

---

### Jar 生成を使用したステートレス・ストアド・プロシージャ Java サービスの追加

WebServicesAssembler で Oracle JPublisher を実行して、ストアド・プロシージャ Web サービスをサポートするクラスを生成できるように、<jar-generation> タグを含む構成ファイルを使用して、Oracle データベース・サーバー接続情報を指定します。Oracle JPublisher により生成されたクラスでは、PL/SQL プロシージャまたはファンクションへのアクセスがサポートされ、Java の型を PL/SQL の型にマップするためのクラスも含まれます。生成されたクラスは、WebServicesAssembler により、ストアド・プロシージャ Web サービスとともにアセンブルされる Jar ファイルにパッケージされます。

表 5-2 に、WebServicesAssembler 構成ファイルの <stateless-stored-procedure-java-service> タグを示します。このタグは、ストアド・プロシージャ Web サービスの作成に Jar 生成を使用する構成ファイルの作成時に使用します。<stateless-stored-procedure-java-service> タグは、構成ファイルの <web-service> タグ内に挿入します。このタグを追加して、ストアド・プロシージャ Web サービスの J2EE の .ear ファイル生成に必要な情報を提供します。

表 5-3 に、<stateless-stored-procedure-java-service> タグ内の <jar-generation> のサブタグを示します。<jar-generation> タグでは、Oracle JPublisher を実行してストアド・プロシージャ Web サービス用の Java クラスを生成できるように、WebServicesAssembler に情報を提供します。WebServicesAssembler では、これらのクラスを使用して、ストアド・プロシージャまたはファンクションの Java マッピングを提供する Jar ファイルが生成されます。

表 5-2 および表 5-3 に示すストアド・プロシージャ Web サービスのタグなど、config.xml ファイルの詳細は、例 5-1 を参照してください。

**表 5-2 ステートレス・ストアド・プロシージャのサブタグ（Jar 生成を使用）**

| タグ | 説明 |
|---|---|
| <database-JNDI-name><br>*source_JNDI_name*<br></database-JNDI-name> | このタグでは、バックエンド・データベースの JNDI 名を指定します。<br>data-sources.xml OC4J 構成ファイルには、指定した source_JNDI_name に関連するデータベース・サーバーのソースを記述します。 |
| <jar-generation><br>*sub-tags*<br></jar-generation> | <jar-generation> についてサポートされる *sub-tags* は、表 5-3 を参照してください。<br>例 :<br><jar-generation><br><schema>scott/tiger</schema><br><db-url>jdbc:oracle:thin:@system1:1521:orcl</db-url><br><prefix>sp.company</prefix><br><db-pkg-name>Company</db-pkg-name><br></jar-generation> |
| <uri><br>*URI*<br></uri> | このタグでは、Web サービスを実装するサーブレットのサーブレット・マッピング・パターンを指定します。*URI* として指定したパスは、Web サービスの位置を指定する <context> に追加されます。 |

**表 5-3 ステートレス・ストアド・プロシージャの <jar-generation> のサブタグ**

| タグ | 説明 |
|---|---|
| <db-pkg-name><br>*pkg_name*<br></db-pkg-name> | *pkg_name* は、エクスポートする PL/SQL パッケージの名前です。<br><jar-generation> を使用する場合、このタグは必須です。 |
| <db-url><br>*url_path*<br></db-url> | *url_path* は、指定したパッケージのエクスポート元となる Oracle データベース・サーバーのデータベース接続文字列です。<schema> および <db-url> の組合せにより、エクスポート対象のストアド・プロシージャを含むデータベースに接続されます。<br><jar-generation> を使用する場合、このタグは必須です。<br>例 :<br><db-url>jdbc:oracle:thin:@system1.us.oracle.com:1521:tv1</db-url> |

**表 5-3 ステートレス・ストアド・プロシージャの <jar-generation> のサブタグ（続き）**

| タグ | 説明 |
|---|---|
| <method-name><br>*method*<br></method-name> | *method* は、エクスポートする PL/SQL メソッドの名前です。<br>このタグはオプションです。複数の <method> タグを使用できます。この場合は、指定した各メソッドがエクスポートされます。<br>このタグを使用しないと、パッケージ内のメソッドがすべてエクスポートされます。指定したメソッドがオーバーロードされる場合は、そのメソッドのバリエーションがすべてエクスポートされます。 |
| <prefix><br>*prefix*<br></prefix> | *prefix* は、生成されるクラスの Java パッケージ接頭辞です。<br>デフォルトでは、PL/SQL パッケージはデフォルトの Java パッケージ内の Java クラスに生成されます。<br>このタグはオプションです。<br>例 :<br><prefix>sp.company</prefix> |
| <schema><br>*user_name/password*<br></schema> | このタグでは、データベース・サーバーの *user_name/password* を指定します。<br>各項目の意味は、次のとおりです。<br>*user_name* は、データベース・ユーザー名です。<br>*password* は、指定したユーザー名のデータベース・パスワードです。<br><jar-generation> を使用する場合、このタグは必須です。<br>例 :<br><schema>scott/tiger</schema> |

**例 5-1 <jar-generation> タグを使用したストアド・プロシージャ用 WebServicesAssembler 構成ファイルのサンプル**

```
<web-service>
    <display-name>Web Services Example</display-name>
    <description>Java Web Service Example</description>
    <!-- Specifies the resulting web service archive will be stored in ./spexample.ear -->
    <destination-path>./spexample.ear</destination-path>
    <!-- Specifies the temporary directory that web service assembly tool can create temporary files. -->
    <temporary-directory>/tmp</temporary-directory>
    <!-- Specifies the web service will be accessed in the servlet context named "/webservices". -->
    <context>/webservices</context>
    <!-- Specifies the web service will be stateless -->

    <stateless-stored-procedure-java-service>
       <jar-generation>
         <schema>scott/tiger</schema>
         <db-url>jdbc:oracle:thin:@system1:1521:orcl</db-url>
         <prefix>sp.company</prefix>
         <db-pkg-name>Company</db-pkg-name>
       </jar-generation>
      <!-- Specifies the web service will be accessed in the uri named
            "statelessSP" within the servlet context. -->
       <uri>/statelessSP</uri>
       <database-JNDI-name>/jdbc/OracleDataSource</database-JNDI-name>
    </stateless-stored-procedure-java-service>
  <wsdl-gen>
    <wsdl-dir>wsdl</wsdl-dir>
    <!--force 'true'  will write over existing wsdl -->
    <option name="force">true</option>
    <!-- change this to point to your soap servers http listener -->
    <option name="httpServerURL">http://localhost:8888</option>
  </wsdl-gen>
  <proxy-gen>
    <proxy-dir>proxy</proxy-dir>
```

```
    <!-- include-source 'true'  will create an additional jar with only the proxy source-->
    <option name="include-source">true</option>
  </proxy-gen>
</web-service>
```

### 事前生成済 Jar を使用したステートレス・ストアド・プロシージャ Java サービスの追加

Web サービスのサポートに必要なクラスを含む事前生成済の Jar ファイルが使用可能な場合は、ストアド・プロシージャの <class-name> および <interface-name> アセンブリ・オプションを指定する構成ファイルを使用します。構成ファイルに指定する <class-name> および <interface-name> タグでは、PL/SQL プロシージャまたはファンクションと Web サービスの間のマッピングを提供する Java クラスを含む、事前に生成された Jar ファイルの使用がサポートされます。

表 5-4 に、WebServicesAssembler 構成ファイルの <stateless-stored-procedure-java-service> タグを示します。このタグは、ストアド・プロシージャ Web サービスの作成に事前生成済 Jar ファイルを使用する構成ファイルの作成時に使用します。<stateless-stored-procedure-java-service> タグは、構成ファイルの <web-service> タグ内に挿入します。このタグを追加して、ストアド・プロシージャ Web サービスの J2EE の .ear ファイル生成に必要な情報を提供します。

<class> および <interface> タグを <stateless-stored-procedure-java-service> に追加するのは、事前生成済 Jar ファイルを使用する場合のみです。

**表 5-4 ステートレス・ストアド・プロシージャのサブタグ（事前生成済 Jar ファイルを使用）**

| タグ | 説明 |
|---|---|
| <class-name><br>*class*<br></class-name> | ストアド・プロシージャ Web サービス・サーブレット定義には、class-name 値を持つ <param-name> を指定し、対応する <param-value> を PL/SQL Web サービス実装にアクセスする Java クラスの完全修飾名に設定する必要があります。<br>このパラメータのクラス名を指定するには、構成ファイルの <class-name> タグを使用する必要があります。クラス名は、トップレベルの <option name="source-path"> タグに指定した、提供する Jar ファイル内で検索できます。 |
| <database-JNDI-name><br>*source_JNDI_name*<br></database-JNDI-name> | このタグでは、バックエンド・データベースの JNDI 名を指定します。<br>data-sources.xml OC4J 構成ファイルには、指定した source_JNDI_name に関連するデータベース・サーバーのソースを記述します。 |
| <interface-name><br>*interface*<br></interface-name> | ストアド・プロシージャ Web サービス・サーブレット定義には、interface-name 値を持つ <param-name> を指定し、対応する <param-value> をストアド・プロシージャ Web サービスに組み込むメソッドを指定する Java インタフェースの完全修飾名に設定する必要があります。<br><interface-name> タグでは、Web サービスとして公開する必要のあるメソッドを Web サービス・サーブレット生成コードに対して指定するインタフェースの名前を指定します。インタフェース名は、トップレベルの <option name="source-path"> タグに指定した、提供する Jar ファイル内で検索できます。 |
| <java-resource><br>*resource*<br></java-resource> | これは下位互換性タグです。<br>関連項目：表 5-1 のトップレベルの <option name="source-path"> タグ。<br>このタグはオプションです。<br>ストアド・プロシージャの事前生成済 Jar ファイルは、<java-resource> タグを使用して指定する必要があります。<class-name> タグで指定するクラスと、<interface-name> タグで指定するインタフェースは、<java-resource> タグで指定するリソース内に存在する必要があります。 |
| <uri><br>*URI*<br></uri> | このタグでは、Web サービスを実装するサーブレットのサーブレット・マッピング・パターンを指定します。*URI* として指定したパスは、Web サービスの位置を指定する <context> に追加されます。 |

**関連資料：**

- 5-4 ページの「Jar 生成を使用したステートレス・ストアド・プロシージャ Java サービスの追加」
- 『Oracle9*i* JPublisher ユーザーズ・ガイド』

### WSDL およびクライアント・サイド・プロキシ生成タグの追加

`WebServicesAssembler` 構成ファイルでは、Web サービス開発者が Web サービス記述用の WSDL ファイルとクライアント・サイド・プロキシ・ファイルを生成できるように、`<wsdl-gen>` および `<proxy-gen>` タグがサポートされます。これらのタグを追加して、WSDL ファイルとクライアント・サイド・プロキシを生成するかどうかを制御できます。また、ストアド・プロシージャ・スタイル Web サービスの J2EE の .ear ファイルとともに、WSDL ファイルをアセンブルするように指定できます。クライアント・サイド開発者は、デプロイされた Web サービスから取得した WSDL ファイルを使用して、その Web サービスを使用するアプリケーションを作成できます。

**関連項目：** 9-4 ページの「WSDL ファイルとクライアント・サイド・プロキシの生成」

## ストアド・プロシージャ Web サービスを使用した WebServicesAssembler の実行

`WebServicesAssembler` 構成ファイルの作成後に、ストアド・プロシージャ Web サービス用の J2EE の .ear ファイルを生成できます。J2EE の .ear ファイルには、`web.xml` ファイルなど、ストアド・プロシージャ Web サービス・サーブレット構成情報と、Oracle JPublisher により生成されたクラスが含まれます（`WebServicesAssembler` では、Oracle JPublisher により生成されたクラスが、生成される J2EE の .ear に含まれる単一の Jar ファイルに収集されます）。

Oracle Application Server Web Services のアセンブリ・ツール `WebServicesAssembler` を次のように実行します。

```
java -jar WebServicesAssembler.jar -config my_pl_service_config
```

*my_pl_service_config* は、`<stateless-stored-procedure-java-service>` タグを含む構成ファイルです。

**関連項目：**

- 5-3 ページの「ストアド・プロシージャ Web サービスをアセンブルするための構成ファイルの作成」
- 9-2 ページの「Web サービス・アセンブリ・ツールの実行」

## Oracle Application Server Web Services（OC4J）でのデータソースの設定

PL/SQL ストアド・プロシージャ・ベースの Web サービスを追加するには、`data-sources.xml` を構成して OC4J 内でデータソースを設定する必要があります。`data-sources.xml` ファイルを構成すると、OC4J はデータベースを指します。データベースには、ストアド・プロシージャ Web サービスを実装する PL/SQL ストアド・プロシージャ・パッケージを含める必要があります。

OC4J により Web サービス・サーブレット・インスタンスが初期化されると、単一のデータベース接続が作成されます。このデータベース接続は、OC4J により Web サービス・サーブレット・インスタンスが削除された時点で破棄されます。各ストアド・プロシージャ Web サービス・サーブレットにより、単一のスレッド・モデルが実装されます。このため、どの Web サービス・サーブレット・インスタンスでも、特定の時点で処理できるのは単一クライアントのデータベース接続リクエストのみとなります。OC4J により Web サービス・サーブレット・インスタンスがプーリングされ、インスタンスが Oracle Application Server Web Services クライアントに割り当てられます。

PL/SQL Web サービスの起動は、それぞれ明示的に別個のデータベース・トランザクションです。単一のデータベース・トランザクションで複数のサービス・メソッド起動を実行すること

はできません。この種のセマンティックが必須の場合は、他のプロシージャやファンクションを内部的に起動する PL/SQL プロシージャを記述してから、それをストアド・プロシージャ Web サービス内で別のメソッドとして公開する必要があります（ただし、Oracle Application Server Web Services には、そのための明示的なサポートやツールは用意されていません）。

エミュレートされたデータ・ソースをストアド・プロシージャの CLOB 型または BLOB 型で使用する場合、エミュレートされたデータ・ソースは、JNDI 名を指定するために `location` 属性を使用する必要があります。`ejb-location` を使用して名前を指定することはできません。

**関連資料：**『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』

# ストアド・プロシージャ Web サービスのデプロイ

ストアド・プロシージャ Web サービス構成、クラス、Jar およびサポート・ファイルを含む J2EE の .ear ファイルの作成後、J2EE の .ear ファイルに格納されている標準的な J2EE アプリケーションの場合と同様に、Web サービスを（OC4J で動作するように）デプロイできます。

**関連資料：**『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』

# Web サービスとして動作するストアド・プロシージャの制限

この項の内容は、次のとおりです。



## Web サービス用にサポートされるストアド・プロシージャ機能

ストアド・プロシージャ Web サービスでは、次の PL/SQL 機能がサポートされます。

1. プロシージャとファンクションの両方を含む PL/SQL ストアド・プロシージャ。
2. IN、OUT、IN、INOUT パラメータ・モード。ストアド・プロシージャに OUT または INOUT パラメータが含まれる場合、戻されるオブジェクトの属性として、INOUT および OUT データがクライアントに戻されます。宣言されたストアド・プロシージャがファンクションである場合、戻されるオブジェクトの INOUT パラメータ・モードの属性として、このストアド・プロシージャの戻り値も含められます。
3. パッケージされたプロシージャのみ（トップレベル・プロシージャを Web サービスとしてエクスポートするには、パッケージ内でラップする必要があります）。
4. オーバーロードされたプロシージャ。Oracle JPublisher では、複数の PL/SQL 型が同じ Java 型にマップされることがあります。たとえば、PL/SQL の様々な NUMBER 型が、すべて Java の `int` 型にマップされることがあります。これは、PL/SQL ではオーバーロードとみなされていたメソッドが、Java ではオーバーロードされなくなることを意味します。この場合、生成されたコードのコンパイル・エラーを防ぐため、Java メソッド名が変更されます。ただし、実行時には、PL/SQL エンジンにより `PLS-00307` エラー（このコールに一致する < メソッド名 > が複数宣言されています。）がレポートされる場合があります。このエラーは、オーバーロード解決における PL/SQL の制限によるものです。
5. 単純な PL/SQL 型。

   次の単純型がサポートされます。NULL 値は、NATURALN および POSITIVEN を除き、次に示すすべての単純型でサポートされます。

これらの単純型のマッピングの詳細は、Oracle JPublisher のマニュアルを参照してください。

VARCHAR2（STRING、VARCHAR）、LONG、CHAR（CHARACTER）、NUMBER（DEC、DECIMAL、DOUBLE PRECISION、FLOAT、INTEGER、INT、NUMERIC、REAL、SMALLINT）、PLS_INTEGER、BINARY_INTEGER（NATURAL、NATURALN、POSITIVE、POSITIVEN）、BOOLEAN

6. TIMESTAMP がサポートされます。バリエーションである TIMESTAMP WITH LOCAL TIME ZONE および TIMESTAMP WITH TIME ZONE もサポートされます。
7. DATE がサポートされます。
8. ユーザー定義のオブジェクト型。
9. Oracle JPublisher および Oracle Application Server Web Services では、BLOB、CLOB および BFILE という LOB 型がサポートされます。

   PL/SQL プロシージャで入出力の型として LOB 型を使用すると、WebServices Assembler ではランタイム・エラーの原因となるストアド・プロシージャは公開されません。たとえば、WebServices Assembler では、IN パラメータとして BFILE が含まれるメソッドは公開されません。
10. SYS.XMLTYPE はサポートされます。SYS.XMLTYPE は、Web サービスで org.w3c.dom.DocumentFragment の型にマップされます。

**関連資料：**『Oracle9*i* JPublisher ユーザーズ・ガイド』

## Web サービス用にサポートされないストアド・プロシージャ機能

ストアド・プロシージャ Web サービスでは、PL/SQL ファンクションおよびプロシージャに次の制限が適用されます。

1. Web サービスとしてエクスポートされるのは、PL/SQL パッケージ内のプロシージャとファンクションのみです。トップレベルのストアド・プロシージャはパッケージ内でラップする必要があります。メソッドはデフォルトの this 参照を使用してパッケージ・レベルのメソッドにラップする必要があります。
2. NCHAR およびその関連型はサポートされません。
3. Oracle JPublisher では、ほぼすべての PL/SQL 型が Java 型に変換されます。ストアド・プロシージャ Web サービス用のデプロイ・ツールでは、組込み型と数値型の場合は JDBC スタイル、ユーザー型と LOB 型の場合は Oracle スタイルが生成されます。LOB 型は、Web サービスによってシリアライズ / デシリアライズ可能な Java 型に変換されます。JavaBeans に準拠するユーザー型も Web サービスによってシリアライズ / デシリアライズされます。これらのスタイルの詳細と関連する通告については、Oracle JPublisher のマニュアルを参照してください。
4. TIMESTAMP 値の小数秒は、ストアド・プロシージャ Web サービス使用時は保持されません。
5. ユーザー定義 ADT のフィールドとしての TIMESTAMP はサポートされません。ただし、ユーザー定義 ADT のフィールドとしての DATE はサポートされます。

**関連資料：**『Oracle9*i* JPublisher ユーザーズ・ガイド』

## Oracle PL/SQL Web サービスで BOOLEAN を使用する場合のデータベース・サーバーのリリースに関する制限

リリース 9.2.0.1 以下の Oracle データベース・サーバー、または Java 対応でないデータベース・サーバーを使用する場合は、PL/SQL の BOOLEAN 引数がサポートされるように、SYS.SQLJUTIL パッケージを SYS スキーマにインストールする必要があります。

このパッケージを定義する PL/SQL スクリプトは、UNIX 上では次の場所にあります。

```
${ORACLE_HOME}/sqlj/lib/sqljutil.sql
```

Windows 上では、このスクリプトは次の場所にあります。

```
%ORACLE_HOME%¥sqlj¥lib¥sqljutil.sql
```

## TIMESTAMP および DATE の精度に関する制限

TIMESTAMP 値の小数秒は、ストアド・プロシージャ Web サービス使用時は保持されません。

## LOB（CLOB/BLOB）のエミュレートされたデータ・ソースに関する制限

エミュレートされたデータ・ソースを CLOB 型または BLOB 型で使用する場合、エミュレートされたデータ・ソースは、JNDI 名を指定する `location` 属性を使用する必要があります。`ejb-location` を使用して名前を指定することはできません。

# 6

# ドキュメント・スタイルの Web サービスの開発とデプロイ

この章では、Java クラスとして実装されるドキュメント・スタイル・メッセージ処理用の Oracle Application Server Web Services を記述してデプロイする手順について説明します。

この章の内容は、次のとおりです。

# ドキュメント・スタイルの Web サービスの使用

この章では、Java クラスを使用して実装されるドキュメント・スタイルの Web サービスと、ステートフルおよびステートレスのドキュメント・スタイルの Java Web サービスの記述方法の違いについて説明します。

サンプル・コードは、次の OTN の Web サイトで提供されています。

`http://otn.oracle.com/tech/java/oc4j/demos/1012/index.html`

Web サービスの `demo.zip` ファイルを解凍すると、ドキュメント・スタイルの Web サービスのサンプルが、UNIX の場合は `/webservices/demo/basic/java_doc_services` の下、Windows の場合は `¥webservices¥demo¥basic¥java_doc_services` の下の `stateless` ディレクトリと `stateful` ディレクトリに格納されます。

Oracle Application Server には、Web サービス実装用に記述する Java クラスにアクセスするサーブレットが用意されています。このサーブレットでは、Web サービス・クライアントにより生成されたメッセージが処理され、ドキュメント・スタイルの Web サービスを実装する Java メソッドを実行するためにディスパッチされます。Web サービスのデプロイ後は、クライアントがサービス・リクエストを発行（サービスを使用）すると、Oracle Application Server Web Services ランタイムでは、自動的に生成される Web サービス・サーブレットを使用して、ドキュメント・スタイルの Web サービスのサポート用に実装したメソッドが起動されます。

**関連項目：**

- 第 3 章「Java クラス Web サービスの開発とデプロイ」
- 第 4 章「EJB Web サービスの開発とデプロイ」
- 第 7 章「JMS Web サービスの開発とデプロイ」
- 第 8 章「Web サービスを使用するクライアントの作成」

# ドキュメント・スタイルの Web サービスの記述

ドキュメント・スタイルの Java Web サービスを記述するには、サポートされるメソッド・シグネチャを使用して、1 つ以上のメソッドを含む Java クラスを作成する必要があります。この Java クラスには、受信メッセージを処理するメソッド、または送信メッセージを戻すメソッドが含まれます。

この項の内容は、次のとおりです。

- ドキュメント・スタイルの Web サービス用にサポートされるメソッド・シグネチャ
- ステートレス・ドキュメント・スタイルの Web サービスとステートフル・ドキュメント・スタイルの Web サービスの記述
- ドキュメント・スタイルの Web サービス用のクラスとインタフェースの記述

# ドキュメント・スタイルの Web サービス用にサポートされるメソッド・シグネチャ

表 6-1 に、ドキュメント・スタイルの Web サービス用にサポートされるメソッド・シグネチャを示します。Oracle Application Server Web Services ランタイムの検証機能では、表 6-1 に示すメソッド・シグネチャに準拠していないドキュメント・スタイルの Web サービスは拒否されます。

表 6-1 のメソッド・シグネチャに示されている `Element` 入力パラメータと `Element` 戻り値は、W3C（`org.w3c.dom.Element`）により指定された Document Object Model（DOM）に準拠する必要があります。

**表 6-1 ドキュメント・スタイルの Java Web サービス用にサポートされるメソッド・シグネチャ**

| メソッド・シグネチャ | 説明 |
|---|---|
| `public Element` *op_Name*`(Element` *e_name*`)` | メソッド *op_Name* は、Java メソッドとして実装されるドキュメント・スタイルの Web サービスの操作であり、入力パラメータとして `Element` *e_name* を使用し、`Element` を戻します。 |
| `public Element` *get_Name*`()` | メソッド *get_Name* は、Java メソッドとして実装されるドキュメント・スタイルの Web サービスの操作であり、入力パラメータを使用せず、`Element` を戻します。 |
| `public void` *set_Name*`(Element` *e_name*`)` | メソッド *set_Name* は、Java メソッドとして実装されるドキュメント・スタイルの Web サービスの操作であり、入力パラメータとして `Element` *e_name* を使用します。戻り値はありません。 |

## ドキュメント・スタイルの Web サービスに関する NULL 値の受渡し

入力 `Element`、またはドキュメント・スタイルの Web サービスが戻す `Element` として、`null` を渡すことができます。

## Element の配列

Oracle Application Server Web Services では、`Element[]`（`org.w3c.dom.Element` の配列）はサポートされません。

**関連項目：**

- 6-7 ページの「ドキュメント・スタイルの Web サービスに関するメッセージの処理」
- W3C の Document Object Model（DOM）の詳細は、`http://www.w3.org/DOM/` を参照してください。

## ステートレス・ドキュメント・スタイルの Web サービスとステートフル・ドキュメント・スタイルの Web サービスの記述

Oracle Application Server Web Services では、Web サービスとして動作するドキュメント・スタイルの Java クラスのステートフル実装とステートレス実装がサポートされます。ステートフル Java 実装の場合、Oracle Application Server Web Services では、単一の Java インスタンスで個々のクライアントからの Web サービス・リクエストを処理できます。

ステートレス Java 実装の場合、Oracle Application Server 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。

---

**注意：** Web サービス開発者は、ステートフル Web サービスとステートレス Web サービスのどちらを実装するかを、設計段階で決定する必要があります。この 2 種類の Web サービスは、パッケージング時の処理方法が多少異なります。この違いについては、6-7 ページの「ドキュメント・スタイルの Web サービスの準備」を参照してください。

---

**注意：** ステートフル Java 実装クラスをステートレス・ドキュメント・スタイルの Web サービスとしてデプロイすると、予測できない結果を生じる可能性があります。

---

## ドキュメント・スタイルの Web サービス用のクラスとインタフェースの記述

ドキュメント・スタイルの Java Web サービスの開発手順は、次のとおりです。

- ドキュメント・スタイルの Web サービスのメソッドの定義
- 明示的なメソッド公開用のインタフェースの定義
- ドキュメント・スタイルの Web サービスに関するメッセージの処理

### ドキュメント・スタイルの Web サービスのメソッドの定義

ドキュメント・スタイルの Web サービスとしてデプロイするメソッドを含む Java クラスを記述または提供して、ドキュメント・スタイルの Web サービスを作成します。`stateful` および `stateless` サンプル・ディレクトリには、ステートレス・ドキュメント・スタイルの Web サービスとステートフル・ドキュメント・スタイルの Web サービスのサンプルが含まれています。`src` ディレクトリ内で、ファイル `StatefulDocImpl.java` にはステートフル Java サービスのサンプル実装、`StatelessDocImpl.java` にはステートレス・ドキュメント・スタイルの Web サービスの実装が含まれています。これらの例では、インタフェース・クラスが使用されます。ドキュメント・スタイルの Web サービスを実装する場合、インタフェース・クラスの使用はオプションです。

ドキュメント・スタイルの Web サービスを実装する Java クラスには、次の制限事項があります。

- Java クラスでは、表 6-1 に示すメソッド・シグネチャに準拠したパブリック・メソッドを定義する必要があります。インタフェースを使用する場合、メソッド・シグネチャの制限に準拠する必要があるのは、インタフェース内で指定されているパブリック・メソッドのみです。インタフェースを組み込まない場合は、クラス内のすべてのパブリック・メソッドが、表 6-1 に示すメソッド・シグネチャの制限に準拠する必要があります。
- Java クラス実装には、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ります。

ドキュメント・スタイル Java クラス・ベースの 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。少なくとも、サービスでは、受信メッセージ（`Element`）の処理または送信メッセージ（`Element`）の生成のために、なんらかのアクションが実行されます。

StatelessDoc Web Service サンプルは、StatelessDocImpl、パブリック・クラスおよびインタフェース StatelessDoc を使用して実装されます。StatelessDocImpl クラスでは、2 つのパブリック・メソッドが定義されます。一方は Web サービスが実行されるサーバー上で受信メッセージを表示する displayElement() で、他方は受信メッセージを使用して変換後のメッセージをクライアントに戻す processElement() です。プライベート・メソッド applyXSLtoXML() は、converter.xsl ファイル内の指定に従って受信メッセージを変換するヘルパー・メソッドです。

例 6-1 に、StatelessDocImpl クラス用のメソッド・シグネチャを示します（StatelessDocImpl のソース・コードの詳細は、src ディレクトリを参照してください）。

**例 6-1 ステートレス・ドキュメント・スタイルの Web サービスの Java メソッドの定義**

```
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import java.io.*;

public class StatelessDocImpl implements StatelessDoc

{
    public StatelessDocImpl()
    {  }

  // Display the Element that was sent
  public void displayElement(Element e)
  {  }

 //method to process the input xml doc
  public Element processElement(Element e)
  {  }

 /**
 * This Method Transforms an XML Document into another using the provided
 * Style Sheet: converter.xsl.  Note : This Method makes use of XSL
 *  Transformation capabilities of Oracle XML Parser Version 2.0
 **/
 private Element applyXSLtoXML(Element e)
  throws Exception
  {}
```

StatefulDoc Web Service サンプルは、StatefulDocImpl、パブリック・クラスおよびインタフェース StatefulDoc を使用して実装されます。StatefulDocImpl クラスでは、2 つのパブリック・メソッドが定義されます。一方は顧客情報の状態を初期化する startShopping() で、他方は顧客情報の状態を変更して更新済の情報をクライアントに戻す makePurchase() です。プライベート・メソッド processElement() は、購買を表す顧客の XML 要素を処理して更新済の XML 要素を戻すヘルパー・メソッドです。

例 6-2 に、StatefulDoc クラス用のメソッド・シグネチャを示します（StatefulDocImpl のソース・コードの詳細は、src ディレクトリを参照してください）。

**例 6-2 ステートフル・ドキュメント・スタイルの Web サービスの Java メソッドの定義**

```
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;

public class StatefulDocImpl implements StatefulDoc
  private Element e ;
  public void startShopping(Element e)
  {
  }
  public Element makePurchase()
  {
  }
  private void  processElement(Element e) {
```

```
}
```

## 明示的なメソッド公開用のインタフェースの定義

Oracle Application Server Web Services では、パブリック・インタフェースを提供し、ドキュメント・スタイルの Web サービスとして公開するメソッドを制限できます。Web サービスで公開するメソッドを制限するには、公開するメソッドのシグネチャをリストするパブリック・インタフェースを組み込みます。例 6-3 にクラス `StatelessDocImpl` 内のメソッド用のインタフェース、例 6-4 にクラス `StatelefulDocImpl` 内のメソッド用のインタフェースを示します。

ドキュメント・スタイルの Web サービスにインタフェースを組み込む場合、表 6-1 に示すメソッド・シグネチャの制限に準拠する必要があるのは、そのインタフェースに指定されているパブリック・メソッドのみです。インタフェースを組み込まない場合は、クラス内のすべてのパブリック・メソッドが、メソッド・シグネチャの制限に準拠する必要があります。例 6-3 に示す `StatelessDoc` など、インタフェースを使用すると、Java クラスが準備され、ドキュメント・スタイルの Web サービスとしてデプロイされた時点で、指定したメソッド・シグネチャを持つメソッドのみが公開されます。

ドキュメント・スタイルの Web サービス・インタフェースの用途は、次のとおりです。

1. メソッドの公開を、あるクラスのパブリック・メソッドのサブセットに限定するため。
2. 公開されるメソッドのセットを拡張し、あるクラスのスーパークラスのメソッドを含めるため。
3. メソッドの公開を、あるクラスのパブリック・メソッドのサブセットに限定するため。この場合、サブセットには、サポート対象のメソッド・シグネチャを使用するメソッドのみが含まれます。表 6-1 に、ドキュメント・スタイルの Web サービスを実装する Java メソッドについてサポートされているシグネチャを示します。

### 例 6-3 パブリック・インタフェースを使用したステートレス Java サービスの公開

```
import  org.w3c.dom.*;

public interface StatelessDoc
{
     //method to display the element
     public void displayElement(Element e) ;

     //method to process the input xml doc
     public Element processElement(Element e) ;
}
```

### 例 6-4 パブリック・インタフェースを使用したステートフル Java サービスの公開

```
import org.w3c.dom.Element;

// Interface that implements getElement and setElement
public interface StatefulDoc {

  // Set the Element
  public void startShopping(Element e);

  // Retrieve the element that was set
  public Element makePurchase();
}
```

### ドキュメント・スタイルの Web サービスに関するメッセージの処理

Web サービス開発者は、ドキュメント・スタイルの Web サービスに関連するメッセージに対する処理を決定する必要があります。

ドキュメント・スタイルの Web サービスに関連するメッセージは、そのサービスに関連する `Element` パラメータまたは `Element` 戻り値に指定されます。ドキュメント・スタイルの Web サービス開発者は、メッセージを処理または生成する必要があります。ドキュメント・スタイルの Web サービスのメッセージに関する制限事項は、`Element` が W3C（`org.w3c.dom.Element`）により指定されている Document Object Model（DOM）に準拠する必要があることのみです。

ドキュメント・スタイルの Web サービス実装、またはサービスを使用するクライアントでは、`null` 値のサポートが必要になることがあります。これは、入力 `Element` または戻り値 `Element` として `null` が使用される場合があるためです。

たとえば、ドキュメント・スタイルの Web サービス実装では、次が有効です。

```
Element get_op () {
    return null;
}
```

## ドキュメント・スタイルの Web サービスの準備

この項では、Oracle Application Server Web Services のツール `WebServicesAssembler` を使用して、Java クラスとして実装されるステートレスおよびステートフル・ドキュメント・スタイルの Web サービス用に、J2EE の .ear ファイルを準備する方法について説明します。

ドキュメント・スタイルの Web サービスを実装する Java クラスをデプロイするには、Oracle Application Server Web Services サーブレットと Java 実装を提供する Java クラスのデプロイメント・ディスクリプタを含む J2EE の .ear ファイルをアセンブルする必要があります。Java クラスを使用して実装する Web サービスには、Oracle Application Server Containers for J2EE（OC4J）で動作する Web サービス・サーブレットの構成情報を提供する .war ファイルが含まれています。この項では、`WebServicesAssembler` で使用する構成ファイルの作成手順について説明します。

この項の内容は、次のとおりです。

- ドキュメント・スタイルの Web サービスをアセンブルするための構成ファイルの作成
- ドキュメント・スタイルの Web サービスを使用した WebServicesAssembler の実行

## ドキュメント・スタイルの Web サービスをアセンブルするための構成ファイルの作成

Oracle Application Server Web Services のアセンブリ・ツール `WebServicesAssembler` を使用すると、Oracle Application Server Web Services をアセンブルできます。この項では、ドキュメント・スタイルの Web サービスのアセンブルに使用する構成ファイルの作成方法について説明します。Web サービスのアセンブリ・ツールでは、ドキュメント・スタイルの Web サービスを記述する XML 構成ファイルが使用されます。`WebServicesAssembler` では、この構成ファイルを使用して、Oracle Application Server Web Services でデプロイできる J2EE の .ear ファイルが生成されます。

次のタグを追加して、`WebServicesAssembler` 構成ファイルを作成します。

- Web サービスのトップレベル・タグの追加
- ドキュメント・メッセージ・スタイルを指定した Java サービス・タグの追加
- WSDL およびクライアント・サイド・プロキシ生成タグの追加

## Web サービスのトップレベル・タグの追加

表 6-2 に、WebServicesAssembler 構成ファイルのトップレベル・タグを示します。これらのタグを追加して、ドキュメント・スタイルの Web サービスを記述するトップレベルの情報を提供します。

ステートレスのサンプル構成ファイルの詳細は例 6-5、ステートフルのサンプル構成ファイルの詳細は例 6-6 を参照してください。サンプル config.xml ファイルは、java_doc_services demo ディレクトリの stateless および stateful ディレクトリにあります。

**表 6-2 トップレベルの WebServicesAssembler 構成タグ**

| タグ | 説明 |
|---|---|
| <context><br>*context*<br></context> | Web サービスのコンテキスト・ルートを指定します。<br>このタグは必須です。 |
| <datasource-JNDI-name><br>*name*<br></datasource-JNDI-name> | Web サービスに関連するデータソースを指定します。 |
| <description><br>*description*<br></description> | Web サービスの簡単な説明を提供します。<br>このタグはオプションです。 |
| <destination-path><br>*dest_path*<br></destination-path> | 生成される J2EE の .ear ファイルの出力名を指定します。*dest_path* には、出力ファイルのフルパスを指定します。<br>このタグは必須です。 |
| <display-name><br>*disp_name*<br></display-name> | Web サービスの表示名を指定します。<br>このタグはオプションです。 |
| <option name=source-path"><br>*path*<br><option> | 指定したファイルを出力 .ear ファイルに含めます。このオプションを使用して Java リソースを指定するか、J2EE の出力 .ear ファイルのソース・ファイルとして使用される既存の .war、.ear または ejb-jar ファイルの名前を指定します。<br>.war ファイルが入力として提供されると、オプションの contextroot によって、.war ファイルの root-context が指定されます。<br>*path1* には、.war の context-root を指定します。<br>*path2* には、組み込むファイルへのパスを指定します。<br>次に例を示します。<br><option name="source-path" contextroot="/test">/myTestArea/ws/src/statefull.war</option><br>このタグはオプションです。 |
| <stateless-java-service><br>*sub-tags*<br></stateless-java-service> | このタグを使用して、ステートレス・サービスを定義するドキュメント・スタイルの Web サービスを追加します。有効な *sub-tags* の詳細は、表 6-3 を参照してください。 |
| <stateful-java-service><br>*sub-tags*<br></stateful-java-service> | このタグを使用して、ステートフル・サービスを定義するドキュメント・スタイルの Web サービスを追加します。有効な *sub-tags* の詳細は、表 6-3 を参照してください。 |
| <temporary-directory><br>*temp_dir*<br></temporary-directory> | アセンブラで一時ファイルを格納できるディレクトリを指定します。<br>このタグはオプションです。 |

## ドキュメント・メッセージ・スタイルを指定した Java サービス・タグの追加

ドキュメント・スタイルの Web サービス開発者は、サービスをステートフルにするかステートレスにするかを決定します。構成ファイルには、サービスのタイプに応じて異なるタグが含まれます。この項では、次のように両方の場合のタグについて説明します。

- ステートフル・ドキュメント・スタイルの Java サービス・タグの追加
- ステートレス・ドキュメント・スタイルの Java サービス・タグの追加

**表 6-3 Java サービスの WebServicesAssembler 構成タグ - ドキュメント・スタイル**

| タグ | 説明 |
|---|---|
| <class-name><br>*value*<br></class-name> | ドキュメント・スタイルの Web サービスの定義には、少なくとも 1 つは <class-name> タグが必要です。*value* には、ドキュメント・スタイルの Web サービス実装を提供する Java クラスの名前を指定します。<br>このタグは必須です。 |
| <interface-name><br>*interface*<br></interface-name> | ドキュメント・スタイルの Web サービスの構成ファイルでは、オプションの <interface-name> タグがサポートされます。対応する *interface* 値には、ドキュメント・スタイルの Web サービスに含めるメソッドをリストする Java インタフェースの名前を指定します。<br>このタグはオプションです。 |
| <java-resource><br>*resource*<br></java-resource> | このタグでは、Java リソースの追加がサポートされます。ドキュメント・スタイルの Web サービスに組み込む Java リソースの位置を指定します。<br>複数の <java-resource> タグを使用して、複数の Java リソースを組み込むことができます。<br>このタグはオプションです。 |
| <message-style><br>*doc*<br></message-style> | ドキュメント・スタイルの Web サービスを定義するには、<message-style> タグを組み込んで値 doc を指定する必要があります。<br>有効な値：doc、rpc<br>ドキュメント・スタイルの Web サービスの場合、このタグは必須です。<br>デフォルト値：rpc（<message-style> タグを指定しない場合） |
| <scope><br>*value*<br></scope> | <scope> タグは、ステートフル・サービスにのみ適用されます。このタグは、<stateful-java-service> タグ内でのみ使用します。<br>このタグはオプションです。<br>有効な値：application、session<br>デフォルト値：session |
| <session-timeout><br>*value*<br></session-timeout> | このオプションのパラメータは、ステートフル・サービスにのみ適用されます。このタグは、<stateful-java-service> タグ内でのみ使用します。<br>*value* には、セッションのタイムアウト秒数を定義する整数を指定します。セッション・タイムアウト指定のないステートフル Java セッションの場合、デフォルトのセッション・タイムアウト値は 60 秒です。<br>このタグはオプションです。 |
| <uri><br>*URI*<br></uri> | このタグでは、ドキュメント・スタイルの Web サービスを実装するサーブレットのサーブレット・マッピング・パターンを指定します。*URI* として指定したパスは、ドキュメント・スタイルの Web サービスの位置を指定する <context> に追加されます。<br>このタグはオプションです。 |

### ステートフル・ドキュメント・スタイルの Java サービス・タグの追加

表 6-3 に、WebServicesAssembler 構成ファイルの <stateful-java-service> のタグを示します。これらのタグは、ステートフル・ドキュメント・スタイルの Web サービス用の構成ファイルを作成するときに使用します。

ステートフル・ドキュメント・スタイルの Web サービスのタグなど、config.xml ファイルの詳細は、例 6-5 を参照してください。

### ステートレス・ドキュメント・スタイルの Java サービス・タグの追加

表 6-3 に、ステートレス・ドキュメント・スタイルの Web サービスの作成時に使用する WebServicesAssembler 構成ファイルの <stateless-java-service> のタグを示します。<stateless-java-service> タグは、構成ファイルの <web-service> タグ内に挿入します。このタグを追加して、ステートレス・ドキュメント・スタイルの Web サービスの J2EE の .ear ファイル生成に必要な情報を提供します。

ステートレス・ドキュメント・スタイルの Web サービスのタグなど、config.xml ファイルの詳細は、例 6-6 を参照してください。

---

**注意：** ステートフル Java 実装クラスをステートレス・ドキュメント・スタイルの Web サービスとしてデプロイすると、予測できない結果を生じる可能性があります。

---

## WSDL およびクライアント・サイド・プロキシ生成タグの追加

WebServicesAssembler 構成ファイルでは、Web サービス開発者が Web サービス記述用の WSDL ファイルとクライアント・サイド・プロキシ・ファイルを生成できるように、<wsdl-gen> および <proxy-gen> タグがサポートされます。これらのタグを追加して、WSDL ファイルとクライアント・サイド・プロキシを生成するかどうかを制御できます。また、ドキュメント・スタイルの Web サービスの .ear ファイルとともに、WSDL ファイルをアセンブルするように指定できます。クライアント・サイド開発者は、デプロイされた Web サービスから WSDL ファイルを取得し、それを使用してアプリケーションを作成できます。

**関連項目：** 9-4 ページの「WSDL ファイルとクライアント・サイド・プロキシの生成」

**例 6-5 ドキュメント・スタイルの Web サービス用のサンプル・ステートフル Java の WebServicesAssembler 構成ファイル**

```
<web-service>
    <display-name>Stateful Java Document Web Service</display-name>
    <description>Stateful Java Document Web Service Example</description>
    <!-- Specifies the resulting web service archive will be stored in ./docws.ear -->
    <destination-path>./docws.ear</destination-path>
    <!-- Specifies the temporary directory that web service assembly tool can create temporary files. -->
    <temporary-directory>./temp</temporary-directory>
    <!-- Specifies the web service will be accessed in the servlet context named "/docws". -->
    <context>/statefuldocws</context>

    <!-- Specifies the web service will be stateful -->

    <stateful-java-service>
        <interface-name>StatefulDoc</interface-name>
        <class-name>StatefulDocImpl</class-name>
        <!-- Specifies the web service will be accessed in the uri named "/docService" within the servlet
        context. -->
        <uri>/docservice</uri>
        <!-- Specifies the location of Java class files ./classes -->
        <java-resource>./classes</java-resource>
        <!-- Specifies that it uses document style SOAP messaging -->
        <message-style>doc</message-style>
   </stateful-java-service>
```

```
    <!-- generate the wsdl -->
    <wsdl-gen>
   <wsdl-dir>wsdl</wsdl-dir>
    <!-- over-write a pregenerated wsdl , turn it 'false' to use the pregenerated wsdl-->
   <option name="force">true</option>
   <option name="httpServerURL">http://localhost:8888</option>
    </wsdl-gen>

    <!-- generate the proxy -->

    <proxy-gen>
   <proxy-dir>proxy</proxy-dir>
   <option name="include-source">true</option>
    </proxy-gen>
</web-service>
```

**例 6-6 ドキュメント・スタイルの Web サービス用のサンプル・ステートレス Java の WebServicesAssembler 構成ファイル**

```
<web-service>
    <display-name>Stateless Java Document Web Service</display-name>
    <description>Stateless Java Document Web Service Example</description>
    <!-- Specifies the resulting web service archive will be stored in ./statelessdocws.ear -->
    <destination-path>./statelessdocws.ear</destination-path>
    <!-- Specifies the temporary directory that web service assembly tool can create temporary files. -->
    <temporary-directory>./temp</temporary-directory>
    <!-- Specifies the web service will be accessed in the servlet context named "/statelessdocws". -->
    <context>/statelessdocws</context>
    <!-- to package the  stylesheet to format input xml -->
    <option name="source-path">converter.xsl</option>

    <!-- Specifies the web service will be stateless -->

    <stateless-java-service>
        <interface-name>StatelessDoc</interface-name>
        <class-name>StatelessDocImpl</class-name>
        <!-- Specifies the web service will be accessed in the uri named "/docService" within the servlet
        context. -->
        <uri>/docservice</uri>
        <!-- Specifies the location of Java class files ./classes -->
        <java-resource>./classes</java-resource>
        <!-- Specifies that it uses document style SOAP messaging -->
        <message-style>doc</message-style>
   </stateless-java-service>

    <!-- generate the wsdl -->
    <wsdl-gen>
   <wsdl-dir>wsdl</wsdl-dir>
    <!-- over-write a pregenerated wsdl , turn it 'false' to use the pregenerated wsdl-->
   <option name="force">true</option>
   <option name="httpServerURL">http://localhost:8888</option>
    </wsdl-gen>

    <!-- generate the proxy -->
    <proxy-gen>
   <proxy-dir>proxy</proxy-dir>
   <option name="include-source">true</option>
    </proxy-gen>

</web-service>
```

## ドキュメント・スタイルの Web サービスを使用した WebServicesAssembler の実行

`WebServicesAssembler` 構成ファイルの作成後に、ドキュメント・スタイルの Web サービス用の J2EE の .ear ファイルを生成できます。この J2EE の EAR ファイルには、生成されるファイル `web.xml` など、ドキュメント・スタイルの Web サービス・サーブレット構成情報と、実装クラスが含まれます。

Oracle Application Server Web Services のアセンブリ・ツール `WebServicesAssembler` を次のように実行します。

```
java -jar WebServicesAssembler.jar -config my_service_config
```

ここで、*my_service_config* は、`<stateless-java-service>` または `<stateful-java-service>` タグを含む構成ファイルです。

**関連項目：**

- 6-7 ページの「ドキュメント・スタイルの Web サービスをアセンブルするための構成ファイルの作成」
- 9-2 ページの「Web サービス・アセンブリ・ツールの実行」

# ドキュメント・スタイルの Web サービスのデプロイ

Java クラスと Web サービス・サーブレットのデプロイメント・ディスクリプタを含む .ear ファイルの作成後に、.ear ファイルに格納されている標準的な J2EE アプリケーションの場合と同様に、Web サービスを（OC4J で動作するように）デプロイできます。

**関連資料：**『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』

# 7

# JMS Web サービスの開発とデプロイ

この章では、JMS キューや JMS トピックなど、JMS 宛先を Web サービスとして公開する Oracle Application Server Web Services を構成、デプロイおよび作成する手順について説明します。また、受信 JMS メッセージの使用と送信 JMS メッセージの生成を行う、バックエンド JMS メッセージ・プロセッサの記述方法についても説明します。

Oracle Application Server Web Services では、JMS Web サービスとともに非同期メッセージ機能がサポートされます。

この章の内容は、次のとおりです。

# JMS Web サービスの概要

この項の内容は、次のとおりです。



## JMS Web サービスの使用

JMS Web サービスのサンプル・コードは、次の OTN の Web サイトで提供されています。

`http://otn.oracle.com/tech/java/oc4j/demos/1012/index.html`

Web サービスの `demo.zip` ファイルを解凍すると、サンプルが、`/webservices/demo/basic/jms_service`（UNIX）の下、または `¥webservices¥demo¥basic¥jms_service` の下の `demo1` ディレクトリと `demo2` ディレクトリに格納されます。

JMS Web サービスの例は、OC4J/JMS と Oracle JMS の両方を示しています。サンプルのうち、`demo1` では OC4J/JMS、`demo2` では Oracle JMS を使用しています。

JMS Web サービスを使用する場合、Oracle Application Server には `send` 操作と `receive` 操作という 2 つのメッセージ操作をサポートするサーブレットが用意されています。この 2 つの操作を使用する場合に、宛先が JMS キューであれば、`send` はエンキュー、`receive` はデキューを意味します。宛先がトピックであれば、`send` はパブリッシュ、`receive` はサブスクライブを意味します。個々の JMS 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 Web サービスでは、その構成と、それを使用するクライアント・サイド・プログラムで指定された操作に基づいて、JMS 宛先への受信メッセージと送信メッセージの処理方法が決定されます。JMS Web サービス・クライアントから提供される操作が無効な場合は、Oracle Application Server Web Services ランタイムの検証機能により例外がスローされます。たとえば、デプロイ操作が `send` でリクエストが `receive` の場合は、例外がスローされます。

JMS Web サービスに関連するクライアント・サイド・メッセージは、W3C（`org.w3c.dom.Element`）により指定されている Document Object Model（DOM）に準拠した XML 文書です。`send` 操作の場合、クライアント・サイド開発者は JMS Web サービスに適切な書式のメッセージを配信する必要があります。同様に、receive 操作の場合、クライアントでは JMS Web サービスから受信するメッセージを処理する必要があります。

> **関連項目：** JMS の詳細は、`http://java.sun.com/products/jms/` を参照してください。

## JMS Web サービスのバックエンド・メッセージ処理

JMS Web サービスは Web サービスを定義する構成情報からなっており、それに加えてサーバー側開発者は JMS Web サービス・クライアントから送信されるメッセージをコンシュームするコード、またはクライアントが受信するメッセージを生成するコードを提供します。

この項では、JMS Web サービスに関連する JMS メッセージを処理するためのアーキテクチャについて説明します。この項の内容は、次のとおりです。

## メッセージ処理への MDB の使用

JMS Web サービスは JMS 宛先との間でメッセージを送受信し、バックエンドで MDB を使用してメッセージを生成および使用できます。たとえば、図 7-1 に示す MDB ベースの JMS Web サービスは、JMS Web サービス・クライアントからみると、メッセージの `send` 操作と `receive` 操作の両方を処理します。

**図 7-1 MDB ベースの JMS Web サービス**

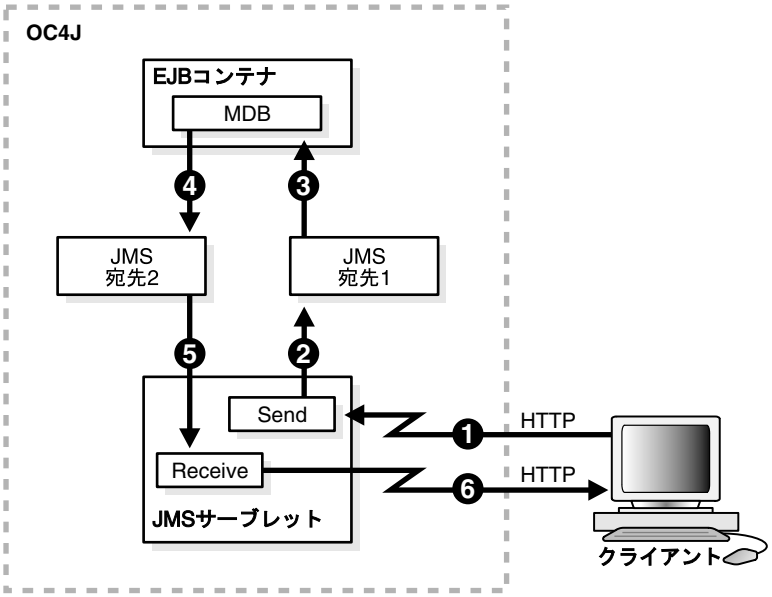



図 7-1 に示す MDB は、JMS の宛先をリスニングするように構成されています。MDB ベースの JMS Web サービスの動作ステップは、次のとおりです。

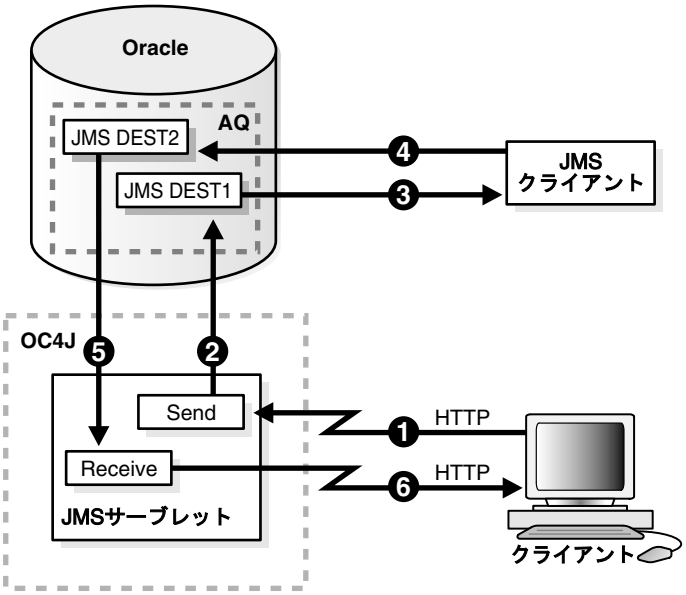
1. JMS Web サービス・クライアントが、JMS Web サービス上で send 操作を実行してメッセージを送信します。
2. JMS Web サービスが受信メッセージを処理し、それを JMS 宛先である JMS 宛先 1 にダイレクトします。
3. EJB コンテナが、JMS 宛先 1 をリスニングしている MDB を起動します。
4. メッセージの処理後に、MDB は JMS 宛先 2 で新規メッセージを生成します。メッセージの生成と使用には、1 つ以上の MDB が関係する場合があります。たとえば、ある MDB が JMS 宛先 1 をリスニングするのみでなく、JMS 宛先 2 へ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）JMS Web サービス・クライアントが、JMS Web サービス上で `receive` 操作を実行してメッセージを受信します。JMS Web サービスが JMS 宛先からのメッセージをコンシュームし、処理して、送信メッセージをクライアントに渡します。

## メッセージ処理への JMS クライアントの使用

メッセージ処理に JMS クライアントを使用する場合、JMS Web サービスはバックエンドで JMS コードをアセンブル、デプロイまたは実行しません。JMS Web サービスに関連する JMS メッセージの生成とコンシュームは、JMS Web サービスの外部でスタンドアロン JMS クライアントとして動作する別個の JMS プログラムが受け持ちます。

たとえば、図 7-2 に示す JMS Web サービスは、メッセージ処理にサーバー・サイド JMS クライアントを使用しています。

**図 7-2 JMS クライアント・ベースの JMS Web サービス**




JMS Web サービスには、メッセージの処理と JMS 宛先の使用をサポートする構成情報のみが含まれています。JMS クライアント・ベースの JMS Web サービスの動作ステップは、次のとおりです。

1. JMS Web サービス・クライアントが、JMS Web サービス上で send 操作を実行してメッセージを送信します。
2. JMS Web サービスが受信メッセージを処理し、それを JMS DEST1 にダイレクトします。
3. JMS クライアントが、JMS DEST 1 で受信メッセージを処理します。この受信メッセージは、メッセージ・リスナーを使用するか他の方法で識別できます。
4. 受信メッセージの処理後に、JMS クライアントは JMS DEST 2 で新規メッセージを生成できます。JMS DEST 2 でメッセージを生成するのは、他の JMS クライアントでも同じ JMS クライアントでもかまいません。
5. （矢印 5 および 6）JMS Web サービス・クライアントが、JMS Web サービス上で `receive` 操作を実行してメッセージを受信します。JMS Web サービスが JMS 宛先からの送信メッセージをコンシュームして、そのメッセージをクライアントに渡します。

# JMS Web サービスの記述とメッセージの処理

JMS Web サービスを記述するには、サーバー・サイド開発者が次の 2 つの作業を実行する必要があります。

1. JMS Web サービス用のバックエンド・メッセージ処理プログラムの作成
2. JMS Web サービスの準備と構成

この項の内容は、次のとおりです。

- バックエンド・メッセージ処理への MDB の使用
- バックエンド・メッセージ処理への JMS スタンドアロン・プログラムの使用
- メッセージ処理と応答メッセージ

**関連項目：**

- 7-9 ページの「JMS Web サービスの準備と構成」
- 第 4 章「EJB Web サービスの開発とデプロイ」

## バックエンド・メッセージ処理への MDB の使用

JMS Web サービスでメッセージの生成または使用に MDB を使用する場合は、MDB を JMS Web サービスとともにアセンブルする必要があります。この場合、MDB は、JMS Web サービスとしてデプロイされる J2EE の .ear ファイルの一部としてパッケージされます。

JMS Web サービスとともに MDB を使用する場合、サーバー側開発者は次の手順を実行する必要があります。

- 受信メッセージを処理する MDB の開発
- 送信メッセージを生成する MDB の開発
- MDB 用 EJB.jar ファイルのコンパイルと準備
- JMS Web サービスと MDB のアセンブル
- サーバー・サイド・リソース参照の定義

---

**注意：** 1 つの JMS Web サービスで受信メッセージの処理、送信メッセージの生成またはその両方を実行できます。

---

### 受信メッセージを処理する MDB の開発

JMS Web サービスの `send` 操作により生成された受信メッセージを処理する MDB には、次の特性を持つ `onMessage()` メソッドを組み込む必要があります。

- `onMessage()` メソッドは、`final` または `static` ではなく `public` として宣言する必要があります。
- `onMessage()` メソッドの戻り型は `void` にする必要があります。
- `onMessage()` メソッドの引数の 1 つは `javax.jms.Message` 型にする必要があります。JMS Web サービスでサポートされるのは `ObjectMessage` 型のメッセージのみなので、MDB 開発者は受信する JMS Web サービス・メッセージを `ObjectMessage` にキャストする必要があります。
- 受信 JMS メッセージで `getObject()` メソッドを使用して `Element` 型にキャストし、メッセージからメッセージ・ペイロードを使用できます。

例 7-1 に、受信 JMS メッセージを処理する MDB のメソッドを示します。コードの詳細は、demo1 ディレクトリにある MessageBean.java も参照してください。

**例 7-1 JMS Web サービス用の受信用 onMessage() メソッドのサンプル**

```
public void onMessage(Message inMessage) {
      ObjectMessage msg = null;
  Element       e;
      try {
    // Message should be of type objectMessage
    if (inMessage instanceof ObjectMessage) {
      // retrieve the object
      msg = (ObjectMessage) inMessage;
      e = (Element)msg.getObject();
      processElement(e);
      this.send2Queue(e);
    } else {
      System.out.println("MessageBean::onMessage() => Message of wrong type: "
      + inMessage.getClass().getName());
    }
  } catch (JMSException ex) {
    ex.printStackTrace();
    mdc.setRollbackOnly();
  } catch (Throwable te) {
    te.printStackTrace();
  }
}
```

## 送信メッセージを生成する MDB の開発

JMS Web サービスの receive 操作で使用される送信メッセージを生成する MDB には、次の特性を指定して、JMS 宛先でメッセージを生成するコードを組み込む必要があります。

- JMS 宛先に置かれるメッセージは、javax.jms.Message.ObjectMessage 型にする必要があります。
- 送信 JMS メッセージ上で setObject() メソッドを使用し、java.io.Serializable 型にキャストして、メッセージのペイロードを設定します。

例 7-2 に、適切な型の送信メッセージを作成するコード部分を示します。この例のコードの詳細は、demo2 ディレクトリにある MessageBean2.java を参照してください。

**例 7-2 JMS Web サービスの送信メッセージのサンプル**

```
// Create an Object Message
message = queueSession.createObjectMessage();
// Stuff the result into the ObjectMessage
((ObjectMessage)message).setObject ((java.io.Serializable)ee);
// Send the Message
queueSender.send(message);
```

### MDB 用 EJB.jar ファイルのコンパイルと準備

MDB クラスのコンパイル後に、MDB と必要なデプロイ情報を含む EJB の .jar ファイルを作成します。

### JMS Web サービスと MDB のアセンブル

`WebServicesAssembler` ツールと、EJB の .jar を指定するトップレベルのタグ `<option name=source-path">` および JMS Web サービス構成を定義するタグ `<jms-doc-service>` を含む構成ファイルを使用して、MDB の EJB.jar ファイルと JMS Web サービスの .ear ファイルをアセンブルします。

**関連項目：**

- 7-9 ページの「JMS Web サービスの準備と構成」
- 7-14 ページの「JMS Web サービスのデプロイ」

### サーバー・サイド・リソース参照の定義

JMS Web サービスで使用する JMS 宛先に関連したリソース参照を定義します。

- MDB で OC4J/JMS が使用される場合は、OC4J の `jms.xml` 構成ファイルにリソース参照を定義します。
- MDB で Oracle JMS が使用される場合は、Oracle JMS 宛先へのアクセスをサポートする sql ファイルを実行します。

**関連資料：**『Oracle Streams アドバンスト・キューイング・ユーザーズ・ガイドおよびリファレンス』の第 3 章「AQ プログラム環境」を参照してください。

## バックエンド・メッセージ処理への JMS スタンドアロン・プログラムの使用

JMS Web サービスにバックエンドで JMS スタンドアロン・プログラムを使用する場合、サーバー・サイド開発者は次の手順を実行する必要があります。

1. JMS 宛先の定義、受信メッセージの取扱いと処理および送信メッセージの生成を行う JMS クライアントを開発します。JMS クライアントでは、MDB をトリガーする JMS 宛先を使用する処理を実行することもできます。
2. `WebServicesAssembler` ツールと、トップレベルのタグ `<jms-doc-service>` を含む構成ファイルを使用して、JMS Web サービスの .ear ファイルをアセンブルします。
3. OC4J/JMS の `jms.xml` 構成ファイル内で、JMS 宛先に関連するリソース参照を定義します。JMS 宛先が Oracle JMS に定義されている場合、開発者は Oracle JMS 宛先へのアクセスを初期化する sql ファイルを実行する必要があります。

**関連項目：**

- 7-5 ページの「バックエンド・メッセージ処理への MDB の使用」
- 7-14 ページの「JMS Web サービスのデプロイ」

---

**注意：** JMS Web サービスでスタンドアロンの JMS クライアントを使用してメッセージを使用または生成する場合、JMS Web サービスを使用してスタンドアロン・クライアントをアセンブルすることはできません。

---

## メッセージ処理と応答メッセージ

JMS Web サービスでは、受信メッセージである JMS Web サービスの send 操作メッセージが処理され、そのメッセージが JMS 宛先に置かれます。この項では、JMS 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 Web サービスに関連するクライアント・サイド・メッセージは、W3C（org.w3c.dom.Element）により指定されている Document Object Model（DOM）に準拠した XML 文書です。JMS Web サービスでは、Web サービス・クライアントから Element を受信すると、その Element を含む JMS の ObjectMessage が作成されます。JMS Web サービスでは、JMS 宛先にメッセージを置く前に、特定のヘッダー値を設定できます。JMS 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タグの値に応じて、JMS 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 Web サービスでは、JMSReplyTo ヘッダーの設定時に、<reply-to-topic-resource-ref> または <reply-to-queue-resource-ref> で指定した値が使用されます（どちらか一方をその JMS Web サービス用に構成する必要があります）。<reply-to-connection-factory-resource-ref> タグで指定した値は、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 7-3 に、onMessage() メソッドが JMS Web サービスの send 操作で生成されたメッセージの ReplyTo 情報を取得することを示すコード部分を示します。

**例 7-3**

```
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();
.
.
.
```

**関連項目：**

- 7-5 ページの「受信メッセージを処理する MDB の開発」
- 7-10 ページの「JMS Doc Service タグの追加」

# JMS Web サービスの準備と構成

この項では、Oracle Application Server Web Services のツール WebServicesAssembler を使用して、JMS Web サービス用に J2EE の .ear ファイルを準備する方法について説明します。

JMS Web サービスをデプロイするには、J2EE の .ear ファイルをアセンブルする必要があります。この .ear ファイルには、次の内容を含めることができます。

- Oracle Application Server Web Services サーブレットのデプロイメント・ディスクリプタ。
- JMS Web サービスに MDB も含まれる場合は、J2EE の .ear にも MDB 実装を提供する Jar ファイルを含めます。このコンポーネントはオプションです。JMS キューまたは JMS トピックを JMS Web サービスとして公開する場合、JMS Web サービスとともに MDB の Jar ファイルを含める必要はありません。

この項では、WebServicesAssembler で使用する構成ファイルの作成手順について説明します。

この項の内容は、次のとおりです。

- JMS Web サービスをアセンブルするための構成ファイルの作成
- JMS Web サービスを使用した WebServicesAssembler の実行

## JMS Web サービスをアセンブルするための構成ファイルの作成

Oracle Application Server Web Services のアセンブリ・ツール WebServicesAssembler を使用すると、Oracle Application Server Web Services をアセンブルできます。この項では、アセンブル対象の JMS Web サービスを記述する XML 構成ファイルの作成方法を説明します。

次のタグを追加して、WebServicesAssembler 構成ファイルを作成します。

- Web サービスのトップレベル・タグの追加
- JMS Doc Service タグの追加
- WSDL およびクライアント・サイド・プロキシ生成タグの追加

### Web サービスのトップレベル・タグの追加

表 7-1 に、WebServicesAssembler 構成ファイルのトップレベル・タグを示します。これらのタグを追加して、JMS Web サービスを記述するトップレベルの情報を提供します。

例 7-4 に、完全な JMS Web サービスのサンプル構成ファイルを示します。JMS Web サービス用の完全な config.xml ファイルは、jms_service ディレクトリの demo1 および demo2 ディレクトリにあります。

**表 7-1 トップレベルの WebServicesAssembler 構成タグ**

| タグ | 説明 |
|---|---|
| <context><br>*context*<br></context> | Web サービスのコンテキスト・ルートを指定します。<br>このタグは必須です。 |
| <datasource-JNDI-name><br>*name*<br></datasource-JNDI-name> | Web サービスに関連するデータソースを指定します。 |
| <description><br>*description*<br></description> | Web サービスの簡単な説明を提供します。<br>このタグはオプションです。 |
| <destination-path><br>*dest_path*<br></destination-path> | 生成される J2EE の .ear ファイルの出力名を指定します。*dest_path* には、出力ファイルのフルパスを指定します。<br>このタグは必須です。 |

**表 7-1 トップレベルの WebServicesAssembler 構成タグ（続き）**

| タグ | 説明 |
|---|---|
| <display-name><br>*disp_name*<br></display-name> | Web サービスの表示名を指定します。<br>このタグはオプションです。 |
| <option name="source-path"><br>*path*<br><option> | 指定したファイルを出力 .ear ファイルに含めます。このオプションを使用して Java リソースを指定するか、J2EE の出力 .ear ファイルのソース・ファイルとして使用される既存の .war、.ear または ejb-jar ファイルの名前を指定します。<br>.war ファイルが入力として提供されると、オプションの contextroot によって、.war ファイルの root-context が指定されます。<br>*path1* には、.war の context-root を指定します。<br>*path2* には、組み込むファイルへのパスを指定します。<br>次に例を示します。<br><option name="source-path" contextroot="/test">/myTestArea/ws/src/statefull.war</option><br>このタグはオプションです。 |
| <jms-doc-service><br>*sub-tags*<br></jms-doc-service> | このタグを使用して JMS Web サービスを追加します。有効な *sub-tags* の詳細は、表 7-2 を参照してください。 |
| <temporary-directory><br>*temp_dir*<br></temporary-directory> | アセンブラで一時ファイルを格納できるディレクトリを指定します。<br>このタグはオプションです。 |

## JMS Doc Service タグの追加

<jms-doc-service> では、JMS Web サービスの構成情報を定義します。JMS Web サービス開発者は、<operation> サブタグの値に基づいて、サービスで send 操作、receive 操作または両方の操作のうち、どれをサポートするかを決定します。構成ファイルの一部のタグが有効かどうかは、Web サービス用に選択する操作に応じて異なります。表 7-2 に、サポートされる <jms-doc-service> サブタグと、特定の操作を指定した場合に各サブタグが有効かどうかを示します。

**表 7-2 JMS サービスの WebServicesAssembler 構成タグ**

| タグ | 説明 |
|---|---|
| <connection-factory-resource-ref><br>*resource-ref*<br></connection-factory-resource-ref> | JMS Web サービスに関連する JMS 宛先について、トピック・コネクション・ファクトリまたはキュー・コネクション・ファクトリのリソース参照 *resource-ref* を指定します。<br>このタグは必須です。 |
| <jms-delivery-mode><br>*delivery-mode*<br></jms-delivery-mode> | JMSDeliveryMode メッセージ・ヘッダーを、send 操作で作成される JMS メッセージについて指定された *delivery-mode* 値に設定します。<br>このタグは、<operation> 値が send または both の場合に有効です。<br>このタグはオプションです。 |
| <jms-expiration><br>*expiration*<br></jms-expiration> | JMSExpiration メッセージ・ヘッダーを、send 操作で作成される JMS メッセージについて指定された *expiration* 値に設定します。<br>このタグは、<operation> 値が send または both の場合に有効です。<br>このタグはオプションです。 |

**表 7-2 JMS サービスの `WebServicesAssembler` 構成タグ（続き）**

| タグ | 説明 |
|---|---|
| `<jms-message-type>`<br>*message-type*<br>`</jms-message-type>` | メッセージの `JMSType` を、`send` 操作で作成される JMS メッセージについて指定された *message-type* に設定します。<br>このタグは、`<operation>` 値が `send` または `both` の場合に有効です。<br>このタグはオプションです。 |
| `<jms-priority>`<br>*priority*<br>`</jms-priority>` | `JMSPriority` メッセージ・ヘッダーを、`send` 操作で作成される JMS メッセージについて指定された *priority* 値に設定します。<br>このタグは、`<operation>` 値が `send` または `both` の場合に有効です。<br>このタグはオプションです。 |
| `<receive-timeout>`<br>*timeout*<br>`</receive-timeout>` | receive のタイムアウト秒数を指定する構成可能なタイムアウト値を提供します。receive 操作で新規メッセージを待機する時間を秒数で指定します。<br>このタグは、`<operation>` 値が `receive` または `both` の場合に有効です。<br>このタグが指定されていないか、または値が 0（ゼロ）に設定されている場合、JMS は操作ブロックを永続的に受信します。有効な値は 0（ゼロ）および正の整数です。<br>デフォルト値 : 0<br>このタグはオプションです。 |
| `<operation>`<br>*op*<br>`</operation>` | JMS Web サービスでサポートされる操作 *op* を指定します。<br>`send` および `receive` 操作を使用する場合の、各操作の意味は次のとおりです。<br>■ 宛先が JMS キューであれば、`send` はエンキュー、`receive` はデキューを意味します。<br>■ 宛先がトピックであれば、`send` はパブリッシュ、`receive` はサブスクライブを意味します。<br>send 操作では、`<connection-factory-resource-ref>` および対応する JMS 宛先の `<queue-resource-ref>` または `<topic-resource-ref>` を使用して、サービス上で send 操作の JMS 宛先が決定されます。<br>`receive` 操作の場合は、`<reply-to-connection-factory-resource-ref>` タグを設定しないと、`<connection-factory-resource-ref>` および対応する JMS 宛先の `<queue-resource-ref>` または `<topic-resource-ref>` が使用されます。`<reply-to-connection-factory-resource-ref>` タグを設定した場合は、`<reply-to-*>` タグで `receive` 操作の JMS 宛先を指定します。<br>有効な値 : `send`、`receive`、`both`<br>デフォルト値 : `both`<br>このタグはオプションです。 |
| `<queue-resource-ref>`<br>*queue-ref*<br>`</queue-resource-ref>` | 宛先 JMS キューのリソース参照 *queue-ref* を指定します。<br>`<topic-resource-ref>` または `<queue-resource-ref>` のどちらか一方を指定する必要があります。両方は指定できません。`<queue-resource-ref>` を指定する場合、`<connection-factory-resource-ref>` では対応するキュー・コネクション・ファクトリを参照する必要があります。 |
| `<reply-to-connection-factory-resource-ref>`<br>*reply-to-conn-factory-res-ref*<br>`</reply-to-connection-factory-resource-ref>` | 指定した `<operation>` が `both` の場合、`receive` 操作では `<reply-to-connection-factory-resource-ref>` が使用されます。*reply-to-conn-factory-res-ref* 値では、`receive` 操作用の JMS 宛先のコネクション・ファクトリを指定します。また、MDB または任意の JMS コンシューマが結果の返送を予期している場合は、このパラメータで応答メッセージの送信先となる宛先コネクション・ファクトリの名前を指定する必要があります。<br>関連項目 : 7-8 ページの「メッセージ処理と応答メッセージ」<br>このタグはオプションです。 |

**表 7-2 JMS サービスの WebServicesAssembler 構成タグ（続き）**

| タグ | 説明 |
|---|---|
| <reply-to-queue-resource-ref><br>*reply-to-queue-res-ref*<br></reply-to-queue-resource-ref> | 宛先 JMS キューのリソース参照 *reply-to-queue-res-ref* を指定します。<br><reply-to-queue-resource-ref> を指定する場合、<reply-to-connection-factory-resource-ref> では対応するキュー・コネクション・ファクトリを参照する必要があります。<br><reply-to-connection-factory-resource-ref> タグを設定する場合は、<reply-to-topic-resource-ref> または <reply-to-queue-resource-ref> タグのどちらか一方を指定する必要があります。両方は指定できません。<br>このタグはオプションです。 |
| <reply-to-topic-resource-ref><br>*reply-to-topic-res-ref*<br></reply-to-topic-resource-ref> | 宛先 JMS トピックのリソース参照 *reply-to-topic-res-ref* を指定します。<br><reply-to-topic-resource-ref> を指定する場合、<reply-to-connection-factory-resource-ref> では対応するトピック・コネクション・ファクトリを参照する必要があります。<br><reply-to-connection-factory-resource-ref> タグを設定する場合は、<reply-to-topic-resource-ref> または <reply-to-queue-resource-ref> タグのどちらか一方を指定する必要があります。両方は指定できません。<br>このタグはオプションです。 |
| <topic-resource-ref><br>*topic-ref*<br></topic-resource-ref> | 宛先 JMS トピックのリソース参照 *topic-ref* を指定します。<br><topic-resource-ref> または <queue-resource-ref> のどちらか一方を指定する必要があります。両方は指定できません。<topic-resource-ref> を指定する場合、<connection-factory-resource-ref> では対応するトピック・コネクション・ファクトリを参照する必要があります。 |
| <topic-subscription-name><br>*topic-name*<br></topic-subscription-name> | JMS プロバイダが永続 JMS トピックをサポートしている場合、JMS Doc Service は、その永続トピックの使用をサポートします。永続トピックを指定するには、このタグを使用して *topic-name* を指定します。このタグは、<topic-resource-ref> が指定されている場合にのみ有効です。<br>このタグはオプションです。 |
| <uri><br>*URI*<br></uri> | このタグでは、JMS Web サービスを実装するサーブレットのサーブレット・マッピング・パターンを指定します。*URI* として指定したパスは、JMS Web サービスの位置を指定する <context> に追加されます。<br>このタグはオプションです。 |

## WSDL およびクライアント・サイド・プロキシ生成タグの追加

WebServicesAssembler では、Web サービス開発者が WSDL ファイルとクライアント・サイド・プロキシ・ファイルを生成できるように、<wsdl-gen> および <proxy-gen> タグがサポートされます。これらのタグを使用して、WSDL ファイルとクライアント・サイド・プロキシを生成するかどうかを制御できます。また、これらのタグを使用すると、生成された WSDL ファイルまたは記述する WSDL ファイルを、Web サービスの J2EE の .ear とともにパッケージするように指定できます。

クライアント・サイド開発者が Web サービスを使用するアプリケーションを作成するには、デプロイされた Web サービスから取得した WSDL ファイルを使用する方法と、WSDL から生成されるクライアント・サイド・プロキシを使用する方法があります。

**関連項目：** 9-4 ページの「WSDL ファイルとクライアント・サイド・プロキシの生成」

**例 7-4 JMS Web サービス用 WebServicesAssembler 構成ファイルのサンプル**

```
<web-service>
  <display-name>JMS Web Service Example</display-name>
  <description>JMS Web Service Example</description>
   <!-- Name of the destination -->
  <destination-path>./jmsws1.ear</destination-path>
  <temporary-directory>./tmp</temporary-directory>
  <!-- Context root of the application -->
  <context>/jmsws1</context>
  <!-- Path of the jar file with MDBs definied/implemented in it -->
  <option name="source-path">MDB/mdb_service1.jar</option>

  <!--  tags for jms doc service  -->
  <jms-doc-service>
    <uri>JmsSend</uri>
    <connection-factory-resource-ref>jms/theQueueConnectionFactory</connection-factory-resource-ref>
    <queue-resource-ref>jms/theQueue</queue-resource-ref>
    <operation>send</operation>x
  </jms-doc-service>

  <jms-doc-service>
    <uri>JmsReceive</uri>
    <connection-factory-resource-ref>jms/logQueueConnectionFactory</connection-factory-resource-ref>
    <queue-resource-ref>jms/logQueue</queue-resource-ref>
    <operation>receive</operation>
   </jms-doc-service>
    <!-- generate the wsdl -->
    <wsdl-gen>
           <wsdl-dir>wsdl</wsdl-dir>
           <!-- over-write a pregenerated wsdl , turn it 'false' to use the pregenerated wsdl-->
           <option name="force">true</option>
           <option name="httpServerURL">http://localhost:8888</option>
           <!-- do not package the wsdl -generate it again on the server-->
           <option name="packageIt">false</option>
    </wsdl-gen>
    <!-- generate the proxy -->
    <proxy-gen>
           <proxy-dir>proxy</proxy-dir>
           <option name="include-source">true</option>
    </proxy-gen>
</web-service>
```

## JMS Web サービスを使用した WebServicesAssembler の実行

`WebServicesAssembler` 構成ファイルの作成後に、JMS Web サービス用の J2EE の .ear ファイルを生成できます。この J2EE EAR ファイルには、生成されるファイル `web.xml` などの Web サービス・サーブレット構成情報と、サービスに MDB が使用される場合は実装クラスを含む ejb.jar ファイルが含まれます。

Oracle Application Server Web Services のアセンブリ・ツール `WebServicesAssembler` を次のように実行します。

```
java -jar WebServicesAssembler.jar -config my_jms_service_config
```

ここで、*my_jms_service_config* は、`<jms-doc-service>` タグを含む構成ファイルです。

**関連項目：**

- 7-9 ページの「JMS Web サービスをアセンブルするための構成ファイルの作成」
- 9-2 ページの「Web サービス・アセンブリ・ツールの実行」

# JMS Web サービスのデプロイ

Java クラスと Web サービス・サーブレットのデプロイメント・ディスクリプタを含む .ear ファイルの作成後に、.ear ファイルに格納されている標準的な J2EE アプリケーションの場合と同様に、Web サービスを（OC4J で動作するように）デプロイできます。

**関連資料：**『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』

# JMS Web サービスの制限

JMS Web サービスでは、`ObjectMessage` 型（`javax.jms.Message.ObjectMessage`）のメッセージのみがサポートされます。

# 8

# Web サービスを使用するクライアントの作成

この章では、Oracle Application Server Web Services を使用するクライアント・アプリケーションを簡単に作成して実行するための、Oracle Application Server Web Services の機能について説明します。

この章の内容は、次のとおりです。

# Web サービスの検索

Web サービスを使用する場合は、クライアント・アプリケーションを開発する必要があります。Web サービス・クライアントには、静的 Web サービス・クライアントと動的 Web サービス・クライアントの 2 種類があります。**静的 Web サービス・クライアント**は、UDDI レジストリ内のサービスを参照することなく Web サービスの位置を認識します。**動的 Web サービス・クライアント**は、検索を実行して UDDI レジストリ内で Web サービスの位置を確認してから、そのサービスにアクセスします。UDDI レジストリ内の Web サービスを検索する方法の詳細は、第 10 章「Web サービスの検出と公開」を参照してください。

静的クライアントを使用する場合は、Oracle Application Server Web Services の検索用に次のオプションが提供されます。

- 既知の URL にある既知の Web サービスを使用する方法。
- Oracle Application Server Web Services と既知のサービス URL を使用してクライアント・サイド・プロキシ Jar を取得するか、他の方法で Web サービス用のクライアント・サイド・プロキシ Jar を取得する方法。Oracle Application Server Web Services により生成されるクライアント・サイド・プロキシ Jar には、関連する Web サービスを検索するための URL が含まれています。
- Oracle Application Server Web Services と既知のサービスの URL を使用して WSDL を取得するか、他の方法で Web サービスを記述する WSDL ファイルを取得する方法。Oracle Application Server Web Services により生成される WSDL ファイルには、関連する Web サービスを検索するための URL が含まれています。

Web サービスを検出するか、WSDL またはクライアント・サイド・プロキシ Jar を取得した後に、その Web サービスを使用するクライアント・サイド・アプリケーションを作成できます。

**関連項目：** 第 10 章「Web サービスの検出と公開」

# Web サービス用の WSDL ファイルとクライアント・サイド・プロキシ Jar の取得

この項の内容は、次のとおりです。



## Web サービス・ホーム・ページを使用した WSDL とクライアント・サイド・プロキシの保存

Oracle Application Server Web Services を使用するには、Web サービスにアクセスするクライアント・サイド・アプリケーションを作成する必要があります。Oracle Application Server Web Services には、デプロイされる Web サービス用に次のファイルが用意されています。

- WSDL サービス記述
- クライアント・サイド・プロキシ Jar（クラス・ファイル）
- クライアント・サイド・プロキシ・ソース

Oracle Application Server Web Services では、デプロイされた Web サービスごとに Web サービス・ホーム・ページが提供されます。ホーム・ページにアクセスするには、サービス・エンド・ポイントを次の書式で入力します。

```
http://host:port/context-root/service
```

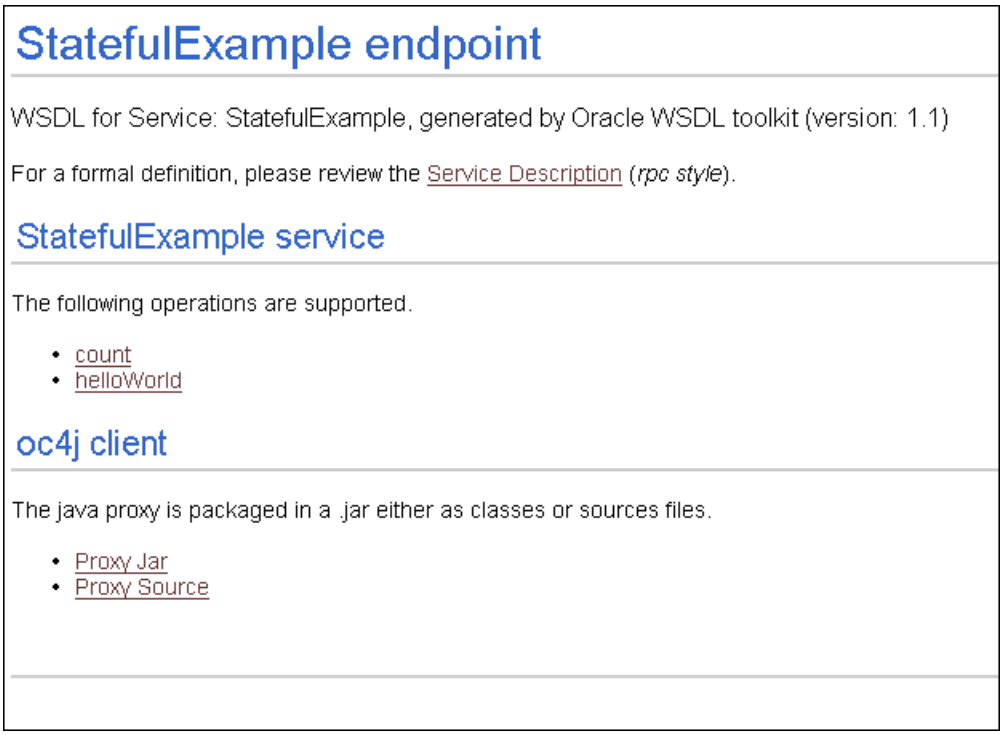
図 8-1 に、次のエンド・ポイントにある StatefulExample の Web サービス・ホーム・ページを示します。

```
http://system1.us.oracle.com/webservices/statefulTest
```

Web サービス・ホーム・ページで提供される要素は、次のとおりです。

- WSDL ファイルへのリンク - Web サービス用の WSDL ファイルを取得するには、「Service Description」リンクを選択してファイルを保存します。
- サポートされる操作ごとの Web サービス・テスト・ページへのリンク - 使用可能な Web サービスの操作をテストするには、その操作のパラメータ値があれば入力し、「Invoke」ボタンを選択します。
- Web サービスのクライアント・サイド・プロキシ Jar およびクライアント・サイド・プロキシ・ソースへのリンク - クライアント・サイド・プロキシ Jar またはクライアント・サイド・プロキシ・ソースを取得するには、「Proxy Jar」または「Proxy Source」のうち適切なリンクを選択し、ファイルを保存します。

**図 8-1 Web サービス・ホーム・ページ**




## Web サービス・テスト・ページの制限

Web サービス・テスト・ページには、次の制限があります。

- RPC スタイル Web サービスの場合、複合入力パラメータのサポート機能はありません。この種のページでは、「Invoke」ボタンはサポートされません。
- ドキュメント・スタイルの Web サービスに対するサポート機能はありません。この種のページでは、「Invoke」ボタンはサポートされません。

# Web サービスの WSDL とクライアント・サイド・プロキシの直接的な取得

Web サービス・ホーム・ページを使用して Web サービス用の WSDL ファイルやクライアント・サイド・プロキシを取得しない場合は、これらのファイルを直接取得できます。

この項の内容は、次のとおりです。



## WSDL サービス記述の取得

Web サービス用の WSDL サービス記述を取得するには、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（URL のコンポーネントの詳細は、表 8-1 を参照）。

```
http://host:port/context-root/service?WSDL
```

または

```
http://host:port/context-root/service?wsdl
```

このコマンドでは、WSDL 記述がフォーム service.wsdl 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。取得した 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。

## クライアント・サイド・プロキシ Jar とクライアント・サイド・プロキシ・ソース Jar の取得

Web サービス用のクライアント・サイド・プロキシ Jar 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。クライアント・サイド・プロキシ Jar ファイルにはプロキシ・スタブ・クラスが含まれており、このクラスにより、通信に SOAP を使用して Web サービスにアクセスするアプリケーションの作成がサポートされます。プロキシ・クラスの機能は、次のとおりです。

- Web サービスの静的位置の提供（UDDI レジストリ内でサービスを検索する必要はありません）
- Web サービスの一部として公開される各メソッド用のプロキシ・メソッドの提供
- パラメータのマーシャリングとアンマーシャリングおよびレスポンスの処理など、SOAP リクエストのすべての構成作業

クライアント・サイド・プロキシ Jar を取得するための URL の書式は、次のとおりです（URL のコンポーネントの詳細は、表 8-1 を参照）。

```
http://host:port/context-root/service?PROXY_JAR
```

または

```
http://host:port/context-root/service?proxy_jar
```

このコマンドでは、ファイル service_proxy.jar が戻されます。service_proxy.jar はクライアント・サイド・プロキシ・クラスを含む Jar ファイルであり、これらのクラスを使用して、Web サービスにアクセスするクライアント・サイド・アプリケーションを作成できます。

Web サービス用のクライアント・サイド・プロキシ・ソース Jar 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。クライアント・サイド・プロキシ・ソース Jar を取得するための URL の書式は、次のとおりです（URL のコンポーネントの詳細は、表 8-1 を参照）。

```
http://host:port/context-root/service?PROXY_SOURCE
```

または

```
http://host:port/context-root/service?proxy_source
```

このコマンドでは、ファイル service_proxysrc.jar が戻されます。service_proxysrc.jar ファイルは、クライアント・サイド・プロキシ・ソース・ファイルを含む Jar ファイルです。このファイルは、サービスに関連するファイル service_proxy.jar のソース・コードを表します。

## パッケージ指定によるクライアント・サイド・プロキシ Jar とクライアント・サイド・プロキシ・ソースの取得

クライアント・サイド・プロキシ Jar ファイルまたはクライアント・サイド・プロキシ・ソース Jar を取得する場合は、生成されるクライアント・サイド・プロキシのクラスまたはソース・ファイルのパッケージ名を指定する request パラメータなどのオプションがあります。Web サービスのクライアント・サイド Java クラスが特定のパッケージに属している場合は、そのパッケージ名をクライアント・サイド・アプリケーションのパッケージ名と一致するように指定する必要があります。

クライアント・サイド・プロキシ Jar を取得してパッケージ名を指定するための URL の書式は、次のとおりです（URL のコンポーネントの詳細は、表 8-1 を参照）。

```
http://host:port/context-root/service?PROXY_JAR&packageName=mypackage
```

または

```
http://host:port/context-root/service?proxy_jar&packageName=mypackage
```

このコマンドでは、ファイル service_proxy.jar が戻されます。service_proxy.jar は、Java の package 文に指定したパッケージ *mypackage* を使用する、クライアント・サイド・プロキシのクラスを含む Jar ファイルです。

クライアント・サイド・プロキシ・ソース Jar を取得してパッケージ名を指定するための URL の書式は、次のとおりです（URL のコンポーネントの詳細は、表 8-1 を参照してください）。

```
http://host:port/context-root/service?PROXY_SOURCE&packageName=mypackage
```

または

```
http://host:port/context-root/service?proxy_source&packageName=mypackage
```

このコマンドでは、ファイル service_proxysrc.jar が戻されます。proxy_jar の場合と同様に、packageName=*name* オプションを使用して、提供されるパッケージ名を request パラメータに指定する方法があります。service_proxysrc.jar は、Web サービスにアクセスするクライアント・サイド・プロキシ用のクライアント・サイド・ソース・ファイルを含む Jar ファイルです。

**表 8-1 クライアント・サイド・プロキシ・スタブにアクセスするための URL**

| URL コンポーネント | 説明 |
|---|---|
| *context-root* | context-root は、Web サービスに関連する Web モジュールについて <context-root> タグで指定する値です。この値を決定するには、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* | これは、Oracle Application Server Web Services を実行する Web サービスのサーバーのホストです。 |
| *mypackage* | ここでは、生成されたプロキシの Jar またはプロキシ・ソース内でパッケージ名に使用する値を指定します。 |
| *port* | これは、Oracle Application Server Web Services を実行する Web サービスのサーバーのポートです。 |
| *service* | service は、Web サービスに関連するサーブレット用に <url-pattern> タグで指定した値です。これはサービス名です。この値を決定するには、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。 |

**関連項目：**

- 第 3 章「Java クラス Web サービスの開発とデプロイ」
- 第 4 章「EJB Web サービスの開発とデプロイ」
- 第 5 章「ストアド・プロシージャ Web サービスの開発とデプロイ」

## WebServicesAssembler を使用したクライアント・サイド・プロキシの生成

Oracle Application Server Web Services の `WebServicesAssembler` ツールを使用すると、クライアント・サイド・プロキシを生成できます。クライアント・サイド・プロキシは、Oracle Application Server Web Services のエンド・ポイントまたはサード・パーティ Web サービスのエンド・ポイントにデプロイされている Web サービスにアクセスします。

`WebServicesAssembler` を使用してクライアント・サイド・プロキシを生成するには、構成ファイル内で `<proxy-gen>` タグを指定します。表 8-2 に、`WebServicesAssembler` 構成ファイルの `<proxy-gen>` のサブタグを示します。

---

**注意：** クライアント・サイド・プロキシの生成中に、ファイアウォール内から外部の WSDL ファイルにアクセスする場合は、`http.proxyHost` や `http.proxyPort` など、表 8-4 に示す適切なセキュリティ・プロパティを設定する必要があります。

---

例 8-1 に、`<proxy-gen>` タグを含むサンプル `WebServicesAssembler` を示します。

**例 8-1 WebServicesAssembler プロキシ生成用構成ファイル**

```
<?xml version="1.0"?>
<web-service>
  <proxy-gen>
    <proxy-dir>/TestArea/Hotel/proxy/outside</proxy-dir>
    <option name="include-source">true</option>
    <option name="wsdl-location" package-name="myPackage.proxy">
          http://terraservice.net/TerraService.asmx?WSDL</option>
    <option name="wsdl-location">
        http://ws.serviceobjects.net/sq/FastQuote.asmx?WSDL</option>
  </proxy-gen>
</web-service>
```

**表 8-2 プロキシ生成用 <proxy-gen> サブタグ**

| タグ | 説明 |
|---|---|
| `<proxy-dir>`<br>*directory*<br>`</proxy-dir>` | 生成されるクライアント・サイド・プロキシ・スタブの Jar ファイル用のディレクトリを指定します。このファイルは、生成される Web サービスの `.ear` ファイルに組み込まれます。<br>このタグは必須です。 |
| `<option name="include-source">`<br>*value*<br>`</option>` | `WebServicesAssembler` に対して、生成されるクライアント・サイド・プロキシにクラスとソースを含めるように指定するには、*value* を `true` に設定します。value が false の場合、生成される Jar にはソースは含まれません。<br>このタグはオプションです。<br>有効な値：`true`、`false`<br>デフォルト値：`false` |
| `<option name="wsdl-location">`<br>*URL*<br>`</option>`<br>または<br>`<option name="wsdl-location"`<br>`package-name="`*package*`">`<br>*URL*<br>`</option>` | このタグでは、ソース WSDL 用の *URL* をクライアント・サイド・プロキシの生成に使用するように設定します。<br>このオプションでは、オプションの属性 `package-name` もサポートされます。`package-name` では、生成されるクライアント・サイド・プロキシ用の *package* 名を指定できます。<br>このタグはオプションです。<br>次に例を示します。<br>`<option name="wsdl-location">`<br>`http://system1:8888/webservice3/TestService?WSDL`<br>`</option>`<br>`<option name="wsdl-location"`<br>`package-name="myPackage.proxy">`<br>`http://system1:8888/webservice3/TestService?WSDL`<br>`</option>` |

**関連項目：** 第 9 章「Web サービスのツール」

# Web サービスを使用するためのクライアント・サイド・プロキシ Jar の操作

この項では、Web サービスにアクセスするクライアント・サイド・アプリケーションを作成するときに、クライアント・サイド・プロキシ Jar を使用する方法について説明します。クライアント・サイド・プロキシ Jar クラスを使用すると、Web サービスを使用するアプリケーションを簡単に作成できます。

クライアント・サイド・プロキシ Jar ファイルには、Web サービス実装のプロキシとして機能する Java クラスが含まれています。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が軽減されます。

例 8-2 に、Web サービスから抽出したソース・コードのサンプルのクライアント・サイド・プロキシを示します。プロキシ・クラスには、Web サービスで使用可能な操作ごとに対応するメソッドがあります。この例は、関連する Web サービス実装内の `helloWorld(String)` メソッドのプロキシとして機能する `helloWorld(String)` メソッドを示しています。

例 8-3 に、例 8-2 に示した提供されるクライアント・サイド・プロキシからの `helloWorld()` メソッドを使用するクライアント・サイド・アプリケーション・コードを示します。

**注意：** ファイアウォール内から外部の Web サービスにアクセスしている場合は、`http.proxyHost` や `http.proxyPort` など、表 8-4 に示す適切なセキュリティ・プロパティを設定する必要があります。

**例 8-2 Web サービス用のクライアント・サイド・プロキシのサンプル・メソッド**

```
public class StatefulExampleProxy {

   public java.lang.String helloWorld(java.lang.String param0) throws Exception
   {
   .
   .
   .
   }
.
.
.
}
```

**例 8-3 Web サービス用のプロキシ・クラスを使用するサンプルのクライアント・サイド・アプリケーション**

```
import oracle.j2ee.ws_example.proxy.*;

public class Client
{
  public static void main(String[] argv) throws Exception
  {
    StatefulExampleProxy proxy = new StatefulExampleProxy();
    System.out.println(proxy.helloWorld("Scott"));
    System.out.println(proxy.count());
    System.out.println(proxy.count());
    System.out.println(proxy.count());
  }
}
```

## Web サービス・プロキシ・クライアントの CLASSPATH の設定

プロキシを使用する Web サービス・クライアントを作成するときは、適切な `CLASSPATH` を使用してクライアントを実行する必要があります。表 8-3 に、`CLASSPATH` に含める必要がある jar ファイルを示します。

**表 8-3 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 Web サービス CLASSPATH コンポーネント**

| コンポーネント Jar | 説明 |
|---|---|
| *proxy*.jar | Web サービスに対するアクセスを提供する *proxy* jar ファイル。 |
| `$ORACLE_HOME/lib/xmlparserv2.jar` | Oracle XML Parser の jar ファイル。 |
| `$ORACLE_HOME/j2ee/home/lib/http_client.jar` | Oracle HTTP クライアントの jar ファイル。 |
| `$ORACLE_HOME/soap/lib/soap.jar` | OracleAS SOAP の jar ファイル。 |
| `$ORACLE_HOME/j2ee/home/lib/mail.jar` | 通常、このファイルは JRE. で使用できます。JRE で使用できない場合は、CLASSPATH に含めてください。 |
| `$ORACLE_HOME/j2ee/home/lib/activation.jar` | 通常、このファイルは JRE. で使用できます。JRE で使用できない場合は、CLASSPATH に含めてください。 |
| `$ORACLE_HOME/jlib/javax-ssl-1_1.jar` | SSL を使用する Web サービスに接続するために、クライアントが SSL を使用する場合に使用されます。このファイルを使用する場合は、`$ORACLE_HOME/lib/jsee.jar` を `CLASSPATH` に含めないでください。 |

**表 8-3 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 Web サービス CLASSPATH コンポーネント（続き）**

| コンポーネント Jar | 説明 |
|---|---|
| $ORACLE_HOME/jlib/jssl-1_1.jar | SSL を使用する Web サービスに接続するために、クライアントが SSL を使用している場合は必須です。この場合は、$ORACLE_HOME/jlib/javax-ssl-1_1.jar または $ORACLE_HOME/lib/jsse.jar を指定する必要があります。 |
| $ORACLE_HOME/lib/jsse.jar | SSL を使用する Web サービスに接続するために、クライアントが SSL を使用する場合に使用されます。このファイルを使用する場合は、$ORACLE_HOME/jlib/javax-ssl-1_1.jar を CLASSPATH に含めないでください。 |
| $ORACLE_HOME/webservices/lib/wsdl.jar | クライアントが動的プロキシを使用している場合は必須です。 |
| $ORACLE_HOME/webservices/lib/dsv2.jar | クライアントが動的プロキシを使用している場合は必須です。 |

## Web サービスのパラメータとしての JavaBeans の使用

JavaBeans を Oracle Application Server Web Services へのパラメータとして使用する場合、クライアント・サイド・コードでは、ダウンロードしたクライアント・サイド・プロキシに含まれている、生成された Bean を使用する必要があります。これは、生成されたクライアント・サイド・プロキシ・コードでは、SOAP 構造の名前空間と完全修飾 Bean クラス名の間で変換が行われることで、SOAP 構造と JavaBean の間での変換が行われるためです。指定した名前の Bean が指定したパッケージに存在しない場合、生成されたクライアント・コードは失敗します。

ただし、クライアントで Web Services Description Language（WSDL）を使用して、クライアント・サイド・プロキシではなく Oracle Application Server Web Services のコールを形成する場合、特別な要件はありません。生成された WSDL ドキュメントでは、SOAP 構造が標準的な方法で記述されます。WSDL ドキュメントから直接動作する Oracle JDeveloper などのアプリケーション開発環境では、JavaBeans をパラメータとして使用し、Oracle Application Server Web Services を適切にコールできます。

## Web サービスのセキュリティ機能の使用

Oracle Application Server Web Services を使用するクライアント・サイド・アプリケーションを実行する場合は、クライアント・アプリケーション内でプロパティを設定して、セキュアな Web サービスにアクセスできます。表 8-4 に、使用可能なプロパティを示します。各プロパティにより、Web サービス・クライアントの証明書と他のセキュリティ情報が提供されます。表 8-3 に、SSL のサポートに必要なファイルも含めて、CLASSPATH に含める必要がある jar ファイルを示します。

Web サービスのクライアント・アプリケーションでは、Java コマンドラインで -D フラグを使用して、表 8-4 のセキュリティ・プロパティをシステム・プロパティとして設定できます。また、これらのプロパティをシステム・プロパティに追加して、Java プログラム内でセキュリティ・プロパティを設定する方法もあります（プロパティを追加するには System.setProperties() を使用します）。さらに、クライアント・サイド・スタブには、クライアント・プロキシ・スタブ内のパブリック・メソッドである _setTranportProperties メソッドが含まれています。このメソッドを使用すると、Properties 引数を指定してセキュリティ・プロパティに適切な値を設定できます。

**表 8-4 Web サービスの HTTP トランスポートのセキュリティ・プロパティ**

| プロパティ | 説明 |
|---|---|
| http.authRealm | HTTP 認証のユーザー名 / パスワードを指定するレルムを指定します。<br>このプロパティは、Basic 認証を使用するときは必須です。 |
| http.authType | HTTP の認証タイプを指定します。指定した値の大 / 小文字区別は無視されます。<br>有効な値 : basic、digest<br>値 basic は、HTTP の Basic 認証を指定します。<br>basic または digest 以外の値を指定すると、このプロパティは未設定とみなされます。 |
| http.password | HTTP 認証のパスワードを指定します。 |
| http.proxyAuthRealm | プロキシ認証のユーザー名 / パスワードを指定するレルムを指定します。 |
| http.proxyAuthType | プロキシ認証タイプを指定します。指定した値の大 / 小文字区別は無視されます。<br>有効な値 : basic、digest<br>basic または digest 以外の値を指定すると、このプロパティは未設定とみなされます。 |
| http.proxyHost | プロキシ・ホストのホスト名または IP アドレスを指定します。 |
| http.proxyPassword | HTTP のプロキシ認証パスワードを指定します。 |
| http.proxyPort | プロキシ・ポートを指定します。整数値を指定する必要があります。このプロパティが使用されるのは http.proxyHost が定義されている場合のみで、他の場合は無視されます。<br>デフォルト値 : 80 |
| http.proxyUsername | HTTP のプロキシ認証のユーザー名を指定します。 |
| http.username | HTTP 認証のユーザー名を指定します。 |
| java.protocol.handler.pkgs | java.net.URLStreamHandlerFactory 用のパッケージ接頭辞のリストを指定します。接頭辞は縦線文字（|）で区切る必要があります。<br>この値には HTTPClient を含める必要があります。<br>この値は Java プロトコル・ハンドラ・フレームワークに必須です。Oracle Application Server では定義されません。HTTPS を使用する場合は、このプロパティを設定する必要があります。HTTPS を使用する場合にこのプロパティを設定しないと、java.net.MalformedURLException がスローされます。<br>**注意 :** このプロパティは、システム・プロパティとして設定する必要があります。<br>たとえば、このプロパティを次のどちらかに設定します。<br>■ java.protocol.handler.pkgs=HTTPClient<br>■ java.protocol.handler.pkgs=sun.net.www.protocol\|HTTPClient |
| oracle.soap.transport.allowUserInteraction | allows user interaction パラメータを指定します。指定した値の大 / 小文字区別は無視されます。次のどちらかの場合に、このプロパティを true に設定すると、ユーザー名とパスワードの入力を求められます。<br>1. プロパティ http.authType、http.username または http.password を設定しておらず、HTTP サーバーから 401 HTTP ステータスが戻される場合<br>2. プロパティ http.proxyAuthType、http.proxyUsername または http.proxyPassword を設定しておらず、HTTP プロキシから 407 HTTP レスポンスが返される場合<br>有効な値 : true、false<br>true 以外の値を指定すると、false とみなされます。 |

**表 8-4 Web サービスの HTTP トランスポートのセキュリティ・プロパティ（続き）**

| プロパティ | 説明 |
|---|---|
| oracle.ssl.ciphers | 有効化する暗号スイートをコロン（:）で区切ったリスト形式で指定します。<br>デフォルト値 : Oracle SSL でサポートされているすべての暗号スイートのリスト。 |
| oracle.wallet.location | エクスポートする Oracle Wallet またはトラスト・ポイントの位置を指定します。<br>**注意 :** 次のように、URL ではなくファイル位置を指定します。<br>/etc/ORACLE/Wallets/system1/exported_wallet（UNIX）<br>d:¥oracle¥system1¥exported_wallet（Windows）<br>SSL 認証、サーバー認証または相互認証で HTTPS をトランスポートとして使用する場合は、このプロパティを設定する必要があります。 |
| oracle.wallet.password | エクスポートする Wallet のパスワードを指定します。クライアント認証、つまり相互認証で HTTPS をトランスポートとして使用する場合は、このプロパティを設定する必要があります。 |

# Web サービスを使用するための WSDL ファイルと Oracle JDeveloper の操作

Web サービスの WSDL を使用すると、手動で、あるいは Oracle JDeveloper や他の IDE を使用して、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 IDE では、WSDL 機能を持つ Oracle Application Server Web Services がサポートされ、エンドツーエンドの J2EE と統合された Web サービス・アプリケーションの作成に高い生産性をもたらします。

Oracle JDeveloper では、次の機能によって Oracle Application Server Web Services がサポートされます。

- 開発者は、Web サービスの WSDL 記述から Java スタブを作成し、既存の Web サービスをプログラムで使用できます。
- 開発者は、Java または EJB クラスから新規 Web サービスを作成できます。必要なデプロイメント・ディスクリプタ、web.xml および WSDL ファイルは自動的に生成されます。
- WSDL ファイルのスキーマ・ドリブン編集機能を提供します。
- Web サービス J2EE の .ear ファイルに対する有効な J2EE デプロイのサポート機能があり、OC4J に自動的にデプロイできます。

Oracle 以外の Web サービス IDE またはクライアント開発ツールでは、提供される WSDL ファイルを使用して、Oracle Application Server Web Services で動作するサービスに対する Web サービス・リクエストを生成できます。現在、サービス用の WSDL 記述について、多数の IDE に SOAP リクエストを作成する機能があります。

# 9

# Web サービスのツール

Oracle Application Server Web Services のアセンブリ・ツール `WebServicesAssembler` を使用すると、Oracle Application Server Web Services をアセンブルできます。この Web サービス・アセンブリ・ツールでは、Java クラス、PL/SQL ストアド・プロシージャやファンクション、J2EE の EAR、WAR または JAR ファイルの位置など、Web サービスを記述する構成ファイルを使用して、Oracle Application Server Web Services でデプロイできる J2EE の EAR ファイルが生成されます。

この章の内容は、次のとおりです。

# Web サービス・アセンブリ・ツールの実行

Web サービス・アセンブリ・ツールを実行するには、次のように入力します。

```
java -jar WebServicesAssembler.jar [-debug] -config [file]
```

または

```
java -jar WebServicesAssembler.jar [-debug]
```

*file* は、Web サービス・アセンブリ・ツールの構成ファイルです。-config オプションを指定しない場合は、WebServicesAssembler.jar が起動されるのと同じディレクトリにファイル config.xml が存在する必要があります。

-debug オプションを指定すると、WebServicesAssembler では冗長なデバッグ・コメントが表示されます。

---

**注意：** コマンドラインから WebServicesAssembler.jar を実行する場合は、PATH 環境変数に JDK/bin ディレクトリ（javac コンパイラが含まれているディレクトリ）を組み込む必要があります。

---

# Web サービス・アセンブリ・ツールのサンプル構成ファイル

例 9-1 のサンプル構成ファイルでは、2 つのサービスをエンタープライズ・アーカイブ・ファイル（EAR: Enterprise ARchive file）内でラップするように定義しています。このサンプルには、<stateless-java-service> および <stateful-java-service> タグで定義されたサービスの構成情報が含まれています。

**関連項目：**

- 3-7 ページの「Java クラス・ベースの Web サービスの準備とデプロイ」
- 4-7 ページの「ステートレス Session EJB ベースの Web サービスの準備とデプロイ」
- 5-3 ページの「ストアド・プロシージャ Web サービスの準備」
- 6-7 ページの「ドキュメント・スタイルの Web サービスの準備」
- 7-9 ページの「JMS Web サービスの準備と構成」

**例 9-1 Web サービス・アセンブリ・ツールのサンプル構成ファイル**

```
<web-service>

    <display-name>Web Services Example</display-name>
    <description>Java Web Service Example</description>
    <!-- Specifies the resulting web service archive will be stored in ./ws_example.ear -->
    <destination-path>./ws_example.ear</destination-path>
    <!-- Specifies the temporary directory that web service assembly
         tool can create temporary files. -->
    <temporary-directory>./tmp</temporary-directory>
    <!-- Specifies the web service will be accessed in the servlet context
         named "/webservices". -->
    <context>/webservices</context>

    <!-- Specifies the web service will be stateless -->
    <stateless-java-service>
        <interface-name>oracle.j2ee.ws_example.StatelessExample</interface-name>
        <class-name>oracle.j2ee.ws_example.StatelessExampleImpl</class-name>
        <!-- Specifies the web service will be accessed in the uri named
             "statelessTest" within the servlet context. -->
        <uri>/statelessTest</uri>
        <!-- Specifies the location of Java class files are under ./src -->
        <java-resource>./src</java-resource>
```

```
    </stateless-java-service>

    <stateful-java-service>
        <interface-name>oracle.j2ee.ws_example.StatefulExample</interface-name>
        <class-name>oracle.j2ee.ws_example.StatefulExampleImpl</class-name>
        <!-- Specifies the web service will be accessed in the uri named
             "statefulTest" within the servlet context. -->
        <uri>/statefulTest</uri>
        <!-- Specifies the location of Java class files are under ./src -->
        <java-resource>./src</java-resource>
    </stateful-java-service>

</web-service>
```

## Web サービス・アセンブリ・ツールの構成ファイルのサンプル出力

例 9-1 のサンプル入力ファイルを使用して Web サービス・アセンブリ・ツールを実行すると、出力として EAR ファイル（/tmp/ws_example.ear）が生成されます。例 9-2 に、生成された J2EE の .ear ファイル ws_example.ear の構造を示します。

**例 9-2 Web サービス・アセンブリ・ツールのサンプル Ear ファイルの構造**

```
ws_example.ear
|---META-INF
|   '---application.xml
'---ws_example_web.war
    |---index.html
    '---WEB-INF
        |------web.xml
        '------classes
               '------oracle
                      '-----j2ee
                            '---ws_example
                                |---StatefulExample.java
                                |---StatefulExample.class
                                |---StatefulExampleImpl.java
                                '---StatefulExampleImpl.class
                                |---StatelessExample.java
                                |---StatelessExample.class
                                |---StatelessExampleImpl.java
                                '---StatelessExampleImpl.class
```

# WSDL ファイルとクライアント・サイド・プロキシの生成

この項では、WebServicesAssembler 構成ファイル内で <wsdl-gen> および <proxy-gen> タグを使用する方法について説明します。これらのタグにより、Web サービス用の WSDL ファイルとクライアント・サイド・プロキシの生成オプションが制御されます。クライアント・サイド開発者は、WSDL ファイルまたはクライアント・サイド・プロキシを取得して、Web サービスを使用するアプリケーションの作成に使用できます。Web サービスをアセンブルするサーバー・サイド開発者は、これらのファイルを Web サービスのテストに使用できます。

この項の内容は、次のとおりです。



## WSDL ファイルの生成とアセンブル

Oracle Application Server Web Services を使用すると、Web サービス開発者は Web サービスに関連する WSDL ファイルの生成方法を選択できます。

1. <wsdl-gen> タグを使用すると、WebServicesAssembler で WSDL ファイルを作成するように指定できます。アセンブリ時に Web サービスが準備されるときには、WebServicesAssembler により WSDL ファイルが生成され、Web サービスとともにパッケージされます。

   例 9-3 に、<wsdl-gen> タグを含む構成ファイルを示します。

2. Oracle Application Server Web Services ランタイムに対して、Web サービス・クライアントから WSDL がリクエストされたとき（Web サービスのデプロイ後）に、WSDL ファイルを生成するように許可します。この場合、構成ファイルでは <wsdl-gen> タグを指定しないでください。

3. WSDL ファイルを手動で作成します。この場合は、J2EE の .ear ファイルのアセンブリ中に、<wsdl-gen> タグを使用して WSDL ファイルへのパスを指定します。アセンブリ時に Web サービスが準備されるときには、WebServicesAssembler により WSDL ファイルが Web サービスとともにパッケージされます。

表 9-1 に、WebServicesAssembler 構成ファイルの <wsdl-gen> のサブタグを示します。

---

**注意：** <wsdl-gen> タグを使用すると、デフォルトで WSDL が J2EE の .ear ファイルにパッケージされます。生成される WSDL を J2EE の .ear ファイルから除外するには、<option name="packageIt"> タグを使用して value を false に設定します。

---

**表 9-1 WSDL 生成用 <wsdl-gen> サブタグ**

| タグ | 説明 |
|---|---|
| <option name="force"><br>*value*<br></option> | *value* を true に設定すると、WebServicesAssembler では、<wsdl-dir> タグで指定した WSDL ディレクトリ内の既存の WSDL ファイルが上書きされます。<br>有効な値：true、false<br>デフォルト値：true |
| <option name="httpServerURL"><br>*URL*<br></option> | このタグでは、生成される WSDL 内で HTTP サーバー・リスナーのエンド・ポイントの値を設定します。*URL* を、Web サービスの HTTP リスナーを指すように設定します。<br>例：<br><option name="httpServerURL">http://localhost:8888</option> |

**表 9-1 WSDL 生成用 <wsdl-gen> サブタグ（続き）**

| タグ | 説明 |
|---|---|
| <option name="packageIt"><br>*value*<br></option> | *value* を true に設定すると、WebServicesAssembler では生成される WSDL がアセンブル後の .ear ファイルに組み込まれます。value を false に設定すると、生成される WSDL ファイルはアセンブル後の .ear ファイルに組み込まれません。<br>有効な値：true、false<br>デフォルト値：true |
| <wsdl-dir><br>*directory*<br></wsdl-dir> | 生成される Web サービスの .ear ファイルに含まれる WSDL ファイルのソースの *directory* を指定します。<br>WSDL ファイルを手動で提供する場合は、指定したディレクトリに WSDL ファイルのコピーを置き、値 false を指定して <option name="force"> タグを使用します。 |

**例 9-3 <wsdl-gen> を含む WebServicesAssembler 構成ファイル**

```
<web-service>

    <display-name>Stateless Java Document Web Service</display-name>
    <description>Stateless Java Document Web Service Example</description>
    <destination-path>./statelessdocws.ear</destination-path>
    <temporary-directory>./temp</temporary-directory>
    <context>/statelessdocws</context>
    <option name="source-path">converter.xsl</option>

    <stateless-java-service>
        <interface-name>StatelessDoc</interface-name>
        <class-name>StatelessDocImpl</class-name>
        <uri>/docservice</uri>
        <java-resource>./classes</java-resource>
        <message-style>doc</message-style>
   </stateless-java-service>

    <!-- generate the wsdl -->
    <wsdl-gen>
       <wsdl-dir>wsdl</wsdl-dir>
       <!-- over-write a pregenerated wsdl , turn it 'false'
            to use the pregenerated wsdl-->
       <option name="force">true</option>
       <option name="httpServerURL">http://localhost:8888</option>
    </wsdl-gen>

</web-service>
```

## 手動による WSDL ファイルの生成

WebServicesAssembler ツールで生成される WSDL ファイルや、Oracle Application Server Web Services ランタイムで生成される WSDL ファイルを使用せずに、Web サービス用 WSDL ファイルの独自のバージョンを用意する場合は、次の手順を実行してください。

1. サービス用の WSDL ファイルを手動で作成します。
2. WSDL ファイル名として、サービス名の後に拡張子 .wsdl を付けたファイル名を指定します。たとえば、サービス service1 の場合は service1.wsdl となります。
3. <option name="force"> を false に設定し、<option name="packageIt"> を true に設定して、<wsdl-gen> タグを含む構成ファイルを作成します。
4. 作成した WSDL ファイルを、<wsdl-dir> タグで指定したディレクトリに置きます。
5. 構成ファイルを指定して WebServicesAssembler を実行します。

## WSDL を使用したクライアント・サイド・プロキシの生成

構成ファイルに <wsdl-gen> とともに <proxy-gen> タグが含まれている場合は、生成される WSDL を使用して、指定したディレクトリにプロキシが生成されます（Web サービスのアセンブリ処理中に WebServicesAssembler を実行する場合）。

<proxy-gen> のサブタグは、表 8-2 を参照してください。

---

**注意：** <proxy-gen> を使用すると、生成されるプロキシは J2EE の .ear ファイルにアセンブルされません。

---

例 9-4 に、<wsdl-gen> および <proxy-gen> タグの両方を含むサンプル構成ファイルを示します。

**例 9-4 <wsdl-gen> を含む WebServicesAssembler 構成ファイル**

```
<web-service>
  <display-name>Test</display-name>
  <description>Test program</description>
  <destination-path>test.ear</destination-path>
<temporary-directory>temp/</temporary-directory><context>/HotelService</context>
  <option name="source-path">Workspace1/common/classes</option>

  <stateless-java-service>
    <interface-name>com.mypackage1.Itest</interface-name>
    <uri>/main</uri>
    <class-name>com.mypackage1.test</class-name>
  </stateless-java-service>

  <wsdl-gen>
    <wsdl-dir>wsdl</wsdl-dir>
    <option name="force">true</option>
    <option name="httpServerURL">http://localhost:8888</option>
    <option name="packageIt">false</option>
  </wsdl-gen>

  <proxy-gen>
    <proxy-dir>proxy</proxy-dir>
    <option name="include-source">true</option>
  </proxy-gen>

  </web-service>
```

# Web サービス・アセンブリ・ツールの構成ファイルの仕様

WebServicesAssembler の入力ファイルは、Web サービス・アセンブリ・ツールの構成ファイル DTD に準拠する XML ファイルです。

例 9-5 に、Web サービス・アセンブリ・ツールの構成ファイル DTD を示します。

**例 9-5 アセンブリ・ツールの入力ファイル DTD**

```
<?xml version="1.0" encoding="UCS-2"?>
<!-- Specify the properties of the web services to be assembled. -->
<!ELEMENT web-service
((display-name)?,(description)?,destination-path,temporary-directory,context,(datasource-JNDI-name)?,(stateful-
java-service)*,(stateless-java-service)*,(stateless-stored-procedure-java-service)*,(stateless-session-ejb-serv
ice)*,(jms-doc-service)*,(option)*,(wsdl-gen)?,(proxy-gen)?)>
<!ELEMENT display-name (#PCDATA)*>
<!ELEMENT description (#PCDATA)*>
<!-- Specify the full path of the resulting EAR file. For example,
"/home/demo/webservices.ear" -->
<!ELEMENT destination-path (#PCDATA)*>
<!-- Specify a directory where the assembly tool can create temporary
directories and files. -->
<!ELEMENT temporary-directory (#PCDATA)*>
<!-- Specify the context root of the web services. For example, "/webservices". -->
<!ELEMENT context (#PCDATA)*>
<!-- for specifying database  resource refs -->
<!ELEMENT datasource-JNDI-name (#PCDATA)*>

<!-- Specify the properties of a stateful  Java service -->
<!ELEMENT stateful-java-service
((interface-name)?,class-name,uri,(java-resource)*,(ejb-resource)*,(scope)*,(session-timeout)*,(message-style)?
)>
<!-- Specify the properties of a stateless Java service -->
<!ELEMENT stateless-java-service
((interface-name)?,class-name,uri,(java-resource)*,(ejb-resource)*,(message-style)?)>
<!-- Specify the properties of a stateless stored procedure Java service -->
<!ELEMENT stateless-stored-procedure-java-service
((interface-name)?,(class-name)?,uri,database-JNDI-name,(java-resource)?,(jar-generation)?)>
<!-- Specify the properties of a stateless session ejb service -->
<!ELEMENT stateless-session-ejb-service (path,uri,ejb-name,(ejb-resource)*)>

<!-- Specify the java interface which defines the public methods to be exposed
in the web service. For example, "com.foo.myproject.helloWorld". -->
<!ELEMENT interface-name (#PCDATA)*>
<!-- Specify the java class to be exposed as a web service. If interface-name is
not specified, all the public methods in this class will be exposed. For example,
     "com.foo.myproject.helloWorldImpl". -->
<!ELEMENT class-name (#PCDATA)*>
<!-- Specify the uri of this service. This uri is used in the URL to access the
WSDL and client jar, and invoke the web service. For example, "/myService". -->
<!ELEMENT uri (#PCDATA)*>
<!--
Specify the java resources used in this service.
The value can be a directory or a file that implements the web services. If it
is a directory, all the files and subdirectories under the directory are copied
and packaged in the Enterprise ARchive. If the java resource should belong to a
java package, you should either package it as a jar file and specify it as a
java resource, or create the necessary directory and specify the directory which
contains this directory structure as java resource. For example, you want to
include  "com.mycompany.mypackage.foo" class as a java resource of the web
services, you can either package this class file in foo.jar and specify
<java-resource>c:/mydir/foo.jar</java-resource>, or place the class under
d:/mydir/com/mycompany/mypackage/foo.class and specify the java resource as
<java-resource>c:/mydir/</java-resource>.
-->
<!ELEMENT java-resource (#PCDATA)*>
```

```
<!-- Specify the ejb resources used in this service. ejb-resource should be a
jar file that implements a enterprise java bean. -->
<!ELEMENT ejb-resource (#PCDATA)*>
<!-- Specify the database JNDI name for stateless PL/SQL web service. -->
<!ELEMENT database-jndi-name (#PCDATA)*>
<!-- Specifies the path of the EJB jar file to exposed as web services. -->
<!ELEMENT path (#PCDATA)*>
<!-- Specify the ejb-name of the session bean to be exposed as web services.
ejb-name should match the <ejb-name> value in the META-INF/ejb-jar.xml of the bean. -->
<!ELEMENT ejb-name (#PCDATA)*>
<!-- Specify scope of Stateful Java service -->
<!ELEMENT scope (#PCDATA)*>
<!-- Specify session timeout  of Stateful Java service -->
<!ELEMENT session-timeout (#PCDATA)*>
<!-- Specify the directory location of the generated wsdl-->
<!ELEMENT wsdl-dir (#PCDATA)*>
<!-- Specify that wsdl generation is to happen 'force' 'httpServerURL' 'packageIt'-->
<!ELEMENT wsdl-gen (wsdl-dir,(option)*)>
<!-- Specifyg the directory location of the generated proxy-->
<!ELEMENT proxy-dir (#PCDATA)*>
<!ELEMENT option (#PCDATA)*>
<!ATTLIST option name CDATA #REQUIRED>

<!-- Specifying that proxy generation is asked for , it can have optional tags as
'include-source' 'wsdl-location' -->
<!ELEMENT proxy-gen (proxy-dir,(option)*)>
<!ELEMENT jar-generation (db-package-name,db-schema,db-url,prefix,(method-name)*)>
<!ELEMENT database-JNDI-name (#PCDATA)*>
<!ELEMENT db-package-name (#PCDATA)*>
<!ELEMENT db-url (#PCDATA)*>
<!ELEMENT db-schema (#PCDATA)*>
<!ELEMENT prefix (#PCDATA)*>
<!ELEMENT method-name (#PCDATA)*>
 <!-- specify the message style ,if this tag is not present it is considered to have 'rpc' ..it can have values
of 'rpc' or 'doc' or 'document' -->
<!ELEMENT message-style (#PCDATA)*>

<!ELEMENT connection-factory-resource-ref (#PCDATA)*>
<!ELEMENT topic-resource-ref (#PCDATA)*>
<!ELEMENT queue-resource-ref (#PCDATA)*>
<!--Resource ref of the return destination factory-->
<!ELEMENT reply-to-connection-factory-resource-ref (#PCDATA)*>
<!--Resource ref of the return destination Topic. -->
<!ELEMENT reply-to-topic-resource-ref (#PCDATA)*>
<!--Resource ref of the return destination Queue. -->
<!ELEMENT reply-to-queue-resource-ref (#PCDATA)*>
<!--jms-priority ,jms-message-type,jms-delvery-mode ,jms-expiration The JMS properties are only set for
enqueuing operations, i..e, for send operations only. -->
<!ELEMENT jms-priority (#PCDATA)*>
<!ELEMENT jms-message-type (#PCDATA)*>
<!ELEMENT jms-delivery-mode (#PCDATA)*>
<!ELEMENT jms-expiration (#PCDATA)*>
<!-- operation property is optional. Possible values for this parameter are: send, receive, and both. If not
provided, the value defaults to both. -->
<!ELEMENT operation (#PCDATA)*>
<!ELEMENT jms-doc-service
(uri,connection-factory-resource-ref,(topic-resource-ref)?,(queue-resource-ref)?,(reply-to-connection-factory-r
esource-ref)?,(reply-to-topic-resource-ref)?,(reply-to-queue-resource-ref)?,(jms-priority)?,(jms-message-type)?
,(jms-delivery-mode)?,(jms-expiration)?,(operation)?)>
```

# Web サービス・アセンブリ・ツールの制限

WebServicesAssembler ツールには、次の制限があります。

- アップロード / ダウンロードの機能はありません。Web サービス・アセンブリ・ツールでは、クライアント・システムからサーバーへの Java クラスのアップロードも、クライアント・システムへの生成された EAR ファイルのダウンロードも行われません。
- 拡張構成タスクはサポートされません。たとえば、Web サービス・アセンブリ・ツールでは、Web サービス・サーブレットのセキュリティ・オプションの制御、EJB や初期ファイルの保護または他の管理タスクの実行はできません。

# 10

# Web サービスの検出と公開

Oracle Application Server Containers for J2EE（OC4J）では、Oracle Application Server UDDI Registry（OracleAS UDDI Registry）と呼ばれる Universal Discovery Description and Integration（UDDI）Web サービス・レジストリが提供されています。

OracleAS UDDI Registry を使用すると、エンタープライズ環境の Web サービス・プロバイダ管理者は、Web サービスのコンシューマ（アプリケーション・プログラマ）が使用できるように Web サービスを公開できます。Web サービスのコンシューマは、UDDI 照会インタフェースを使用して、公開された Web サービスを検出できます。また、これらのサービスを特定のエンタープライズ・プロセスのアプリケーションで使用することもできます。

この章の主な内容は、次のとおりです。

# UDDI レジストリの理解

UDDI レジストリに用意されている情報を使用すると、次の 3 種類の検索ができます。

- ホワイト・ページの検索 : アドレス、連絡先および既知の識別子。たとえば、名前や一意の識別子（ID）など、すでになんらかの情報を知っているビジネスを検索します。
- イエロー・ページのトピック別検索 : NAICS、ISO 3166、UNSPSC など、標準的な分類に基づく産業分類。
- グリーン・ページのサービス検索 : Web サービス用インタフェースの仕様の参照など、企業により公開されている Web サービスに関する技術情報と、各種ファイルおよび URL ベースの検出メカニズムへのポインタのサポート。

UDDI レジストリでは、共通インターネット・プロトコル（TCP/IP と HTTP）、XML および SOAP など、業界標準に準拠したテクノロジが使用されています。これらのテクノロジは、単純なメッセージ・ベースの情報交換に XML を使用するための仕様です。UDDI は標準的な Web サービスの記述形式および Web サービス検出プロトコルであり、UDDI レジストリにはあらゆるタイプのサービスのメタデータを、Web Services Description Language（WSDL）で記述されたメタデータについて最良と定義された方法で含めることができます。

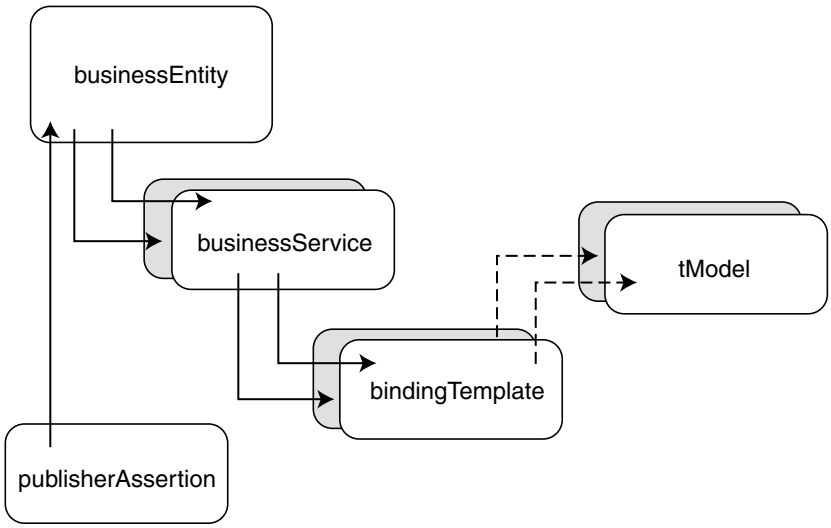
## UDDI レジストリのデータ構造

UDDI レジストリは次の 5 つのデータ構造タイプで構成されており、迅速に検索して理解できるようにレジストリ情報がグループ化されています。

- businessEntity: トップレベルの論理的な親データ構造です。このデータ構造には、ビジネス・サービス、カテゴリ、連絡先、検出用 URL および識別子など、Web サービスに関する情報と、検索に役立つカテゴリ情報を公開するビジネスの記述情報が含まれています。
- businessService: 単一の businessEntity データ構造の論理的な子であり、bindingTemplate 構造の論理的な親です。このデータ構造には、名前、簡単な説明、技術サービスの記述および検索に役立つカテゴリ情報など、技術サービスの特定ファミリに関するビジネス・サービスの記述情報が含まれています。
- bindingTemplate: 単一の businessService データ構造の論理的な子です。このデータ構造には、Web サービスのエントリ・ポイントとインタフェース仕様の参照に関する技術情報が含まれています。
- tModel: Web サービスの仕様や技術的な識別の基礎となる分類の説明です。このデータ構造は、Web サービスの技術仕様を表します。Web サービス・コンシューマ（プログラマ）が特定の技術仕様と互換性のある登録済 Web サービスを検索できるようにします。つまり、Web サービス・コンシューマは、tModel データ構造に含まれる Web サービスの仕様の記述に基づいて、他の互換 Web サービスを簡単に識別できます。
- publisherAssertion: 一方または双方が表明する 2 者間の関係についての情報です。

図 10-1 に、UDDI 情報モデルと、その 5 つのデータ構造タイプの関係を示します。

**図 10-1 UDDI レジストリ情報モデル**




UDDI レジストリでは XML と SOAP を使用しているため、これらのデータ構造にはそれぞれ、ビジネス記述や技術的な用途を持った多数の要素と属性が含まれています。

UDDI サービス記述のフレームワークの詳細は、次の URL で入手できる『UDDI Version 2.03, Data Structure Reference Published Specification, Dated 19 July 2002』および『UDDI Version 2.04 API, Published Specification Dated 19 July 2002』を参照してください。

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

# OracleAS UDDI Registry の概要

OracleAS UDDI Registry を使用すると、Web サービス・プロバイダ管理者は、Web サービスのコンシューマ（プログラマ）が使用できるように Web サービスを公開できます。Web サービスのコンシューマは、UDDI 照会インタフェースを使用し、OracleAS UDDI Registry を参照、検索およびドリルダウンして、公開された Web サービスを検出できます。また、コンシューマは、登録されている Web サービスの中から 1 つ以上を選択し、特定のエンタープライズ・プロセスのアプリケーションで使用することもできます。

たとえば、管理者は、すべてのメタデータとインタフェース仕様へのポインタを OracleAS UDDI Registry に提供することで、Web サービスを公開できます。また、J2EE スタック（Enterprise JavaBeans（EJB）、JavaBeans、JavaServer Pages（JSP）またはサーブレット）を使用して Web サービスの実装を完了し、その実装を Simple Object Access Protocol（SOAP）に基づいた Web サービスとして公開しているコンシューマと協力して作業することもできます。このようにして、管理者は Web サービスの可用性を公開し、コンシューマが Web サービスを検索して独自のアプリケーション用に選択できるようにします。

OracleAS UDDI Registry は、次の仕様を含め、UDDI バージョン 2 仕様に準拠しています。

- UDDI バージョン 2.04 API 仕様
- UDDI バージョン 2.03 データ構造リファレンス仕様
- UDDI バージョン 2.03 レプリケーション仕様

OC4J にデプロイされる Web サービス用の OracleAS UDDI Registry サポートは、次のパートに分かれています。

- Web サービスの検出 : コンシューマは、照会 API を使用して独自の Web サービス検出ツールを実装できます。このツールでは、OracleAS UDDI Registry 内と、アクセス可能な他の UDDI v1.0 仕様または v2.0 仕様と互換性のある UDDI レジストリ内で、J2EE Web サービ

スの検索、特定およびドリルダウンができます。照会 API の使用方法および Javadoc マニュアルの検索方法の詳細は、10-10 ページの「OracleAS UDDI Registry 照会 API の使用」を参照してください。

- Web サービスの公開 : 管理者は、Oracle Enterprise Manager 10*g* を使用して J2EE Web サービスをデプロイできます。このデプロイ・プロセスの一部として、管理者は Web サービスを OracleAS UDDI Registry に公開できます。

  プログラマは、公開 API を使用し、主要な 5 つの UDDI データ構造（businessEntity、businessService、bindingTemplate、tModel および publisherAssertion）ごとに保存コールと削除コールを提供して、Web サービスを公開できます。公開 API の使用方法および Javadoc マニュアルの検索方法の詳細は、10-28 ページの「OracleAS UDDI Registry の公開 API の使用」を参照してください。

- Web サービスの更新 : 管理者は、Oracle Enterprise Manager 10*g* を使用して、J2EE Web サービスを検索、特定およびドリルダウンし、公開されている Web サービスを更新できます。

- 追加ツールの開発 : コンシューマは、Java ベースのクライアント・ライブラリを使用して、追加ツールやアプリケーションの開発を円滑に進めることができます。この API の詳細は、『Oracle Application Server Web Services UDDI Client API Reference』を参照してください。

- レプリケーション管理 : 管理者は、1 つ以上の OracleAS UDDI Registry 実装、および UDDI v2.03 レプリケーション仕様も実装している他のベンダーからの UDDI 実装で構成される論理的なレジストリを作成できます。詳細は、10-40 ページの「UDDI レプリケーション」を参照してください。

- データベース・サポート : Microsoft SQL Server、IBM DB2 および Oracle（インフラストラクチャ以外の）データベース用に、UDDI オープン・データベース・サポートが提供されています。詳細は、10-49 ページの「UDDI オープン・データベース・サポート」を参照してください。

## 標準的な分類 / 識別子体系のサポート

OracleAS UDDI Registry では、tModels、businessEntities および businessServices を分類し（カテゴリに分け）、tModels および BusinessEntities を識別するための標準的な分類がサポートされています。

OracleAS UDDI Registry には、tModels、businessEntities および businessServices をカテゴリに分けるための、次の組込み済およびチェック済の標準的な分類が用意されています。

- North American Industry Classification System（NAICS）1997 リリース

  これは、各産業および対応するコードの分類体系です。NAICS の詳細は、次の Web サイトを参照してください。

  `http://www.census.gov/epcd/www/naics.html`

- United Nations Standard Products and Services Codes（UNSPSC）バージョン 7.3

  これは、国際市場全体を対象として製品とサービスの両方を分類する初のコーディング・システムです。UNSPSC の詳細は、次の Web サイトを参照してください。

  `http://www.unspsc.org/`

- ISO 3166 Geographic Code System（ISO 3166）

  これはすべての国および地域のリストです。ISO 3166 の詳細は、次の Web サイトを参照してください。

  `http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/index.html`

管理者は、Web サービスを公開するときに、登録先となる分類とカテゴリを選択できます。Web サービスをこれらの分類の一部またはすべてに公開するか、各分類の必要な数のカテゴリとサブカテゴリに公開するかをオプションで選択できます。

また、OracleAS UDDI Registry には、次の組込みの識別子体系も用意されています。

- Dun & Bradstreet D-U-N-S Number Identifier System（D-U-N-S）

  この体系では、一意の 9 桁の連番を使用して、世界各国の企業を識別しています。詳細は、次の Web サイトを参照してください。

  `http://www.dnb.com`

- Thomas Register Supplier Identifier Code System

  この体系では、製造会社およびサプライヤの識別コードを提供しています。詳細は、次の Web サイトを参照してください。

  `http://www.thomasregister.com/`

表 10-1 に分類、名前および tModel キーの Universal Unique Identifier（UUID）を示します。

**表 10-1 分類 / 識別子システム**

| 分類 | 名前 | tModel キー |
|---|---|---|
| NAICS | ntis-gov:naics:1997 | uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2 |
| UNSPSC | unspsc-org:unspsc | uuid:CD153257-086A-4237-B336-6BDCBDCC6634 |
| ISO 3166 | uddi-org:iso-ch:3166-1999 | uuid:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88 |
| D-U-N-S | dnb-com:D-U-N-S | uuid:8609C81E-EE1F-4D5A-B202-3EB13AD01823 |
| Thomas Register | thomasregister-com:supplierID | uuid:B1B1BAF5-2329-43E6-AE13-BA8E97195039 |

分類の詳細は、次の Web サイトを参照してください。

`http://www.uddi.org/taxonomies/UDDI_Taxonomy_tModels.htm`

## UUID 生成

OracleAS UDDI Registry では、乱数からバージョン 4 の Unique Universal Identifier（UUID）を生成するアルゴリズムを使用します。

UDDI v2 仕様に定められている組込み tModel データ構造は、すべて含まれます。OracleAS UDDI Registry 自体を表すブートストラップ・ノードの businessEntity を分類するために、UDDI v2 仕様に定義されている追加の tModel データ構造 `uddi-org:operators` も含まれます。

# OracleAS UDDI Registry の使用開始

この項では、OracleAS UDDI Registry の使用開始方法について説明します。内容は、次のとおりです。



## OracleAS UDDI Registry の構成

Oracle Application Server Infrastructure と、OracleAS Portal および Wireless 中間層をインストールすると、OracleAS UDDI Registry がインストールされます。OracleAS UDDI Registry は OC4J_Portal インスタンスに自動的に配置され、UDDI データベース・スキーマは OracleAS Infrastructure データベースに組み込まれます。

OracleAS UDDI Registry を OC4J スタンドアロンまたは Oracle Application Server Core インストールとともにインストールする場合、次の URL の OTN でスタンドアロンの OracleAS UDDI Registry キットを参照してください。

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

OracleAS UDDI Registry を初期化および構成するには、ブラウザ経由またはプログラムで SOAP を起動し、UDDI 照会サーブレットのエンド・ポイントまたは公開 SOAP のエンド・ポイントにアクセスする必要があります。別の方法では、統合された Web サービス公開を含め、Oracle Enterprise Manager 10*g* から OracleAS UDDI Registry を使用できなくなります。

UDDI 照会サーブレットのエンド・ポイントをブラウザから ping して、OracleAS UDDI Registry を初期化および構成するには、次の URL を入力します。

```
http://OracleAS-host:OracleAS-port/uddi/inquiry
```

「OracleAS UDDI Registry」ページが表示されます。「Welcome! Your registry is now up and running.」というメッセージが表示されます。

このインストールおよび構成のステップでは、次の設定が行われます。

- UDDI コア tModel データ構造
- レジストリ・ノードを表す businessEntity ノード
- businessEntity の discoveryURL プリフィックスおよび operatorName プロパティ

デフォルトでは、インストール時に UDDI ユーザーおよびユーザー・グループが作成されます。表 10-2 に、ユーザーのタイプ、ユーザー名およびパスワードを示します。

**表 10-2 デフォルトの UDDI ユーザーおよびパスワード**

| タイプ | ユーザー名 | デフォルト・パスワード |
|---|---|---|
| 管理者 | ias_admin | ias_admin123 |
| パブリッシャ | uddi_publisher | uddi_publisher123 |
| パブリッシャ | uddi_publisher1 | uddi_publisher1 |
| レプリケータ | uddi_replicator | パスワードなし（明示的に使用せず） |

これらのユーザー名およびパスワードを使用して、UDDI のエンド・ポイントに接続できます。インストール時に設定される UDDI ユーザーおよびグループについては、10-35 ページの「ユーザーの管理」で詳しく説明します。

OracleAS UDDI Registry は、次の URL から使用できます。

- ベースとなる情報 :

  http://*OracleAS-host*:*OracleAS-port*/uddi/

- UDDI 照会 SOAP のエンド・ポイント

  http://*OracleAS-host*:*OracleAS-port*/uddi/inquiry

- UDDI 公開 SOAP のエンド・ポイント

  http://*OracleAS-host*:*OracleAS-port*/uddi/publishing

- UDDI 管理のエンド・ポイント

  http://*OracleAS-host*:*OracleAS-port*/uddi/admin

- UDDI レプリケーション SOAP のエンド・ポイント

  http://*OracleAS-host*:*OracleAS-port*/uddirepl/replication

- UDDI レプリケーション HTTPS の Wallet パスワード管理のエンド・ポイント

  http://*OracleAS-host*:*OracleAS-port*/uddirepl/admin/wallet

## インストール時または初期のプロパティの変更

管理操作の多くは、コマンドライン・ツール uddiadmin.jar を使用して実行します。このツールは、UNIX の場合は uddi/lib/uddiadmin.jar ファイル、Windows の場合は uddi¥lib¥uddiadmin.jar ファイルにあります。通常、コマンドライン・ツールでは次の形式を使用します。

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
[-verbose] options_and_their_parameters
```

コマンドライン・ツール uddiadmin.jar の setProperty オプションを使用すると、データベース接続の最大数やデフォルト言語など、構成プロパティの値を設定できます。

次の 2 つのプロパティは、インストール直後に変更する必要があります。ただし、これらのプロパティは一度変更したら、ホストの設定に変更がある場合またはステージング環境から本番環境にシステムを移行する場合以外は、二度と変更しないでください。

- operatorName: OracleAS UDDI Registry のオペレータの名前を指定します。この名前はレスポンスの operator 属性に表示されます。このプロパティ設定は、データベース内の既存のエンティティに遡及して適用されます。たとえば、オペレータ名を変更すると、すべてのビジネスおよび tModel データ構造の古いオペレータ名が、新しいオペレータ名に変更されます。

  次の例では、operatorName プロパティを OracleUddiServerIT_Dept に設定しています。

  ```
  java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
   -setProperty oracle.uddi.server.operatorName=OracleUddiServerIT_Dept
  ```

- businessEntityURLPrefix: レジストリに保存される各 businessEntity データ構造に対して自動的に生成される discoveryURL のプリフィックスを指定します。このプリフィックスは、デプロイ環境に応じてカスタマイズする必要があります。このパラメータ設定は、データベース内の既存のエンティティに遡及して適用されます。たとえば、discoveryURL のプリフィックスを変更すると、すべての businessEntity usetype の古い discoveryURL が新しい discoveryURL に変更されます。

  ホスト名とポートには、Web サーバーのホスト名とポートを使用する必要があります (サーブレット・コンテナと同じでも異なっていてもかまいません)。

  次の例では、discoveryURL のプリフィックスを http://*uddihost:port*/uddi/inquiryget に設定しています。

  ```
  java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
  -setProperty oracle.uddi.server.businessEntityURLPrefix=http://uddihost:port/uddi/
  inquiryget
  ```

**注意：** これらのプロパティは、UDDI レプリケーションを有効にする前に設定してください。

また、defaultLang プロパティを使用して、レジストリのデフォルト言語を設定できます。詳細は、10-71 ページの「defaultLang」を参照してください。

コマンドライン・ツールの詳細は、10-33 ページの「コマンドライン・ツール uddiadmin.jar の使用」を参照してください。`setProperty` オプションの詳細は、10-65 ページの「setProperty」を参照してください。

## 本番環境での考慮事項

この項では、インストール直後に実行する必要がある、インストール後の構成手順について説明します。これらの手順は必須ではありませんが、本番環境では実行することをお薦めします。

- **エンド・ポイント公開のセキュリティ：**デフォルトでは、HTTP アクセスが使用可能です。ただし、セキュリティを考慮して HTTPS アクセスの使用をお薦めします。HTTP アクセスを使用不可にする方法の詳細は、10-49 ページの「トランスポート・セキュリティ」を参照してください。
- **データベース接続プールのサイズ変更および文キャッシュ：**実際のデータベース・サーバーの負荷に対処するために、データベース接続の最大数や文キャッシュの使用方法など、データベース接続プールのパラメータを構成する必要があります。

  OracleAS Infrastructure 以外の Oracle データベースをバックエンド記憶域として使用している場合、パラメータは、データ・ソース `jdbc/OracleUddi` を編集することによって構成できます。詳細は、『Oracle Application Server Containers for J2EE サービス・ガイド』の「データ・ソース」を参照してください。

  OracleAS Infrastructure データベースをバックエンド記憶域として使用している場合、パラメータは、次の UDDI サーバー構成プロパティを変更することによって構成できます。

  - minConnections
  - maxConnections
  - jdbcDriverType
  - stmtCacheType
  - stmtCacheSize
- **operatorName および businessEntity の discoveryURL プリフィックスの変更：**システムをステージング環境から本番環境に移行するとき、businessEntity の discoveryURL プリフィックスまたは operatorName（あるいは両方のパラメータ値）の変更が必要になる場合があります。詳細は、10-69 ページの「businessEntityURLPrefix」および 10-75 ページの「operatorName」を参照してください。

# Web サービスの検出

OracleAS UDDI Registry で Web サービスを検出するには、ツールまたは照会 API を使用してレジストリを参照します。これらのメソッドは、次の各項で説明します。

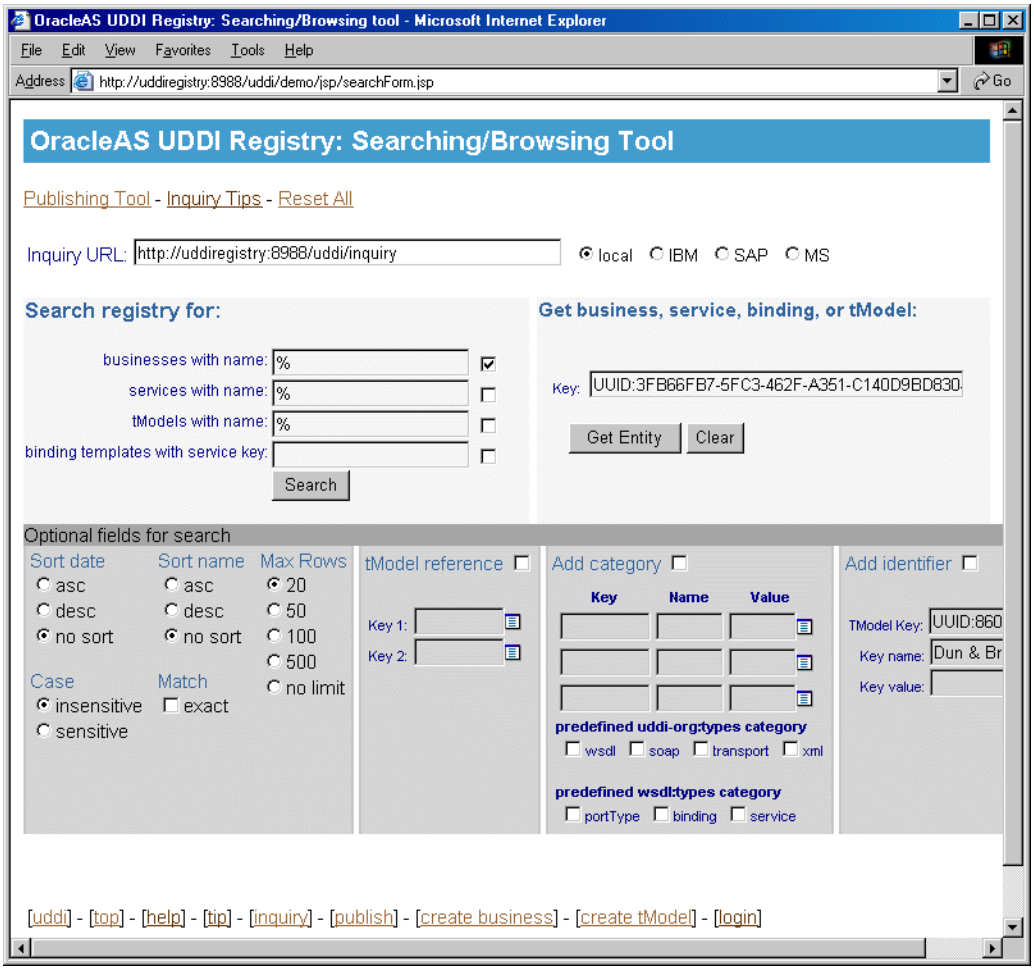
- OracleAS UDDI Registry の検索および参照ツールの使用
- 他のツールを使用した Web サービスの検出
- OracleAS UDDI Registry 照会 API の使用

## OracleAS UDDI Registry の検索および参照ツールの使用

OracleAS UDDI Registry には、検索および参照ツールが用意されています。このツールを使用すると、businessEntity、businessService、tModel または bindingTemplate 別にレジストリを検索できます。検索および参照ツールにアクセスするには、次の URL を入力します。

`http://OracleAS-host:OracleAS-port/uddi`

次に、「**UDDI Inquiry/Publishing tool**」のリンクをクリックします。次の図に示すような「Searching and Browsing Tool」ページが表示されます。



## 他のツールを使用した Web サービスの検出

コンシューマとして Oracle JDeveloper を使用し、OracleAS UDDI Registry を参照できます。また、サード・パーティのツールを使用すると、OracleAS UDDI Registry およびアクセス可能な他の UDDI v1.0 の Web サービス・レジストリから Web サービスの情報を参照およびドリルダウンできます。

## OracleAS UDDI Registry 照会 API の使用

Java プログラマが使用できる照会 API をコンシューマとして使用し、独自の Web サービス検出インタフェースを実装できます。照会 API は、UDDI クライアント API の一部で、図 10-1 の 5 つの各データ構造にある情報を検索して取得するための `find`（参照およびドリルダウン）コールおよび `get` コールを提供します。

照会 API は、Java ベースの API で、Java プログラマの便宜を図るために用意されています。ただし、プログラムを任意の言語で記述し、SOAP を使用して Web サービスを検出することもできます。

OracleAS UDDI Registry の照会 API の URL は、次のとおりです。

`http://`*`OracleAS-http-server-hostname`*`:`*`OracleAS-port`*`/uddi/inquiry`

URL の *`OracleAS-http-server-hostname`* は、Oracle HTTP Server がインストールされているホストです。*`OracleAS-port`* は、Oracle HTTP Server のポート番号です。

照会 API は、Oracle Application Server のインストール・ディレクトリにあります。このディレクトリは、UNIX の場合は `${`*`ORACLE_Home`*`}/uddi/`、Windows の場合は *`%ORACLE_Home_ORACLE`*`%¥uddi¥` です。照会 API のリファレンス・マニュアルについては、『Oracle Application Server Web Services UDDI Client API Reference』を参照してください。

サンプル・デモ・ファイルのセット（`uddidemo.zip`）は、次の URL の Oracle Technology Network（OTN）の Web サイトにあります。

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

`uddidemo.zip` ファイルには Java プログラム・ファイル `UddiInquiryExample.java` が含まれています。Java プログラマは、このファイルを基にして OracleAS UDDI Registry のクライアント・ライブラリを使用し、主要な構成とシーケンスのデモを実行できます。

サンプル・プログラムで実行される内容は、次のとおりです。

- SoapTransportLiaison インタフェースの実装を取得します。これは、SOAP および基礎となるトランスポート・プロトコル（この場合は HTTP）を使用して、UDDI クライアントとサーバー間で通信の詳細を処理する実装です。

  ```
  SoapTransportLiaison transport = new OracleSoapHttpTransportLiaison();
  ```

- 必要に応じて、ヘルパー・メソッドをコールし、プロキシ情報を設定します。`-Dhttp.proxyHost=`*`hostname`* `-Dhttp.proxyPort=`*`portnum`* などのパラメータを使用すると、OracleAS UDDI Registry にアクセスするための HTTP プロキシ情報をコマンドラインで指定できます。

  ```
  setHttpProxy((SoapHttpTransportLiaison)transport);
  ```

- SoapTransportLiaison インスタンスと UDDI 照会レジストリの URL を使用して、指定の OracleAS UDDI Registry に接続する UddiClient のインスタンスを初期化します。UddiClient インスタンスは、クライアントから OracleAS UDDI Registry へのリクエスト送信に使用されるプライマリ・インタフェースです。

  ```
  UddiClient uddiClient = new UddiClient(szInquiryUrl, null, transport);
  ```

**注意：** UddiClient インスタンスは、デフォルトでは、UDDI v2 クライアント（サポートされている最新バージョン）として動作します。特定のバージョンが必要な場合は、別のコンストラクタ、または JVM プロパティ oracle.uddi.client.defaultVersion を使用してバージョンを指定できます。

次に例を示します。

```
-Doracle.uddi.client.defaultVersion=1
```

- UddiClient インスタンスを使用してビジネス検索リクエストを実行します。特に、英字 T で始まるビジネス・エンティティをすべて検索し、レスポンスに出力します。入力パラメータと戻り値は、UDDI 仕様に定義されている XML 要素を正確に疑似実行するオブジェクトであることに注意してください。

```
// Find a business with a name that starts with "T"
String szBizToFind = "T";
System.out.println("¥nListing businesses starting with " + szBizToFind);
// Actual find business operation:
// First null means no specialized FindQualifier.
// Second null means no max number of entries in response.
// (For example, maxRows attribute is absent.)
BusinessList bl = uddiClient.findBusiness(szBizToFind, null, null);
// Print the response.
System.out.println("The response is: ");
 List listBusinessInfo = bl.getBusinessInfos().getUddiElementList();
      for (int i = 0; i < listBusinessInfo.size(); i++) {
        BusinessInfo businessInfo = (BusinessInfo)listBusinessInfo.get(i);
        System.out.println(businessInfo.getBusinessKey());
        Name name = businessInfo.getFirstNameAsName();
        if (name != null) {
          System.out.println("name=" + name.getContent() +
                             " ; xml:lang=" + name.getLang());
        }

 Description description =
    businessInfo.getFirstDescriptionAsDescription();
 if (description != null) {
    System.out.println("description=" + description.getContent()
                  +  " ; xml:lang=" + description.getLang());
 }
```

- UddiClient インスタンスを使用して UddiElementFactory インスタンスを取得します。照会に必要な UDDI オブジェクトの作成には、常にこのファクトリを使用する必要があります。

```
UddiElementFactory uddiEltFactory = uddiClient.getUddiElementFactory();
```

- UddiElementFactory インスタンスを使用して、検索に使用する CategoryBag インスタンスとその KeyedReference を作成します。

```
CategoryBag cb = (CategoryBag)uddiEltFactory.createCategoryBag();
KeyedReference kr =
(KeyedReference)uddiEltFactory.createKeyedReference();
kr.setTModelKey(szCategoryTModelKey);
kr.setKeyValue(szCategoryKeyValue);
kr.setKeyName("");
cb.addUddiElement(kr);
```

- UddiClient インスタンスを使用してサービス検索リクエストを実行します。特に、すべてのビジネス・エンティティ（businessKey が未指定）で、UNSPSC 分類によりアプリケーション・サービス・プロバイダ（コード 81.11.21.06.00）として分類されるサービスを最大 30 まで検索します。

```
ServiceList serviceList =
      uddiClient.findService("", cb, null, new Integer(30));
```

- UddiElementFactory インスタンスを使用して XmlWriter オブジェクトを取得します。UddiElement を拡張するオブジェクトによって表される RAW XML データを表示するには、要素の内容をライターにマーシャリングし、ライターをフラッシュしてクローズします。

```
XmlWriter writerXmlWriter =
        uddiEltFactory.createWriterXmlWriter(new PrintWriter(System.out));
serviceList.marshall(writerXmlWriter);
writerXmlWriter.flush();
```

- 複数の引数を持つ tModel 操作を検索します。これは、UDDI v2 の機能です。find_xx リクエストで複数の引数を使用できるようになりました。たとえば、名前パターンが "uddi%inquiry%" で、uddi-org:types 分類で wsdlSpec または xmlSpec に分類されている tModel 操作を検索できます。

```
System.out.println("¥nListing tModels with the name pattern \"uddi%inquiry%\" ");
System.out.println("and classified as ¥"wsdlSpec¥" or ¥"xmlSpec¥" ");
System.out.println("under uddi-org:types taxonomy.");
// Use UddiElement factory to create UDDI-specific objects
// that are needed in inquiries.
CategoryBag cbTM = (CategoryBag)uddiEltFactory.createCategoryBag();
KeyedReference krTM1 =
 (KeyedReference)uddiEltFactory.createKeyedReference();
krTM1.setTModelKey(CoreTModelConstants.TAXONOMY_KEY_UDDI_TYPE);
krTM1.setKeyValue(CoreTModelConstants.UDDI_TYPE_VALUE_WSDL_SPEC);
cbTM.addUddiElement(krTM1);

KeyedReference krTM2 =
 (KeyedReference)uddiEltFactory.createKeyedReference();
krTM2.setTModelKey(CoreTModelConstants.TAXONOMY_KEY_UDDI_TYPE);
krTM2.setKeyValue(CoreTModelConstants.UDDI_TYPE_VALUE_XML_SPEC);
cbTM.addUddiElement(krTM2);

FindQualifiers fqTM = (FindQualifiers)uddiEltFactory.createFindQualifiers();
List listFQTM = uddiEltFactory.createList();
listFQTM.add(FindQualifiers.OR_ALL_KEYS);
fqTM.setFindQualifierStringList(listFQTM);

// Actual find tModel operation:
// Integer(10) means a maximum of 10 tModel operations are to be returned.
TModelList tModelList =  uddiClient.findTModel("uddi%inquiry%",
                                               null,
                                               cbTM,
                                               fqTM,
                                               new Integer(10));

// Print some response information.
System.out.println("The response is: ");
List listTModelInfo = tModelList.getTModelInfos().getUddiElementList();
for (int i = 0; i < listTModelInfo.size(); i++) {
  TModelInfo tModelInfo = (TModelInfo)listTModelInfo.get(i);
  System.out.println(tModelInfo.getTModelKey());
  System.out.println("name=" + tModelInfo.getName());
 }
```

- 完了後、リソースを解放するために UddiClient インスタンスをクローズします。

```
uddiClient.close();
```

- OracleAS UDDI Registry と 4 つのパブリック UDDI レジストリに URL を（コメントで）提供します。

# Web サービスの公開

Web サービスは、次のいずれかのインタフェースを使用して OracleAS UDDI Registry に公開できます。

- Oracle Enterprise Manager 10*g*。10-13 ページの「Web サービスの公開における Oracle Enterprise Manager の使用」を参照してください。
- OracleAS UDDI Registry の公開ツール。10-19 ページの「OracleAS UDDI Registry の公開ツールの使用」を参照してください。
- OracleAS UDDI Registry の公開 API。10-28 ページの「OracleAS UDDI Registry の公開 API の使用」を参照してください。

## Web サービスの公開における Oracle Enterprise Manager の使用

Oracle Enterprise Manager 10*g* を使用すると、管理者は、Web サービスを OracleAS UDDI Registry に公開したり、検出された Web サービスを更新したりできます。

- アプリケーションのデプロイ・ウィザードの使用。アプリケーションのデプロイ・ウィザードの指示に従って、J2EE アプリケーションを OC4J コンテナにデプロイします。

  10-13 ページの「アプリケーションのデプロイ・ウィザードを使用した Web サービスの公開」を参照してください。

- 「UDDI レジストリ」ページの使用。「UDDI レジストリ」ページを使用すると、公開された Web サービスを検出し、更新できます。

  10-16 ページの「OracleAS UDDI Registry の公開済 Web サービスの更新」を参照してください。

---

**注意：** Oracle Enterprise Manager で UDDI の機能を使用するには、UDDI 照会サーブレットのエンド・ポイントを ping して、UDDI レジストリを初期化しておく必要があります。詳細は、10-6 ページの「OracleAS UDDI Registry の構成」を参照してください。

UDDI レジストリを初期化する前に UDDI 関連ページにアクセスした場合は、Oracle Enterprise Manager を再起動する必要があります。

---

### アプリケーションのデプロイ・ウィザードを使用した Web サービスの公開

管理者として Oracle Enterprise Manager のアプリケーションのデプロイ・ウィザードを使用し、OracleAS Web Services のアセンブリ・ツールで生成された J2EE Web サービスを公開できます。

J2EE Web サービスを公開するには、最初に J2EE エンタープライズ・アーカイブ（EAR）ファイルとしてアセンブルする必要があります。詳細は、第 9 章を参照してください。J2EE Web アプリケーションの EAR ファイル・ベースのデプロイの詳細は、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』を参照してください。

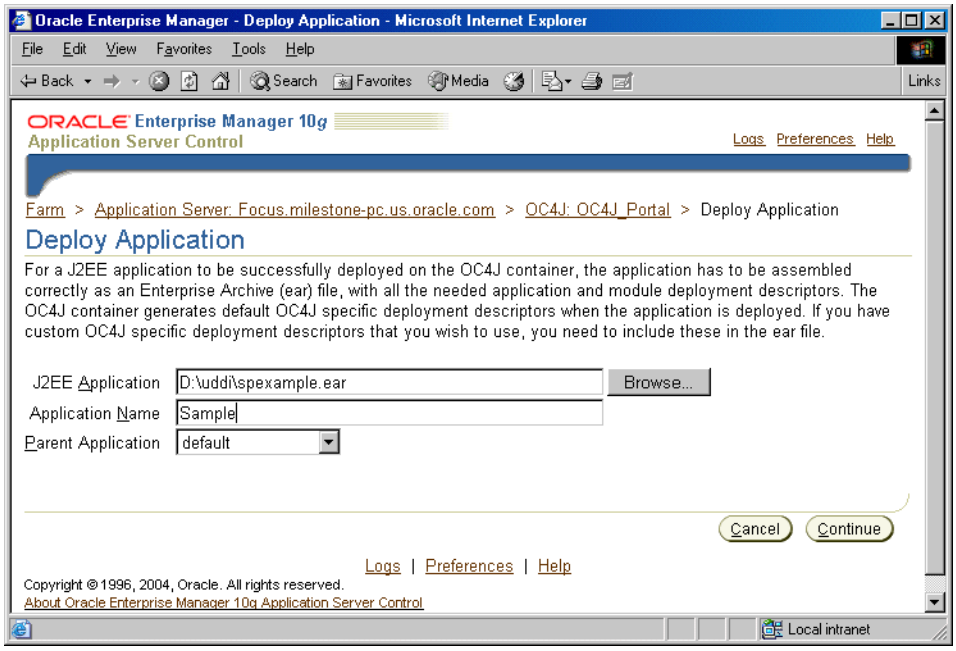
ウィザードを使用して、EAR ファイル内の Web サービス・サーブレットを公開できます。アクセス先となるアプリケーション内のすべての Web サービス・サーブレットを、OracleAS UDDI Registry に用意されている 1 つ以上の分類の 1 つ以上の必要なカテゴリに公開する必要があります。アプリケーション内の非公開の Web サービス・サーブレットは「未公開」ステータスで表示されます。公開されると、Web Services のステータスは「公開済」に変更されます。

OracleAS UDDI Registry を初期化したら、次の手順に従います。

1. Oracle Enterprise Manager を起動し、「アプリケーション・サーバー : < インスタンス名 >」ページにナビゲートします。「**システム・コンポーネント**」表から OC4J インスタンスを選択します。デフォルトでは、UDDI レジストリは、OC4J_PORTAL インスタンスに配置されます。
2. 「OC4J: <OC4J 名 >」ページで、「**アプリケーション**」をクリックします。

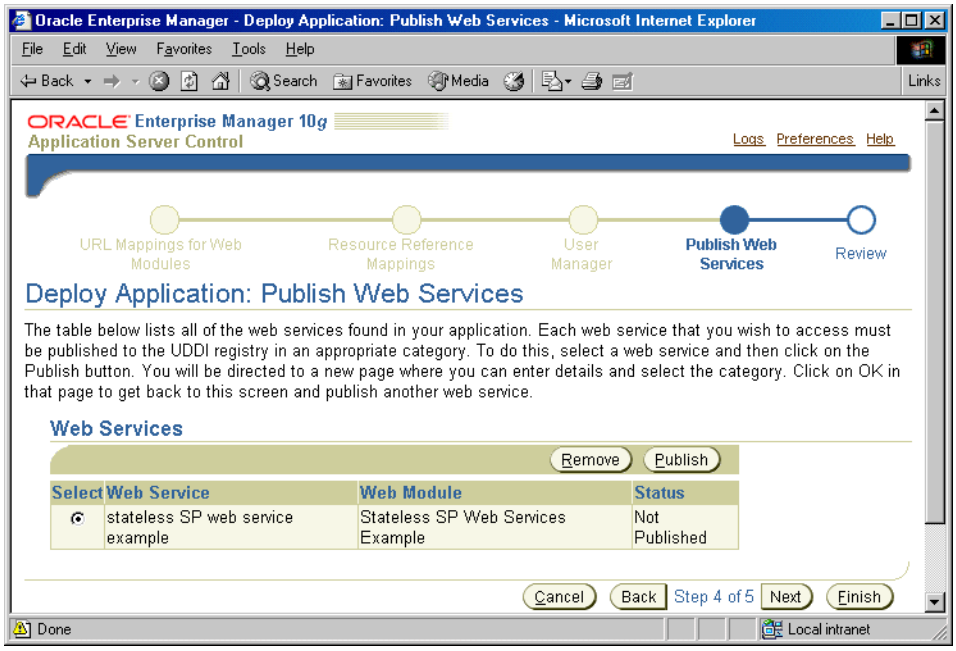
3. 「**デプロイ済アプリケーション**」セクションで、「**EAR ファイルのデプロイ**」をクリックして**アプリケーションのデプロイ**・ウィザードを起動します。

4. 次の図に示すように、**アプリケーションのデプロイ**・ウィザードの「アプリケーションのデプロイ」ページで、EAR ファイルの場所を「**J2EE アプリケーション**」フィールドに、アプリケーション名を「**アプリケーション名**」フィールドに指定します。




「**続行**」をクリックします。

5. 次ページ以降、次の図に示す「Web サービスの公開」ページが表示されるまで、デフォルトを使用します。



ウィザードで「Web サービスの公開」ページが表示されない場合は、UDDI レジストリが初期化されていないか、または OracleAS Infrastructure が実行されていません。UDDI レジストリの初期化の詳細は、10-6 ページの「OracleAS UDDI Registry の構成」を参照してください。その場合は、Oracle Enterprise Manager の再起動が必要になることがあります。

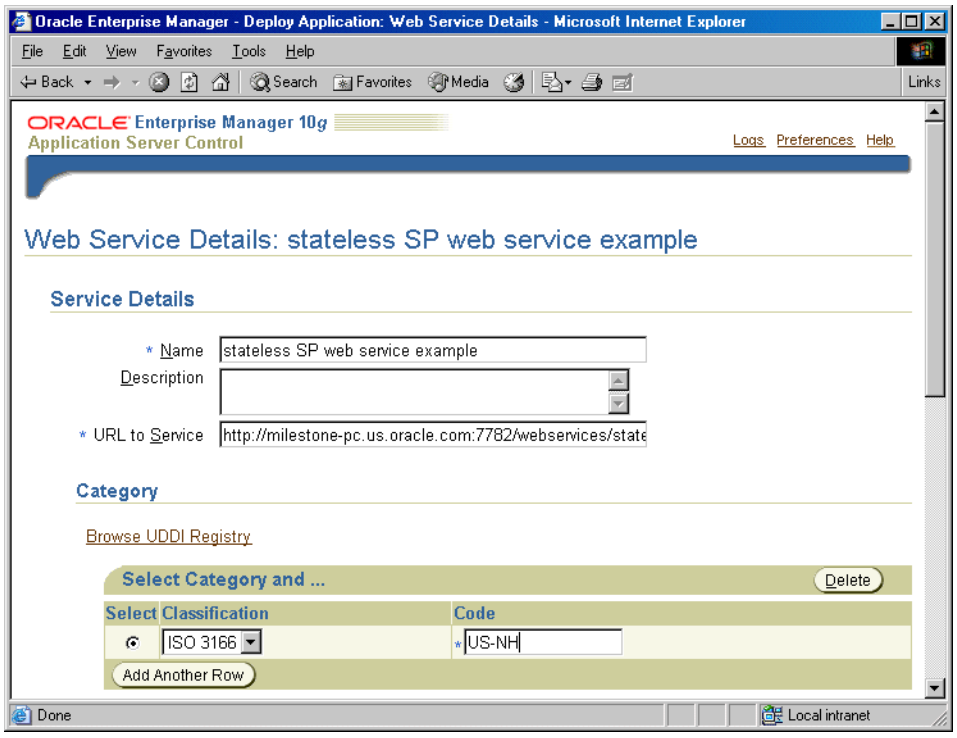
6. 「Web サービスの公開」ページで、アプリケーションに認識されており、「**ステータス**」が「**未公開**」になっている Web サービスのリストから、登録対象の Web サービスを選択します。

   次に、「**公開**」をクリックして「Web サービスの詳細」ページに進みます。

7. 「Web サービスの詳細」ページで、「**サービスの詳細**」セクションと「**TModel の詳細**」セクションの各フィールドの情報を確認し、必要に応じて編集または入力します。「**名前**」と「**サービスへの URL**」には、値が自動的に挿入されます。

   - カテゴリを指定するには、「カテゴリ」セクションで「**分類**」と「**コード**」の値を選択します。
   - サービスまたは tModel の登録先となるカテゴリを指定するには、「**サービスの詳細**」セクションまたは「**TModel の詳細**」セクションで、「**UDDI レジストリの参照**」をクリックします。次に、必要な分類を参照し、必要に応じて各カテゴリをドリルダウンして、対象となるカテゴリ名と値をすべてメモします。
   - 空のカテゴリ情報行を追加するには、「**行の追加**」をクリックします。必要な分類を選択し、必要なカテゴリの値コードおよび対応するカテゴリ名を入力します。

   次の図に、「Web サービスの詳細」ページの上部を示します。




8. 「Web サービスの詳細」ページに必要な情報をすべて入力して「**OK**」をクリックし、Web サービスを OracleAS UDDI Registry に公開します。「Web サービスの公開」ページに戻ります。

9. 「Web サービスの公開」ページで、公開する Web サービスをもう 1 つ選択し、手順 6 および手順 7 で説明したすべての処理を繰り返します。

10. このアプリケーションに使用する Web サービスをすべて公開したら、「**次へ**」をクリックして「確認」ページに進みます。このページでは、アプリケーション・デプロイ情報を確認できます。

11. すべての変更を完了した後、「**デプロイ**」をクリックして、J2EE アプリケーションを OC4J コンテナにデプロイします。この操作によって、Oracle Enterprise Manager の OC4J ホーム・ページに戻ります。

デプロイ後は、公開対象として選択した Web サービスを記述するメタデータが、OracleAS UDDI Registry に追加されています。

## OracleAS UDDI Registry の公開済 Web サービスの更新

Oracle Enterprise Manager を使用すると、OracleAS UDDI Registry のカテゴリに公開されている Web サービスの情報を参照、ドリルダウンおよび取得できます。検出された公開済 Web サービスは更新できます。

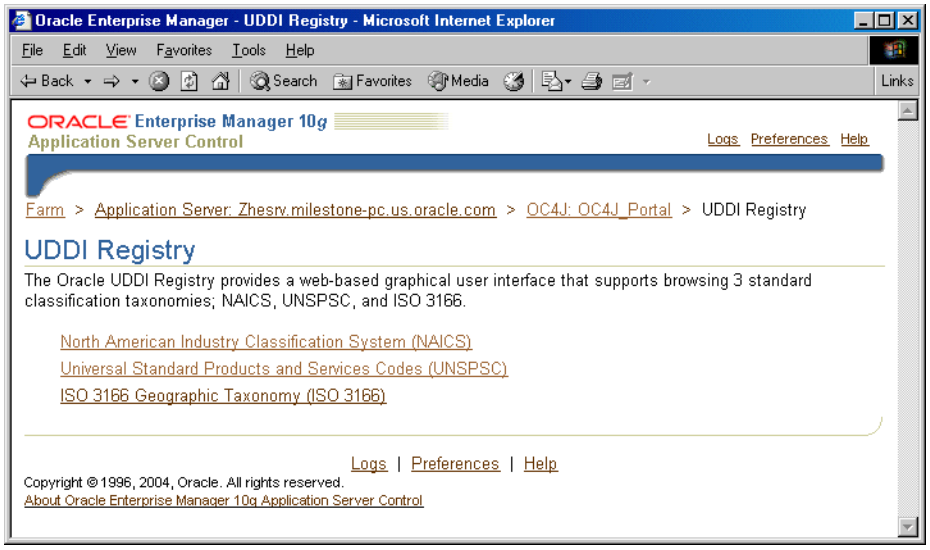
Oracle Enterprise Manager 10*g* を使用して公開済 Web サービスを更新する手順は、次のとおりです。

1. Oracle Enterprise Manager を起動し、「アプリケーション・サーバー : < インスタンス名 >」ページにナビゲートします。「**システム・コンポーネント**」表から OC4J インスタンスを選択します。

2. 「OC4J: <OC4J 名 >」ページで、「**管理**」リンクをクリックします。「管理」ページで、「**関連リンク**」セクションの「**UDDI レジストリ**」リンクをクリックします。

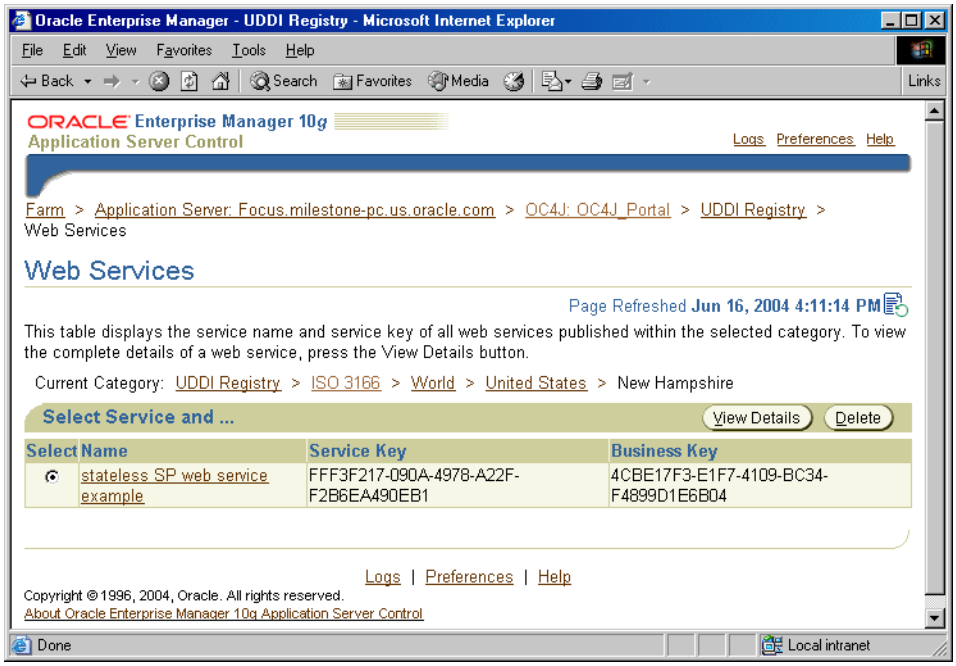
   「**UDDI レジストリ**」が「**関連リンク**」列に表示されていない場合、UDDI 照会サーブレットのエンド・ポイントを ping して OracleAS UDDI Registry を初期化します（10-6 ページの「OracleAS UDDI Registry の構成」を参照してください）。次に、Oracle Enterprise Manager を再起動します。

3. 「UDDI レジストリ」ページでは、対応するリンクをクリックして、3 つの標準的な分類である NAICS、UNSPSC または ISO 3166 のうち 1 つを選択します。次の図に、「UDDI レジストリ」ページを示します。




4. 「UDDI レジストリ : < 分類名 >」ページでは、カテゴリからサブカテゴリへとドリルダウンして、そのカテゴリまたはサブカテゴリに関連する公開済 Web サービスを検出できます。各分類は階層ツリー形式で編成されています。カテゴリ名をクリックして特定のブランチにナビゲートし、すべてのサブカテゴリ名を判断できます。ブランチをナビゲートするにつれて、カテゴリ・コード値が変化します。

次の図に、そのページを示します。

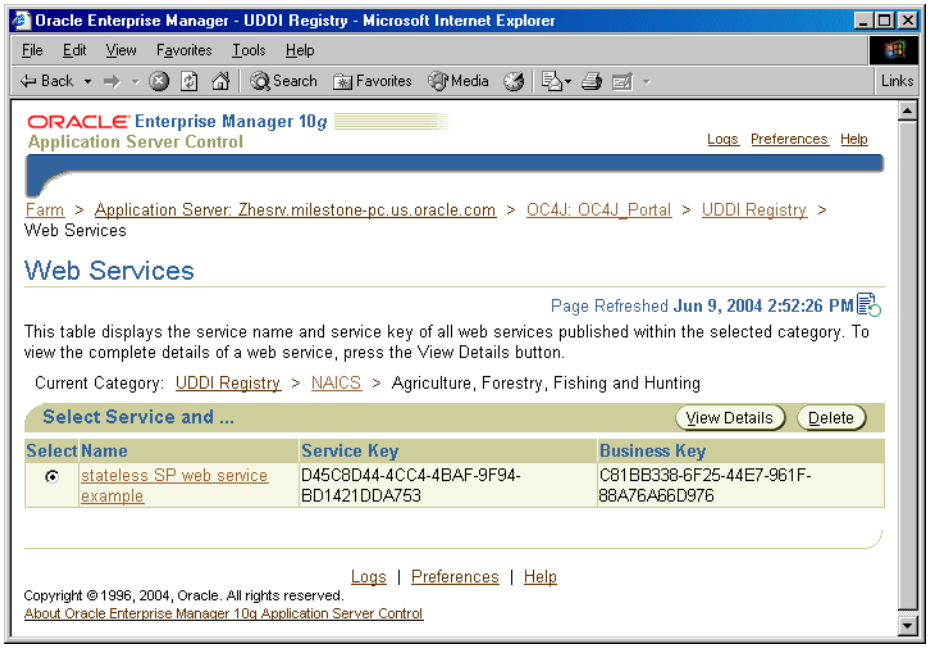



必要なカテゴリをクリックし、そのカテゴリまたはサブカテゴリにナビゲートします。

5. 特定のカテゴリに公開されているすべての Web サービスを表示するには、そのカテゴリの「**選択**」列で対応するラジオ・ボタンを選択し、「**詳細の表示**」をクリックします。

「Web サービス」ページには、そのカテゴリ名で公開されている Web サービスがすべて表示されます。選択したカテゴリに表示される Web サービスについて、対応するサービス名、サービス・キーおよびビジネス・キーも表示されます。選択したカテゴリまたはサブカテゴリに公開されている Web サービスがない場合は、何も表示されません。

次の図に、「Web サービス」ページを示します。



6. あるカテゴリに表示される特定の公開済 Web サービスの詳細を表示するには、サービス名をクリックする方法と、「**選択**」列で対応するラジオ・ボタンを選択して「**詳細の表示**」をクリックする方法があります。

   必要なサービス名をクリックします。

7. 「Web サービスの詳細」ページに、OracleAS UDDI Registry に公開されている、選択した Web サービスの詳細な情報が表示されます。これには次の情報が含まれます。

   - サービスの詳細 : Web サービス名、Web サービス記述および Web サービスのアクセス・ポイントの URL などの情報

     サービス・カテゴリ : 分類、対応するコード値およびカテゴリ名。

   - tModel の詳細 : tModel 名、tModel 記述、インタフェース仕様（通常は WSDL ドキュメント）への URL など、Web サービスにより実装されるインタフェースの記述情報

     tModel カテゴリ : 分類、対応するコード値およびカテゴリ名。

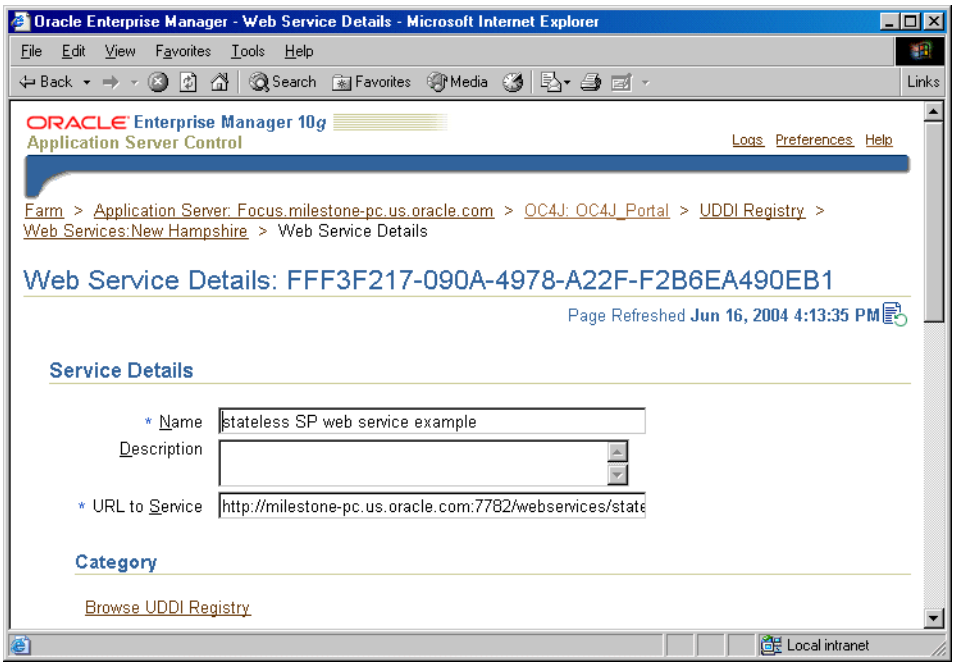
   このページでは、次の操作ができます。

   - OracleAS UDDI Registry を参照し、Web サービスの登録先カテゴリを探すことができます。「**UDDI レジストリの参照**」をクリックします。
   - Web サービスまたは tModel の登録先カテゴリを追加できます。「**行の追加**」をクリックします。
   - Web サービスと tModel の登録先カテゴリを削除できます。「**削除**」をクリックします。

8. この Web サービスについての選択または変更をすべて完了した時点で、「**適用**」をクリックして変更内容を保存します。

いずれかのフィールドを変更した後に、すべての選択値を元の設定に戻す場合は、「**元に戻す**」をクリックします。ページがリフレッシュされ、現行のセッションを始める前と同じように、すべての選択値が元の設定に戻ります。

同じカテゴリに公開されている他の Web サービスを更新するには、「**Web サービスの詳細**」ページの上部で「**Web サービス : < 分類名 >**」リンクを選択し、必要な「Web サービス : < 分類名 >」ページに戻ります。次の図に、「Web サービスの詳細」ページを示します。




必要な「Web サービス : < 分類名 >」ページに戻ると、別の Web サービスを選択して詳細を表示し、必要な変更を行い、最後に「**適用**」をクリックして変更内容を保存します。

また、「**UDDI レジストリの参照**」をクリックして「UDDI レジストリ」ページに戻り、別の分類にナビゲートして他のカテゴリの Web サービスを検出することもできます。必要な各カテゴリで、対象となる Web サービスを選択して詳細を表示し、必要な変更を行い、最後に「**適用**」をクリックして変更内容を保存します。

## OracleAS UDDI Registry の公開ツールの使用

OracleAS UDDI Registry には、公開ツールが用意されています。このツールを使用すると、新しい businessServices および bindingTemplates、または tModel を含む、新しい businessEntity を作成できます。公開ツールにアクセスするには、次の URL をブラウザに入力します。

`http://OracleAS-host:OracleAS-port/uddi`

「OracleAS UDDI Registry」ページが表示されます。

この項では、Google ベースの検索にサービスを公開する例を取り上げます。この例では、WSDL インタフェース仕様にマップされる tModel の作成方法、businessEntity および businessService の作成方法を含め、UDDI レジストリを公開する方法を示します。

次の手順に従います。

1. 「**UDDI Inquiry/Publishing tool**」のリンクをクリックします。次に、「OracleAS UDDI Registry: Searching and Browsing Tool」ページで「**Publishing Tool**」をクリックします。
2. 「Log Into Publishing Service」ページで、uddi_publisher ユーザーのユーザー名とパスワードを入力し、「**Login**」をクリックします。（デフォルトのパスワードは、表 10-2 を参照してください。）

   次の図に示すような「Publishing Tool」ページが表示されます。


3. 「Publishing Tool」ページでは、次を作成できます。
   - 新規の businessEntity
   - 新規の tModel

   OracleAS UDDI Registry に Web サービスを登録する最初のステップは、tModel の作成です。この例では、Google 互換サービスを表す tModel を作成します。

   新しい tModel を作成するには、「**tModel**」をクリックします。「Publish tModel」ページが表示されます。
4. 「**Basic information**」セクションで、次の情報を指定します。
   a. 「**Name**」には、この tModel の検索ができるように名前を指定します。たとえば、**urn:google.com:search-interface** と入力します。
   b. 「**Description**」には、この tModel の説明を指定します。たとえば **Google search interface** と入力します。

c. 「**Overview Document URL**」には、インタフェース仕様（通常は WSDL ドキュメント）への URL を指定します。この例では、Google より WSDL ドキュメント **http://api.google.com/GoogleSearch.wsdl** が提供されます。

d. 「**Overview Document Description**」には、サマリー・ドキュメントの説明を入力します。

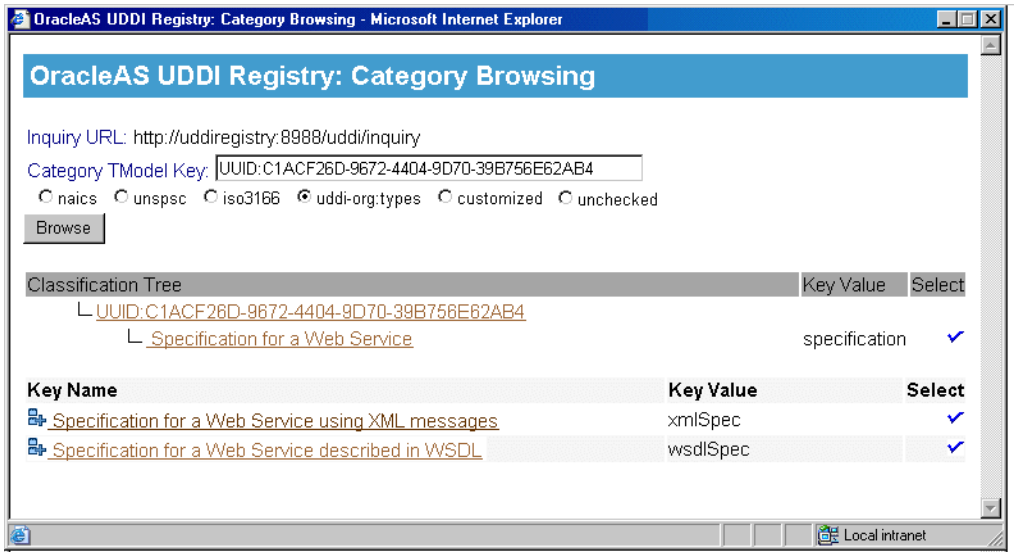
5. 「**Category**」セクションで、tModel をカテゴリに分類します。この例では、WSDL ベースのインタフェースとして分類します。tModel キー、キー名およびキー値の情報を検索するには、1 行目の末尾にあるカテゴリ参照のアイコンをクリックします。「Category Browsing」ページが表示されます。

   次の手順に従います。

   a. 「**uddi-org:types**」を選択し、「**Browse**」をクリックします。

      該当セクションの「Classification Tree」が表示されます。

   b. 「**Specification for a Web Service**」をクリックします。次の図に示すような該当セクションの「Classification Tree」が表示されます。



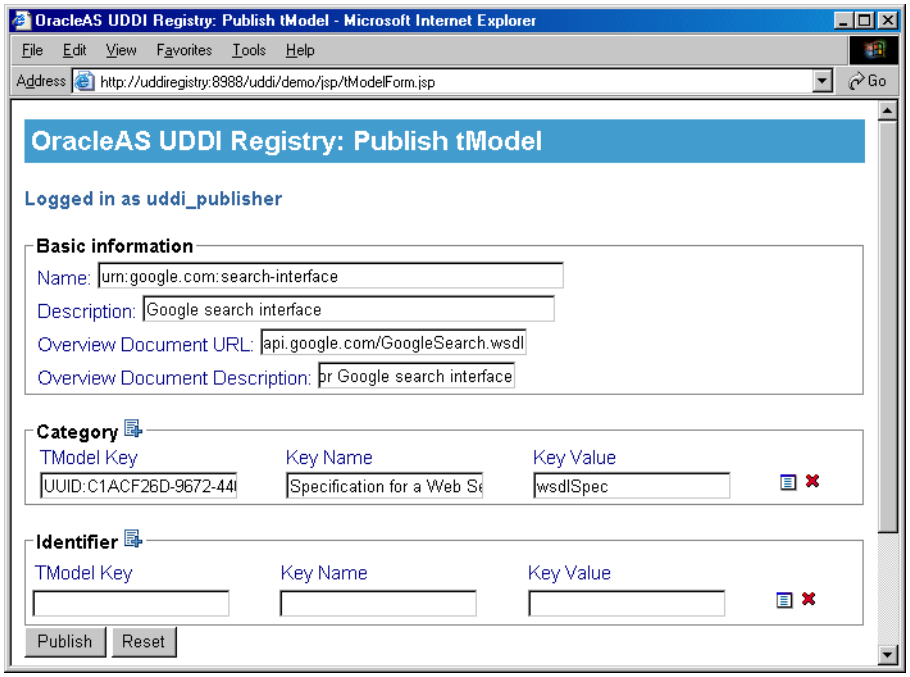

   c. 「**Specification for a Web Service described in WSDL**」の行で、「**Select**」アイコンをクリックします。

   「Publish tModel」ページで、選択した情報が「**Category**」セクションに自動的に挿入されます。ここでは、次の情報が表示されます。

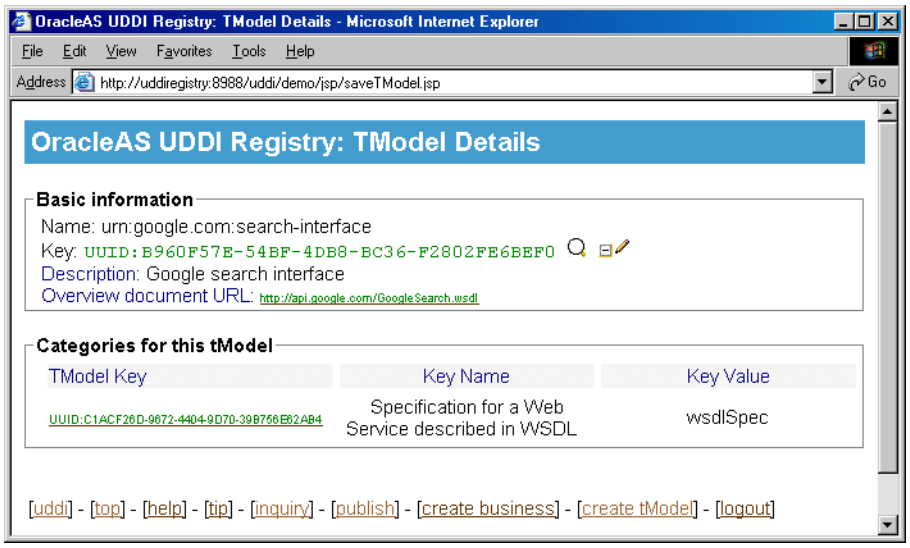
   - **TModel Key: uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4**
   - **Key Name: Specification for a Web Service described in WSDL**
   - **Key Value: wsdlSpec**

6. 「**Identifier**」セクションでは、D-U-N-S 体系または Thomas Register 体系などの識別情報を指定します（詳細は 10-4 ページの「標準的な分類 / 識別子体系のサポート」を参照してください）。この例では、このセクションは空白のままにしておきます。

次の図に、「Publish tModel」ページを示します。




7. 「**Publish**」ボタンをクリックして、tModel を公開します。次の図に示すような「TModel Details」ページが表示されます。




この tModel のキー（UUID:B960F57E-54BF-4DB8-BC36-F2802FE6BEF0）は、レジストリによりランダムに割り当てられたキーであることに注意してください。

8. ページの下部にある「**publish**」リンクをクリックして、「Publishing Tool」のページに戻ります。

9. 名称、連絡先、businessEntity と関連付けられるカテゴリなどのビジネスに関する詳細を含む businessEntity を作成します。「**business entity (or service provider)**」をクリックします。「Publish Business Entity (Service Provider)」ページが表示されます。

businessEntity がすでに存在する場合、新しいエンティティを作成する必要はありません。「Publishing Tool」のページで、既存の businessEntity へのリンクをクリックします。次に手順 15 に進みます。

10. 「Publish Business Entity (Service Provider)」ページの「**Basic Information**」セクションに、次の情報を指定します。

    a. 「**Name**」には、このビジネスの検索ができるように名前を指定します。たとえば、**Google Service Provider** と入力します。

    b. 「**Description**」には、ビジネスの説明を入力します。

11. 「**Contacts**」セクションで、UDDI レジストリに表示する情報を入力します。次のフィールドに情報を指定します。

    a. 「**Name**」には、連絡先の名前を入力します。

    b. 「**Voice**」には、ボイスメール番号を指定します。

    c. 「**Email**」には、電子メール・アドレスを指定します。

    d. 「**Address**」には、郵送先住所を指定します。

12. 「**Category**」セクションで、ビジネスと関連付けるカテゴリを追加します。この例では、ISO 3166 地域分類を使用し、ビジネスの地域カテゴリを指定します。

    カテゴリの tModel キー、キー名およびキー値の情報を検索するには、1 行目の末尾にあるカテゴリ参照のアイコンをクリックします。「Category Browsing」ページが表示されます。次の手順に従います。

    a. 「**iso3166**」を選択し、「**Browse**」をクリックします。該当セクションの「Classification Tree」が表示されます。

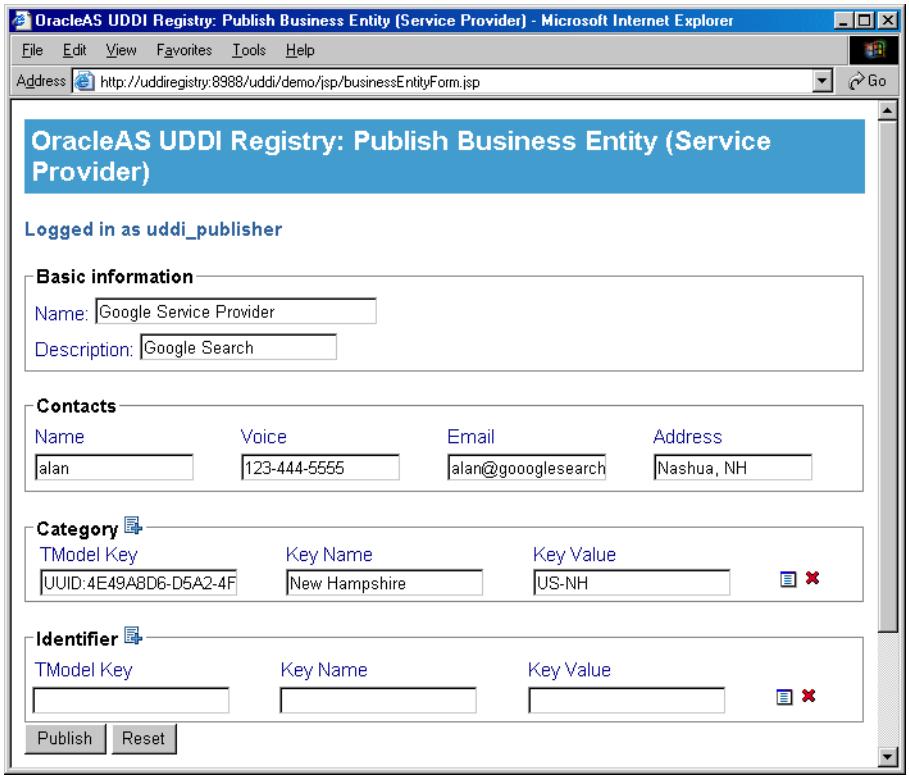
    b. 地域カテゴリを選択するためにドリルダウンします。この例では、「**World**」、「**United States**」を順にクリックし、「**New Hampshire**」の「Select」アイコンをクリックします。

    「Publish Business Entity」ページで、選択した情報が「**Category**」セクションに自動的に挿入されます。ここでは、次の情報が表示されます。

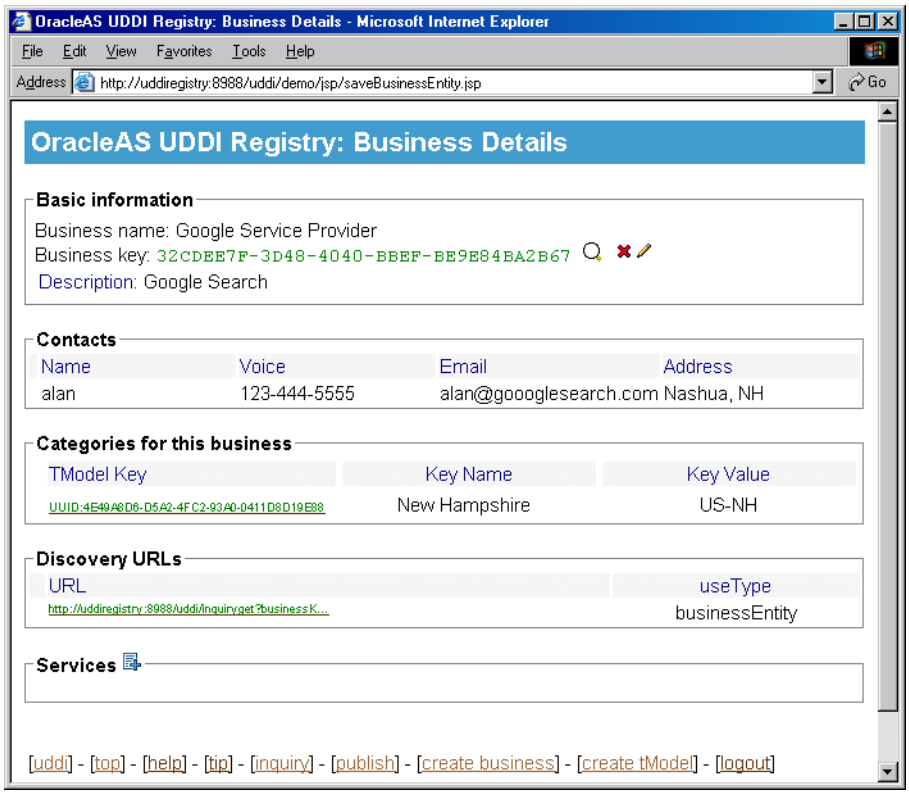
    - **TModel Key: uuid:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88**
    - **Key Name: New Hampshire**
    - **Key Value: US-NH**

13. 「**Identifier**」セクションの情報はオプションです。この例では、空白のままにしておきます。

次の図に、「Publish Business Entity (Service Provider)」ページを示します。




14. 「**Publish**」をクリックし、ビジネスを公開します。「Business Details」ページが表示されます。



「Business Key」および「Discovery URL」は OracleAS UDDI Registry により生成されることに注意してください。「Discovery URL」を使用すると、HTTP get メソッドを使用して businessEntity を取得できます。

15. 「Business Details」ページの「**Services**」セクションで、「**Services**」の右側にあるアイコンをクリックし、businessEntity により提供される businessServices とサービスのカテゴリに関する情報を入力します。「Publish Business Service」ページが表示されます。

16. 「**Basic information**」セクションで、次の情報を指定します。

    a. 「**Owning business**」には、ビジネスのキーが自動的に挿入されます。

    b. 「**Name**」には、businessService の名前を入力します。

    c. 「**Description**」には、businessService の説明を入力します。

17. 「**Category**」セクションで、カテゴリの詳細を追加し、このサービスの用途を指定します。この例では、NAICS 分類のカテゴリを追加し、そのカテゴリがオンライン検索ツールであることを指定します。

    カテゴリの tModel キー、キー名およびキー値の情報を検索するには、1 行目の末尾にあるカテゴリ参照のアイコンをクリックします。「Category Browsing」ページが表示されます。次の手順に従います。

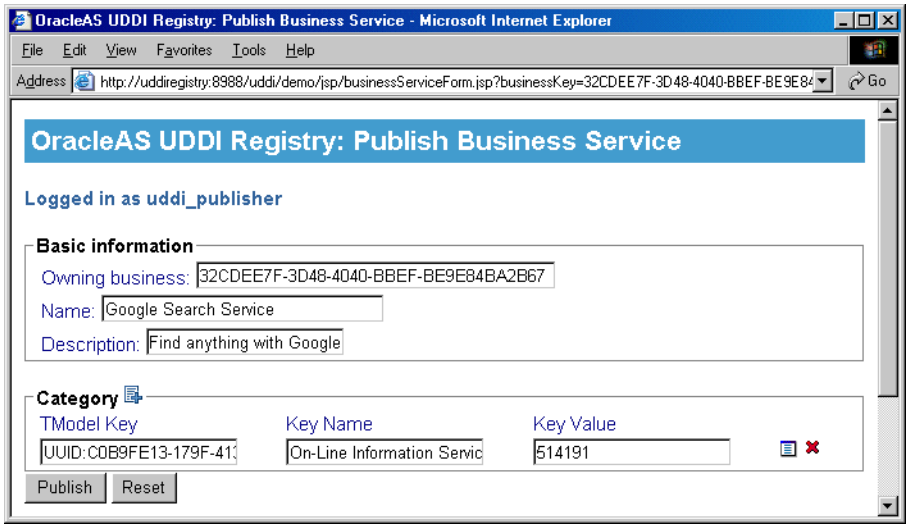
    a. 「**naics**」を選択し、「**Browse**」をクリックします。該当セクションの「Classification Tree」が表示されます。

    b. カテゴリを選択するためにドリルダウンします。「**Information**」、「**Information Services and Data Processing Services**」、「**Information Services**」および「**Other Information Services**」を順にクリックします。次に「**On-Line Information Services**」の「**Select**」アイコンをクリックします。

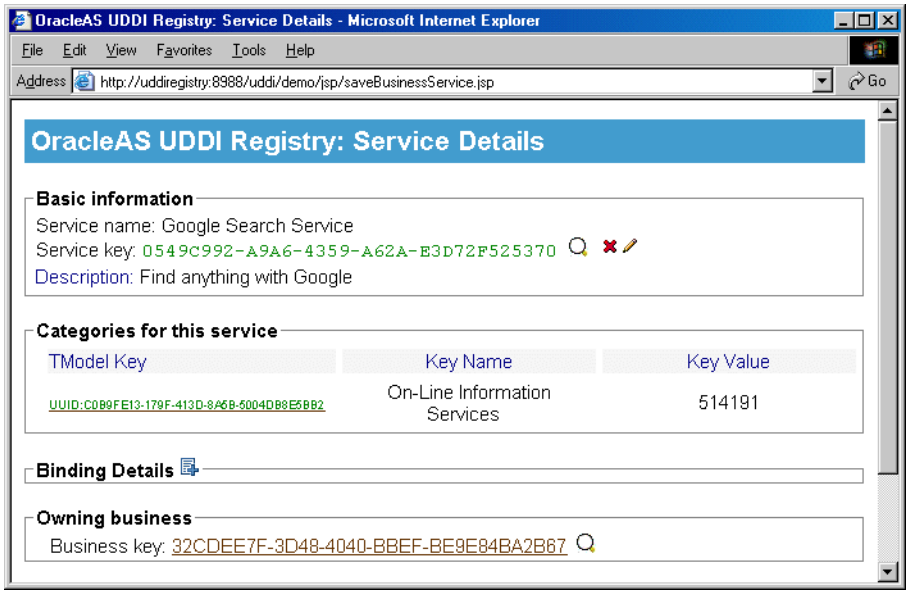
    「Publish Business Service」ページで、選択した情報が「**Category**」セクションに自動的に挿入されます。ここでは、次の情報が表示されます。

    - **TModel Key: uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2**
    - **Key Name: On-Line Information Services**
    - **Key Value: 514191**

    次の図に、「Publish Business Service」ページを示します。



18. 「**Publish**」をクリックして、サービスを公開します。次の図に示すような「Service Details」ページが表示されます。




19. この businessService にアクセスすると提供されるサービスの内容と場所の詳細を含む bindingTemplate を作成します。「Service Details」ページで、「**Binding Details**」の右側にあるアイコンをクリックします。

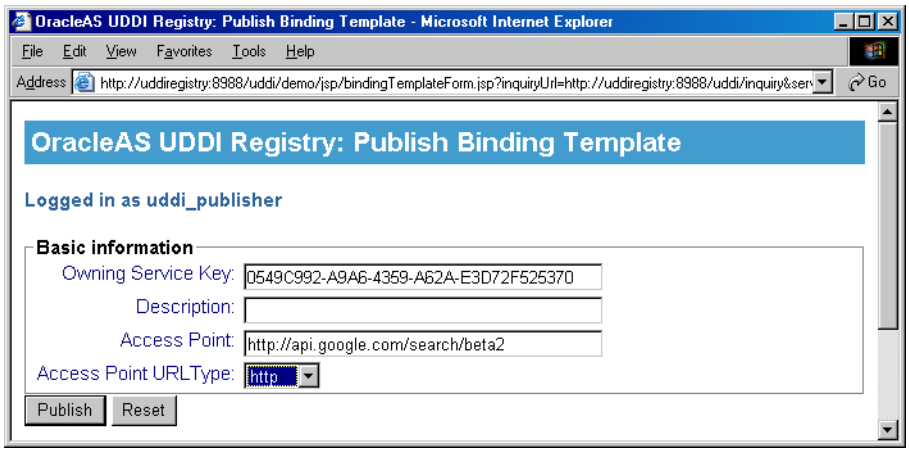
20. 「Publish Binding Template」ページの「**Owning Service Key**」フィールドには、作成したサービスのキーが自動的に挿入されます。「**Basic information**」セクションで、次の情報を入力します。

    - 「**Description**」には、bindingTemplate の説明を指定します。
    - 「**Access Point**」には、サービスにアクセスするためのアクセス・ポイントを指定します。この例では、`http://api.google.com/GoogleSearch.wsdl` のソース・コードを確認して、アクセス・ポイントを見つけます。WSDL ファイルには、次のように指定されています。

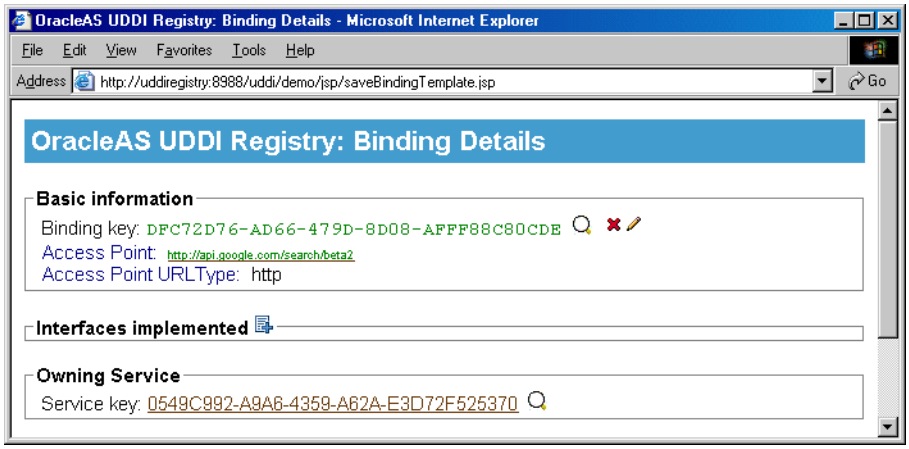
      ```
      <soap:address location="http://api.google.com/search/beta2"/>
      ```

      **http://api.google.com/search/beta2** をアクセス・ポイントとして入力します。
    - 「**Access Point URL Type**」には、「**HTTP**」を選択します。

    次の図に、「Publish Binding Template」ページを示します。



21. 「**Publish**」をクリックし、bindingTemplate を公開します。次の図に示すような「Binding Details」ページが表示されます。



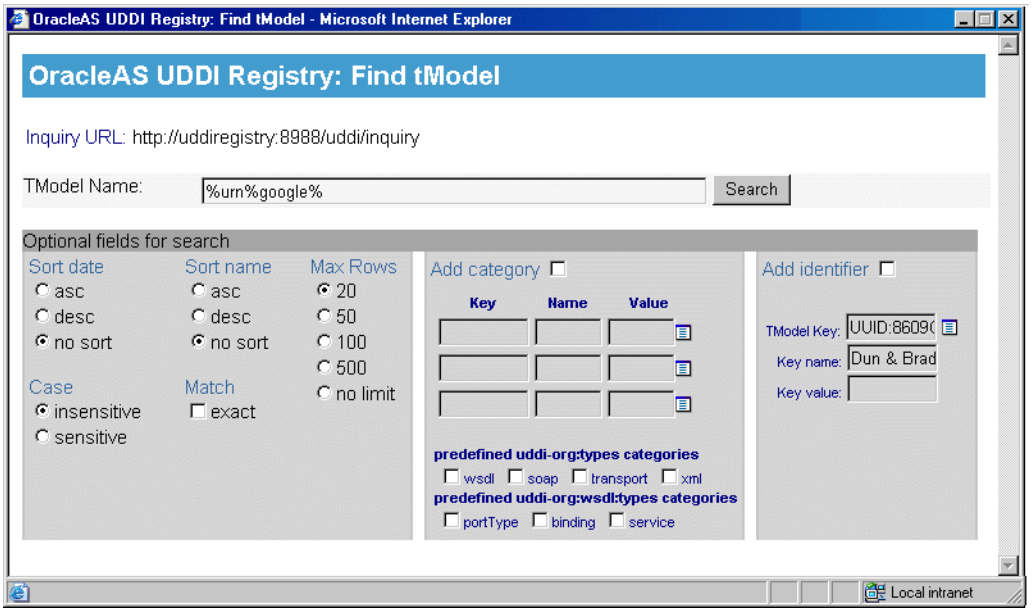

22. 「**Interfaces Implemented**」の右側にあるアイコンをクリックして、新しいインタフェース参照を追加します。

「Publish Interface Reference」ページが表示されます。

23. このページの「**Binding Key**」フィールドに値が自動的に挿入されます。「Interface (tModel) Key」の行の末尾にある、適切な tModel の検索アイコンをクリックします。

「Find tModel」ページが表示されます。

24. 「**TModel Name**」には、作成した tModel の名前の始めの部分、**%urn%google%** などを入力します。次の図に、「Find tModel」ページを示します。

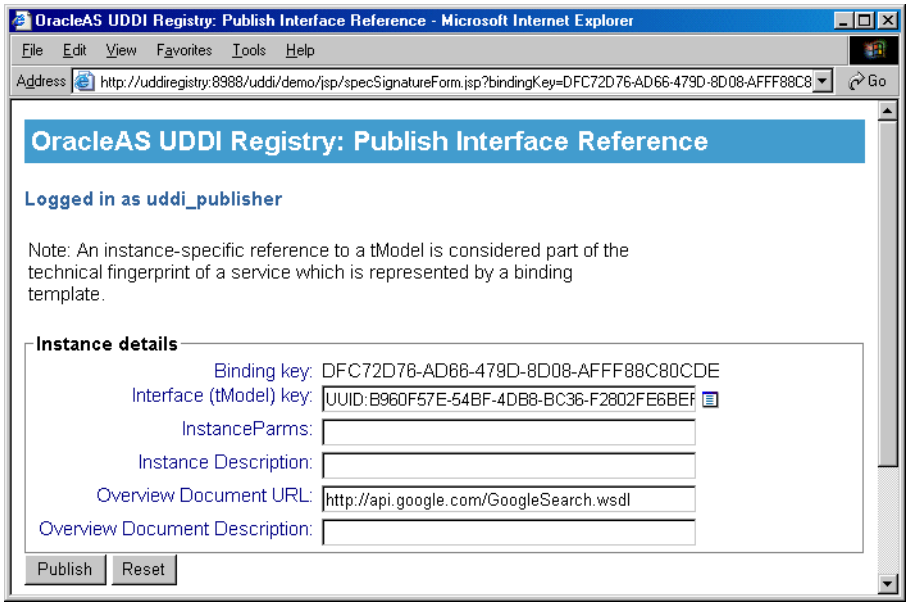



25. 「**Search**」をクリックします。

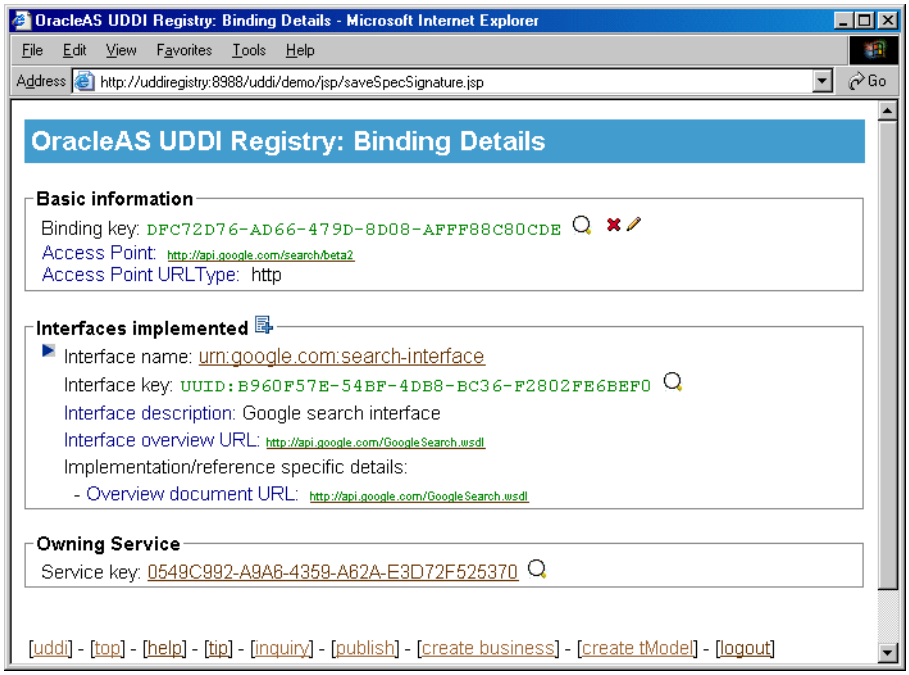
26. すでに作成されている tModel が返されます。「**Select**」をクリックします。

「Publish Interface Reference」ページが表示されます。このページの「**Interface (tModel) key**」フィールドに tModel キーが自動的に挿入されます。

27. 「**Overview Document URL**」には、インタフェース仕様の URL を指定します。この例では、次の図に示すように **http://api.google.com/GoogleSearch.wsdl** と入力します。




28. 「**Publish**」ボタンをクリックします。「Binding Details」ページが表示されます。次の図に示すように、ここでは実装されているインタフェースに関する情報が表示されます。



## OracleAS UDDI Registry の公開 API の使用

OracleAS UDDI Registry の公開 API を使用すると、コンシューマ（プログラマ）は認証後に主要な 5 つの UDDI データ構造（businessEntity、businessService、bindingTemplate、tModel および publisherAssertion）ごとに保存コールと削除コールを提供して、Web サービスを公開できます。

公開 API は、UDDI クライアント API の一部で、これを使用するとプログラマは Java 言語で Web サービスを公開できます。プログラムを任意の言語で記述し、SOAP を使用して Web サービスを公開することもできます。Java プログラマの便宜を図るために、Java API が用意されています。

公開 API は、Oracle Application Server のインストール・ディレクトリにあります。このディレクトリは、UNIX の場合は `${ORACLE_Home}/uddi/`、Windows の場合は `%ORACLE_Home_ORACLE%¥uddi¥` です。API のマニュアルは、『Oracle Application Server Web Services UDDI Client API Reference』を参照してください。

サンプル・デモ・ファイルのセット（`uddidemo.zip`）は、次の URL の Oracle Technology Network（OTN）の Web サイトにあります。

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

#### UddiPublishingExample.java の例

`uddidemo.zip` ファイルには Java プログラム・ファイル `UddiPublishingExample.java` が含まれています。Java プログラマは、このファイルを基にして OracleAS UDDI Registry のクライアント・ライブラリを使用し、主要な構成とシーケンスのデモを実行できます。

サンプル・プログラムで実行される内容は、次のとおりです。

- SoapHTTPTransportLiaison のインスタンスを取得します。これは、SOAP および基礎となるトランスポート・プロトコル（この場合は HTTP）を使用して、UDDI クライアントとサーバー間で通信の詳細を処理する実装です。

  ```
  SoapHttpTransportLiaison transport = new OracleSoapHttpTransportLiaison();
  ```

- システム・プロパティ http.proxyHost および http.proxyPort が設定されている場合は、トランスポートのプロキシ情報を設定します。これらのプロパティは、コマンドラインで設定できます。これらのプロパティが設定されていない場合、このコマンドは機能しません。

  ```
  setHttpProxy((SoapHttpTransportLiaison)transport);
  ```

- SoapTransportLiaison インスタンスと UDDI 公開レジストリの URL を使用して、指定の OracleAS UDDI Registry に接続する UddiClient のインスタンスを初期化します。UddiClient インスタンスは、クライアントから OracleAS UDDI Registry へのリクエスト送信に使用されるプライマリ・インタフェースです。この例では、認証は UDDI get_authToken メッセージを使用して行われます。

  ```
  SimpleAuthenticationLiaison auth =
              new SimpleAuthenticationLiaison(szUserName, szPassword);
  UddiClient uddiClient = new UddiClient(null, szPublishingUrl,
                                         transport, auth);
  ```

  **注意：** UddiClient インスタンスは、デフォルトでは、UDDI v2 クライアント（サポートされている最新リリース）として動作します。特定のリリースが必要な場合は、別のコンストラクタ、または JVM プロパティ oracle.uddi.client.defaultVersion を使用してリリースを指定できます。

- 認証を実行します。このコールは、公開前に行う必要があります。

  ```
  UddiClient.authenticate();
  ```

- UddiClient を使用して UddiElementFactory インスタンスを取得します。必要な UDDI オブジェクトの作成には、常にこのファクトリを使用する必要があります。

  ```
  UddiElementFactory uddiEltFactory = uddiClient.getUddiElementFactory();
  ```

- UddiElementFactory インスタンスを使用して、OverviewDoc データ構造を作成し、tModel データ構造に含めます。

```
OverviewDoc overviewDocTm =
      (OverviewDoc)uddiEltFactory.createOverviewDoc();
tModel.setOverviewDoc(overviewDocTm);
overviewDocTm.setOverviewURL("http://api.google.com/GoogleSearch.wsdl");
```

- UddiElementFactory インスタンスを使用して、Google 互換サービスを表す tModel データ構造を作成します。

```
TModel tModel = (TModel)uddiEltFactory.createTModel();
tModel.setName("urn:google.com:search-interface");
```

- tModel データ構造内で UddiElementFactory インスタンスを使用して、検索に使用する CategoryBag データ構造とその KeyedReference データ構造を作成します。tModel データ構造を SOAP/WSDL ベースのインタフェースとして分類し、"applicable service providers" カテゴリに置きます。

```
CategoryBag catBagTm =
 (CategoryBag)uddiEltFactory.createCategoryBag();
tModel.setCategoryBag(catBagTm);

KeyedReference krTm1 =
 (KeyedReference)uddiEltFactory.createKeyedReference();

catBagTm.addUddiElement(krTm1);
krTm1.setTModelKey(CoreTModelConstants.TAXONOMY_KEY_UDDI_TYPE);
krTm1.setKeyName("soapSpec");
krTm1.setKeyValue("soapSpec");

KeyedReference krTm2 =
 (KeyedReference)uddiEltFactory.createKeyedReference();
catBagTm.addUddiElement(krTm2);
krTm2.setTModelKey(CoreTModelConstants.TAXONOMY_KEY_UDDI_TYPE);
krTm2.setKeyName("wsdlSpec");
krTm2.setKeyValue("wsdlSpec");

KeyedReference krTm3 =
 (KeyedReference)uddiEltFactory.createKeyedReference();
catBagTm.addUddiElement(krTm3);
krTm3.setTModelKey(CoreTModelConstants.TAXONOMY_KEY_UNSPSC_7_3);
krTm3.setKeyName("application service providers");
krTm3.setKeyValue("81.11.21.06.00");
```

- Google 検索インタフェースの tModel ビジネス操作を公開します。

```
System.out.println("¥nPublish the google search interface tModel.");
TModel tMSaved = uddiClient.saveTModel(tModel);
String szGoogleTModelKey = tMSaved.getTModelKey();
System.out.println("The tModel is saved with tModelKey assigned to be " +
                    szGoogleTModelKey);
```

- UddiElementFactory インスタンスを使用して、Google 互換サービスを表す businessEntity データ構造を作成します。

```
BusinessEntity businessEntity =
  (BusinessEntity)uddiEltFactory.createBusinessEntity();
businessEntity.setName("ACME search Inc.", "en");
```

businessEntity データ構造内で UddiElementFactory インスタンスを使用して、検索に使用する CategoryBag データ構造とその KeyedReference データ構造を作成します。データ構造を "information services and data processing services" カテゴリに分類します。

```
KeyedReference krBe1 =
 (KeyedReference)uddiEltFactory.createKeyedReference();
catBagBe.addUddiElement(krBe1);
```

```
krBe1.setTModelKey(CoreTModelConstants.TAXONOMY_KEY_NAICS_1997);
krBe1.setKeyName("Information Services and Data Processing Services");
krBe1.setKeyValue("514");
```

- UddiElementFactory インスタンスを使用して、Google 互換サービスを表す businessService データ構造を作成します。

```
BusinessServices businessServices =
(BusinessServices)uddiEltFactory.createBusinessServices();
businessEntity.setBusinessServices(businessServices);
BusinessService businessService =
(BusinessService)uddiEltFactory.createBusinessService();
businessServices.addUddiElement(businessService);
businessService.setName("ACME Web Search service", "en");
```

businessService データ構造内で UddiElementFactory インスタンスを使用して、検索に使用する CategoryBag データ構造とその KeyedReference データ構造を作成します。businessService データ構造を "application service providers" カテゴリに分類します。

```
CategoryBag catBagBs =
 (CategoryBag)uddiEltFactory.createCategoryBag();
businessService.setCategoryBag(catBagBs);
KeyedReference krBs1 =
 (KeyedReference)uddiEltFactory.createKeyedReference();
catBagBs.addUddiElement(krBs1);
krBs1.setTModelKey(CoreTModelConstants.TAXONOMY_KEY_UNSPSC_7_3);
krBs1.setKeyName("application service
providers");krBs1.setKeyValue("81.11.21.06.00");
```

- UddiElementFactory インスタンスを使用して、Google 互換サービスを表す bindingTemplate データ構造を作成します。

```
BindingTemplates bindingTemplates =
 (BindingTemplates)uddiEltFactory.createBindingTemplates();
businessService.setBindingTemplates(bindingTemplates);
BindingTemplate bindingTemplate =
 (BindingTemplate)uddiEltFactory.createBindingTemplate();
bindingTemplates.addUddiElement(bindingTemplate);
```

  - UddiElementFactory インスタンスを使用して、アクセス・ポイントを作成し、bindingTemplate データ構造に含めます。

```
AccessPoint accessPoint =
(AccessPoint)uddiEltFactory.createAccessPoint();
bindingTemplate.setAccessPoint(accessPoint);
accessPoint.setUrlType("http");
accessPoint.setContent("http://foobar.net/search-g");
```

  - UddiElementFactory インスタンスを使用して、tModel インスタンスの詳細を作成し、bindingTemplate データ構造に含めます。

```
TModelInstanceDetails tModelInstanceDetails =
(TModelInstanceDetails)uddiEltFactory.createTModelInstanceDetails();
bindingTemplate.setTModelInstanceDetails(tModelInstanceDetails);
```

  - bindingTemplate データ構造により Google 検索インタフェースが実装されるように宣言します。

```
TModelInstanceInfo tModelInstanceInfo =
 (TModelInstanceInfo)uddiEltFactory.createTModelInstanceInfo();
tModelInstanceDetails.addUddiElement(tModelInstanceInfo);
tModelInstanceInfo.setTModelKey(szGoogleTModelKey);
```

- businessEntity データ構造とそこに含まれる businessService および bindingTemplate データ構造を公開します。

```
System.out.println("Publish the ACME Search Inc. businessEntity...");
BusinessEntity bESaved = uddiClient.saveBusiness(businessEntity);
System.out.println("The saved businessEntity (in XML) is:");

bESaved.setName("The ACME search Inc.", "en");
BusinessEntity bEUpdated = uddiClient.saveBusiness(bESaved);
```

- UddiElementFactory インスタンスを使用して XmlWriter オブジェクトを取得します。UddiElement を拡張するオブジェクトによって表される RAW XML データを表示するには、要素の内容をライターにマーシャリングし、ライターをフラッシュしてクローズします。

```
XmlWriter writerXmlWriter =
 uddiEltFactory.createWriterXmlWriter(new PrintWriter(System.out));
bESaved.marshall(writerXmlWriter);
writerXmlWriter.flush();
writerXmlWriter.close();
```

- 完了後、リソースの解放とレジストリからのログアウトのために、UddiClient インスタンスをクローズします。

```
uddiClient.close();
```

#### UddiPublisherAssertionExample.java の例

uddidemo.zip ファイルには Java プログラム・ファイル UddiPublisherAssertionExample.java が含まれています。Java プログラマは、このファイルを基にして OracleAS UDDI Registry のクライアント・ライブラリを使用し、主要な構成とシーケンスのデモを実行できます。**パブリッシャ・アサーション**は UDDI v2 の機能で、ビジネス登録およびそのビジネス登録と OracleAS UDDI Registry 内の他のビジネス・データとの関連について特定の事実を表明するパブリッシャによるアサーションです。登録済データ間の関係は、publisherAssertion を使用して確立します。関係を確立した後の一連のアサーションは、findRelatedBusinesses という一般照会メッセージで表示できます。

サンプル・プログラムで実行される内容は、次のとおりです。

- 2 つの UddiClient のインスタンスを初期化します。

```
UddiClient uddiClient1 = createUddiClient(szInquiryUrl,
                              szPublishingUrl, szUserName1, szPassword1);
UddiClient uddiClient2 = createUddiClient(szInquiryUrl, szPublishingUrl,
                              szUserName2, szPassword2);
DispositionReport dispositionReport = null;
```

- 使用する businessEntity データ構造を作成します。

```
String bEKey1 = createBusinessEntity(uddiClient1,
                      "bE1 - UddiPublisherAssertionExample");
String bEKey2 =  createBusinessEntity(uddiClient2,
                      "bE2 - UddiPublisherAssertionExample");
```

- uddiClient1 について、bE1 から bE2 への peer-to-peer 関係を表す publisherAssertion を作成します。

```
System.out.println("");
System.out.println("uddiClient1 attempts to create a peer-to-peer relationship ");
System.out.println("from bE1 to bE2...");
dispositionReport = uddiClient1.addPublisherAssertion
          (createPeerToPeerPublisherAssertion(uddiClient1, bEKey1, bEKey2));
System.out.println("Done.");
```

- 確立する関係について uddiClient1 を問い合せます。つまり、toKey 側で認識されていない関係を調べます。

```
AssertionStatusReport assertionStatusReport1 =
        uddiClient1.getAssertionStatusReport
                  (AssertionStatusItem.COMPLETION_STATUS_TOKEY_INCOMPLETE);
printOutXml("pending relationships for uddiClient1: case toKey incomplete",
             assertionStatusReport1);
```

- 確立する関係について uddiClient2 を問い合せます。つまり、toKey 側で認識されていない関係を調べます。

```
AssertionStatusReport assertionStatusReport2 =

uddiClient2.getAssertionStatusReport(AssertionStatusItem.COMPLETION_STATUS_TOKEY_IN
COMPLETE);
printOutXml("pending relationships for uddiClient2: case toKey incomplete",
             assertionStatusReport2);
```

- publisherAssertion を作成して、リクエストされた peer-to-peer 関係に uddiClient2 が同意していることを示します。

```
System.out.println("");
System.out.println("uddiClient2 agrees to the peer-to-peer relationship ");
System.out.println("from bE1 to bE2");
dispositionReport = uddiClient2.addPublisherAssertion
        (createPeerToPeerPublisherAssertion(uddiClient2, bEKey1, bEKey2));
System.out.println("Done.");
```

- 確立する他の peer-to-peer 関係があるかどうを調べるために、確立する関係について uddiClient2 を再度問い合せます。確立する必要がある保留中の関係はありません。

```
AssertionStatusReport assertionStatusReport2After =
          uddiClient2.getAssertionStatusReport
          (AssertionStatusItem.COMPLETION_STATUS_TOKEY_INCOMPLETE);
printOutXml("pending relationships for client2: toKey incomplete (should be
  none)", assertionStatusReport2After);
```

- 一般照会メッセージ findRelatedBusinesses をコールして、peer-to-peer 関係を確立した（アサーションを公開した）関連ビジネスを検索します。

```
RelatedBusinessesList rbList = uddiClient1.findRelatedBusinesses(bEKey1,
                                createPeerToPeerKeyedReference(uddiClient1),
                                null);
printOutXml("find all businesses that are peers to " + bEKey1, rbList);
```

- uddiClient1 が所有する、bE1 と bE2 間のパブリッシャ・アサーション関係を削除します。

```
System.out.println("");
System.out.println("Delete a publisherAssertion...");
dispositionReport = uddiClient1.deletePublisherAssertion
             (createIdentityPublisherAssertion(uddiClient1,bEKey1, bEKey2));
System.out.println("Done");
```

- uddiClient1 が所有するすべてのパブリッシャ・アサーション関係を削除する別の方法として、setPublisherAssertions コールを使用する方法を示します。

```
System.out.println("");
System.out.println("Delete all publisherAssertions of uddiClient1 ");
System.out.println("by using setPublisherAssertions...");
publisherAssertions = uddiClient1.setPublisherAssertions(null);
printOutXml("Done. The current list:", publisherAssertions);
```

# OracleAS UDDI Registry の管理

この項では、OracleAS UDDI Registry の管理機能について説明します。



## コマンドライン・ツール uddiadmin.jar の使用

管理者としてコマンドライン・ツール uddiadmin.jar を使用し、管理操作の多くを実行します。

コマンドライン・ツール uddiadmin.jar は、UNIX の場合は ${*ORACLE_Home*}/uddi/lib/uddiadmin.jar ファイル、Windows の場合は %*ORACLE_Home_ORACLE*%¥uddi¥lib¥uddiadmin.jar ファイルにあります。通常、このコマンドライン・ツールでは、次のコマンド形式を使用します。

```
java -jar uddiadmin.jar  registry_admin_URL username password
 [-verbose] options_and_their_parameters
```

この形式のパラメータの意味は次のとおりです。

- *registry_admin_URL*: 管理のエンド・ポイントを指す URL です。

  http://*OracleAS-host:OracleAS-port*/uddi/admin

- *username*: デフォルト・ユーザー名は ias_admin です。username は、uddiadmin グループに属している必要があります。
- *password*: デフォルト・パスワードは ias_admin123 です。
- -verbose: 指定すると、例外の発生時にスタック・トレース情報が出力されます。
- *options_and_their_parameters*: 表 10-3 に示されるオプションとそのパラメータです。

表 10-3 に、コマンドライン・ツール uddiadmin.jar のコマンドライン・オプションを示します。

**表 10-3 uddiadmin.jar のコマンドライン・オプション**

| オプション | 説明 |
|---|---|
| changeOwner | 指定したエンティティの所有権を、指定したユーザーに変更します。 |
| correctChangeRecord | changeRecordCorrectionfile ファイルの内容と changeRecordNewDatafile ファイルの内容を UDDI ノードに適用します。 |
| deleteEntity | 指定したエンティティをその所有者に関係なく削除します。 |
| deleteRoleQuotaLimits | 指定した割当てグループについて、グループから割当て制限へのマッピングを削除します。 |
| destroyTModel | 指定した tModel をレジストリから永続的に削除します。 |
| doPing | UDDI レプリケーションの do_ping メッセージを、指定したレプリケーション・エンド・ポイント URL に送信します。 |
| downloadReplicationConfiguration | 現在使用しているレプリケーション構成を、指定の UDDI ノードからダウンロードします。 |
| getChangeRecord | local_usn で指定された変更レコードの詳細を取得します。 |
| getHighWaterMarks | 最高水位標ベクターを指定の UDDI ノードから取得します。 |
| getProperties | 現行のレジストリ構成パラメータをリスト表示します。 |

**表 10-3 uddiadmin.jar のコマンドライン・オプション（続き）**

| オプション | 説明 |
|---|---|
| getRoleQuotaLimits | 現行の J2EE ロールから割当て制限へのマッピングを表示します。 |
| getUserDetail | 指定したユーザーの詳細を取得します。 |
| getUsers | レジストリにエンティティがある既存のユーザーをすべてリスト表示します。 |
| import | 指定したファイル内のすべての businessEntity と tModel データ構造、および publisherAssertion をインポートします。 |
| setOperationalInfo | エンティティの操作情報を設定します。 |
| setProperty | 指定した構成パラメータの値を変更します。 |
| setRoleQuotaLimits | 指定した割当てグループに割当て制限を設定します。 |
| setWalletPassword | UDDI レプリケーション用の UDDI ノード間の HTTPS 通信で使用する、Wallet のパスワードを設定します。 |
| transferCustody | tModel または businessEntity データ構造の管理を、新規のオペレータと新規の認可名に転送します。 |
| uploadReplicationConfiguration | 指定のレプリケーション構成を、特定の UDDI ノードにアップロードします。 |

これらのオプションのリファレンス情報は、10-61 ページの「uddiadmin.jar ツールのコマンドライン・オプション」を参照してください。

## サーバーの構成

コマンドライン・ツール `uddiadmin.jar` の次のオプションを使用してサーバーを構成します。

- `getProperties`: 現行のレジストリ構成パラメータをリスト表示します。次に例を示します。

  ```
  java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
  [-verbose] -getProperties
  ```

  このオプションの詳細は、10-63 ページの「getProperties」を参照してください。

- `setProperty`: 指定した構成パラメータの値を変更します。パラメータを有効にするには、OracleAS UDDI Registry の J2EE アプリケーションを再起動する必要があります。

  **注意：** `setProperty` オプションを使用してサーバー構成のプロパティ値を変更する場合は、注意が必要です。プロパティの設定が不適切な場合は、レジストリの整合性に重大な障害を与える可能性があります。

  `setProperty` オプションを使用すると、10-7 ページの「インストール時または初期のプロパティの変更」に説明されているように、サーバー構成を指定できます。

## ユーザーの管理

OracleAS UDDI Registry では、デフォルト・ユーザー・リポジトリとして、OracleAS Infrastructure の Oracle Internet Directory（OID）を使用します。使用するには、OC4J の Java Authentication and Authorization Service（JAAS）の LDAP ベース・プロバイダを使用します。

UDDI 固有の OID グループは、OID デフォルト・サブスクライバのグループ・サブツリーの `cn=uddi_groups` サブツリーの下にあります。

表 10-4 に、UDDI ユーザー・グループをまとめます。

**表 10-4 デフォルトの UDDI ユーザー・グループ**

| グループ | 説明 |
|---|---|
| uddiadmin | 管理エンド・ポイントにアクセスして、管理アクティビティを実行できます。<br>また、uddipublisher グループで指定したすべてのアクティビティを実行できます。 |
| uddipublisher | 公開エンド・ポイントにアクセスして、レジストリの UDDI エンティティを保存、更新または削除できます。 |
| uddireplicator | レプリケーションのスケジュールに基づいて、レプリケーション・アクティビティを実行できます。また、`get_changeRecords` などのレプリケーション・リクエストを他の UDDI ノードに送信して、受信した changeRecords を適用できます。 |

**注意：** これらのデフォルトの UDDI グループは削除しないでください。

デフォルトの UDDI グループ以外に、ユーザー割当て用の一連のデフォルト・グループがあります。管理者として、適用する必要がある特定のユーザー割当てポリシーに基づき、グループの追加、更新または削除ができます。

デフォルトでは、表 10-5 に示すユーザーがインストール時に作成されます。これらのグループに対して、ユーザーの追加または削除ができます。

**表 10-5 デフォルトの UDDI ユーザー**

| グループ | ユーザー名 | コメント |
|---|---|---|
| uddiadmin | ias_admin | 管理ユーザー。通常、Oracle Enterprise Manager 管理者も `ias_admin` でログインし、Oracle Enterprise Manager の統合された J2EE Web サービスのデプロイおよび公開ウィザードを使用して、UDDI レジストリを公開します。 |
| uddipublisher | uddi_publisher<br>uddi_publisher1 | 公開およびデフォルトの割当てグループのデモで使用するサンプル・ユーザー。 |
| uddireplicator | uddi_replicator | OracleAS UDDI Registry がバックグラウンドで UDDI レプリケーション・アクティビティを実行するために使用するデフォルト・ユーザー。このユーザーは削除しないでください。このユーザーを削除する必要がある場合は、必ず別のユーザーを uddireplicator グループに追加してください。その場合、`oraudrepl.ear` アプリケーションの `orion-application.xml` ファイルを変更して、レプリケーション・クライアント・モジュールを起動するユーザーも更新する必要があります。 |

作成、削除、一時停止などの一般的なユーザー管理は、OID とその Oracle Delegated Administration Service によって実行されます。詳細は、『Oracle Identity Management 委任管理ガイド』を参照してください。

作成、削除、一時停止、ロール管理などの操作を含め、ユーザー管理は OC4J の JAAS サービスにより実行されます。詳細は、『Oracle Application Server Containers for J2EE サービス・ガイド』を参照してください。

ただし、ユーザーの認可名を検索するには、`uddiadmin.jar` コマンドライン・ツールの `getUsers` または `getUserDetail` オプションを使用します。

- `getUsers`: レジストリにエンティティがある既存のユーザーをすべてリスト表示します。次に例を示します。

  ```
  java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
  [-verbose] -getUsers
  ```

  このオプションの詳細は、10-64 ページの「getUsers」を参照してください。

- `getUserDetail`: 指定したユーザーの詳細、現在は各ユーザーの認可名を取得します。次に例を示します。

  ```
  java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
  [-verbose] -getUserDetail username
  ```

  このオプションの詳細は、10-63 ページの「getUserDetail」を参照してください。

## 割当て制限の適用

OracleAS UDDI Registry は、パブリッシャが所有できるエンティティの数を適用するメカニズムを備えています。パブリッシャは、そのパブリッシャに関連付けられている割当てグループに基づいて、tModel、publisherAssertion、businessEntity、businessEntity ごとの businessService データ構造および businessService ごとの bindingTemplate データ構造を特定の数まで所有できます。割当てグループは、パブリッシャに割り当てられているユーザー・グループによって指定されます。

OracleAS UDDI Registry は、グループ・ベースのメカニズムを使用して、割当て制限をパブリッシャに割り当てます。新規のパブリッシャが追加されたとき、OracleAS UDDI Registry の管理者は、そのパブリッシャを割当てグループに関連付ける必要があります。表 10-6 に、パブリッシャが所有できるエンティティごとに事前定義されている割当てグループと割当て制限を示します。

**表 10-6 事前定義されている割当てグループと割当て制限**

| 割当てグループ | エンティティごとの割当て制限 | | | | |
|---|---|---|---|---|---|
| | businessEntity | businessEntity ごとの businessService | businessService ごとの bindingTemplate | tModel | publisherAssertion |
| Default | 1 | 4 | 2 | 100 | 10 |
| uddi_unlimited_quota_group | 無制限 | 無制限 | 無制限 | 無制限 | 無制限 |
| uddi_lowlimits_quota_group | 2 | 2 | 1 | 3 | 3 |
| <Implicit> UDDI_Administrators | 無制限 | 無制限 | 無制限 | 無制限 | 無制限 |

明示的な Default 割当てグループは削除できません。OracleAS UDDI Registry 管理者であるユーザーの割当て制限は、常に無制限です。

また、OracleAS UDDI Registry 管理者として、割当てグループの更新、新規割当てグループの追加、割当てグループの削除、割当てグループとその割当て制限のリスト表示、およびパブリッシャと割当てグループの関連付けを行うことができます。次の各項では、管理者の各作業について説明します。

## 割当てグループの制限の更新

割当てグループの制限を更新するには、コマンドライン・ツール uddiadmin.jar の setRoleQuotaLimits オプションを使用します。

指定した割当てグループに割当て制限値を設定します。このオプションを使用して、グループから割当て制限への新規マッピングの作成、または既存のマッピングの更新ができます。パラメータは次のように定義されています。

- roleName: 指定の割当て制限にマップする割当てグループの名前
- maxBE: businessEntity データ構造の許容最大数
- maxBSperBE: businessEntity ごとの businessService データ構造の許容最大数
- maxBTperBS: businessEntity ごとの bindingTemplate データ構造の許容最大数
- maxTM: tModel データ構造の許容最大数
- maxPA: publisherAssertion データ構造の許容最大数

値 -1 は無制限を意味します。

次に、コマンドの形式を示します。

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setRoleQuotaLimits roleName maxBE maxBSperBE maxBTperBS maxTM maxPA
```

このオプションの詳細は、10-66 ページの「setRoleQuotaLimits」を参照してください。

## 新規割当てグループの追加（拡張操作）

新規割当てグループを追加する手順は、次のとおりです。

1. この処理を始める前に、構成ファイルの application.xml、web.xml および orion-application.xml のバックアップを作成することをお薦めします。
2. グループをユーザー・ストア（通常は OID）に追加します。
3. orauddi.ear アプリケーションで、作成する新しいグループ名に対応する J2EE セキュリティ・ロール partnerGroup を定義します。この設定は、orauddi.ear アプリケーションの application.xml ファイルおよび web.xml ファイルの両方に追加する必要があります。
4. orauddi.ear アプリケーションの orion-application.xml ファイルで、J2EE セキュリティ・ロールからユーザー・ストアへのマッピングを定義します。
5. コマンドライン・ツール uddiaddmin.jar の setRoleQuotaLimits オプションを使用して、割当てグループの実際の制限を定義します。詳細は、10-37 ページの「割当てグループの制限の更新」を参照してください。

## 割当てグループの削除（拡張操作）

割当てグループを削除する手順は、次のとおりです。

1. orauddi.ear アプリケーションから削除する partnerGroup の J2EE セキュリティ・ロールを削除します。この設定は、orauddi.ear アプリケーションの application.xml ファイルおよび web.xml ファイルの両方から削除する必要があります。
2. orauddi.ear アプリケーションの orion-application.xml ファイルで、J2EE セキュリティ・ロールからユーザー・ストアへのマッピングを削除します。
3. 次の例に示すように、コマンドライン・ツール uddiadmin.jar の deleteRoleQuotaLimits オプションを使用して、割当てグループの実際の制限を削除します。

   ```
   java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
     -deleteRoleQuotaLimits roleName [roleName...]
   ```

   このオプションの詳細は、10-61 ページの「deleteRoleQuotaLimits」を参照してください。

4. グループをユーザー・ストア（通常は OID）から削除します。

### 割当てグループとその割当て制限のリスト表示

割当てグループとその割当て制限をリスト表示するには、コマンドライン・ツール `uddiadmin.jar` の `setRoleQuotaLimits` オプションを使用します。このオプションは、次の例に示すように、現在レジストリに設定されている J2EE ロールから割当て制限へのすべてのマッピングを表示します。

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -getRoleQuotaLimits
```

このオプションの詳細は、10-63 ページの「getRoleQuotaLimits」を参照してください。

### パブリッシャと割当てグループの関連付け

ユーザーをユーザー・ストア（OID または `jazn-data.xml`）に追加する場合、適切な割当て制限が適用されるように、そのユーザーをグループに配置する必要があります。たとえば、管理者は、事前定義の設定を使用してユーザーを `uddi_lowlimits_quota_group` グループに割り当てることで、ユーザーに低い割当て制限を適用できます。

ユーザーが特定のグループに属していない場合、そのユーザーには Default グループの割当て制限が適用されます。OracleAS UDDI Registry 管理者の割当て制限は、常に無制限です。

## 管理エンティティの管理

管理エンティティの管理には、コマンドライン・ツール `uddiaddmin.jar` の次のオプションを使用します。

- deleteEntity: 指定したエンティティをその所有者に関係なく削除します。tModel エンティティの場合、この操作では非永続的な削除（非表示）操作が実行されることに注意してください。

  このオプションのリファレンス情報は、10-61 ページの「deleteEntity」を参照してください。

- destroyTModel: 指定した tModel をレジストリから永続的に削除します（UDDI 定義の delete_tModel コールの場合、tModel エンティティは単に非表示になります）。

  このオプションのリファレンス情報は、10-62 ページの「destroyTModel」を参照してください。

- changeOwner: 指定したエンティティの所有権を、新規に指定したユーザーに変更します。

  このオプションのリファレンス情報は、10-61 ページの「changeOwner」を参照してください。

## エンティティのインポート

ファイルからエンティティをインポートするには、コマンドライン・ツール uddiaddmin.jar の import オプションを使用します。指定したファイル内のすべての businessEntity、tModel および publisherAssertion データ構造をインポートできます。

businessEntity データ構造をインポートする場合、インポート用に指定するファイルには、UDDI の businessDetail の XML 文書が含まれている必要があります。

tModel データ構造をインポートする場合、指定するファイルには UDDI の tModelDetail の XML 文書が含まれている必要があります。各ファイルをインポートすることによって、エンティティ・キー（businessKey、serviceKey、bindingKey、tModelKey など）が保たれます。ただし、operatorName および authorizedName フィールドは保たれません。operatorName フィールドは、レジストリの operatorName 構成パラメータで置換されます。インポートされるエンティティの所有者は管理者であるため、authorizedName フィールドは管理者の authorizedName となります。

import オプションが特に役立つのは、なんらかの認可ソースから既知のサービス・インタフェース仕様の tModel と分類の tModel データ構造をインポートする場合です。

エンティティ・キーが保たれるため、エンティティのソースを慎重に評価して、エンティティ・キーに衝突が発生しないことを確認する必要があります。

publisherAssertion をインポートする場合は、2 つのブール値が必要です。この 2 つのブール値は、publisherAssertion の挿入元のサイド（2 つのブール値が true の場合は両方のサイド）を示すために使用されます。

インポートは、単一モード（-s）または複数モード（-m）で実行できます。単一モードでは、部分的な成功（エンティティの一部がインポートされ、残りはエラーが発生したためインポートされない）が許容されません。複数モードでは、部分的な成功が許容されます。

次に、import オプションの形式を示します。

```
-import [-s|-m] {-businesses filename | -tmodels filename |
 -assertions filename  -fromBusinessCheck {true|false}
 -toBusinessCheck {true|false} }
```

たとえば、publisherAssertion ファイル assert.xml が次のような内容であるとします。

```
 <publisherAssertion generic="2.0" xmlns="urn:uddi-org:api_v2">
   <fromKey>22A5A0304C64-11D8-AB19-B8A03C50A862</fromKey>
   <toKey>27CC6702-7F6E-4395-A0B8-97D2FFB5F7634  </toKey>
   <keyedReference tModelKey="UUID:807A2C6A-EE22-470D-ADC7-E0424A337C03"
                   keyName="subsidiary"
                   keyValue="parent-child" />
 </publisherAssertion>
```

この場合、publisherAssertion データ構造をインポートするには、次のコマンドを使用します。

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -import -s -assertions assert.xml -fromBusinessCheck true
 -toBusinessCheck true
```

## 操作情報の設定

setOperationalInfo オプションを使用して、エンティティの操作情報の一部（キーで指定する businessEntity または tModel のオペレータ名、認可名、タイムスタンプなど）を設定します。たとえばインポート操作の後などに設定できます。

setOperationalInfo オプションでは、次の 2 つの構文形式を使用します。

- キーで指定する businessEntity または tModel のオペレータ名、認可名、タイムスタンプのいずれか、あるいは 3 つすべてを変更するには、次の形式を使用します。

  ```
  -setOperationalInfo {-businessKey key | -tModelKey key}
  [-newOperator OperatorName] [-newAuthorizedname authName] [-newTime timestamp]]
  ```

  businessEntity または tModel データ構造のオペレータ名、認可名、タイムスタンプは任意に組み合せることができます。

- businessService または bindingTemplate データ構造のタイムスタンプのみ変更する場合は、次の形式を使用します。

  ```
  -setOperationalInfo {-serviceKey key | -bindingKey key} [-newTime timestamp]
  ```

このオプションの詳細は、10-65 ページの「setOperationalInfo」を参照してください。

## UDDI レプリケーション

OracleAS UDDI Registry により、管理者は、1 つ以上の OracleAS UDDI Registry 実装、および UDDI v2.0 レプリケーション仕様も実装している他のベンダーからの UDDI 実装で構成される論理的なレジストリを作成できます。

この項では、UDDI サービスを構成する UDDI オペレータ・ノード間での完全なデータ・レプリケーションの実行に必要な、データ・レプリケーション・プロセスとプログラム・インタフェースを説明します。UDDI レプリケーションによって、すべてのオペレータ・ノードは、個別のオペレータ・ノードで発生したすべての変更を参照できます。さらに、UDDI サービス内のいずれかのオペレータ・ノードで照会が実行されると、UDDI サービス内の他のオペレータ・ノードで実行された照会との一貫性が維持されます。このため、論理的な OracleAS UDDI Registry と呼ばれます。

レプリケーション処理、UDDI の新規オペレータをオンラインにする方法、レプリケートされたデータのチェックと妥当性チェックなど、UDDI レプリケーションに関する概念と定義の技術的な説明は、UDDI v2.0.3 のレプリケーション仕様を参照してください。次の各項では、UDDI レプリケーションの Oracle 実装について説明します。

### UDDI レプリケーションの有効化

UDDI レプリケーションを有効にするには、管理者として、次の手順を実行する必要があります。

1. 他のオペレータ・ノードの UDDI 管理者とともに、レプリケーション・トポロジに参加して同意します。この手順では、レプリケーション構成を編集し（UDDI v2 レプリケーション仕様で指定されている書式を使用）、コマンドライン・ツール uddiadmin.jar の uploadReplicationConfiguration オプションと downloadReplicationConfiguration オプションを使用します。

   ```
   java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
   -uploadReplicationConfiguration xml_file_containing_replication_configuration
   ```

   このオプションのリファレンス情報は、10-67 ページの「uploadReplicationConfiguration」を参照してください。

   ダウンロードを正常に完了するには、レプリケーション構成をアップロードする必要があります。

   ```
   java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
   -downloadReplicationConfiguration
   ```

   このオプションのリファレンス情報は、10-62 ページの「downloadReplicationConfiguration」を参照してください。

2. プロパティ status の値を 1 に設定して、レプリケーション・スケジュールを有効にします。

   ```
   java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
    -setProperty oracle.uddi.server.scheduler.status=1
   ```

   このオプションのリファレンス情報は、10-65 ページの「setProperty」を参照してください。このプロパティのリファレンス情報は、10-78 ページの「status」を参照してください。

3. プロパティ startMaintainingUpdateJournal を true に設定して、更新ジャーナルの保持を有効にします。

   ```
   java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
   -setProperty oracle.uddi.server.replication.startMaintainingUpdateJournal=true
   ```

   このオプションのリファレンス情報は、10-65 ページの「setProperty」を参照してください。このプロパティのリファレンス情報は、10-77 ページの「startMaintainingUpdateJournal」を参照してください。

UDDI レプリケーションの開始後、管理者として `oraudrepl.ear` アプリケーションを停止または開始することによって、レプリケーション操作を一時停止したり再開することができます。

HTTPS クライアント証明書を使用する場合、次の手順を実行する必要があります。

1. Oracle Wallet Manager を使用してエクスポート済の Oracle Wallet ファイルを取得し、プロパティ walletLocation を設定して、エクスポート済の Wallet の位置を指定します。次の例では、ewallet.p12 の位置は、UNIX の場合が `${ORACLE_Home}/uddi/config`、Windows の場合が `%ORACLE_Home_ORACLE%¥uddi¥config` であり、相対的な位置です。

   ```
   java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
    -setProperty oracle.uddi.server.replication.walletLocation=ewallet.p12
   ```

   このオプションを設定するのは 1 回のみです。

   このオプションのリファレンス情報は、10-65 ページの「setProperty」を参照してください。このプロパティのリファレンス情報は、10-80 ページの「walletLocation」を参照してください。

2. `oraudrepl.ear` アプリケーションが起動または再起動するたびに、`setWalletPassword` オプションを使用して Wallet のパスワードを指定します。次の例に示すように、`uddirepl/admin/wallet` パスを指定します。

   ```
   java -jar uddiadmin.jar http://OracleAS-host:port/uddirepl/admin/wallet username
   password
    -setWalletPassword=walletpassword
   ```

   セキュリティ上の理由でパスワードが永続的に保持されないため、アプリケーションが再起動されるたびにこのオプションを起動する必要があります。

   このオプションのリファレンス情報は、10-66 ページの「setWalletPassword」を参照してください。

エラーが発生した場合、管理者は、エラーの原因になった無効な changeRecord 操作を訂正する必要があります。管理者は、コマンドライン・ツール `uddiadmin.jar` の `correctChangeRecord` オプションを使用して、適切な changeRecord データを指定できます。詳細は、10-42 ページの「レプリケーション例外の処理」を参照してください。

## 管理の転送

tModel または businessEntity の管理を、新規のオペレータと新規の認可名に転送するには、コマンドライン・ツール uddiadmin.jar の transferCustody オプションを使用します。このオプションは、UDDI 仕様で定義される管理転送の一部です。このオプションのリファレンス情報は、10-67 ページの「transferCustody」を参照してください。

## UDDI レプリケーション・スケジューラのプロパティの設定

次の UDDI サーバー・プロパティを使用して、UDDI レプリケーション・スケジューラのプロパティを設定できます。

- timer_pool_size: スケジューラで使用する、同時にアクティブなスレッドの数を指定します。次の例では、スレッド数を 1 に設定しています。

  ```
  java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
    -setProperty oracle.uddi.server.scheduler.timer_pool_size=1
  ```

  このプロパティのリファレンス情報は、10-79 ページの「timer_pool_size」を参照してください。

- status: レプリケーション・リクエストを送信するためにスケジューラを有効にするかどうかを指定します。値 0 はスケジューラをオフに、値 1 はオンに設定します。次の例では、スケジューラをオンに設定しています。

  ```
  java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
    -setProperty oracle.uddi.server.scheduler.status=1
  ```

  このプロパティのリファレンス情報は、10-78 ページの「status」を参照してください。

## レプリケーション例外の処理

レプリケーション操作中にエラーが発生すると、OracleAS UDDI Registry は、そのエラーを oraudrepl.ear アプリケーションの application.log ファイルに記録します。エラーの原因を調べ、各問題を訂正する必要があります。

変更レコードを訂正するには、correctChangeRecord オプションを使用します。このオプションは、changeRecordCorrectionfile ファイルの内容と changeRecordNewDatafile ファイルの内容を UDDI ノードに適用します。これらのファイルの内容は、UDDI レプリケーションの XML Schema に準拠する必要があります。このオプションは、UDDI レプリケーションのエラー・リカバリの一部です。

## UDDI レプリケーションの拡張構成およびチューニング

前の各項で説明した UDDI サーバー・プロパティ以外に、次のサーバー・プロパティがレプリケーションで使用できます。

- changeRecordWantsAck: ローカル・ノードから送信された変更レコードで ACK が必須かどうかを制御します。このプロパティのリファレンス情報は、10-70 ページの「changeRecordWantsAck」を参照してください。
- maxChangeRecordsSentEachTime: 受信 getChangeRecords リクエストに応答して送信する変更レコードの最大数を制御します。このプロパティのリファレンス情報は、10-73 ページの「maxChangeRecordsSentEachTime」を参照してください。
- pushEnabled: UDDI レプリケーションに対して送信タスクを実行するかどうかを制御します。このプロパティのリファレンス情報は、10-75 ページの「pushEnabled」を参照してください。
- pushTaskExecutionPeriod: 送信タスクの実行期間（ミリ秒）を制御します。このプロパティのリファレンス情報は、10-76 ページの「pushTaskExecutionPeriod」を参照してください。
- soapRequestAuthMethod: レジストリ・ノードでレプリケーション SOAP リクエストを他のノードに送信するときに使用する認証方式を制御します。HTTP クライアント証明書（CLIENT-CERT）を使用する場合、レジストリ・ノードが起動または再起動するたびに Wallet のパスワードを設定する必要があります。

このプロパティのリファレンス情報は、10-77 ページの「soapRequestAuthMethod」を参照してください。

- soapRequestTimeout: 各 SOAP レプリケーション・リクエストのタイムアウト値（ミリ秒）を制御します。このプロパティのリファレンス情報は、10-77 ページの「soapRequestTimeout」を参照してください。
- taskExecutionPeriod: レプリケーション・タスクを実行する期間をミリ秒で制御します。このプロパティのリファレンス情報は、10-79 ページの「taskExecutionPeriod」を参照してください。

コマンドライン・ツール `uddiadmin.jar` の次のオプションを使用して拡張構成を実行できます。

- doPing: UDDI レプリケーションの do_ping メッセージを、指定したレプリケーション・エンド・ポイント URL に送信します。これは、他のエンド・ポイントがアクティブであるかどうかをチェックするために使用する、TCP/IP の ping コマンドに類似しています。do_ping メッセージを受信する JVM に Wallet の有効なパスワード・セットがない場合は、オプションの walletPassword パラメータを使用すると便利です。

  このオプションのリファレンス情報は、10-62 ページの「doPing」を参照してください。
- getChangeRecord: local_usn（整数）で指定された変更レコードの詳細を取得します。この API を `correctChangeRecord` オプションとともに使用して、OracleAS UDDI Registry の異なる UDDI ノード間で不適切なデータまたは一貫性のないデータを訂正します。

  このオプションのリファレンス情報は、10-62 ページの「getChangeRecord」を参照してください。
- getHighWaterMarks: 最高水位標ベクターを指定の UDDI ノードから取得します。do_ping メッセージを受信する JVM に Wallet の有効なパスワード・セットがない場合は、オプションの walletPassword パラメータを使用すると便利です。

  このオプションのリファレンス情報は、10-63 ページの「getHighWaterMarks」を参照してください。

## レジストリ・ベースのカテゴリ妥当性チェック

OracleAS UDDI Registry では、スペル・チェック形式でカテゴリ値の妥当性チェックを実行できます。管理者は、レジストリにより検証されるカテゴリのセットを追加または削除できます。詳細は、UDDI v2 仕様を参照してください。

### レジストリ・ベースの妥当性チェック用の新規カテゴリの追加

新規カテゴリを追加するには、カテゴリ値をデータベースにロードして、カテゴリをレジストリに登録する必要があります。手順は次のとおりです。

1. 新規の tModel データ構造を保存してカテゴリをレジストリに公開します。たとえば、`ntis-gov:naics:1997` という tModel データ構造を調べます。サード・パーティのツールまたは付属のサンプル Web アプリケーションの次のリンクを使用できます。

   `http://`*`OracleAS-host`*`:`*`OracleAS-port`*`/uddi/`

   tModel データ構造が他のレジストリに定義されている場合は、新規のデータ構造を作成するかわりに、`uddiadmin.jar` コマンドライン・ツールを使用してそれをインポートすることもできます。新規のデータ構造を作成すると、異なる tModelKeys エンティティが作成されます。インポート操作の詳細は、10-39 ページの「エンティティのインポート」を参照してください。

   公開された tModel データ構造は、uddi-org:types 分類で unvalidatable に分類される必要があります。特に、次の keyedReference は、tModel データ構造の CategoryBag 要素に表示される必要があります。

   ```
   <keyedReference tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4" keyName=""
   keyValue="unvalidatable" />
   ```

2. カテゴリ値をデータベースにロードします。そのためには、すべてのカテゴリ値が次の形式でファイルに指定されている必要があります。

   - ファイルの各行で、カテゴリ内のカテゴリ値を 1 つずつ記述します。書式は次のとおりです。

     ```
     <category value> | <description of category value>
      | <category value of the parent>
     ```

   - カテゴリ値に親がないなど、ルート値の場合は、親のカテゴリ値をそれ自体に設定する必要があります。

   - ファイル内では、カテゴリ値の行をそのすべての子孫の行の前に置く必要があります。

     例は、UNIX の場合は `uddi/taxonomy` ディレクトリ、Windows の場合は `uddi¥taxonomy` ディレクトリにあります。NAICS ファイルからの抜粋を次に示します。

     ```
     22|Utilities|22
     221|Utilities|22
     2211|Electric Power Generation, Transmission|221
     ```

     ファイルで複数言語の異なる文字を使用している場合は、文字の破損などの問題が発生するのを回避するために、UTF-8 エンコーディングを使用してファイルを保存することをお薦めします。

3. カテゴリ・ファイルをロードするための SQL*Loader 制御ファイルを作成します。このファイルの例は、UNIX の場合は *`${ORACLE_Home}`*`/uddi/admin/naics-97.ctl`、Windows の場合は *`%ORACLE_Home_ORACLE%`*`¥uddi¥admin¥naics-97.ctl` です。このファイルをコピーし、制御ファイル内のカテゴリ・ファイル名を作成したファイル名に置き換えてください。新規カテゴリの tModel に対して一意の ID を生成する方法の詳細は、UDDI v2 仕様を参照してください。

4. SQL*Loader を使用して、カテゴリ・ファイルをデータベースにロードします。SQL*Loader の使用方法の詳細は、Oracle Database マニュアルの 1 つである『Oracle Database ユーティリティ』を参照してください。

5. コマンドライン管理ツール `uddiadmin.jar` を使用して、検証する必要のあるカテゴリが認識されるようにレジストリを構成します。たとえば、キー `UUID:FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFF0` を持つ新規の tModel エンティティを追加するには、次のように `setProperty` オプションを使用してプロパティ categoryValidationTModelKeys を設定します。

   ```
   java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
   -setProperty "oracle.uddi.server.categoryValidationTModelKeys=
   'UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4',
   'UUID:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88',
   'UUID:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2',
   'UUID:CD153257-086A-4237-B336-6BDCBDCC6634',
   'UUID:FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFF0' "
   ```

   コマンドは必ず 1 行で入力し、改行や余分な空白を入れないでください。

   `setProperty` オプションでは検証が必要なカテゴリがすべて定義されるため、新規カテゴリを追加するには、既存のすべての tModelKey 値と新規の tModelKey 値を指定する必要があります。

   次のサーバー・プロパティを使用することもできます。

   - identifierValidation: すべての IdentifierBag エンティティの妥当性チェックを制御します。

   - operatorCategory: categoryValidation の値が true の場合に、追加エンティティをオペレータ・ノードとして分類するかどうかを決定します。

   - categoryValidation: すべての CategoryBag エンティティの妥当性チェックを制御します。

- assertionKeyedRefValidation: すべての publisherAssertion の keyedReference データ構造の妥当性チェックを制御します。
- tModelInstanceInfoKeyValidation: tModelInstanceInfo 要素内で tModelKey の存在の妥当性チェックを行うかどうかを決定します。
- addressTModelKeyValidation: address 要素内で tModelKey の存在の妥当性チェックを行うかどうかを決定します。
- hostingRedirectorValidation: bindingTemplate 要素内で hostingRedirector の妥当性チェックを行うかどうかを決定します。妥当性チェックによって、参照先の bindingTemplate 要素が存在し、hostingRedirector 要素が含まれていないことが確認されます。

これらのプロパティの詳細は、10-68 ページの「サーバー構成のプロパティ」を参照してください。

6. レジストリ・ユーザーは、手順 1 の unvalidatable 分類の削除によって公開されたカテゴリ tModel を使用できます。特に、次の keyedReference 要素を tModel データ構造の CategoryBag 要素から削除する必要があります。

```
<keyedReference tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"
 keyName="" keyValue="unvalidatable" />
```

## レジストリ・ベースの妥当性チェックからのカテゴリの削除

カテゴリをレジストリ・ベースの妥当性チェックから削除するには、レジストリからカテゴリを登録解除し、カテゴリ値をデータベースから削除する必要があります。手順は次のとおりです。

1. カテゴリをレジストリから登録解除するには、`uddiadmin.jar` コマンドの `setProperty` オプションを使用してプロパティ categoryValidationTModelKeys を設定し、検証済カテゴリのリストから削除します。

   tModel データ構造をレジストリから削除する必要はありません（通常は、削除しないでください）。

2. カテゴリ値をデータベースから削除するには、SQL*Plus スクリプト `wurvcrm.sql` を使用します。このスクリプトは、UNIX の場合は `uddi/admin` ディレクトリ、Windows の場合は `uddi¥admin` ディレクトリにあります。次に例を示します。

   ```
   sqlplus sys/sys-password @wurvcrm.sql
   ```

   このスクリプトを実行すると、削除するカテゴリの tModelKey 値を求めるプロンプトが表示されます。行セットが削除されたことを確認する必要があります。その結果、1 行も削除されていない場合は、入力した tModelKey 値が無効です。スクリプトを再度実行します。

## 外部検証

サード・パーティは、新規のカテゴリと識別子分類体系を登録し、OracleAS UDDI Registry で使用する妥当性チェック・プロセスを制御して外部検証または外部チェックを実行できます。これによって、エンティティが分類されるとき、またはエンティティがカテゴリによって識別されるときに、サード・パーティのカテゴリ・プロバイダは、validate_values SOAP Web サービスを指定して、保存する UDDI エンティティを検証できます。

validate_values サービスをコールするオペレータは、単一の引数として businessEntity、businessService または tModel 要素をこのコールに渡します。これは、save_business、save_service または save_tModel API コールで渡されるデータと同じです。外部検証は、サード・パーティのカテゴリ・プロバイダ、および「チェック済」に分類された識別子分類体系に対して実行されます。「チェック済」とマークされた tModel 要素は、適切に登録された妥当性チェック・サービスがある分類、識別子または名前空間の tModel 要素であることを表明します。

エラーが発生していない場合のレスポンスは、E_success の errorCode 値と 0（ゼロ）の `errno` 値を戻す dispositionReport メッセージです。エラーが発生した場合、または外部サービス・プロバイダが選択した妥当性チェック・アルゴリズムに基づいて、コールされたサービスで保存する情報が有効でないことを示す必要がある場合、そのサービスでは、SOAP 障害の発生と E_invalidValue または E_valueNotAllowed の errorCode 値を示す必要があります。いずれの場合も、エラー・テキストには、拒否された keyedReference データとその理由が示されます。

コマンドライン・ツール `uddiadmin.jar` の `setProperty` オプションを使用して、次の操作を実行します。

- 外部検証の有効化
- 外部で検証されたカテゴリをレジストリに追加
- 外部で検証されたカテゴリをレジストリから削除

### 外部カテゴリ検証の有効化

外部カテゴリ検証を有効にするには、次のように `setProperty` オプションを発行してサーバー・プロパティ externalValidation を設定します。

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.externalValidation=true
```

### 外部で検証されたカテゴリをレジストリに追加

外部で検証されたカテゴリをレジストリに追加する手順は、次のとおりです。

1. 新規カテゴリを tModel データ構造としてレジストリに公開します。このデータ構造は、uddi-org:types 分類で「チェック済」に分類される必要があります。
2. 次のように `setProperty` オプションを使用して、サーバー・プロパティ externalValidationTModelList を更新し、カテゴリの外部検証サービスをレジストリに登録します。

   ```
   java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
    -setProperty oracle.uddi.server.externalValidationTModelList=key-value,
   URL-validation-service
   ```

   たとえば、公開されたカテゴリ tModel にキー uuid:acme-taxonomy-key があり、検証サービスの URL が `http://acme.com/externalValidation` の場合、次のコマンドを使用します。

   ```
   java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
   -setProperty
   oracle.uddi.server.externalValidationTModelList=uuid:acme-taxonomy-key,
   http://acme.com/externalValidation
   ```

3. 必要な場合は、次のようにサーバー・プロパティ externalValidationTimeout を使用して、外部検証サービスへのコールのタイムアウト制限（ミリ秒単位）をチューニングできます。

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.externalValidationTimeout=5000
```

### 外部で検証されたカテゴリをレジストリから削除

外部で検証されたカテゴリをレジストリから削除する手順は、次のとおりです。

1. 次のように *URL-validation-service* には NULL 値を指定して、setProperty オプションを使用し、サーバー・プロパティ externalValidationTModelList を更新します。

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.externalValidationTModelList=key-value,""
```

   たとえば、公開されたカテゴリ tModel にキー uuid:acme-taxonomy-key があり、検証サービスの URL が http://acme.com/externalValidation の場合、NULL 値を指定したコマンドは次のようになります。

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty
 oracle.uddi.server.externalValidationTModelList=uuid:acme-taxonomy-key,""
```

2. 対応する tModel データ構造を無効にするか、または更新します。tModel が更新されない場合、レジストリでは、save_business、save_service または save_tModel API への後続の保存コールで、削除されたカテゴリで分類または識別された新規の UDDI エントリを拒否します。

## パフォーマンス・モニタリングおよびチューニング

Oracle データベースのバックエンドでは、Oracle Enterprise Manager 10*g* や他の標準的なデータベース・モニターおよびチューニング・ユーティリティを使用して、UDDI サーブレットおよび関連する JDBC 接続プールをモニターできます。

OC4J のスタンドアロン環境では、通常、パフォーマンス情報は次の URL にアクセスして参照できます。

```
http://oc4j-host-name:port-number/dmsoc4j/Spy
```

## データのバックアップおよびリストア操作

UDDI レジストリ・データのバックアップおよびリストアは、標準的な Oracle データベースのバックアップおよびリストア操作を使用して実行できます。Oracle Database マニュアルの『Oracle Database バックアップおよびリカバリ基礎』を参照してください。

# データベース構成

この項では、データベース固有の構成情報について説明します。

## 必須のデータベース・キャラクタ・セットである UTF-8

データベース・キャラクタ・セットには、考えられる文字すべてに対処するために UTF-8 を使用する必要があります。ただし、レジストリに格納されるデータに特定の国や地域（西欧など）の文字が確実に含まれている場合は、適切なデータベース・キャラクタ・セットを使用できます。

## データベース・キャラクタ・セットと組込みの ISO 3166 分類

UDDI 仕様では、レジストリで UTF-8 キャラクタ・セット全体をサポートするように定められています。必須ではありませんが、OracleAS UDDI Registry を使用する場合は、Oracle Application Server Infrastructure データベースにキャラクタ・セットとして UTF-8 を使用することをお薦めします。

データベースが UTF-8 キャラクタ・セットや、それと同等のキャラクタ・セット、あるいはスーパーセットを使用して構成されていない場合、UTF-8 との間のキャラクタ・セット変換中にデータが破損し、消失によるエラーが発生する可能性があります。詳細は、『Oracle Database グローバリゼーション・サポート・ガイド』を参照してください。

特に、UDDI 組込み分類 ISO 3166 の記述には、都市名や地域名を表す一部の西欧文字や一部の東欧文字など、非 ASCII 文字による記述が含まれています。非 UTF-8 データベースをサポートするために、記述に含まれる非 ASCII 文字はすべて近似の ASCII 文字に置換されます。

UTF-8 データベースを使用している場合は、組込みの ISO 3166 分類を次の手順で正確な記述を含む用語にアップグレードできます。

- 次のように、SQL スクリプト `wurrmiso.sql` を実行して既存の ISO 3166 分類を削除します。

  ```
  cd ${ORACLE_Home}/uddi/admin
  sqlplus system/manager @wurrmiso.sql
  ```

- 次のように、SQL*Loader の制御ファイル `iso3166-99.ctl` を使用して、正確な記述を含む ISO 3166 分類をロードします。

  ```
  cd ${ORACLE_Home}/uddi/admin
  sqlldr userid=system/manager control=iso3166-99.ctl
  ```

## ファンクション・ベースの有効化の必要性

大 / 小文字区別なしのファンクション・ベースの検索をサポートするために、ファンクション・ベースを使用可能にする必要があります。そのためには、初期化パラメータ `query_rewrite_enabled=true` を使用します。

また、UDDISYS スキーマ内のすべての表または索引を分析できるように、コスト・ベース・オプティマイザをオンにする必要があります。次に例を示します。

```
execute dbms_stats.gather_schema_stats(ownname=>'UDDISYS',cascade=>true);
```

## UDDI エンティティの変更後のタイムスタンプの精度

UDDI エンティティの変更後のタイムスタンプの精度は、データベースのバージョンと互換性に依存します。データベースの互換性がリリース 9.0.1 以上の場合、変更後のタイムスタンプは最大精度がマイクロ秒の SQL TIMESTAMP 型です。データベースの互換性がリリース 9.0.1 より前の場合、変更後のタイムスタンプは最大精度が秒の SQL DATE 型です。

## トランスポート・セキュリティ

通常、照会 API には認証は不要です。ただし、照会エンド・ポイントを保護する必要がある場合は、web.xml ファイルを構成して、HTTP BASIC 認証や HTTPS Secure Sockets Layer（SSL）クライアント認証など、トランスポート・レベルの認証を使用可能にすることができます。セキュリティ・ロール uddiguest は、保護された照会エンド・ポイントへのアクセス用に予約されています。セキュリティ・ロールおよび関連セキュリティ構成の詳細は、『Oracle Application Server Containers for J2EE サービス・ガイド』および『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』を参照してください。

公開エンド・ポイント URL の場合は、HTTPS アクセスのみ許可することを検討します。HTTP アクセスを使用不可にするには、orauddi アプリケーションの web.xml ファイルを編集してデータの機密保護を規定し、HTTP サーバーを調整します。たとえば、web.xml ファイル内で HTTP アクセスを使用不可にするには、次のコードを使用します。

```
<user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
</user-data-constraint>
```

詳細は、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』のセキュリティに関する章、および『Oracle Application Server Containers for J2EE サービス・ガイド』を参照してください。

管理エンド・ポイントと UDDI レプリケーション・エンド・ポイントの HTTPS アクセスは、同じ方法で設定できます。

# UDDI オープン・データベース・サポート

Oracle Application Server Infrastructure データベース以外に、次のデータベースも OracleAS UDDI Registry でサポートされています。

- Microsoft SQL Server
- IBM DB2
- その他の Oracle データベース（Infrastructure 以外）

Microsoft SQL Server および IBM DB2 を使用する場合は、Oracle Application Server DataDirect Connect JDBC ドライバが必要です。

次の各項で説明する手順では、関連するデータベース・サーバーがインストール済であることを前提にしています。また、Oracle Application Server Portal がインストール済で、関連する UDDI ファイルが、UNIX の場合は ${*ORACLE_Home*}/uddi/admin に、Windows の場合は %*ORACLE_Home_ORACLE*%¥uddi¥admin にコピーされていることを前提にしています。

---

**注意：** OracleAS UDDI Registry は、スタンドアロン OC4J インストールに配置することもできます。詳細は、次の URL の OTN で『Oracle UDDI Support for Web Services Readme for Standalone Kit』を参照してください。

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

このマニュアルの指示に従って、ant コマンドの db.host、db.port および db.sid オプションに任意の値を指定できます。これらの値は、次の各項でデータ・ソースを定義する際に変更します。

---

# Microsoft SQL Server

次の各項では、SQL Server におけるインストールおよび構成情報について説明します。

## スクリプト・ソース・ディレクトリ

インストールは Windows マシンで実行する必要があります。SQL Server マシンから *%ORACLE_Home%*¥uddi¥admin¥mssql ディレクトリにアクセスできない場合は、このディレクトリをアクセス可能な位置にコピーしてください。このディレクトリ（コピーする必要がない場合は元の *%ORACLE_Home%*¥uddi¥admin¥mssql）は、*%MSSQL_HOME_DB%* として参照されます。

## データベースとユーザーの作成

提供されている *%MSSQL_HOME_DB%*¥wurcreatedb_mssql.sql スクリプトを使用して、SQL Server インスタンスの uddisys データベースと uddisys ユーザーを mixed-authentication モードで作成します。Windows 認証を使用する場合、またはこのスクリプトの設定の一部を変更する必要がある場合は、次の要件を満たしている必要があります。

- uddisys データベースの照合は、大 / 小文字の区別が必要です。
- uddisys データベースでは、再帰的トリガーが使用可能であることが必要です。
- uddisys ユーザーのデフォルト・データベースは、uddisys データベースであることが必要です。
- uddisys ユーザーは、uddisys データベースの db_owner ロールのメンバーであることが必要です。

Microsoft osql ユーティリティを使用してスクリプトを実行するには、管理者のログイン（sa）とパスワードを使用します。次の例では、管理者のパスワードが sa であることを前提にしていますが、異なる場合は、使用している環境に適したパスワードをかわりに使用してください。

```
osql -S server -U sa -P sa  -i wurcreatedb_mssql.sql
```

この例の *server* は、SQL Server インスタンスをホスティングするサーバーです。

次のエラーを受け取った場合は、SQL Server の認証モードを SQL Server and Windows モードに必ず変更してください。

```
"Login failed for user 'sa'. Reason: Not associated with a trusted SQL Server Server
Connection."
```

認証モードを変更するには、SQL Server Enterprise Manager をオープンし、使用しているサーバーにナビゲートし、右クリックしてプロパティを選択し、「Security」タブをクリックして「SQL Server and Windows」を選択します。次に、SQL Server を再起動します。

## スキーマのインストール

*%MSSQL_HOME_DB%* ディレクトリにアクセスします。10-50 ページの「データベースとユーザーの作成」で作成した uddisys/uddisys アカウントを osql ユーティリティで使用して、SQL スクリプト wurinst_mssql.sql を実行します。

構文は次のとおりです。

```
osql -S server -U user -P password -d database -i wurinst_mssql.sql
```

この例の *server* は、SQL Server インスタンスをホスティングするサーバーです。

次に例を示します。

```
osql -S server-machine -U uddisys -P uddisys -d uddisys -i wurinst_mssql.sql
```

## BUILTIN_CHECKED_CATEGORY 表のエントリのインポート

iso3166-99_tModelKey.txt、naics-97_tModelKey.txt および unspsc-73_tModelKey.txt ファイルを %MSSQL_HOME_DB% ディレクトリから BUILTIN_CHECKED_CATEGORY 表にインポートする手順は、次のとおりです。

1. SQL Server のスタート・メニュー・オプションから「**Import and Export Data**」オプションを選択します。「**Next**」をクリックします。
2. データ・ソースとして、最後のオプション「**Text File**」を選択します。次に、適切なテキスト・ファイルの名前と位置を *%MSSQL_HOME_DB%*¥iso3166-99_tModelKey.txt と指定します。「**Next**」をクリックします。
3. デフォルトのファイル形式は「**Delimited**」です。「**Next**」をクリックして、このデフォルトを受け入れます。
4. デリミタを縦線文字（|）に設定します。「**Next**」をクリックします。
5. 宛先に **uddisys** データベースを選択します。適切な認証メカニズムと資格証明を指定します。デフォルトは、ユーザー uddisys とパスワード uddisys を使用する SQL Server Authentication です。**uddisys** データベースが選択されていることを確認します。「**Next**」をクリックします。
6. 「**Destination**」をクリックし、**BUILTIN_CHECKED_CATEGORY** 表を選択します。
7. 「**Transform**」をクリックします。TMODEL_KEY を Col001 に、KEY_NAME を Col003 に、KEY_VALUE を Col002 に、PARENT_VALUE を Col004 にマップします。「**OK**」をクリックします。
8. 「**Next**」をクリックします。
9. すぐに実行するには、「**Next**」をクリックし、「**Finish**」をクリックするとインポートが開始します。
10. naics-97_tModelKey.txt ファイルおよび unspsc-73_tModelKey.txt ファイルについて、この手順を繰り返します。

---

**注意：** 使用しているデータベースのキャラクタ・セットが UTF-8 でない場合は、スクリプト iso3166-99.txt を使用して ISO 3166 分類をロードしないでください。これは、異なる言語の文字が分類に含まれているためです。かわりに、スクリプト iso3166-99-ascii.txt を使用して、ASCII 専用バージョンの分類をロードしてください。

---

## SQL Server を使用するための OC4J の構成

SQL Server が必要なデータベースであることを反映するために、次のように、jdbc/OracleUddi に設定されたデータ・ソースを名前と位置を指定して定義します。

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="jdbc/OracleUddi"
    location="jdbc/OracleUddi"
    connection-driver="com.oracle.ias.jdbc.sqlserver.SQLServerDriver"
    username="uddisys"
    password="uddisys"
url="jdbc:oracle:sqlserver://server:1433;SelectMethod=cursor;User=uddisys;Password=uddisys"
/>
```

*server* は、OracleAS UDDI Registry で使用する SQL Server インスタンスをホスティングするサーバーのネットワーク名または IP アドレスであることに注意してください。url= で始まる行は、必ず 1 行で入力してください。

データ・ソースは、orauddi.ear アプリケーションおよび oraudrepl.ear アプリケーションからアクセスできる必要があります。

詳細は、『Oracle Application Server Containers for J2EE サービス・ガイド』の「データ・ソース」を参照してください。

UDDI サーバーを再起動して、変更内容を有効にします。

## IBM DB2

次の各項では、IBM DB2 に関係する OracleAS UDDI Registry のインストールおよび構成情報について説明します。

### スクリプト・ソース・ディレクトリ

関連する DB2 ツールを使用して、マシンから ${*ORACLE_Home*}/uddi/admin/db2 ディレクトリにアクセスできない場合は、このディレクトリをアクセス可能な位置にコピーしてください。このディレクトリは、UNIX の場合は ${*DB2_HOME_DB*}、Windows の場合は *%DB2_HOME_DB%* として参照されます。

### データベースとユーザーの作成

UNIX の場合は ${*DB2_HOME_DB*} ディレクトリに、Windows の場合は *%DB2_HOME_DB%* ディレクトリにアクセスします。uddisys データベースを作成するために、wurcreatedb_db2.sql スクリプトが用意されています。ユーザーは、DB2 で使用する認証方法に基づいて、パスワードが uddisys の uddisys ユーザーを作成する必要があります。デフォルトでは、オペレーティング・システム・レベルで uddisys ユーザーを作成する必要があります。Windows では、uddisys ユーザーはローカル管理者グループに属す必要があります。

このスクリプトの設定の一部を変更する必要がある場合は、次の要件を満たしている必要があります。

- uddisys データベースのデフォルト表領域は、8KB 以上のページが必要です。また、8KB 以上のページ・サイズをサポートするバッファ・プールも提供される必要があります。
- applheapsz パラメータの値を約 12800 ページに増やす必要があります。

スクリプトを実行するには、UNIX の場合は db2、Windows の場合は db2cmd を入力して、DB2 コマンドライン・プロセッサを起動します。次に、スクリプトを実行します。

```
db2 -t +p < wurcreatedb_db2.sql
```

SQL 文を終了するにはオプション -t でセミコロンを使用します。+p はプロンプト表示を抑制します。

### スキーマのインストール

wurinst_db2.sql スクリプトを実行します。これによって、wurcreat.sql、wurdbsql.sql および wurpopul.sql スクリプトもトリガーされます。

これらのスクリプトを実行するには、前述のコマンドライン・プロセッサを起動し、次のように入力します。

```
db2 -t +p < wurinst_db2.sql
```

## BUILTIN_CHECKED_CATEGORY 表のエントリのインポート

iso3166-99_tModelKey.txt、naics-97_tModelKey.txt および unspsc-73_tModelKey.txt ファイルを BUILTIN_CHECKED_CATEGORY 表にインポートする手順は、次のとおりです。

1. コントロール・センターから **BUILTIN_CHECKED_CATEGORY** 表を右クリックし、「**IMPORT**」を選択します。
2. インポート・ファイルとして、UNIX の場合は ${*DB2_HOME_DB*}/iso3166-99_tModelKey.txt を、Windows の場合は *%DB2_HOME_DB%*¥iso3166-99_tModelKey.txt を指定します。
3. 「**Delimited ASCII format（DEL）**」を選択します。「**Options**」をクリックし、列デリミタ（COLDEL）のデリミタとして縦線文字（|）を選択します。
4. INSERT インポート・モード（デフォルト）を使用します。
5. 「Commit」レコードを 500 と等しくなるように設定します。
6. メッセージ・ファイルとして、UNIX の場合は ${*DB2_HOME_DB*}/uddi/admin/db2/iso3166-99_tModelKey.log と、Windows の場合は *%DB2_HOME_DB%*¥uddi¥admin¥db2¥iso3166-99_tModelKey.log と入力します。
7. 「**Columns**」タブにアクセスします。「**Include Columns by Position**」を選択します。TMODEL_KEY を 1 に、KEY_NAME を 3 に、KEY_VALUE を 2 に、PARENT_VALUE を 4 にマップします。
8. 「**OK**」をクリックして、インポート・プロセスを実行します。
9. naics-97_tModelKey.txt ファイルおよび unspsc-73_tModelKey.txt ファイルについて、この手順を繰り返します。

---

**注意：** 使用しているデータベースのキャラクタ・セットが UTF-8 でない場合は、スクリプト iso3166-99.txt を使用して ISO 3166 分類をロードしないでください。これは、異なる言語の文字が分類に含まれているためです。かわりに、スクリプト iso3166-99-ascii.txt を使用して、ASCII 専用バージョンの分類をロードしてください。

---

## DB2 を使用するための OC4J の構成

次の各項では、DB2 パッケージの作成方法、および通常使用する URL の変更方法について説明します。

**DB2 パッケージの作成** DB2 が必要なデータベースであることを反映するために、次のように、jdbc/OracleUddi に設定されたデータ・ソースを名前と位置を指定して定義します。

```
<data-source
    class="com.evermind.sql.DriverManagerDataSource"
    name="jdbc/OracleUddi"
    location="jdbc/OracleUddi"
    connection-driver="com.oracle.ias.jdbc.db2.DB2Driver"
    username="uddisys"
    password="uddisys"
url="jdbc:oracle:db2://servername:50000;databaseName=UDDISYS;
PackageName=JDBCPKG;DynamicSections=512;
CreateDefaultPackage=TRUE;ReplacePackage=true"
/>
```

*servername* は、UDDI レジストリで使用する DB2 インスタンスをホスティングするサーバーのネットワーク名または IP アドレスであることに注意してください。また、url で始まる行とその後の 2 行は、必ず 1 行で入力し、空白や改行を入れないでください。ここでは、読みやすいように 3 行で表示しています。

データ・ソースは、対応する orauddi.ear アプリケーションおよび oraudrepl.ear アプリケーションの data-sources.xml ファイルを編集してアクセスできるようにする必要があります。

詳細は、『Oracle Application Server Containers for J2EE サービス・ガイド』の「データ・ソース」を参照してください。

ここで、次の例に示すように、UDDI 照会サーブレットのエンド・ポイントに接続すると、これらの初期 URL の接続文字列が使用され、適切なデフォルトパッケージが DB2 に作成されます。

```
http://OracleAS-host:port/uddi/inquiry
```

照会サーブレットのエンド・ポイントへのリクエストが停止または失敗した場合は、DB2 コントロール・センターから、JDBCPKGA パッケージおよび JDBCPKGB パッケージが uddisys データベースのアプリケーション・オブジェクトの下にあるかどうかを確認します。パッケージが作成されている場合は、OC4J インスタンスを停止し、次の項で説明するように URL を修正します。

**通常使用する URL の変更** すでに DB2 パッケージが作成されている場合、前述の手順（10-53 ページの「DB2 パッケージの作成」を参照）で定義したデータ・ソースを更新し、次の URL 属性を変更します。

```
url="jdbc:oracle:db2://servername:50000;databaseName=uddisys;
PackageName=JDBCPKG;DynamicSections=512;
CreateDefaultPackage=TRUE;ReplacePackage=true"
```

次のように変更します。

```
url="jdbc:oracle:db2://servername:50000;databaseName=uddisys;
PackageName=JDBCPKG;DynamicSections=512"
```

この例のテキストは必ず 1 行で入力し、空白や改行を入れないでください。ここでは、読みやすいように複数行で表示しています。

最後の 2 つのパラメータ CreateDefaultPackage と ReplacePackage は、変更後の URL 属性では削除されていることに注意してください。

orauddi.ear アプリケーションおよび oraudrepl.ear アプリケーションの data-sources.xml ファイルを変更した後、サーバーを再起動して変更内容を有効にします。

次の例に示すように、UDDI 照会サーブレットのエンド・ポイントに再度接続します。

```
http://OracleAS-host:port/uddi/inquiry
```

「OracleAS UDDI Registry」ページが表示されます。「Welcome! Your registry is now up and running.」というメッセージが表示されます。

# その他の Oracle データベース（Infrastructure 以外）

次の各項では、OracleAS Infrastructure データベース以外の Oracle データベースにおけるインストールおよび構成情報について説明します。

## スクリプト・ソース・ディレクトリ

関連する Oracle ツールを使用して、サーバーから /uddi/admin ディレクトリにアクセスできない場合は、このディレクトリをアクセス可能な位置にコピーしてください。このディレクトリは、UNIX の場合は ${*ORACLE_Home*}/uddi/admin、Windows の場合は *%ORACLE_Home_ORACLE%*¥uddi¥admin にあります。

## データベースとユーザーの作成

次の手順に従って、uddisys データベースおよび uddisys ユーザーを作成します。

1. UNIX の場合は ${*ORACLE_Home*} ディレクトリに、Windows の場合は *%ORACLE_Home_ORACLE%* ディレクトリにアクセスします。
2. sys ユーザー・アカウントを SQL*Plus で使用して、SQL スクリプト wurinst.sql を実行します。次に例を示します。

```
sqlplus "sys/change_on_install as sysdba" @wurinst.sql
```

パスワードが uddisys のスキーマ uddisys が作成されます。また、ログ・ファイル wurinst.log が作成されます。

## 検証された分類コードの移入

naics-97.ctl、iso3166-99.ctl および unspsc-73.ctl の 3 つの control スクリプトを SQL*Loader で使用して、検証された分類コードを移入します。次に例を示します。

```
sqlldr userid=uddisys/uddisys control=naics-97.ctl
sqlldr userid=uddisys/uddisys
   control=unspsc-73.ctl
sqlldr userid=uddisys/uddisys  control=iso3166-99.ctl
```

---

**注意：** 使用しているデータベースのキャラクタ・セットが UTF-8 でない場合は、スクリプト iso3166-99.ctl を使用して ISO 3166 分類をロードしないでください。これは、異なる言語の文字が分類に含まれているためです。かわりに、次のスクリプトを使用して、ASCII 専用バージョンの分類をロードしてください。

```
sqlldr userid=uddisys/uddisys control=iso3166-99-ascii.ctl
```

---

## OracleAS Infrastructure 以外のデータベースを使用するための OC4J の構成

OracleAS Infrastructure 以外のデータベースが必要なデータベースであることを反映するために、次のように、jdbc/OracleUddi に設定されたデータ・ソースを名前と位置を指定して定義します。

```
<data-source
    class="oracle.jdbc.pool.OracleConnectionCacheImpl"
    name="jdbc/OracleUddi"
    location="jdbc/OracleUddi"
    connection-driver="oracle.jdbc.driver.OracleDriver"
    username="uddisys"
    password="uddisys"
    url="jdbc:oracle:thin:@servername:1521:oracle sid"
/>
```

*servername* は、UDDI レジストリで使用する OracleAS Infrastructure 以外のデータベース・インスタンスをホスティングするサーバーのネットワーク名または IP アドレスであることに注意してください。

データ・ソースは、orauddi.ear アプリケーションおよび oraudrepl.ear アプリケーションからアクセスできる必要があります。

詳細は、『Oracle Application Server Containers for J2EE サービス・ガイド』の「データ・ソース」を参照してください。

UDDI サーバーを再起動して、変更内容を有効にします。

# OracleAS UDDI Registry サーバーのエラー・メッセージ

この項に示すエラー・コードは UDDI 管理者用です。通常、UDDI エラー・コード E_fatalError は、管理者による処理が必要な様々なサーバー・サイド・エラーを表します。

特定のサーバー・サイド・エラーは、J2EE アプリケーションのログ・ファイルに記録されます。orauddi.ear アプリケーションのログ・ファイル application.log は、通常、*J2EE_HOME*/application-deployments/orauddi ディレクトリにあります。

oraudrepl.ear アプリケーションのログ・ファイル application.log は、通常、*J2EE_HOME*/application-deployments/oraudrepl ディレクトリにあります。このリファレンスでは、管理者が問題を診断して解決するための追加情報を提供します。

**WUR-00010: WUR-00010: 存在しない構成パラメータ "{0}" を更新しようとしました。**

**原因 :** 指定した UDDI サーバー構成パラメータが存在しません。

**処置 :** 更新する構成パラメータの名前のスペルを訂正してください。詳細は、構成パラメータのリファレンス情報を参照してください。

**WUR-00011: WUR-00011: uddiserver.config の構成パラメータ "{0}" を更新しようとしました。このファイルは見つかりません。**

**原因 :** UDDI サーバー構成ファイル uddiserver.config が見つかりません。

**処置 :** OC4J インスタンスの JVM プロパティ oracle.home が正しく定義されていることを確認してください。

**WUR-00012: WUR-00012: 指定されたユーザー名 "{0}" は、レジストリで認識される名前ではありません。**

**原因 :** 指定したユーザーがレジストリ内に存在しません。

**処置 :** 指定したユーザー名のスペルを訂正してください。

**WUR-00013: パブリッシュ制限の ' デフォルト ' ロールは削除されない場合があります。**

**原因 :** システム定義のユーザー割当てロール 'Default' を削除しようとしました。

**処置 :** ユーザー割当てロール 'Default' は削除しないでください。ユーザー割当てロール 'Default' が不要な場合は、割当て制限を 0（ゼロ）に設定してこのロールを無効にしてください。

**WUR-00100: レスポンスをマーシャリング中に内部エラーが発生しました。**

**原因 :** クライアントへのレスポンスの書込み中に、予期しない内部エラーが発生しました。

**処置 :** 内部エラーを識別して訂正してください。内部エラーはエラーの詳細に埋め込まれています。

**WUR-00101: リクエストをアンマーシャリング中に内部エラーが発生しました。**

**原因 :** クライアントが送信したレスポンスの解析中に、予期しない内部エラーが発生しました。

**処置 :** 内部エラーを識別して訂正してください。内部エラーはエラーの詳細に埋め込まれています。

**WUR-00104: "{0}" という名前の構成パラメータの値が無効です。**

**原因 :** 指定した UDDI サーバー構成パラメータの値が無効です。

**処置 :** 有効値については、構成パラメータのリファレンス情報を参照してください。UDDI 管理ツールを使用して、構成パラメータを更新してください。

**WUR-00105: "{1}" を試行中に、SQL コード "{0}" のデータベース・エラーが発生しました。**

**原因 :** 指定したアクションの実行中に、予期しないデータベース・エラーが発生しました。

**処置 :** データベース・エラーを識別して訂正してください。データベース・エラーはエラーの詳細に埋め込まれています。

**WUR-00106: 内部エラーのため、リクエストが指定された更新の実行に失敗しました。変更をロールバック中に別のエラーが発生しました。そのため、データは予測不可能な状態になっています。**

**原因 :** エラー処理のロールバック・フェーズで、予期しないデータベース・エラーが発生しました。

**処置 :** データベース・エラーを識別して訂正してください。データベース・エラーはエラーの詳細に埋め込まれています。

**WUR-00107: リクエストされた変更をレジストリにコミット中に内部エラーが発生しました。そのため、データは予測不可能な状態になっています。**

**原因 :** リクエストされた変更のコミット中に、予期しないデータベース・エラーが発生しました。

**処置 :** データベース・エラーを識別して訂正してください。データベース・エラーはエラーの詳細に埋め込まれています。

**WUR-00108: 基礎となるデータベースに接続しようとしたときに、内部エラーが発生しました。**

**原因 :** リクエストを処理するためにデータベース接続を取得中に、予期しない内部エラーが発生しました。

**処置 :** データベース・エラーを識別して訂正してください。データベース・エラーはエラーの詳細に埋め込まれています。

**WUR-00109: 基礎となるデータベースへの接続をクローズしようとしたときに、内部エラーが発生しました。**

**原因 :** リクエストの処理後、データベース接続の解放中に、予期しないデータベース・エラーが発生しました。

**処置 :** データベース・エラーを識別して訂正してください。データベース・エラーはエラーの詳細に埋め込まれています。

**WUR-00110: 基礎となるデータベースのデータソース抽象化を作成および設定しようとしたときに、内部エラーが発生しました。**

**原因 :** データベース接続プールの作成中に、予期しない内部エラーが発生しました。

**処置 :** 内部エラーを識別して訂正してください。内部エラーはエラーの詳細に埋め込まれています。

**WUR-00111: JNDI 参照を実行し、オブジェクト "{0}" を検索しようとしたときに、内部エラーが発生しました。**

**原因 :** 指定したオブジェクトを JNDI コンテキストから取得中に、内部エラーが発生しました。考えられるオブジェクトの例として、データベース接続プール、メッセージ・キューなどがあります。

**処置 :** 内部エラーを識別して訂正してください。内部エラーはエラーの詳細に埋め込まれています。

**WUR-00113: データソース抽象化を設定するためリポジトリ API にアクセスしようとしたときに、内部エラーが発生しました。**

**原因 :** Oracle Application Server のメタデータ・リポジトリ・アクセス API を使用してデータベース接続プールを作成中に、予期しない内部エラーが発生しました。

**処置 :** 内部エラーを識別して訂正してください。内部エラーはエラーの詳細に埋め込まれています。

**WUR-00114: Universal Unique Identifier (UUID) を生成しようとしたときに、内部エラーが発生しました。**

**原因 :** UUID の生成中に、予期しない内部エラーが発生しました。

**処置 :** 内部エラーを識別して訂正してください。内部エラーはエラーの詳細に埋め込まれています。

**WUR-00115: レジストリが J2EE コンテナから OC4J 固有の環境設定を取得できません。ユーザー "{0}" を認証できません。**

**原因 :** ユーザーの認証中に、予期しない内部エラーが発生しました。このエラーは、通常、web.xml の設定が不適切な場合、または使用する OC4J コンテナのバージョンがサポートされていない場合に発生します。

**処置 :** 内部エラーを識別して訂正してください。内部エラーはエラーの詳細に埋め込まれています。

**WUR-00116: UDDI レジストリのインストール後の自動構成の実行中に内部エラーが発生しました。レジストリが正常に設定されていないと、標準のレジストリ処理を進めることができません。**

**原因 :** UDDI レジストリに対するインストール後の自動構成を実行中に、予期しない内部エラーが発生しました。

**処置 :** 内部エラーを識別して訂正してください。内部エラーはエラーの詳細に埋め込まれています。

**WUR-00117: データ・ソースを正常にクローズできません。**

**原因 :** UDDI レジストリを停止するとき、データベース接続プールのクローズ中に予期しない内部エラーが発生しました。

**処置 :** 内部エラーを識別して訂正してください。内部エラーはエラーの詳細に埋め込まれています。

**WUR-00200: 外部検証中に内部エラーが発生しました。**

**原因 :** 外部検証サービスへの妥当性チェック・コールの実行中に、予期しない内部エラーが発生しました。

**処置 :** 内部エラーを識別して訂正してください。内部エラーはエラーの詳細に埋め込まれています。

**WUR-00201: メモリー内リクエストの処理中に、外部検証中に内部エラーが発生しました。**

**原因 :** UDDI エンティティを送信して外部検証を行う前に、リクエスト内の UDDI エンティティを処理中に予期しない内部エラーが発生しました。

**処置 :** 内部エラーを識別して訂正してください。内部エラーはエラーの詳細に埋め込まれています。

**WUR-00202: tModel リスト・プロパティ "{0}" の形式が正しくないため、外部検証中に内部エラーが発生しました。**

**原因 :** UDDI サーバー構成プロパティ oracle.uddi.server.externalValidationTModelList の値が無効です。

**処置 :** 値を訂正してください。詳細は、構成パラメータのリファレンス情報を参照してください。

**WUR-00203: タイムアウト・プロパティ "{0}" の整数フォーマットが正しくないため、外部検証中に内部エラーが発生しました。**

**原因 :** UDDI サーバー構成プロパティ oracle.uddi.server.externalValidationTimeout の値が無効です。

**処置 :** 値を訂正してください。詳細は、構成パラメータのリファレンス情報を参照してください。

**WUR-00204: レスポンスが正しい DispositionReport ではないため、外部検証中に内部エラーが発生しました。**

**原因 :** 外部検証サービスで戻された DispositionReport が無効です。たとえば、DispositionReport が空です。

**処置 :** 外部検証サービス・プロバイダに連絡してください。

**WUR-00205: 予期せぬレスポンスのため、外部検証中に内部エラーが発生しました。レスポンスはメッセージ "{1}" のコード "{0}" です。**

**原因 :** 外部検証サービスで戻された DispositionReport に、予期しない DispositionReport エラー番号が含まれています。

**処置 :** 外部検証サービス・プロバイダに連絡してください。

**WUR-00300: DB スキーマのバージョンがありません。VERSION 表で DB を確認してください。**

**原因 :** 永続記憶域用のデータベース・スキーマのバージョンが欠落しています。

**処置 :** オラクル社カスタマ・サポート・センターに連絡してください。

**WUR-00301: DB スキーマのバージョン "{0}" は中間層のバージョンと互換性がありません。DB スキーマを更新して UDDI レジストリ関数を作成する必要があります。**

**原因 :** 永続記憶域用のデータベース・スキーマのバージョンが、使用しているレジストリのバージョンでサポートされていません。

**処置 :** データベース・スキーマを最新バージョンにアップグレードしてください。詳細は、UDDI データベース・スキーマのアップグレードのマニュアルを参照してください。

**WUR-00302: UDDI DELTA サーバー・プロパティ・ファイルを取得およびロードしようとしたときに、内部エラーが発生しました。**

**原因 :** 古いバージョンのデータベース・スキーマを使用して下位互換性モードで UDDI レジストリを初期化中に、内部エラーが発生しました。

**処置 :** オラクル社カスタマ・サポート・センターに連絡してください。

**WUR-00303: この操作は DB スキーマのバージョン "{0}" では許可されていません。この操作を実行するには、DB スキーマを最新のバージョンにアップグレードする必要があります。**

**原因 :** UDDI レジストリが古いバージョンのデータベース・スキーマを使用して下位互換性モードで実行されているため、リクエストされた操作はサポートされませんでした。

**処置 :** データベース・スキーマを最新バージョンにアップグレードしてください。詳細は、UDDI データベース・スキーマのアップグレードのマニュアルを参照してください。

**WUR-05001: たった今保存された UDDI エンティティが見つかりません。**

**原因 :** 更新ジャーナルの更新中に、予期しない内部エラーが発生しました。

**処置 :** オラクル社カスタマ・サポート・センターに連絡してください。

**WUR-05002: businessEntity または tModel ではないエンティティを管理転送することはできません。不正なエンティティのキーは "{0}" です。**

**原因 :** 管理転送の変更レコードで、指定した UDDI エンティティが businessEntity または tModel ではありません。

**処置 :** 変更レコードが作成された UDDI ノードの管理者に連絡してください。

**WUR-05003: 警告 : USN が "{1}" のノード "{0}" から、重複した変更レコードを受信しました。**

**原因 :** 指定した UDDI ノードから送信された変更レコードが重複しています。

**処置 :** 処置は必要ありません。これは情報メッセージです。

**WUR-05004: USN が "{1}" のノード "{0}" から、正しくない変更レコードを受信しました。USN が "{2}" の変更レコードが処理されました。**

**原因 :** 指定した変更レコードより更新順序番号（USN）が大きい変更レコードの処理後に、指定した変更レコードを受信しました。

**処置 :** 変更レコードが作成された UDDI ノードの管理者に連絡してください。

**WUR-05005: USN が "{1}" のノード "{0}" の変更レコードは、この名前付きノードが認識されないため無効です。**

**原因 :** 指定した変更レコードを作成したノードが認識されませんでした。つまり、レプリケーション通信グラフにノードが記録されていません。

**処置 :** 変更レコードを提供した UDDI ノードの管理者に連絡してください。

# uddiadmin.jar ツールのコマンドライン・オプション

次の各項では、uddiadmin.jar コマンドライン・ツールのオプションについて説明します。ほとんどの場合、コマンドラインでは次の形式を使用します（setWalletPassword では別の URL を使用します）。

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
[-verbose] options_and_their_parameters
```

コマンドは必ず 1 行で入力してください。uddiadmin.jar コマンドライン・ツールの詳細は、10-33 ページの「コマンドライン・ツール uddiadmin.jar の使用」を参照してください。

## changeOwner

**形式**

```
-changeOwner new_username {-businessKey business_Key | -tModelKey
tModel_Key}
```

**説明**

指定したエンティティの所有権を、新規に指定したユーザーに変更します。

## correctChangeRecord

**形式**

```
-correctChangeRecord changeRecordCorrectionfile
changeRecord_NewDatafile
```

**説明**

changeRecordCorrectionfile ファイルの内容と changeRecordNewDatafile ファイルの内容を UDDI ノードに適用します。これらのファイルの内容は、UDDI レプリケーションの XML Schema に準拠する必要があります。このオプションは、UDDI レプリケーションのエラー・リカバリの一部です。

## deleteEntity

**形式**

```
-deleteEntity {-businessKey business_Key | -serviceKey serviceKey |
-bindingKey binding_Key | -tModelKey tModel_Key}
```

**説明**

指定したエンティティをその所有者に関係なく削除します。tModel エンティティの場合、この操作では非永続的な削除（非表示）操作が実行されることに注意してください。

## deleteRoleQuotaLimits

**形式**

```
-deleteRoleQuotaLimits role_Name [role_Name ...]
```

**説明**

指定した割当てグループについて、グループから割当て制限へのマッピングを削除します。このオプションの使用方法の詳細は、10-37 ページの「割当てグループの削除（拡張操作）」を参照してください。

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -deleteRoleQuotaLimits role_Name
```

## destroyTModel

**形式**

```
-destroyTModel tModel_Key
```

**説明**

指定した tModel をレジストリから永続的に削除します（UDDI 定義の delete_tModel コールの場合、tModel エンティティは単に非表示になります）。

## doPing

**形式**

```
-doPing replicationEndPointUrl [-timeout timeInMilliseconds ] [
-walletPassword wallet_password]
```

**説明**

UDDI レプリケーションの do_ping メッセージを、指定したレプリケーション・エンド・ポイント URL に送信します。これは、他のエンド・ポイントがアクティブであるかどうかをチェックするために使用する、TCP/IP の ping コマンドに類似しています。do_ping メッセージを受信する JVM に Wallet の有効なパスワード・セットがない場合は、オプションの walletPassword パラメータを使用すると便利です。

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-doPing http://OracleAS-host:port/uddirepl/replication
```

## downloadReplicationConfiguration

**形式**

```
-downloadReplicationConfiguration
```

**説明**

現在使用しているレプリケーション構成を、OracleAS UDDI Registry 内の指定の UDDI ノードからダウンロードします。ダウンロードを正常に完了するには、レプリケーション構成をアップロードする必要があります。このオプションの使用方法の詳細は、10-40 ページの「UDDI レプリケーションの有効化」を参照してください。

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-downloadReplicationConfiguration
```

## getChangeRecord

**形式**

```
-getChangeRecord local_usn
```

**説明**

local_usn（整数）で指定された変更レコードの詳細を取得します。この API を CorrectChangeRecord オプションとともに使用して、OracleAS UDDI Registry の異なる UDDI ノード間で不適切なデータまたは一貫性のないデータを訂正します。

## getHighWaterMarks

**形式**

```
-getHighWaterMarks replicationEndPointUrl [ -walletPassword
wallet_password]
```

**説明**

最高水位標ベクターを、*replicationEndPointUrl* パラメータで指定された UDDI ノードから取得します。do_ping メッセージを受信する JVM に Wallet の有効なパスワード・セットがない場合は、オプションの walletPassword パラメータを使用すると便利です。

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-getHighWaterMarks http://OracleAS-host:port/uddirepl/replication
```

## getProperties

**形式**

```
-getProperties
```

**説明**

現行のレジストリ構成パラメータをリスト表示します。このオプションの使用方法の詳細は、10-34 ページの「サーバーの構成」を参照してください。

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -getProperties
```

## getRoleQuotaLimits

**形式**

```
-getRoleQuotaLimits
```

**説明**

現在レジストリに設定されている J2EE ロールから割当て制限へのすべてのマッピングを表示します。このオプションの使用方法の詳細は、10-38 ページの「割当てグループとその割当て制限のリスト表示」を参照してください。

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -getRoleQuotaLimits
```

## getUserDetail

**形式**

```
-getUserDetail username_to_retrieve
```

**説明**

指定したユーザーの詳細、現在は各ユーザーの authorizedName を取得します。このオプションの使用方法の詳細は、10-35 ページの「ユーザーの管理」を参照してください。

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -getUserDetail username
```

## getUsers

**形式**

```
-getUsers
```

**説明**

レジストリにエンティティがある既存のユーザーをすべてリスト表示します。このオプションの使用方法の詳細は、10-35 ページの「ユーザーの管理」を参照してください。

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
[-verbose] -getUsers
```

## import

**形式**

```
-import [-s|-m]

 -businesses filename |

 -tmodels filename    |

 -assertions filename -fromBusinessCheck {true|false}

          -toBusinessCheck {true|false} ]
```

**説明**

指定したファイル内のすべての businessEntity と tModel データ構造、および publisherAssertion データ構造をインポートします。businessEntity データ構造をインポートする場合、インポート用に指定するファイル（`filename`）には、UDDI の businessDetail の XML 文書が含まれている必要があります。tModel データ構造をインポートする場合、指定するファイルには UDDI の tModelDetail の XML 文書が含まれている必要があります。各ファイルをインポートすることによって、エンティティ・キー（businessKey、serviceKey、bindingKey、tModelKey など）が保たれます。ただし、operatorName および authorizedName フィールドは保たれません。operatorName フィールドは、レジストリの operatorName 構成パラメータで置換されます。インポートされるエンティティの所有者は管理者であるため、authorizedName フィールドは管理者の認可名となります。

インポートは、単一モード（`-s`）または複数モード（`-m`）で実行できます。単一モードでは、部分的な成功（エンティティの一部がインポートされ、残りはエラーが発生したためインポートされない）が許容されません。複数モードでは、部分的な成功が許容されます。

インポート・パラメータが特に役立つのは、なんらかの認可ソースから既知のサービス・インタフェース仕様の tModel と分類の tModel データ構造をインポートする場合です。

エンティティ・キーが保たれるため、管理者はエンティティのソースを慎重に評価して、エンティティ・キーに衝突が発生しないことを確認する必要があります。

publisherAssertion をインポートする場合は、2 つのブール値が必要です。この 2 つのブール値は、publisherAssertion の挿入元のサイド（2 つのブール値が true の場合は両方のサイド）を示すために使用されます。

このオプションの使用方法の詳細は、10-39 ページの「エンティティのインポート」を参照してください。

**例**

次の例では、ファイル `assert.xml` に含まれる publisherAssertion をインポートしています。

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -import -s -assertions assert.xml -fromBusinessCheck true
 -toBusinessCheck true
```

## setOperationalInfo

**形式 1**

```
-setOperationalInfo {-businessKey key | -tModelKey key } [-newOperator
OperatorName] [-newAuthorizedname authName] [-newTime timestamp]
```

**形式 2**

```
-setOperationalInfo {-serviceKey key  | -bindingKey key } [-newTime
timestamp]
```

**説明**

たとえば、インポート操作に従って、キーで指定する businessEntity または tModel データ構造のオペレータ名、認可名、タイムスタンプなどの操作情報を設定します。setOperationalInfo オプションを使用してオペレータ名、認可名、タイムスタンプを任意に組み合せることができます。

形式 1 を使用すると、キーで指定する businessEntity または tModel のオペレータ名、認可名、タイムスタンプのいずれか、あるいは 3 つすべてを変更できます。

形式 2 を使用すると、businessService または bindingTemplate のタイムスタンプのみ変更できます。

---

**注意：** タイムスタンプの書式は、java.sql.Timestamp で 'yyyy-mm-dd hh.mm:ss.ffffffff' と定義されています。次に例を示します。

```
'2002-12-01 00:00:00'
```

タイムスタンプ値の yyyy-mm-dd と hh.mm:ss.ffffffff の間には空白が 1 つ入るため、コマンドラインでは値全体を一重引用符で囲む必要があります。

---

**注意：** 通常、レプリケーションが有効な場合、setOperationalInfo オプションは使用しないでください。

---

このオプションの使用方法の詳細は、10-40 ページの「操作情報の設定」を参照してください。

## setProperty

**形式**

```
-setProperty property_name=value
```

**説明**

指定したサーバー構成のプロパティ値を変更します。変更内容を有効にするには、OracleAS UDDI Registry の J2EE アプリケーションを再起動する必要があります。

**例**

次の例では、operatorName プロパティを OracleUddiServerIT_Dept に設定しています。

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
  -setProperty oracle.uddi.server.operatorName=OracleUddiServerIT_Dept
```

---

**注意：** setProperty オプションを使用してサーバー構成のプロパティ値を変更する場合は、注意が必要です。プロパティの設定が不適切な場合は、レジストリの整合性に重大な障害を与える可能性があります。

---

このオプションの詳細は、10-7 ページの「インストール時または初期のプロパティの変更」を参照してください。

## setRoleQuotaLimits

**形式**

```
-setRoleQuotaLimits roleName [maxBE] [maxBSperBE] [ maxBTperBS] [maxTM]
[maxPA]
```

**説明**

指定した割当てグループに割当て制限値を設定します。このオプションを使用して、グループから割当て制限への新規マッピングの作成、または既存のマッピングの更新ができます。パラメータは次のように定義されています。

- roleName: 指定の割当て制限にマップする割当てグループの名前
- maxBE: businessEntity データ構造の許容最大数
- maxBSperBE: businessEntity ごとの businessService データ構造の許容最大数
- maxBTperBS: businessEntity ごとの bindingTemplate データ構造の許容最大数
- maxTM: tModel データ構造の許容最大数
- maxPA: publisherAssertion データ構造の許容最大数

値 -1 は無制限を意味します。

このオプションの詳細は、10-37 ページの「割当てグループの制限の更新」を参照してください。

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setRoleQuotaLimits roleName maxBE maxBSperBE maxBTperBS maxTM maxPA
```

## setWalletPassword

**形式**

```
-setWalletPassword wallet_password
```

**説明**

UDDI レプリケーション用の UDDI ノード間の HTTPS 通信で使用する、Wallet のパスワードを設定します。セキュリティ上の理由で Wallet のパスワードは永続的に保持されないため、アプリケーションが再起動するたびにこのオプションを起動する必要があります。レジストリ・レプリケーションの Wallet 管理 URL は、次のとおりです。

```
http://OracleAS-host:port/uddirepl/admin/wallet
```

このオプションの UDDI レプリケーションとの使用方法の詳細は、10-40 ページの「UDDI レプリケーションの有効化」を参照してください。

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddirepl/admin/wallet username
password
 -setWalletPassword=walletpassword
```

## transferCustody

**形式**

```
-transferCustody oldOperatorName newOperatorName newAuthorizedName
{-tModelKey tModel_Key | -businessKey businessKey}
```

**説明**

tModel または businessEntity の管理を、新規のオペレータと新規の認可名に転送します。このオプションは、UDDI 仕様で定義される管理転送の一部です。

## uploadReplicationConfiguration

**形式**

```
-uploadReplicationConfiguration
xml_file_containing_replication_configuration
```

**説明**

指定のレプリケーション構成を、OracleAS UDDI Registry 内の特定の UDDI ノードにアップロードします。新しいレプリケーション構成を使用するには、アプリケーションを再起動する必要があります。このオプションの使用方法の詳細は、10-40 ページの「UDDI レプリケーションの有効化」を参照してください。

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-uploadReplicationConfiguration xml_file_containing_replication_configuration
```

# サーバー構成のプロパティ

この項では、UDDI サーバー構成プロパティのリファレンス情報について説明します。プロパティはクラス oracle.uddi.server およびそのサブクラスのものです。これらのプロパティは、コマンドライン・ツール uddiadmin.jar のオプションを使用して設定します。uddiadmin.jar コマンドライン・ツールの詳細は、10-33 ページの「コマンドライン・ツール uddiadmin.jar の使用」を参照してください。

## addressTModelKeyValidation

**説明**
address 要素内で tModelKey の存在の妥当性チェックを行うかどうかを決定します。

**プロパティ・タイプ / 許容値**
ブール（true、false）

**初期値**
true

**標準値**
true

**例**
```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.addressTModelKeyValidation=true
```

## assertionKeyedRefValidation

**説明**
すべての publisherAssertion の keyedReference エンティティの妥当性チェックを制御します。

**プロパティ・タイプ / 許容値**
- full: すべての妥当性チェック条件をチェックします。
- tmodel_existence: tModelKey の存在のみチェックします。
- none: 条件はチェックしません。

**初期値**
full

**標準値**
full

**例**
```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.assertionKeyedRefValidation=full
```

## businessEntityURLPrefix

**説明**
レジストリに保存される各 businessEntity データ構造に対して自動的に生成される検出用 URL のプリフィックスを指定します。このプリフィックスは、デプロイ環境に応じてカスタマイズする必要があります。このパラメータ設定は、データベース内の既存のエンティティに遡及して適用されます。たとえば、discoveryURL のプリフィックスを変更すると、古い URL プリフィックスで始まる usetype businessEntity のすべての検出用 URL が、新しい URL プリフィックスに変更されます。

**プロパティ・タイプ / 許容値**
有効な URL

**初期値**
サーバーを初期化すると、OracleAS UDDI Registry により初期値が生成されます。

**標準値**
ホスト名とポートには、Web サーバーのホスト名とポートを使用する必要があります（サーブレット・コンテナと同じでも異なっていてもかまいません）。

**注意**
このプロパティの使用方法の詳細は、10-7 ページの「インストール時または初期のプロパティの変更」を参照してください。

**例**
```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty
oracle.uddi.server.businessEntityURLPrefix="http://uddihost:port/uddi/inquiryget"
```

## categoryValidation

**説明**
すべての CategoryBag エンティティの妥当性チェックを制御します。

**プロパティ・タイプ / 許容値**
- full: すべての妥当性チェック条件をチェックします。
- tmodel_existence: tModelKey の存在のみチェックします。
- none: 条件はチェックしません。

**初期値**
full

**標準値**
full

**例**
```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.categoryValidation=full
```

## categoryValidationTModelKeys

**説明**
分類と識別子の tModel キーを表します。このキーは、保存操作中にレジストリによって検証されます。

**プロパティ・タイプ / 許容値**
'<tModelKey1>', '<tModelKey2>', '<tModelKey3>' 形式のリスト

**初期値**
'UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4'。この値は uddi-org:types 分類を表します。ただし、事前インストール済の値は、UDDI タイプの分類に UDDI v1.0 仕様に定義されている 3 つの分類（uddi-org:types, uddi-org:iso-ch:3166-1999、ntis-gov:naics:1997、unspsc-org:unspsc）を付けたものです。

**標準値**
事前インストール済の値

**注意**
uddi-org:types 分類は、リストから削除しないでください。また、コマンドは必ず 1 行で入力し、改行や余分な空白を入れないでください。

**例**
```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty
"oracle.uddi.server.categoryValidationTModelKeys=
'UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4',
'UUID:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88',
'UUID:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2',
'UUID:CD153257-086A-4237-B336-6BDCBDCC6634' "
```

## changeRecordWantsAck

**説明**
ローカル・ノードから送信された変更レコードで ACK が必須かどうかを制御します。

**プロパティ・タイプ / 許容値**
ブール（true、false）

**初期値**
false

**標準値**
false

**例**
```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.replication.changeRecordWantsAck=false
```

## defaultLang

**説明**
UDDI v1.0 記述要素は言語修飾がないため、この記述要素で指定するレジストリのデフォルト言語を指定します。UDDI v2 リクエストの言語はデフォルトで設定されていません。xml:lang 属性の値が有効値になります。

**プロパティ・タイプ / 許容値**
xml:lang の値

**初期値**
en

**標準値**
レジストリで管理されるプライマリ・リージョンの位置

**例**
```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.defaultLang=en
```

## externalValidation

**説明**
外部検証を行うかどうかを決定します。

**プロパティ・タイプ / 許容値**
ブール（true、false）

**初期値**
false

**標準値**
false

**注意**
このプロパティの使用方法の詳細は、10-46 ページの「外部カテゴリ検証の有効化」を参照してください。

**例**
```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.externalValidation=true
```

## externalValidationTimeout

**説明**
外部検証のタイムアウトが発生するまでの時間をミリ秒で定義します。

**プロパティ・タイプ / 許容値**
long

**初期値**
5000

**標準値**
該当なし

**注意**
このプロパティの使用方法の詳細は、10-46 ページの「外部で検証されたカテゴリをレジストリに追加」を参照してください。

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.externalValidationTimeout=5000
```

## externalValidationTModelList

**説明**

外部 SOAP サービスで検証される分類と識別子の tModel データ構造を表す tModel キーと URL のペアのリストを指定します。ペア内の tModelKey と URL はカンマ（,）で区切られ、値ペアはセミコロン（;）で区切られます。

**プロパティ・タイプ / 許容値**

該当なし

**初期値**

NULL 値 ""

**標準値**

NULL 値 ""

**注意**

このプロパティの使用方法の詳細は、10-46 ページの「外部で検証されたカテゴリをレジストリに追加」を参照してください。

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.externalValidationTModelList=uuid:acme-taxonomy-key,
 http://acme.com/externalValidation
```

## hostingRedirectorValidation

**説明**

bindingTemplate 要素内で hostingRedirector の妥当性チェックを行うかどうかを決定します。妥当性チェックによって、参照先の bindingTemplate 要素が存在し、hostingRedirector 要素が含まれていないことが確認されます。

**プロパティ・タイプ / 許容値**

ブール（true、false）

**初期値**

true

**標準値**

true

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.hostingRedirectorValidation=true
```

## identifierValidation

**説明**
すべての IdentifierBag エンティティの妥当性チェックを制御します。

**プロパティ・タイプ / 許容値**
- full: すべての妥当性チェック条件をチェックします。
- tmodel_existence: tModelKey の存在のみ検証します。
- none: 条件はチェックしません。

**初期値**
full

**標準値**
full

**例**
```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
  -setProperty oracle.uddi.server.identifierValidation=full
```

## jdbcDriverType

**説明**
OracleAS Infrastructure データベースにアクセスするために使用する JDBC ドライバのタイプを定義します。このプロパティが適用できるのは、OracleAS Infrastructure データベースをバックエンド記憶域として使用する場合のみです。

**プロパティ・タイプ / 許容値**
thin または oci

**初期値**
thin

**標準値**
該当なし

**注意**
クラスタ環境では、OC4J インスタンスごとにこのプロパティを設定する必要があります。

**例**
```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.db.jdbcDriverType=thin
```

## maxChangeRecordsSentEachTime

**説明**
受信 getChangeRecords リクエストに応答して送信する変更レコードの最大数を制御します。

**プロパティ・タイプ / 許容値**
整数

**初期値**
100

**標準値**
該当なし

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.replication.maxChangeRecordsSentEachTime=100
```

## maxConnections

**説明**

接続プール内のデータベース接続の最大数を決定します。このプロパティが適用できるのは、OracleAS Infrastructure データベースをバックエンド記憶域として使用する場合のみです。

**プロパティ・タイプ / 許容値**

正の整数

**初期値**

8

**標準値**

同時リクエストの最大数、および必要なパフォーマンスに応じて決定します。

**注意**

同時リクエストの予測最大数、およびバッファのパーセンテージに応じて決定した値を入力します。

クラスタ環境では、OC4J インスタンスごとにこのプロパティを設定する必要があります。

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.db.maxConnections=10
```

## minConnections

**説明**

接続プール内のデータベース接続の最小数を決定します。このプロパティが適用できるのは、OracleAS Infrastructure データベースをバックエンド記憶域として使用する場合のみです。

**プロパティ・タイプ / 許容値**

`maxConnections` の値より小さく、負でない整数

**初期値**

1

**標準値**

1

**注意**

クラスタ環境では、OC4J インスタンスごとにこのプロパティを設定する必要があります。

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.db.minConnections=1
```

## operatorCategory

**説明**
categoryValidation プロパティの値が true の場合に、追加エンティティをオペレータ・ノードとして分類するかどうかを決定します。

**プロパティ・タイプ / 許容値**
ブール（true、false）

**初期値**
true

**標準値**
true

**例**
```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.categoryValidation.operatorCategory=true
```

## operatorName

**説明**
OracleAS UDDI Registry のオペレータの名前を指定します。この名前はレスポンスの operator 属性に表示されます。このパラメータ設定は、データベース内の既存のエンティティに遡及して適用されます。たとえば、オペレータ名を変更すると、現在、古いオペレータ名を使用しているすべてのビジネスおよび tModel データ構造が、新しいオペレータ名に変更されます。

**プロパティ・タイプ / 許容値**
非 NULL 文字列

**初期値**
OracleUddiServer

**標準値**
<UDDI レジストリのドメイン >/uddi

**注意**
このパラメータは、レプリケーションを有効にする前に設定してください。

このプロパティの使用方法の詳細は、10-46 ページの「外部で検証されたカテゴリをレジストリに追加」を参照してください。

**例**
```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
  -setProperty oracle.uddi.server.operatorName=OracleUddiServerIT_Dept
```

## pushEnabled

**説明**
UDDI レプリケーションに対して送信タスクを実行するかどうかを制御します。

**プロパティ・タイプ / 許容値**
ブール（true、false）

**初期値**
true

**標準値**
true

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.replication.pushEnabled=true
```

## pushTaskExecutionPeriod

**説明**
送信タスクの実行期間（ミリ秒）を制御します。

**プロパティ・タイプ / 許容値**
45000

**初期値**
該当なし

**標準値**
該当なし

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.replication.pushTaskExecutionPeriod=45000
```

## quotaLimitChecking

**説明**
公開の割当て制限（レジストリで作成可能なユーザーごとのエンティティ数の制限）を適用するかどうかを決定します。

**プロパティ・タイプ / 許容値**
ブール（true、false）

**初期値**
true

**標準値**
true

**注意**
割当て制限の詳細は、10-36 ページの「割当て制限の適用」を参照してください。

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.quotaLimitChecking=true
```

## schemaValidationUponIncomingRequests

**説明**
サーバーで受信リクエストを UDDI XML Schema に対して検証するかどうかを決定します。

**プロパティ・タイプ / 許容値**
ブール（true、false）

**初期値**
true

**標準値**
true

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
-setProperty oracle.uddi.server.schemaValidationUponIncomingRequests=true
```

## soapRequestAuthMethod

**説明**
レジストリ・ノードでレプリケーション SOAP リクエストを他のノードに送信するときに使用する認証方式を制御します。値 CLIENT-CERT を使用する場合、管理者は、レジストリ・ノードが起動または再起動するたびに Wallet のパスワードを設定する必要があります。

**プロパティ・タイプ / 許容値**
NONE または CLIENT-CERT

**初期値**
NONE

**標準値**
CLIENT-CERT

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.replication.soapRequestAuthMethod=NONE
```

## soapRequestTimeout

**説明**
各 SOAP レプリケーション・リクエストのタイムアウト値（ミリ秒）を制御します。

**プロパティ・タイプ / 許容値**
long

**初期値**
180000

**標準値**
該当なし

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.replication.soapRequestTimeout=180000
```

## startMaintainingUpdateJournal

**説明**
UDDI レプリケーションに対して更新ジャーナルを保持するかどうかを制御します。レプリケーションが有効な場合、このプロパティは true に設定する必要があります。

**プロパティ・タイプ / 許容値**
ブール（true、false）

**初期値**
false

**標準値**
true

**注意**
このプロパティを `true` に設定する前に、適切なレプリケーション構成をアップロードしてください。

このプロパティを true に設定した後、false に設定できるのは、UDDI レプリケーションを実行する必要がなくなった場合のみです。このプロパティを不注意に true から false に設定すると、変更レコードに致命的な障害が発生します。

このプロパティの使用方法の詳細は、10-40 ページの「UDDI レプリケーションの有効化」を参照してください。

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.replication.startMaintainingUpdateJournal=false
```

## status

**説明**
レプリケーション・リクエストを送信するためにスケジューラを有効にするかどうかを指定します。

**プロパティ・タイプ / 許容値**
ブール（0 = オフ、1 = オン）

**初期値**
1

**標準値**
1

**注意**
このプロパティの使用方法の詳細は、10-40 ページの「UDDI レプリケーションの有効化」を参照してください。

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
  -setProperty oracle.uddi.server.scheduler.status=1
```

## stmtCacheType

**説明**
文キャッシュのタイプを定義します。このプロパティは、OracleAS Infrastructure データベースおよび JDBC ドライバでのみ使用します。

**プロパティ・タイプ / 許容値**
NONE、IMPLICIT、EXPLICIT

**初期値**
NONE

**標準値**
EXPLICIT

**注意**
クラスタ環境では、OC4J インスタンスごとにこのプロパティを設定する必要があります。

**例**

```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.db.stmtCacheType=NONE
```

## stmtCacheSize

**説明**
接続ごとの文キャッシュのサイズ（キャッシュされる文の数）を定義します。このプロパティは、OracleAS Infrastructure データベースおよび JDBC ドライバでのみ使用します。

**プロパティ・タイプ / 許容値**
整数

**初期値**
50

**標準値**
50

**例**
```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.db.stmtCacheSize=50
```

## taskExecutionPeriod

**説明**
レプリケーション・タスクを実行する期間をミリ秒で制御します。

**プロパティ・タイプ / 許容値**
long

**初期値**
5000

**標準値**
該当なし

**例**
```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
  -setProperty oracle.uddi.server.replication.taskExecutionPeriod=5000
```

## timer_pool_size

**説明**
スケジューラで使用する、同時にアクティブなスレッドの数を指定します。

**プロパティ・タイプ / 許容値**
該当なし

**初期値**
1

**標準値**
1

**注意**
このプロパティの使用方法の詳細は、10-42 ページの「UDDI レプリケーション・スケジューラのプロパティの設定」を参照してください。

**例**
```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.scheduler.timer_pool_size=1
```

## tModelInstanceInfoKeyValidation

**説明**
tModelInstanceInfo 要素内で tModelKey の存在の妥当性チェックを行うかどうかを決定します。

**プロパティ・タイプ / 許容値**
ブール（true、false）

**初期値**
true

**標準値**
true

**例**
```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.tModelInstanceInfoKeyValidation=true
```

## walletLocation

**説明**
Wallet のファイル名を定義します。Wallet のファイルは、`uddiserver.config` ファイルと同じ場所にあります。

**プロパティ・タイプ / 許容値**
該当なし

**初期値**
ewallet.p12

**標準値**
該当なし

**注意**
このプロパティの使用方法の詳細は、10-40 ページの「UDDI レプリケーションの有効化」を参照してください。

**例**
```
java -jar uddiadmin.jar http://OracleAS-host:port/uddi/admin username password
 -setProperty oracle.uddi.server.replication.walletLocation=ewallet.p12
```

# 11

# J2EE アプリケーションでの Web サービスの使用

この章では、Java 2 Platform, Enterprise Edition（J2EE）アプリケーションで Web サービスを使用する方法について説明します。次の 1 種類の Web ベースの情報またはサービスがサポートされています。

- WSDL を使用して記述された SOAP ベースの Web サービス。「WSDL を使用した SOAP ベースの Web サービスの利用」を参照してください。

また、J2EE アプリケーションでは、実行時に WSDL ドキュメントを取得する場合、動的起動 API を使用して、WSDL ドキュメントに記述された SOAP 操作が起動されます。動的起動 API の使用方法は、「Web サービスの動的起動」を参照してください。

# WSDL を使用した SOAP ベースの Web サービスの利用

J2EE 開発者は、wsdl2ejb ユーティリティを使用して、Web Services Description Language（WSDL）ドキュメントに記述された Web サービスをアプリケーションで使用できます。このユーティリティでは、WSDL ドキュメントとオプションの追加パラメータを使用して、OC4J にデプロイできる EJB EAR ファイルが生成されます。EJB リモート・インタフェースは、WSDL の portType に基づいて生成されます。各 WSDL 操作は、EJB メソッドにマップされます。EJB メソッドのパラメータは WSDL 操作の入力メッセージ部から導出されますが、EJB メソッドの戻り値は WSDL 操作の出力メッセージ部からマップされます。OracleAS SOAP のマッピング・レジストリを使用して、XML の型が対応する Java の型にマップされます。

WSDL と SOAP に関する追加の参照情報は、次の場所にあります。

- WSDL 1.1 仕様については、次の URL を参照してください。

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

- SOAP 1.1 仕様については、次の URL を参照してください。

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

表 11-1 に、wsdl2ejb ユーティリティを実行する場合のコマンドライン・オプションを示します。

**表 11-1 wsdl2ejb ユーティリティのコマンドライン・オプション**

| オプション | 説明 |
|---|---|
| -conf <*config file*> | wsdl2ejb ユーティリティで構成ファイルをロードできます。 |
| -d <*destDir*> | 生成された EJB EAR ファイルの書込み先ディレクトリを指定できます。 |
| -Dhttp.proxyHost | HTTP URL を使用して WSDL ドキュメントの位置を指定し、そのドキュメントへのアクセスに HTTP プロキシ・サーバーが必要な場合に、プロキシ・ホスト名を指定できます。 |
| -Dhttp.proxyPort | HTTP URL を使用して WSDL ドキュメントの位置を指定し、そのドキュメントへのアクセスに HTTP プロキシ・サーバーが必要な場合に、プロキシ・ポート番号を指定できます。 |
| -jar | wsdl2ejb ユーティリティを JAR ファイルとして指定できます。 |

wsdl2ejb ユーティリティを実行するには、次のコマンドを入力します。*<destDir>* は生成された EJB EAR ファイルの書込み先ディレクトリで、ファイル mydoc.wsdl は WSDL ドキュメントの位置です。

```
java -jar wsdl2ejb.jar -d <destDir> mydoc.wsdl
```

> **注意：** wsdl2ejb.jar ファイルはインストール・ディレクトリにあります。このディレクトリは、UNIX の場合は *$ORACLE_HOME*/webservices/lib、Windows の場合は *%ORACLE_HOME*¥webservices¥lib です。

HTTP URL を使用して WSDL ドキュメントの位置を指定し、そのドキュメントへのアクセスに HTTP プロキシが必要な場合は、このユーティリティの実行に次のコマンドと構文を使用する必要があります。

```
java -Dhttp.ProxyHost=myProxyHost -Dhttp.proxyPort=80 -jar wsdl2ejb.jar -d <destDir>
http://myhost/mydoc.wsdl
```

この例では、指定した WSDL を使用して、宛先ディレクトリ（*<destDir>*）に EJB EAR ファイルが生成されます。EJB クラス名、Java Naming and Directory Interface（JNDI）バインディング・キーおよび Java パッケージ名は、WSDL に記述された SOAP サービスの位置から導出されます。

このコマンド構文では、wsdl2ejb ユーティリティにより、OracleAS SOAP マッピング・レジストリでデフォルトでサポートされている XML の型がマップされます。

wsdl2ejb ユーティリティでは、コマンドラインに指定した宛先ディレクトリ（*<destDir>*）に次のファイル・セットが生成されます。生成されたファイルは、次のディレクトリ・レイアウトを使用して保存されます。

```
Root /
     + app.ear
     + src/
       + ... generated java sources ...
     + classes/
       + META-INF/
         + ejb-jar.xml
       + ... compiled classes and xml resources ....
     + deploy/
       + ejb.jar
       + META-INF/
         + application.xml
```

- .ear ファイル（OC4J にデプロイできる J2EE アプリケーションを含む JAR アーカイブ）は、コマンドラインに指定した宛先ディレクトリ（*<destDir>*）にあります。.ear ファイルには、アプリケーション用に生成された EJB、JAR および XML ファイルが含まれています。EAR のマニフェスト・ファイルとして機能する application.xml ファイルは、UNIX の場合は /deploy/META-INF ディレクトリ、Windows の場合は ¥deploy¥META-INF ディレクトリにあります。
- EJB アプリケーションのクラス・ファイルを含むアーカイブ JAR ファイルは、UNIX の場合は /deploy ディレクトリ、Windows の場合は ¥deploy ディレクトリにあります。JAR ファイルには、すべての EJB アプリケーションのクラス・ファイルとデプロイメント・ディスクリプタ・ファイルが含まれています。
- モジュール内で生成される Bean 用の標準 J2EE EJB デプロイメント・ディスクリプタ（ejb-jar.xml）は、UNIX の場合は /classes/META-INF ディレクトリ、Windows の場合は ¥classes¥META-INF ディレクトリにあります。XML デプロイメント・ディスクリプタには、アプリケーション・コンポーネントの記述と、コンテナでのアプリケーションの管理を可能にするための追加情報が含まれています。
- Java アプリケーションで使用できる Java クラス・セットのソース・コードは、UNIX の場合は /src ディレクトリ、Windows の場合は ¥src ディレクトリにあります。生成された JavaBeans と EJB の Java ソース・コードは、Java パッケージ名に従ってサブディレクトリに格納されます。EJB クライアント・スタブも生成されます。
- コンパイル済の生成クラスと、生成コードにより使用される追加の XML リソースは、UNIX の場合は /classes ディレクトリ、Windows の場合は ¥classes ディレクトリにあります。

## 拡張構成

WSDL ドキュメントから生成される EJB をより厳密に制御するために、wsdl2ejb ユーティリティに XML 構成ファイルを提供できます。この構成ファイルを使用すると、WSDL ソースに対する複数のオプションと、生成される EJB に対するオプションを制御できます。

また、複合型を使用する WSDL ドキュメントをサポートできるように、構成ファイルを使用して xml から Java の型への追加のマッピングを提供することもできます。

wsdl2ejb 構成ファイルの構文は、Document Type Definition（DTD）で次のように示されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Specify the properties of the source WSDL document and of the target EJB. -->
<!ELEMENT wsdl2ejb (useProxy?, useWallet?, wsdl, ejb?, mapTypes?)>

<!-- Specify if the generated EJB should use the supplied HTTP proxy when accessing HTTP URLs -->
<!ELEMENT useProxy (#PCDATA)>
<!ATTLIST useProxy
          proxyHost CDATA   #REQUIRED
          proxyPort CDATA   #REQUIRED>

<!-- Specify the location of the wallet credential file used by the generated EJB for opening HTTPS connection
-->
<!ELEMENT useWallet (#PCDATA)>
<!ATTLIST useWallet
          location  CDATA   #REQUIRED>

<!--
    Specify how the wsdl2ejb tools should process the source WSDL document.
     In additional to the mandatory location of the WSDL document, the name of the WSDL service and
     its port can be specified. In this case, an EJB will be generated only for the supplied service and
    port.
     An alternative: the name of a WSDL service binding and the SOAP location to be used can be supplied.
     In the latter case, an EJB using the specified binding and the supplied SOAP location will be used.
     This is particularly useful when generating an EJB from a WSDL stored in a UDDI registry.
     In fact, following a UDDI best practice, the WSDL SOAP location will be managed separately from the
    WSDL document.
 -->
<!ELEMENT wsdl (location, ((service-name, service-port) | (service-binding, soap-location))?)>

<!-- Specify the location of the source WSDL document (for example, "/home/mywsdl.wsdl",
"http://myhost/mywsdl.wsdl") -->
<!ELEMENT location (#PCDATA)>


<!-- Specify the name of the WSDL service to be used for the generation.
     It is the name of one of the services defined in the source WSDL. -->
<!ELEMENT service-name (#PCDATA)>


<!-- Specify the service port of the WSDL service to be used for the generation.
     It is the name of one ports of the service name defined above in the source WSDL. -->
<!ELEMENT service-port (#PCDATA)>


<!-- Specify the name of the WSDL binding to be used for the generation.
     It is the name of one of the bindings defined in the source WSDL. -->
<!ELEMENT service-binding (#PCDATA)>

<!-- Specify the SOAP location service port of the WSDL service to be used for the generation.
     It is the name of one ports of the service name defined above in the source WSDL. -->
<!ELEMENT soap-location (#PCDATA)>

<!-- Specify the properties related to the generated EJB. -->
<!ELEMENT ejb (application-name?, ejb-name?, package-name?, remote-name?, session-type?)>
```

```
<!-- Specify the name of the J2EE application for the generated EAR.  -->
<!ELEMENT application-name (#PCDATA)>

<!-- Specify the JNDI binding key name for the generated EJB.  -->
<!ELEMENT ejb-name (#PCDATA)>

<!-- Specify the name for Java package under which the generated EJB will belong. (for example, com.oracle)
-->
<!ELEMENT package-name (#PCDATA)>

<!-- Specify the class name for the EJB Remote Interface (for example, MyWsdlEjb)  -->
<!ELEMENT remote-name (#PCDATA)>

<!-- Specify the if the generated EJB should be stateless or stateful (for example, Stateless | Stateful)  -->
<!ELEMENT session-type (#PCDATA)>

<!--
    Specify the custom Java types and map them to XML types.
     The JAR attribute value will point to a JAR file containing the definition of the custom
     types or the serializer/deserializer to be used for the custom type.
-->
<!ELEMENT mapTypes (map*)>
<!ATTLIST mapTypes
          jar           CDATA   #IMPLED>

!--
     Specify a new XML to JAR type map.
     EncodingStyle: name of the encodingStyle under which this map will belong
                    (for example, http://schemas.xmlsoap.org/soap/encoding/)
     namespace-uri      : uri of the namespace for the XML type defined in this map
     local-name         : localname of the XML type defined in this map
      java-type          : Java class name to which this type is mapped to (for example, com.org.MyBean)
     java2xml-class-name: Java class name of the type serializer
                          (for example, org.apache.soap.encoding.soapenc.BeanSerializer)
     xml2java-class-name: Java class name of the type deserializer
                          (for example, org.apache.soap.encoding.soapenc.BeanSerializer)
-->
<!ELEMENT map (#PCDATA)>
<!ATTLIST map
          encodingStyle       CDATA   #REQUIRED
          namespace-uri       CDATA   #REQUIRED
          local-name          CDATA   #REQUIRED
          java-type           CDATA   #REQUIRED
          java2xml-class-name CDATA   #REQUIRED
          xml2java-class-name CDATA   #REQUIRED>
```

表 11-2 に、DTD で定義されている wsdl2ejb XML 構成ファイルの要素、副要素および属性を示します。必須の要素と属性は、**太字**で示されています。

**表 11-2 DTD で定義されている wsdl2ejb XML 構成ファイルの要素、副要素および属性**

| 要素 | 副要素 | 属性 | 説明 |
|---|---|---|---|
| useProxy | | | オプションの要素。プロキシ・サーバーの属性を指定します。 |
| | | **proxyHost** | 必須属性。プロキシ・サーバーのホスト名を指定します。 |
| | | **proxyPort** | 必須属性。プロキシ・サーバーのポート番号を指定します。 |
| useWallet | | | オプションの要素。Oracle Wallet の属性を指定します。 |
| | | **location** | 必須属性。EJB で HTTPS 接続のオープンに使用される Oracle Wallet 資格証明ファイルの位置を指定します。 |
| **wsdl** | | | 必須要素。wsdl2ejb ユーティリティでソース WSDL ドキュメントを処理する方法を指定します。location 要素を指定する必要があります。また、オプションで、指定する要素の service-name/service-port ペアまたは service-binding/soap-location ペアを指定できます。 |
| | **location** | | 必須要素。ソース WSDL ドキュメントの位置を指定します。ファイル・パスまたは URL を指定できます。 |
| | service-name | | オプションの要素。生成される EJB に使用する WSDL サービスの名前を指定します。要素のペアとして service-port 要素も指定する必要があります。 |
| | service-port | | オプションの要素。生成される EJB に使用する WSDL サービスのサービス・ポートを指定します。要素のペアとして service-name 要素も指定する必要があります。 |
| | service-binding | | オプションの要素。生成される EJB に使用する WSDL バインディングの名前を指定します。要素のペアとして soap-location 要素も指定する必要があります。 |
| | soap-location | | オプションの要素。生成される EJB に使用する WSDL サービスの SOAP ロケーション・サービス・ポートを指定します。要素のペアとして service-binding 要素も指定する必要があります。 |
| ejb | | | オプションの要素。生成される EJB 関連のプロパティを指定します。 |
| | application-name | | オプションの要素。生成される EAR ファイル用の J2EE アプリケーションの名前を指定します。 |
| | ejb-name | | オプションの要素。生成される EJB 用の JNDI バインディング・キー名を指定します。 |
| | package-name | | オプションの要素。生成される EJB が付属する Java パッケージの名前を指定します。 |
| | remote-name | | オプションの要素。EJB リモート・インタフェースのクラス名を指定します。 |
| | session-type | | オプションの要素。生成される EJB をステートレスにするかステートフルにするかを指定します。 |

**表 11-2 DTD で定義されている wsdl2ejb XML 構成ファイルの要素、副要素および属性（続き）**

| 要素 | 副要素 | 属性 | 説明 |
|---|---|---|---|
| mapTypes | | | オプションの要素。Java のカスタム型を指定して XML の型にマップします。 |
| | map | | オプションの要素。XML から JAR の型へのマップを指定します。 |
| | | **encodingStyle** | 必須属性。このマップが付属するエンコーディング・スタイルの名前を指定します。 |
| | | **namespace-uri** | 必須属性。このマップで定義する XML の型用の名前空間の URI を指定します。 |
| | | **local-name** | 必須属性。このマップで定義する XML の型のローカル名を指定します。 |
| | | **java-type** | 必須属性。この型のマップ先となる Java クラス名を指定します。 |
| | | **java2xml-class-name** | 必須属性。型シリアライザの Java クラス名を指定します。 |
| | | **xml2java-class-name** | 必須属性。型デシリアライザの Java クラス名を指定します。 |

次のコマンドを使用し、構成ファイルを指定して `wsdl2ejb` ユーティリティを実行できます。

```
java -jar wsdl2ejb.jar -conf wsdlconf.xml
```

## サポートされる WSDL ドキュメント

`wsdl2ejb` ユーティリティでは、SOAP バインディングを使用してほとんどの WSDL ドキュメントがサポートされます。このサポートには、リモート・プロシージャ・コール（RPC）とドキュメント・スタイルのドキュメント、エンコードされた型またはリテラルの型が含まれます。表 11-3 に、サポートされる XML Schema の型および対応する Java の型とのデフォルトのマッピングを示します。他に必要な型をサポートするには、前述したカスタムの型マッピングが必要です。

**表 11-3 サポートされる XML Schema の型および対応する Java の型**

| サポートされる XML Schema の型 | 対応する Java の型 |
|---|---|
| string | java.lang.String |
| int | int |
| decimal | BigDecimal |
| float | float |
| double | double |
| Boolean | Boolean |
| long | long |
| short | short |
| byte | byte |
| date | GregorianCalendar |
| timeInstant | java.util.Date |

**注意：** 表 11-3 に示したサポートされる型の配列もサポートされます。

## wsdl2ejb ユーティリティにおける既知の制限

この項では、wsdl2ejb ユーティリティにおける既知の制限について説明します。

- サポートされるのは、W3C 勧告の XML Schema の、名前空間が http://www.w3.org/2001/XMLSchema であるバージョンで定義されている型のみです。
- サポートされるのは、WSDL 1.1 仕様で定義されている一方向のリクエスト / レスポンス送信用プリミティブ型のみです。
- 他の WSDL ドキュメントを含めるために <import> タグを使用する WSDL ドキュメントはサポートされません。
- HTTP、MIME および他のカスタム・バインディングはサポートされません。

## デモの実行

wsdl2ejb デモ・ディレクトリには、wsdl2ejb ユーティリティの使用例が含まれています。すべてのコマンドは、$ORACLE_HOME/webservices/demo/basic/wsdl2ejb ディレクトリから実行することを想定しています。デモでは、一部のサンプル WSDL ドキュメントがソースとして使用され、Web サービス操作の起動に使用できる EJB が生成されます。

各デモは Jakarta ant を使用して実行できます。build.xml ファイルを調べ、初期プロパティ（RMI_HOST、RMI_PORT、RMI_ADMIN、RMI_PWD）が構成に合せて適切に設定されていることを確認してください。build.xml ファイルにより、デモ用 WSDL ドキュメントに対して wsdl2ejb ユーティリティが実行され、生成された EJB がデプロイされて、EJB クライアントが実行されます。

---

**注意：** デモをファイアウォールの内側で実行し、外部 HTTP サイトにアクセスするためにプロキシ情報を設定する必要がある場合は、このプロキシ情報が wsdl2ejb 構成ファイル（rpc_doc_conf.xml、base_conf.xml）に指定されていることを確認してください。

---

---

**注意：** 各デモは WSDL/SOAP 相互運用性テスト・パッケージに基づいています。どちらも、インターネット上で SOAP 相互運用性テスト・ケースとして使用可能な稼働中の SOAP サービスにアクセスします。これらのデモが正常に実行されるかどうかは、これらのサービスを使用できるかどうかによって決まります。

---

デモのディレクトリ構造は次のとおりです。

```
demo/web_services/wsdl2ejb:
   - README.txt                : Readme file
   - build.xml                 : Jakarta ant build file to run all the demos
   - rpc_doc                   : directory for simple RPC and document style operations
       - rpc_doc_conf.xml      : wsdl2ejb configuration file for the rpc_doc demo
       - TestRpcDocClient.java : client for the rpc_doc demo
       - DocAndRpc.wsdl        : sample WSDL for the rpc_doc demo
       - (generated)           : directory where the EJB will be generated
   - base
       - base_conf.xml         : wsdl2ejb configuration file for the base interoperability demo
       - TestInteropBaseClient.java : client for the base interoperability demo
       - InteropTest.wsdl           : WSDL document for the base interoperability demo
       - MySoapStructBean.java      : bean utilized to map the custom type used
                                        in the example defined in the WSDL document
       - MySoapStructBean.jar       : packaged-compiled custom type bean
       - (generated)                : directory where the EJB will be generated
```

## 単純型を使用する RPC スタイルおよびドキュメント・スタイルの例

この例では、Add 操作と Multiply 操作を示す単純な WSDL ドキュメントを使用します。Add はリテラル部を使用するドキュメント・スタイル操作で、Multiply は RPC スタイルでエンコード部を使用する RPC スタイルです。

EJB スタブを生成するには、次のコマンドを使用します。

UNIX の場合

```
cd $ORACLE_HOME/webservices/demo/basic/wsdl2ejb
java -jar ../../../lib/wsdl2ejb.jar -conf rpc_doc/rpc_doc_conf.xml
```

Windows の場合

```
cd %ORACLE_HOME%¥webservices¥demo¥basic¥wsdl2ejb
java -jar ..¥..¥..¥lib¥wsdl2ejb.jar -conf rpc_doc¥rpc_doc_conf.xml
```

ユーティリティにより、Web サービス用のプロキシとして使用可能なステートレス EJB の定義を含む `TestApp.ear` ファイルが生成されます。EAR ファイルは、標準 EJB として OC4J にデプロイできます。EJB のデプロイ方法の詳細は、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』を参照してください。

生成された EJB リモート・インタフェースを調べると、WSDL portType DocAndRpc.wsdl ファイルが Java にどのようにマップされているかがわかります。

WSDL PortType の内容は、次のとおりです。

```
<types>
  <s:schema elementFormDefault="qualified" targetNamespace="http://soapinterop.org">
    <s:element name="Add">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="a" type="s:int" />
          <s:element minOccurs="1" maxOccurs="1" name="b" type="s:int" />
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="AddResponse">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="AddResult" type="s:int" />
        </s:sequence>
      </s:complexType>
    </s:element>
  </s:schema>
</types>
<message name="AddSoapIn">
  <part name="parameters" element="s0:Add" />
</message>
<message name="AddSoapOut">
  <part name="parameters" element="s0:AddResponse" />
</message>
<message name="MultiplySoapIn">
  <part name="a" type="xsd:int" />
  <part name="b" type="xsd:int" />
</message>
<message name="MultiplySoapOut">
  <part name="MultiplyResult" type="s:int" />
</message>
<portType name="TestSoap">
  <operation name="Add">
    <input message="s0:AddSoapIn" />
    <output message="s0:AddSoapOut" />
  </operation>
  <operation name="Multiply">
    <input message="s0:MultiplySoapIn" />
    <output message="s0:MultiplySoapOut" />
  </operation>
</portType>
```

Test.java ファイルからの EJB リモート・インタフェースは、次のとおりです。

```
public org.w3c.dom.Element add(org.w3c.dom.Element parameters)
  throws RemoteException;

public int multiply(int a, int b)
  throws RemoteException;
```

WSDL 操作に RPC スタイルが使用されており、その各部がエンコードされている場合、各部の XML Schema の型は対応する Java 固有の型にマップされます。この例では、xsd:int は Java int にマップされています。リテラル部を使用するドキュメント・スタイルでは、各部は単に org.w3c.dom.Element にマップされます。

TestRpcDocClient.java ファイル内の次のクライアント・コードを使用すると、Web サービスの Add および Multiply 操作を起動できます。このコードは、wsdl2ejb ユーティリティにより生成されたクライアント・コード・スタブを変更して生成されています。

```
import java.io.*;
import java.util.*;
import javax.naming.*;

import org.w3c.dom.*;
import oracle.xml.parser.v2.*;

import org.mssoapinterop.asmx.Test;
import org.mssoapinterop.asmx.TestHome;


/**
 * This is a simple client template. To compile it,
 * please include the generated EJB jar file as well as
 * EJB and JNDI libraries in classpath.
 */
public class TestRpcDocClient
{
  // replace the values
  private static String RMI_HOST  = "localhost";
  private static String RMI_PORT  = "23791";
  private static String RMI_ADMIN = "admin";
  private static String RMI_PWD   = "welcome";

  public TestRpcDocClient () {}

  public static void main(String args[]) {

    TestRpcDocClient client = new TestRpcDocClient();

    try {

      RMI_HOST  = args[0];
      RMI_PORT  = args[1];
      RMI_ADMIN = args[2];
      RMI_PWD   = args[3];

      Hashtable env = new Hashtable();
      env.put(Context.INITIAL_CONTEXT_FACTORY, "com.evermind.server.rmi.RMIInitialContextFactory");
      env.put(Context.SECURITY_PRINCIPAL, RMI_ADMIN);
      env.put(Context.SECURITY_CREDENTIALS, RMI_PWD);
      env.put(Context.PROVIDER_URL, "ormi://" + RMI_HOST + ":" + RMI_PORT + "/Wsdl2EjbTestApp1");
      Context ctx = new InitialContext(env);
      TestHome home = (TestHome) ctx.lookup("mssoapinterop.org/asmx/DocAndRpc.asmx");

      Test service = home.create();

      // call any of the Remote methods that follow to access the EJB

      //
```

```
      // Add test
      //
      Document  doc = new XMLDocument();
      Element elAdd = doc.createElementNS("http://soapinterop.org", "s:Add");
      Element   elA = doc.createElementNS("http://soapinterop.org", "s:a");
      Element   elB = doc.createElementNS("http://soapinterop.org", "s:b");
      elA.appendChild(doc.createTextNode("4"));
      elB.appendChild(doc.createTextNode("3"));
      elAdd.appendChild(elA);
      elAdd.appendChild(elB);
      doc.appendChild(elAdd);

      Element elAddResponse = service.add(elAdd);
      Node tNode = elAddResponse.getFirstChild().getFirstChild();
      System.out.println("AddResponse: "+tNode.getNodeValue());

      //
      // Multiply Test
      //
      int a = 4;
      int b = 3;
      int iMultiplyResponse = service.multiply(a, b);
      System.out.println("MultiplyResponse: "+iMultiplyResponse);


    }
    catch (Throwable ex) {
      ex.printStackTrace();
    }
  }
}
```

クライアントの実行結果は、次のとおりです。

```
AddResponse: 7
MultiplyResponse: 12
```

## 相互運用性サービス・ラウンド 2: 基本テスト・パッケージの例

この例は、SOAP 相互運用性テスト・ラウンド 2 の基本テスト・パッケージで定義されている WSDL ドキュメントのサブセットから始まります。このデモ・サンプルでは、SOAP マッピング・レジストリでの組込みタイプの使用方法と、カスタムの型マッピングを追加する方法を示します。

まず、`InteropTest.wsdl` ファイル内の WSDL portType を調べます。

```
<types>
   <schema xmlns="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://soapinterop.org/xsd">
      <complexType name="ArrayOfstring">
         <complexContent>
            <restriction base="SOAP-ENC:Array">
               <attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="string[]"/>
            </restriction>
         </complexContent>
       </complexType>
       <complexType name="ArrayOfint">
          <complexContent>
            <restriction base="SOAP-ENC:Array">
               <attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="int[]"/>
            </restriction>
          </complexContent>
       </complexType>
       <complexType name="ArrayOffloat">
          <complexContent>
             <restriction base="SOAP-ENC:Array">
```

```
                <attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="float[]"/>
              </restriction>
            </complexContent>
         </complexType>
         <complexType name="ArrayOfSOAPStruct">
            <complexContent>
               <restriction base="SOAP-ENC:Array">
                  <attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="s:SOAPStruct[]"/>
               </restriction>
            </complexContent>
         </complexType>
         <complexType name="SOAPStruct">
               <all>
                  <element name="varString" type="string"/>
                  <element name="varInt" type="int"/>
                  <element name="varFloat" type="float"/>
               </all>
         </complexType>
      </schema>
   </types>

   <message name="echoStringRequest">
      <part name="inputString" type="xsd:string"/>
   </message>
   <message name="echoStringResponse">
      <part name="return" type="xsd:string"/>
   </message>
   <message name="echoStringArrayRequest">
      <part name="inputStringArray" type="s:ArrayOfstring"/>
   </message>
   <message name="echoStringArrayResponse">
      <part name="return" type="s:ArrayOfstring"/>
   </message>
   <message name="echoIntegerRequest">
      <part name="inputInteger" type="xsd:int"/>
   </message>
   <message name="echoIntegerResponse">
      <part name="return" type="xsd:int"/>
   </message>
   <message name="echoIntegerArrayRequest">
      <part name="inputIntegerArray" type="s:ArrayOfint"/>
   </message>
   <message name="echoIntegerArrayResponse">
      <part name="return" type="s:ArrayOfint"/>
   </message>
   <message name="echoFloatRequest">
      <part name="inputFloat" type="xsd:float"/>
   </message>
   <message name="echoFloatResponse">
      <part name="return" type="xsd:float"/>
   </message>
   <message name="echoFloatArrayRequest">
      <part name="inputFloatArray" type="s:ArrayOffloat"/>
   </message>
   <message name="echoFloatArrayResponse">
      <part name="return" type="s:ArrayOffloat"/>
   </message>
   <message name="echoStructRequest">
      <part name="inputStruct" type="s:SOAPStruct"/>
   </message>
   <message name="echoStructResponse">
      <part name="return" type="s:SOAPStruct"/>
   </message>
```

```
<message name="echoStructArrayRequest">
   <part name="inputStructArray" type="s:ArrayOfSOAPStruct"/>
</message>
<message name="echoStructArrayResponse">
   <part name="return" type="s:ArrayOfSOAPStruct"/>
</message>
<message name="echoVoidRequest"/>
<message name="echoVoidResponse"/>
<message name="echoBase64Request">
   <part name="inputBase64" type="xsd:base64Binary"/>
</message>
<message name="echoBase64Response">
   <part name="return" type="xsd:base64Binary"/>
</message>
<message name="echoDateRequest">
   <part name="inputDate" type="xsd:dateTime"/>
</message>
<message name="echoDateResponse">
   <part name="return" type="xsd:dateTime"/>
</message>
<message name="echoDecimalRequest">
   <part name="inputDecimal" type="xsd:decimal"/>
</message>
<message name="echoDecimalResponse">
   <part name="return" type="xsd:decimal"/>
</message>
<message name="echoBooleanRequest">
   <part name="inputBoolean" type="xsd:boolean"/>
</message>
<message name="echoBooleanResponse">
   <part name="return" type="xsd:boolean"/>
</message>

<portType name="InteropTestPortType">
   <operation name="echoString" parameterOrder="inputString">
      <input message="tns:echoStringRequest"/>
      <output message="tns:echoStringResponse"/>
   </operation>
   <operation name="echoStringArray" parameterOrder="inputStringArray">
      <input message="tns:echoStringArrayRequest"/>
      <output message="tns:echoStringArrayResponse"/>
   </operation>
   <operation name="echoInteger" parameterOrder="inputInteger">
      <input message="tns:echoIntegerRequest"/>
      <output message="tns:echoIntegerResponse"/>
   </operation>
   <operation name="echoIntegerArray" parameterOrder="inputIntegerArray">
      <input message="tns:echoIntegerArrayRequest"/>
   <output message="tns:echoIntegerArrayResponse"/>
   </operation>
   <operation name="echoFloat" parameterOrder="inputFloat">
      <input message="tns:echoFloatRequest"/>
   <output message="tns:echoFloatResponse"/>
   </operation>
   <operation name="echoFloatArray" parameterOrder="inputFloatArray">
      <input message="tns:echoFloatArrayRequest"/>
      <output message="tns:echoFloatArrayResponse"/>
   </operation>
   <operation name="echoStruct" parameterOrder="inputStruct">
      <input message="tns:echoStructRequest"/>
      <output message="tns:echoStructResponse"/>
   </operation>
   <operation name="echoStructArray" parameterOrder="inputStructArray">
```

```
      <input message="tns:echoStructArrayRequest"/>
      <output message="tns:echoStructArrayResponse"/>
   </operation>
   <operation name="echoVoid">
      <input message="tns:echoVoidRequest"/>
      <output message="tns:echoVoidResponse"/>
   </operation>
   <operation name="echoBase64" parameterOrder="inputBase64">
      <input message="tns:echoBase64Request"/>
      <output message="tns:echoBase64Response"/>
   </operation>
   <operation name="echoDate" parameterOrder="inputDate">
      <input message="tns:echoDateRequest"/>
      <output message="tns:echoDateResponse"/>
   </operation>
   <operation name="echoDecimal" parameterOrder="inputDecimal">
      <input message="tns:echoDecimalRequest"/>
      <output message="tns:echoDecimalResponse"/>
   </operation>
   <operation name="echoBoolean" parameterOrder="inputBoolean">
      <input message="tns:echoBooleanRequest"/>
      <output message="tns:echoBooleanResponse"/>
   </operation>
</portType>
```

この WSDL ドキュメントには、前のデモより多数の複合型が含まれていることに注意してください。このデモでは、プリミティブ型の配列と構造体のプリミティブ型が使用されています。SOAPStruct 複合型を除き、他の型はすべて SOAP マッピング・レジストリで組込みタイプとしてサポートされます。SOAPStruct 複合型を処理するには、SOAP マッピング・レジストリに新規の複合型定義を追加する必要があります。

SOAPStruct スキーマ定義は、次のとおりです。

```
<complexType name="SOAPStruct">
      <all>
            <element name="varString" type="string"/>
            <element name="varInt" type="int"/>
            <element name="varFloat" type="float"/>
      </all>
</complexType>
```

`MySoapStructBean.java` ファイル内で、この SOAPStruct 複合型を次のように単純な JavaBeans クラスにマップできます。また、BeanSerializer により処理されるマーシャリング・アクションとアンマーシャリング・アクションがあります。

```
public class MySoapStructBean implements java.io.Serializable
{
  private String m_varString = null;
  private int m_varInt = 0;
  private float m_varFloat = 0;

  public MySoapStructBean() {}
  public MySoapStructBean(String s, int i, float f) {
    m_varString = s;
    m_varInt    = i;
    m_varFloat  = f;
  }

  public String getVarString () { return m_varString; }
  public int getVarInt() { return m_varInt; }
  public float getVarFloat() { return m_varFloat; }

  public void setVarString (String s) { m_varString = s; }
  public void setVarInt(int i) { m_varInt = i; }
  public void setVarFloat(float f) { m_varFloat = f; }
```

```
}
```

マッピング用の JavaBeans クラスが用意され、使用するシリアライザとデシリアライザが識別されているため、Java マップへの新規スキーマを追加できるように wsdl2ejb ユーティリティを構成できます。そのためには、wsdl2ejb 構成ファイル base_conf.xml に次のコードを追加します。

```
<mapTypes jar="base/MySoapStructBean.jar" >
  <map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       local-name="SOAPStruct"
       namespace-uri="http://soapinterop.org/xsd"
       java-type="MySoapStructBean"
       java2xml-class-name="org.apache.soap.encoding.soapenc.BeanSerializer"
       xml2java-class-name="org.apache.soap.encoding.soapenc.BeanSerializer" />
</mapTypes>
```

MySoapStructBean.jar ファイルには、MySoapStructBean クラスの定義が含まれています。このマップを使用して、http://soapinterop.org/xsd 名前空間に付属する SOAPStruct 複合型と MySoapStructBean JavaBean クラスの間でマッピングが行われます。SOAP のシリアライザとデシリアライザの詳細は、OracleAS SOAP マニュアルを参照してください。

この追加構成を使用し、次のコマンドを使用して wsdl2ejb ユーティリティを実行できます。

```
UNIX の場合
cd $ORACLE_HOME/webservices/demo/basic/wsdl2ejb
java -jar ../../../lib/wsdl2ejb.jar -conf base/base_conf.xml
Windows の場合
cd %ORACLE_HOME%¥webservices¥demo¥basic¥wsdl2ejb
java -jar ..¥..¥..¥lib¥wsdl2ejb.jar -conf base¥base_conf.xml
```

wsdl2ejb ユーティリティにより、Web サービス用のプロキシとして使用可能なステートレス EJB の定義を含む InteropLabApp.ear ファイルが生成されます。EAR ファイルは、標準 EJB として OC4J にデプロイできます。EJB のデプロイ方法の詳細は、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』を参照してください。

ベース・ディレクトリに保存された TestInteropBaseClient.java クラス・ファイルを使用すると、生成された EJB をデプロイ後にテストできます。クライアントの実行結果は、次のとおりです。

```
echoString: Hello World!
echoStringArray[0]: Hello World!
echoStringArray[1]: Seems to work!
echoStringArray[2]: Fine!
echoStringArray[3]: WOW
echoInteger: 7
echoIntegerArray[0]: 1
echoIntegerArray[1]: 2
echoIntegerArray[2]: 3
echoIntegerArray[3]: 4
echoFloat: 1.7777
echoFloatArray[0]: 1.1
echoFloatArray[1]: 1.2
echoFloatArray[2]: 1.3
echoFloatArray[3]: 1.4
echoStruct: varString=Hello World , varInt=1 , varFloat=1.777
echoStructArray: varString[0]=Hello World , varInt[0]=0 , varFloat=[0]=1.7771
echoStructArray: varString[1]=Hello World 1 , varInt[1]=1 , varFloat=[1]=1.7772
echoStructArray: varString[2]=Hello World 2 , varInt[2]=2 , varFloat=[2]=1.7773
echoStructArray: varString[3]=Hello World 3 , varInt[3]=3 , varFloat=[3]=1.7774
echoVoid.
echoDecimal: 1.77709999999999990194510246510617434978485107421875
echoBoolean: true
echoBase64[0]: 1
echoBase64[1]: 2
echoBase64[2]: 3
echoBase64[3]: 4
```

```
echoDate: Sat Nov 10 12:30:00 EST 2001
```

# Web サービスの動的起動

Java 2 Platform, Enterprise Edition（J2EE）アプリケーションでは、実行時に WSDL ドキュメントを取得する場合、動的起動 API を使用して、WSDL ドキュメントに記述された SOAP 操作が起動されます。動的起動 API では、WebServiceProxy のインスタンス作成に使用可能な WebServiceProxyFactory ファクトリ・クラスが記述されます。作成される各 WebServiceProxy インスタンスは、WSDL ドキュメントの位置（およびオプションで追加の修飾子）に基づいており、どのサービスとポートを使用するかを識別します。WebServiceProxy クラスでは、WSDL サービスで公開されるすべての操作の構文とシグネチャなど、WSDL portType を判別するためのメソッドと、定義された操作を起動するためのメソッドが公開されます。

この項では、動的起動 API とその使用方法について説明します。

Java のサンプルは、Oracle Application Server Web Services に付属するコードを参照してください。このコードは、UNIX の場合は `$ORACLE_HOME/webservices/demo/basic/java_services/dynamicproxy`、Windows の場合は `%ORACLE_HOME%¥webservices¥demo¥basic¥java_services¥dynamicproxy` にあります。EJB のサンプルは、次のディレクトリに用意されているコードを参照してください。
UNIX の場合は `$ORACLE_HOME/webservices/demo/basic/stateless_ejb`、Windows の場合は `%ORACLE_HOME%¥webservices¥demo¥basic¥stateless_ejb` にあります。

## 動的起動 API

動的起動 API には、2 つのパッケージ oracle.j2ee.ws.client および oracle.j2ee.ws.client.wsdl が含まれています。各パッケージには、表 11-4 および表 11-5 に示すように、追加のクラスがインタフェース別、クラス別および例外別に含まれています。

**表 11-4 oracle.j2ee.ws.client パッケージ**

| クラス | 説明 |
|---|---|
| **クラス** | |
| WebServiceProxyFactory | このクラスでは、WSDL ドキュメントが指定された WebServiceProxy クラスが作成されます。 |
| **インタフェース** | |
| WebServiceProxy | このインタフェースは、WSDL ドキュメントに定義されているサービスを表します。 |
| WebServiceMethod | このインタフェースでは Web サービスのメソッドが起動されます。 |
| **例外** | |
| WebServiceProxyException | このクラスでは、WebServiceProxy API に発生する例外が記述されます。 |

**表 11-5 oracle.j2ee.ws.client.wsdl パッケージ**

| クラス | 説明 |
|---|---|
| **インタフェース** | |
| PortType | このインタフェースはポートの形式を表します。 |
| Operation | このインタフェースは WSDL 操作を表します。 |
| Input | このインタフェースは入力メッセージを表し、入力名とメッセージ自体が含まれます。 |
| Output | このインタフェースは出力メッセージを表し、出力名とメッセージ自体が含まれます。 |

**表 11-5 oracle.j2ee.ws.client.wsdl パッケージ（続き）**

| クラス | 説明 |
|---|---|
| Fault | このインタフェースは障害メッセージを表し、障害名とメッセージ自体が含まれます。 |
| Message | このインタフェースでは、操作との通信に使用されるメッセージが記述されます。 |
| Part | このインタフェースはメッセージ部を表し、部分名、elementName および typeName が含まれます。 |
| **クラス** | |
| OperationType | このクラスは操作タイプを表します。操作タイプは request-response、solicit response、one way または notification のいずれかです。 |

この項では、oracle.j2ee.ws.client パッケージの詳細を説明します。

WebServiceProxyFactory クラスには、WSDL ドキュメントの URL または Java 入力ストリームが指定された WebServiceProxy クラスをインスタンス化できるメソッドが含まれています。4 つのメソッドを使用して、提供された WSDL ドキュメント内の最初のサービスとその最初のポートを使用するか、サービス名の 1 つとそのポート名の 1 つを使用して WebServiceProxy インスタンスを作成できます。また、2 つのメソッドを使用して、WSDL 用の UDDI に従って作成された WSDL ドキュメントの WebServiceProxy インスタンスを作成できます。1 つのメソッドを使用すると、WebServiceProxy インスタンスに追加オプションの初期化パラメータを提供できます。

表 11-6 に、WebServiceProxyFactory ファクトリ・クラスのメソッドと、各メソッドの必須パラメータを示します。このファクトリ・クラスとそのメソッドの詳細は、JavaDoc を参照してください。

**表 11-6 WebServiceProxyFactory ファクトリのメソッドとパラメータ**

| メソッド | パラメータ |
|---|---|
| `createWebServiceProxy()` | `java.io.InputStream isWsdl`<br>`java.net.URL baseURL` |
| `createWebServiceProxy()` | `java.net.URL wsdlURL` |
| `createWebServiceProxyFromBinding()` | `java.io.InputStream wsdlis`<br>`java.net.URL baseUrl`<br>`java.lang.String szBindingName`<br>`java.lang.String szSoapLocation` |
| `createWebServiceProxyFromService()` | `java.io.InputStream wsdlis`<br>`java.net.URL baseUrl`<br>`java.lang.String szServiceName`<br>`java.lang.String szServicePort` |
| `createWebServiceProxyFromBinding()` | `java.net.URL wsdlUrl`<br>`java.lang.String szBindingName`<br>`java.lang.String szSoapLocation` |
| `createWebServiceProxyFromService()` | `java.net.URL wsdlUrl`<br>`java.lang.String szServiceName`<br>`java.lang.String szServicePort` |
| `setProperties()` | `java.util.Hashtable ht` |

表 11-7 に、WebServiceProxy インタフェースを示します。WebServiceProxyFactory ファクトリのメソッドは、オプションで追加のパラメータを取ります。これらのパラメータは、WSDL ドキュメント内の操作の動的起動に使用できる WebServiceProxy インタフェースで提供されています。

**表 11-7 WebServiceProxy インタフェースのメソッドとパラメータ**

| メソッド | パラメータ | 説明 |
|---|---|---|
| `getXMLMapping Registry()` | なし | WebServiceProxy で使用される SOAP マッピング・レジストリを戻します。クライアントでは、戻された情報に基づいてこのレジストリを使用し、XML と Java の型マッピングを問い合せたり、マッピング・レジストリを新規のマップ定義で拡張できます。 |
| `getPortType()` | なし | このプロキシで使用される WSDL portType を記述する構造体を戻します。このポートの形式に関連する操作の情報が含まれています。 |
| `getMethod()` | | WebServiceMethod メソッドを戻します。このメソッドを使用すると、Web サービスのメソッドを起動できます。 |
| | szOperationName<br>szInputName<br>szOutputName | 実行される WSDL 操作の名前。<br>実行される操作用の wsdl:input タグの名前。<br>実行される操作用の wsdl:output タグの名前。 |
| `getMethod()` | | WebServiceMethod メソッドを戻します。このメソッドを使用して Web サービスのメソッドを起動し、オーバーロードされない WSDL 操作に使用可能なシグネチャを提供できます。 |
| | szOperationName | 実行される WSDL 操作の名前。 |

表 11-8 に、Web サービスのメソッドの起動に使用する WebServiceMethod インタフェースを示します。

**表 11-8 WebServiceMethod インタフェースのメソッドとパラメータ**

| メソッド | パラメータ | 説明 |
|---|---|---|
| `getInputEncodingStyle()` | なし | 入力メッセージ部に使用されるエンコーディング・スタイルを戻します。ソース WSDL 内で指定されていない場合は NULL が戻されます。 |
| `getOutputEncodingStyle()` | なし | 出力メッセージ部に使用されるエンコーディング・スタイルを戻します。ソース WSDL 内で指定されていない場合は NULL が戻されます。 |
| `invoke()` | | 指定された入力部セットを使用し、サービス操作の 1 つを実行してオブジェクトを戻します。レスポンス・メッセージに入力部が 1 つしかない場合はレスポンス部が戻され、それ以外の場合は出力メッセージ部の配列が戻されます。起動される WSDL 操作に出力部がない場合は、NULL が戻されます。 |
| | inMsgPartNames<br>inMsgPartValues | 入力メッセージで提供する部分の名前。<br>パラメータで名前を指定した部分の対応する値。起動される WSDL 操作に入力部がない場合は、NULL または空の配列パラメータを指定できます。 |

oracle.j2ee.ws.client.wsdl パッケージでは、WSDL サービスで公開されるすべての操作の構文とシグネチャなど、WSDL portType を判別するためのメソッドが公開されます。

## WebServiceProxy クライアント

次のクライアント・コードに、動的起動 API の使用とクライアント実行の出力を示します。このクライアント・コードの内容は、次のとおりです。

- WebServiceProxyFactory 内のプロキシ・パラメータを初期化します。
- WSDL ドキュメントの URL を示すプロキシのインスタンスを作成します。
- WSDL のイントロスペクションを実行します。
- 入力メッセージ部を表示します。
- 指定された入力部セットを使用して Web サービス操作を実行し、結果を戻します。

WSDL ドキュメントの記述内容は、次のとおりです。

```
  <?xml version="1.0" encoding="utf-8" ?>
- <definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://soapinterop.org"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" targetNamespace="http://soapinterop.org"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types />
  - <message name="AddSoapIn">
      <part name="a" type="s:int" />
      <part name="b" type="s:int" />
    </message>
  - <message name="AddSoapOut">
      <part name="AddResult" type="s:int" />
    </message>
  - <portType name="TestSoap">
    - <operation name="Add">
        <input message="tns:AddSoapIn" />
        <output message="tns:AddSoapOut" />
      </operation>
    </portType>
  - <binding name="TestSoap" type="tns:TestSoap">
     <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc" />
    - <operation name="Add">
         <soap:operation soapAction="http://soapinterop.org/Add" style="rpc" />
       - <input>
        <soap:body use="encoded" namespace="http://soapinterop.org"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
         </input>
       - <output>
          <soap:body use="encoded" namespace="http://soapinterop.org"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
        </output>
      </operation>
    </binding>
  - <service name="Test">
    - <port name="TestSoap" binding="tns:TestSoap">
        <soap:address location="http://mssoapinterop.org/asmx/Rpc.asmx" />
      </port>
    </service>
  </definitions>
```

```
package oracle.j2ee.ws.client.impl;

import java.util.*;
import java.io.*;
import java.net.*;
import oracle.j2ee.ws.client.*;
import oracle.j2ee.ws.client.wsdl.*;
import org.apache.soap.util.xml.QName;
import org.apache.soap.util.xml.XMLJavaMappingRegistry;
```

```
public class Client {

  public static void main(String[] args) throws Exception {

    String szWsdlUrl = "http://mssoapinterop.org/asmx/Rpc.asmx?WSDL";

    URL urlWsdl = new URL(szWsdlUrl);
    System.err.println("Wsdl url = " + urlWsdl);

    WebServiceProxyFactory wsfact= new WebServiceProxyFactory();

    //
    // Set some initial parameters
    //
    Hashtable ht = new Hashtable();
    ht.put("http.proxyHost", "www-proxy.us.oracle.com");
    ht.put("http.proxyPort", "80");
    wsfact.setProperties(ht);

    //
    // Create an instance of the proxy
    //
    WebServiceProxy wsp = wsfact.createWebServiceProxy(urlWsdl);

    //
    // Optional: Wsdl Introspection
    //
    PortType pt = wsp.getPortType();
    List opList = pt.getOperations();
    for (int i = 0; i < opList.size(); i++) {

      Operation op = (Operation) opList.get(i);
      String szOpName = op.getName();
      String szInput  = op.getInput().getName();
      String szOutput = op.getOutput().getName();

      System.err.println("operation["+i+"] = [" + szOpName +
                         "," + szInput + "," + szOutput + "]");

      //
      // show input message parts
      //
      Message msgIn = op.getInput().getMessage();
      Map     mapParts = msgIn.getParts();
      Collection colParts   = mapParts.values();
      Iterator itParts = colParts.iterator();

      WebServiceMethod wsm = wsp.getMethod(szOpName);
      String szInEncStyle = wsm.getInputEncodingStyle();
      XMLJavaMappingRegistry xmr = wsp.getXMLMappingRegistry();

      while (itParts.hasNext()) {
        Part part = (Part) itParts.next();
        String szPartName = part.getName();
        QName qname       = part.getTypeName();
        String szJavaType = xmr.queryJavaType(qname,
szInEncStyle).getName();
        System.err.println("part name = " + szPartName +
                           ", type = " + qname +
                           ", java type = " + szJavaType);
      }
    }
```

```
    //
    // invoke operation/method Add(2,10)
    //
    String[] inMsgPartNames = new String[2];
    inMsgPartNames[0] = "a";
    inMsgPartNames[1] = "b";
    Object[] inMsgPartValues = new Object[2];
    inMsgPartValues[0] = new Integer(2);
    inMsgPartValues[1] = new Integer(10);

    WebServiceMethod wsm = wsp.getMethod("Add");
    Object objRet = wsm.invoke(inMsgPartNames,
                               inMsgPartValues);

    System.err.println("Calling  method Add(" +inMsgPartValues[0] + ","
+
                       inMsgPartValues[1] +")" );
    System.err.println("return = " + objRet);
  }
}
```

クライアント実行の出力は、次のとおりです。

```
Wsdl url = http://mssoapinterop.org/asmx/Rpc.asmx?WSDL
operation[0] = [Add,,]
part name = b, type = http://www.w3.org/2001/XMLSchema:int, java type = int
part name = a, type = http://www.w3.org/2001/XMLSchema:int, java type = int
Calling  method Add(2,10)
return = 12
```

## 既知の制限

この項では、動的起動 API における既知の制限について説明します。

- サポートされるのは、W3C 勧告の XML Schema の、名前空間が `http://www.w3.org/2001/XMLSchema` であるバージョンで定義されている WSDL ドキュメントで定義されている起動操作です。
- 他の WSDL ドキュメントを含めるために `<import>` タグを使用する WSDL ドキュメントはサポートされません。
- HTTP、MIME および他のカスタム・バインディングはサポートされません。

# 12

# Web サービスの高度な機能

この章では、Oracle Application Server Web Services の高度な機能について説明します。内容は次のとおりです。

- Web サービスのデバッグ・プロパティ ws.debug の設定
- 型未指定のリクエストの処理オプション
- SOAP ヘッダーのサポート

# Web サービスのデバッグ・プロパティ ws.debug の設定

Oracle Application Server Web Services のデバッグ情報を取得するには、Java プロパティ ws.debug を使用し、その値を true に設定します。ws.debug の値を true に設定するには、Oracle Enterprise Manager を使用して OC4J の起動オプションを指定します。デバッグ出力は、Oracle Application Server Web Services が動作しているアイランドに対応する OC4J インスタンスのログ・ファイルに送信されます。

例 12-1 に、サンプルのデバッグ出力を示します。

**例 12-1 Web サービスのデバッグ出力**

```
WS Debug: initQnameMap('null')
WS Debug: operation name is: helloWorld
WS Debug: QueryString is: invoke=helloWorld&param0=test
WS Debug: Operation Name is: helloWorld
WS Debug: Port Type Local name is: StatelessExamplePortType
WS Debug: Port Type Namespace URI is:
http://oracle.j2ee.ws_example/StatelessExample.wsdl
WS Debug: Operation Local name is: helloWorld
WS Debug: Operation Namespace URI is:
http://oracle.j2ee.ws_example/StatelessExample.wsdl
WS Debug: Operation Get parameter order: null
```

**関連資料：** デバッグ・オプションの設定方法およびデバッグ出力の表示方法は、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』を参照してください。

# 型未指定のリクエストの処理オプション

Oracle Application Server Web Services では、次の場合に RPC スタイル Web サービスに対するリクエストがサポートされます。

- 型指定のリクエスト。SOAP エンコードされたパラメータ付きの受信 RPC リクエストに、各受信パラメータの型情報を指定する型属性が含まれています。例 12-2 に、型指定の RPC リクエストのサンプルを示します。
- 型未指定のリクエスト。SOAP エンコードされたパラメータ付きの受信 RPC リクエストに、各受信パラメータに対する型属性が含まれていない場合があります。例 12-3 に、型未指定の RPC リクエストのサンプルを示します。この型の RPC リクエストでは、.NET クライアントとの相互運用性が向上します。

Oracle Application Server Web Services のクライアント・サイド・アプリケーションおよびツールでは、型未指定のリクエストは生成されませんが、一部の外部ツールまたはアプリケーションで、このようなリクエストが生成される場合があります。型未指定のリクエストのサポートによるパフォーマンスへの影響を考慮して、デフォルトではこのサポートは有効化されていません。

型未指定のパラメータが含まれるリクエストをサポートするには、WebServicesAssembler でオプションの <accept-untyped-request> タグを使用します。このタグは、対応する <message-style> タグの値が RPC に設定されている場合に、<stateful-java-service> タグおよび <stateless-java-service> タグとともにサブタグとして適用されます。<accept-untyped-request> タグは、<stateless-session-ejb-service> タグのサブタグとしても適用されます。

表 12-1 に、<accept-untyped-request> タグの仕様を示します。

**例 12-2 型指定の RPC リクエストのサンプル**

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="
   http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:sayHello xmlns:ns1="urn:Hello"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
       <param0 xsi:type="xsd:string">Scott</param0>
       <param1 xsi:type="xsd:int">27</param1>
    </ns1:sayHello>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**例 12-3 型未指定の RPC リクエストのサンプル**

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="
      http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
     <ns1:sayHello xmlns:ns1="urn:Hello"
       SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
       <param0>Scott</param0>
       <param1>27</param1>
     </ns1:sayHello>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**表 12-1 WebServicesAssembler の <accept-untyped-request> タグ**

| タグ | 説明 |
|---|---|
| <accept-untyped-request><br>*value*<br></accept-untyped-request> | *value* に true を設定すると、Web サービスで型未指定のリクエストが受入可能であることを WebServicesAssembler に通知します。value を false に設定すると、Web サービスは型未指定のリクエストを受け入れません。<br>有効な値：true、false（大 / 小文字は区別されません。TRUE および FALSE も有効です）<br>デフォルト値：false |

# SOAP ヘッダーのサポート

この項では、Web サービスのクライアントからエンド・ポイントに送信される SOAP リクエスト・ヘッダーに関する Oracle Application Server Web Services サポートについて説明します。この項の内容は、次のとおりです。



## クライアント・サイドの SOAP リクエスト・ヘッダーのサポート

Oracle Application Server Web Services 生成のクライアント・サイド・プロキシ・コードには、SOAP リクエスト・ヘッダーを使用するためのメソッドが含まれています。SOAP リクエスト・ヘッダーを含む SOAP リクエスト・メッセージは、Web サービスのプロキシ・コードが起動されたときに、サービスのエンド・ポイントに送信されます。

Oracle Application Server Web Services で、Web サービスのドキュメント・スタイルまたは RPC スタイルのサービス用の WSDL からプロキシが生成される場合は、プロキシ・コードによって、次の 2 つの SOAP リクエスト・ヘッダー・サポート・メソッドが提供されます。

```
void _setSOAPRequestHeaders(org.apache.soap.Header headers)
org.apache.soap.Header _getSOAPRequestHeaders()
```

これらのメソッドによって、`org.apache.soap.Header` オブジェクトに対するアクセスが提供されます。デフォルトでは、org.apache.soap.Header オブジェクトの値は `null` に設定されます。これは、SOAP リクエスト・メッセージにヘッダーが存在しないことを示します。リクエスト・ヘッダーが必要な場合は、`_setSOAPRequestHeaders()` メソッドを使用して、SOAP リクエスト・メッセージとともに送信される `Header` オブジェクトを指定します。

---

**注意：** プロキシが、ストアド・プロシージャまたは JMS ドキュメント・スタイルの Web サービスに対して生成される場合は、`_setSOAPRequestHeaders()` メソッドおよび `_getSOAPRequestHeaders()` メソッドは提供されません。

---

SOAP リクエスト・ヘッダーの情報は、すべてのプロキシ操作で共有されます。`_setSOAPRequestHeaders()` を使用してヘッダーが設定された後は、そのプロキシを使用する後続のすべての操作起動で同じヘッダー値が使用されます。新しいヘッダー値を設定するには、新しい `Header` オブジェクトを使用するか、または `null` 値を指定して `_setSOAPRequestHeaders()` をコールします。

---

**注意：** SOAP リクエスト・ヘッダーが設定された後は、`_setSOAPRequestHeaders()` を使用してオブジェクトがリセットされるまで、後続の各操作起動に対して同じヘッダー・オブジェクトが使用されます。

---

SOAP リクエスト・ヘッダーを作成および操作するには、ヘッダー・オブジェクトを移入する必要があります。`org.apache.soap.Header` オブジェクトには、1 つ以上の SOAP ヘッダー・ブロックの内容を指定するためのメソッドが用意されています。次のように定義されます。

```
public void setHeaderEntries(java.util.Vector headerEntries)
```

ベクターには、個々の SOAP ヘッダー・ブロックを指定する `org.w3c.dom.Element` オブジェクトが移入されます。

ヘッダー・エントリに、値が 1 に設定された `mustUnderstand` 属性が含まれている場合は、受信者がヘッダー・エントリを処理する必要があります。受信者がヘッダー・エントリを処理できない場合は、値 `FAULT_CODE_MUST_UNDERSTAND` の SOAP 障害が戻されます。

**関連項目：** SOAP 1.1 のヘッダー・エントリの詳細は、http://www.w3.org/TR/SOAP/ の第 4.2 項「SOAP Header」を参照してください。

### クライアント・サイド・プロキシの SOAP ヘッダーの設定

この項では、プロキシ・クラス EmployeeProxy を使用するサンプルを示します。このコードが含まれる完全なサンプルは、ディレクトリ $ORACLE_HOME/web_services/demo/header_demo/client にあります。このサンプルでは、単一のヘッダー・ブロックが Header オブジェクトに追加されます。次に、Header オブジェクトは、プロキシの _setSOAPRequestHeaders() メソッドに対する引数として提供されます。

**例 12-4 SOAP リクエスト・ヘッダー付きのメッセージを使用したクライアントの一部**

```
.
.
.
// Create an intance of the proxy
EmployeeProxy proxy = new EmployeeProxy();
// Create a Header objecy
Vector v = new Vector();
v.add (e);
Header hdr = new Header();
hdr.setHeaderEntries(v);

// Set the Header
proxy._setSOAPRequestHeaders(hdr);
// Invoke the request
System.out.println("Salary of MILLER is: " + proxy.getEmployeeSalary("MILLER"));
```

## サーバー・サイドの SOAP リクエスト・ヘッダーのサポート

サーバー・サイドで SOAP リクエスト・ヘッダーを処理するには、Web サービスで oracle.j2ee.ws.HeaderCallback インタフェースを実装する必要があります。このインタフェースは、Oracle Application Server Web Services が提供する wsserver.jar の一部です。このインタフェースには、1 つの引数 org.apache.soap.Header を取る 1 つのメソッドが含まれています。

Oracle Application Server Web Services Infrastructure によって、関連サービスのメソッドがコールされる前に processHeaders() メソッドがコールされます。

受信 SOAP リクエスト・ヘッダーに、mustUnderstand 属性が 1、true または TRUE のいずれかの値に設定された 1 つ以上のヘッダー・エントリが含まれている場合は、Web サービスの実装で、oracle.j2ee.ws.HeaderCallback インタフェースを実装する必要があります。このインタフェースが実装されない場合、Oracle Application Server Web Services では、障害コード FAULT_CODE_MUST_UNDERSTAND で SOAP 障害がスローされます。

Web サービス実装で HeaderCallback インタフェースを実装している場合に、mustUnderstand 属性が 1、true または TRUE に設定されたヘッダー・エントリの処理方法をサービスが認識しないと、実装によって、障害コード FAULT_CODE_MUST_UNDERSTAND で SOAP 例外がスローされる場合があります。その後、Oracle Application Server Web Services によって例外が変換され、障害コード FAULT_CODE_MUST_UNDERSTAND で SOAP 障害がスローされます。

この項では、Employee サービス用の実装を提供するサーバー・サイドの Web サービスのコードを示します。この Web サービスが含まれる完全なサンプルは、ディレクトリ $ORACLE_HOME/web_services/demo/basic/header_demo/client（$ORACLE_HOME/webservices/demo/demo.zip を解凍するとこのディレクトリに格納されます）にあります。

例 12-5 に、HeaderCallback を拡張するインタフェースを示します。

例 12-6 に、サンプルの getEmployeeSalery インタフェース用のサービス実装の一部を示します。このインタフェースには、次の形式の受信 SOAP リクエスト・ヘッダーを処理できる processHeaders() メソッドが含まれています。

```
<SOAP-ENV:Header>
  <credentials>
    <username>scott</username>
    <password>tiger</password>
   <datasource>jdbc/OracleCoreDS</datasource>
  </credentials>
</SOAP-ENV:Header>
```

**例 12-5 HeaderCallback を拡張する Employee インタフェース**

```
import oracle.j2ee.ws.HeaderCallback;
/**
 *  Employee java class being exposed as Web Services
 *  This service also extends HeaderCallback so as to
 *  access Headers.
 */
public interface Employee
  extends  HeaderCallback
{
  // Get the salary for a given Employee
  int getEmployeeSalary(String  ename);
}
```

**例 12-6 HeaderCallback の processHeaders() を含む実装**

```
  public void processHeaders(Header header)
     throws java.io.IOException,
            oracle.xml.parser.v2.XSLException
  {
    // Get all the Elements
    Vector entries = header.getHeaderEntries();
    Element e = (Element) entries.firstElement();
    System.out.println("Element received is: " );
    ((XMLElement)e).print(System.out);

    // Get independent nodes and retrieve node values.
    Node userNode;
    userNode = ((XMLNode)e).selectSingleNode("username");
    userName = ((XMLElement)userNode).getText();

    Node passwordNode;
    passwordNode = ((XMLNode)e).selectSingleNode("password");
    password = ((XMLElement)passwordNode).getText();

    Node dsNode;
    dsNode = ((XMLNode)e).selectSingleNode("datasource");
    datasourceName = ((XMLElement)dsNode).getText();

    System.out.println("User name is: " + userName);
    System.out.println("Password is: " + password);
    System.out.println("Datasource is: " + datasourceName);
  }
```

## SOAP ヘッダー・サポートの制限

次に、SOAP ヘッダーのサポートに関する制限事項を示します。

1. Oracle Application Server Web Services では、WSDL 定義で指定されたヘッダー情報を処理または変換するためのサポートは用意されていません。
2. Oracle Application Server Web Services では、`org.apache.soap.Header` オブジェクトで提供される SOAP リクエスト・ヘッダー情報については、XML の妥当性チェック、またはこれにかわる方法を提供しません。このオブジェクトには、整形式の XML をユーザーが移入する必要があります。
3. Oracle Application Server Web Services では、SOAP レスポンス・ヘッダーはサポートされません。
4. プロキシが JMS ドキュメント・スタイルの Web サービスに対して生成される場合、SOAP リクエスト・ヘッダーの `_setSOAPRequestHeaders()` および `_getSOAPRequestHeaders()` メソッドは提供されません。JMS の Web サービスを使用した場合、SOAP リクエスト・ヘッダーを処理するためのサーバー側の機能はありません。
5. プロキシがストアド・プロシージャ Web サービスに対して生成される場合、SOAP リクエスト・ヘッダーの `_setSOAPRequestHeaders()` および `_getSOAPRequestHeaders()` メソッドは提供されません。ストアド・プロシージャ Web サービスを使用した場合、SOAP リクエスト・ヘッダーを処理するためのサーバー側の機能はありません。

# A

# Oracle Application Server SOAP の使用

この付録の内容は、次のとおりです。

# Oracle Application Server SOAP の概要

Oracle Application Server には、Web サービス用に一意のサーブレット・インタフェースと J2EE デプロイを使用する前述の Oracle Application Server Web Services に加えて、Apache 2.3.1 SOAP に由来する、多数の拡張機能を含む Oracle Application Server SOAP（OracleAS SOAP）が用意されています。

SOAP メッセージ・プロセッサ（OracleAS SOAP）には、次の機能があります。

- SOAP プロトコルの処理 - 相互運用可能な SOAP 仕様の実装を提供します。これには、Cookie とセッションのサポートが含まれます。この機能は、ステートフル Web サービスのリクエスト / レスポンスに関するステータス情報の受渡しに特に役立ちます。
- 添付ファイル（XML ペイロード）付きの SOAP リクエストのサポート。
- 解析 - OracleAS SOAP プロセッサにより Oracle XML Parser が統合されます。RPC スタイルのリクエストの場合、OracleAS SOAP プロセッサでは受信 XML 文書を効率的に解析し、リクエストの書式が適切であることを確認し、リクエストを検証できます。同様に、Java レスポンスを SOAP メッセージにエンコードまたはシリアライズできます。
- カスタマイズされた Web サービス・サーブレットを使用した Web サービスの起動 - SOAP プロセッサでは、メッセージの内容がアンマーシャリングされ、サーブレットに応じて Web サービスの実装がコールされます。Web サービスは、Java クラス、EJB または PL/SQL ストアド・プロシージャとして実装できます。
- セキュリティ・マネージャを使用した送信者の認証 - Web サービス実装を起動する前に、OracleAS SOAP プロセッサ（サーブレット）では標準 JAAS ベースのユーザー・マネージャ・プラグインを使用してユーザーが認証されます。また、OracleAS SOAP プロセッサでは、Oracle の Single Sign-On Server とサード・パーティの認証サービスもサポートされ、Web サービス用のシングル・サインオン機能を提供します。
- 例外処理 - 処理中に例外が発生すると、Java 例外が SOAP 障害に変換され、サービス・クライアントに配信されます。

# Apache SOAP ドキュメント

OracleAS SOAP は、Apache SOAP 2.3.1 の修正バージョンです。Apache SOAP 2.3.1 を対象とするドキュメントのほとんどは、OracleAS SOAP にも適用されます。Apache SOAP 2.3.1 のドキュメントは、次のサイトで入手できます。

```
http://xml.apache.org/soap/docs/
```

# SOAP リクエスト・ハンドラ・サーブレットの構成

OracleAS SOAP リクエスト・ハンドラでは、XML 構成ファイルを使用して必須のサーブレット・パラメータを設定します。デフォルトでは、このファイルは `soap.xml` という名前で、UNIX の場合はディレクトリ `$SOAP_HOME/lib`、Windows の場合は `%SOAP_HOME%¥lib` の `soap.ear` ファイルに格納されています。

このファイルの XML 名前空間は、次のとおりです。

```
http://xmlns.oracle.com/soap/2001/04/config
```

SOAP インストール用に別の構成ファイルを使用するには、`soap.ear` ファイルを展開します。UNIX の場合はディレクトリ `webapps/soap/WEB-INF` で、Windows の場合はディレクトリ `webapps¥soap¥WEB-INF` で、`soap.properties` ファイルの `SoapConfig` パラメータに指定されているパス名を変更します。次に、更新した `soap.ear` ファイルを再デプロイします。

たとえば、構成ファイルをデフォルトの `soap.xml` から `newConfig.xml` に変更するには、`soap.properties` 内で `soapConfig` に設定されている値を変更します。

```
servlet.soaprouter.initArgs=soapConfig=soap_home/soap/webapps/soap/WEB-INF/newConfig.xml
```

*soap_home* は、システムの SOAP インストール・ディレクトリへのフルパスです。

pathAuth ブール属性を true に設定すると、クライアントがデプロイ済のサービスにメッセージを転送するには、一意のサービス URL を指定する必要があります。サービス URL は、SOAP サーブレット URL の末尾にサービス URI を追加したものです。この属性のデフォルト値（未指定の場合）は、false です。

表 A-1 に、SOAP リクエスト・ハンドラ・サーブレットの XML 構成ファイルの要素を示します。

**表 A-1 SOAP リクエスト・ハンドラ・サーブレットの構成ファイルのパラメータ**

| パラメータ | 説明 |
|---|---|
| errorHandlers | エラー・ハンドラ・チェーン用のハンドラ・リストを指定します。 |
| faultListeners | これは、faultListener 要素のリストを定義するオプションの要素です。faultListener 要素では、障害発生時に起動するクラスを指定します。ユーザーに戻される SOAP 障害にスタック・トレースを追加するには、faultListener として org.apache.soap.server.DOMFaultListener を指定します。 |
| handler | これは、handler 要素のリストを定義するオプションの要素です。handler 要素では、リクエスト、レスポンス、エラーという 3 つのコンテキストのいずれかで、各 SOAP リクエスト時に起動するように構成できるグローバル・ハンドラを定義します。任意の数のハンドラを定義できます。ハンドラの name 属性では、ハンドラ名を指定します。各ハンドラには一意名が必要です。ハンドラの class 属性では、ハンドラを実装する Java クラスを指定します。このクラスでは、インタフェース oracle.soap.server.Handler を実装する必要があります。各ハンドラには、任意の数のオプションを名前 / 値ペアとして指定できます。コンテキストは、要素 requestHandlers、responseHandlers および errorHandlers 内で構成されます。これらの各要素では、ハンドラ名の順序付きリスト、つまりハンドラ・チェーンを定義します。<br>SOAP では、一意に識別されたハンドラごとに 1 つのインスタンスが作成されることに注意してください。どのチェーンに含まれる特定のハンドラ名も、すべてハンドラの同一インスタンスを参照します。SOAP サーブレットが破棄されると、ハンドラも破棄されます。 |
| logger | エラー・メッセージと情報メッセージは、logger 要素に定義されたクラスを使用してロギングされます。logger クラスでは、oracle.soap.server.Logger を拡張する必要があります。<br>OracleAS SOAP には、サーブレットのログ・メソッドを収集するクラス oracle.soap.server.impl.ServletLogger が含まれているため、SOAP メッセージはサーブレットのログ・ファイルにロギングされます。ServletLogger は、デフォルトのログ出力です。デフォルトのログ出力の場合は、severity オプションを値 status、error、debug のいずれかに設定できます。<br>error を指定すると、status および error メッセージの両方を取得します。同様に、debug を指定すると、3 種類のメッセージすべてを取得します。<br>OracleAS SOAP には、2 つのログ出力実装が含まれています。ログをサーブレット・ログに記録するには oracle.soap.server.impl.ServletLogger、ログを標準出力に記録するには oracle.soap.server.impl.StdOutLogger を使用します。<br>oracle.soap.server.Logger インタフェースを実装すると、独自のログ出力を実装できます。 |
| providerManager | providerManager は、構成マネージャを定義できるオプションの要素です。この要素では、サーバーからプロバイダのデプロイ情報へのアクセス方法を定義します。<br>providerManager の class 属性では、oracle.soap.server.ProviderManager を実装する Java クラスを指定します。デフォルトの構成マネージャ oracle.soap.server.impl.XMLProviderConfigManager では、デプロイされたプロバイダが XML フォーマットのファイルに保持されます。ここでは filename オプションを指定できます。filename はレジストリ・ファイル名へのパスであり、単純ファイル名、相対パスまたはフルパスで指定できます。フルパスを指定しない場合、そのパスはファイル名とサーブレット・コンテキストから判断されます。デフォルトのファイル名は WEB-INF/providers.xml です。<br>代替のプロバイダ構成マネージャ oracle.soap.server.impl.BinaryProviderConfigManager では、デプロイされたプロバイダはシリアライズ・オブジェクトとしてファイルに保持されます。デフォルトのファイル名は WEB-INF/providers.dd です。<br>別の構成マネージャを指定するには、configManager 要素に class 属性を追加します。次に例を示します。<br><osc:configManager class="*fully.qualified.classname*"> |

**表 A-1 SOAP リクエスト・ハンドラ・サーブレットの構成ファイルのパラメータ（続き）**

| パラメータ | 説明 |
|---|---|
| requestHandlers | リクエスト・ハンドラ・チェーン用のハンドラ・リストを指定します。 |
| responseHandlers | レスポンス・ハンドラ・チェーン用のハンドラ・リストを指定します。 |
| serviceManager | serviceManager はオプションの要素であり、構成マネージャを定義して ServiceManager オプションを設定できます。この要素では、サーバーからサービスのデプロイ情報へのアクセス方法を定義します。serviceManager の class 属性では、oracle.soap.server.ServiceManager を実装する Java クラスを指定します。<br><br>OracleAS SOAP 構成マネージャのデフォルト・クラスは、サービスのデプロイ情報が XML ファイルに格納される oracle.soap.server.impl.XMLServiceConfigManager です。XMLServiceConfigManager で filename オプションを指定して、ファイル名を指定します。filename はレジストリ・ファイル名へのパスであり、単純ファイル名、相対パスまたはフルパスで指定できます。フルパスを指定しない場合、そのパスはファイル名とサーブレット・コンテキストから判断されます。デフォルトのファイル名は、WEB-INF/services.xml です。<br><br>別の構成マネージャを指定するには、configManager 要素に class 属性を追加します。<br><br>次に例を示します。<br><br><osc:configManager class="*fully.qualified.classname*"><br><br>代替のサービス構成マネージャ oracle.soap.server.impl.BinaryServiceConfigManager では、デプロイされたサービスはシリアライズ・オブジェクトとしてファイルに保持されます。デフォルトのファイル名は、WEB-INF/services.dd です。<br><br>サービス・マネージャでは、プロバイダ・マネージャとサービス・マネージャを SOAP サービスとして自動的にデプロイできます。これらのマネージャをサービスとして公開可能にするには、autoDeploy オプションを true に設定します。autoDeploy のデフォルト値は false です。 |

# OracleAS SOAP の管理ユーティリティとスクリプトの使用

OracleAS SOAP の管理ユーティリティを使用するには、提供されるクライアント・サイド・スクリプトの 1 つを使用して SOAP 管理ユーティリティを実行するための環境を設定する必要があります。clientenv スクリプトでは CLASSPATH が設定され、パスに $SOAP_HOME/bin ディレクトリが追加されます。

UNIX 上でクライアント環境を設定するには、次のコマンドを使用します。

```
cd $SOAP_HOME/bin
source clientenv.csh
```

Windows の場合は、次のコマンドを使用します。

```
cd %SOAP_HOME%¥bin
clientenv.bat
```

clientenv スクリプトでは、他のスクリプトとサンプルで使用される環境変数が設定されます。これらの環境変数は、手動で設定するとオーバーライドできます。変数 SOAP_URL は SOAP サーバーの URL で、JAXP は Oracle XML Parser に DocumentBuilderFactory を使用するように設定されます。

## プロバイダの管理

providerMgr スクリプトでは、プロバイダを管理する SOAP クライアントが実行されます。このスクリプトは、使用方法に関する情報のパラメータを指定せずに実行してください。

UNIX の場合は、次のコマンドを使用します。

```
providerrMgr.sh options
```

Windows の場合は、次のコマンドを使用します。

```
providerMgr.bat options
```

providerMgr の *options* は、次のとおりです。

deploy *ProviderDescriptorFile*

この場合は、*ProviderDescriptorFile* に記述されているプロバイダがデプロイされ、使用可能になります。

undeploy *ProviderID*

この場合は、指定した *ProviderID* を持つプロバイダが削除されます。*ProviderID* は、プロバイダのディスクリプタ・ファイルに指定されている ID 属性です。

Java プロバイダは、id=java-provider としてインストールするとデプロイされますが、独自に作成したプロバイダは明示的にデプロイする必要があります。たとえば、UNIX の場合、プロバイダをデプロイするときに、そのプロバイダのデプロイメント・ディスクリプタ provider.xml を使用するには、次のコマンドを使用します。

```
providerMgr.sh deploy provider.xml
```

## サービス・マネージャを使用した Java サービスのデプロイとアンデプロイ

ServiceMgr は、SOAP サービスのデプロイとアンデプロイに使用する管理ユーティリティです。サービスをデプロイするには、最初に SOAP 環境を設定してから deploy コマンドを使用します。UNIX の場合は、次のコマンドを使用します。

```
source clientenv.csh
ServiceMgr.sh deploy ServiceDescriptorFile
```

Windows の場合は、次のコマンドを使用します。

```
clientenv.bat
ServiceManager.bat deploy ServiceDescriptorFile
```

deploy オプションを使用すると、ServiceDescriptorFile に指定したサービスが使用可能になります。

サービスをアンデプロイする準備が完了してから、登録済サービス名を引数として指定し、undeploy コマンドを使用します。UNIX の場合は、次のコマンドを使用します。

```
ServiceManager.sh undeploy ServiceID
```

Windows の場合は、次のコマンドを使用します。

```
ServiceManager.bat undeploy ServiceID
```

これにより、指定した ID を持つサービスが使用不能になります。*ServiceID* は、サービス・ディスクリプタ・ファイルに指定されているサービス ID 属性です。

ServiceMgr では、SOAP サービスのリストと問合せがサポートされます。使用可能なサービスをリストするには、最初に SOAP 環境を設定してから list コマンドを使用します。UNIX の場合は、次のコマンドを使用します。

```
source clientenv.csh
ServiceMgr.sh list
```

Windows の場合は、次のコマンドを使用します。

```
clientenv.bat
ServiceMgr.bat list
```

サービスを問い合せて、サービス・デプロイメント・ディスクリプタ・ファイルに設定されているディスクリプタ・パラメータを取得するには、query コマンドを使用します。UNIX の場合は、次のコマンドを使用します。

```
ServiceMgr.sh query ServiceID
```

Windows の場合は、次のコマンドを使用します。

```
ServiceMgr.bat query ServiceID
```

ServiceID は、サービス・ディスクリプタ・ファイルに設定されているサービス ID 属性です。

## WSDL ドキュメントからのクライアント・プロキシの生成

wsdl2java スクリプトは、入力として WSDL ドキュメントを使用し、サービスのコールに使用できる Java クラスを戻します。Java クラスには、WSDL ドキュメントに記述されているのと同じ名前を持つメソッドが含まれています。生成されたコードにより、Apache クライアント・サイド・ライブラリがコールされます。

UNIX の場合は、次のコマンドを使用します。

```
wsdl2java.sh options
```

Windows の場合は、次のコマンドを使用します。

```
wsdl2java.bat options
```

wsdl2java の *options* は、次のとおりです。

wsdl2java.sh *WsdlDocumentURL OutputDir* [-k *PackageName*] [-s *ServiceName*] [-p *PortName*]

各項目の意味は、次のとおりです。

*WsdlDocumentURL* は、WSDL ドキュメントの URL です。

*OutputDir* は、生成されるプロキシ Java コードの出力ディレクトリです。

-k *PackageName* は、生成されるプロキシ Java コードのパッケージ名です。

-s *ServiceName* は、プロキシが生成されるサービスの名前です。

-p *PortName* は、サービスのポート名です。サービスのうち指定したポートのプロキシが生成されます。

出力ディレクトリの構造は、次のとおりです。

*output root dir/service name/port name/package name/java proxy source code*

デフォルトでは、*PackageName* は WSDL サービス名と同じになります。

-s および -p の両方のオプションを指定しないと、すべてのサービスのすべてのポートのプロキシが生成されます。-p オプションを指定しないと、指定したサービスのすべてのポートのプロキシが生成されます。

## Java サービス実装からの WSDL ドキュメントの生成

java2wsdl スクリプトは、入力として Java クラスを使用し、クラスを RPC サービスとして記述する WSDL ドキュメントを出力します。Java クラスが Web サービスとして使用される場合は、そのサービスのコールを必要とする開発者に、関連する WSDL ドキュメントを送信できます。

UNIX の場合は、次のコマンドを使用します。

```
java2wsdl.sh options
```

Windows の場合は、次のコマンドを使用します。

```
java2wsdl.bat options
```

wsdl2java の *options* は、次のとおりです。

```
java2wsdl.sh ClassName OutputFile SoapURL ClassURL1 ClassURL2 ...
```

各項目の意味は、次のとおりです。

*ClassName* は、Web サービスとなる Java の .class ファイルのフルパス名です。

*OutputFile* は、出力の WSDL ドキュメント名です。

*SoapURL* は、SOAP のエンド・ポイントです。

*ClassURL* リストは、参照先クラスを検索するためのクラス・パスとして機能します。

# OracleAS SOAP サービスのデプロイ

この項では、OracleAS SOAP サービスのデプロイとアンデプロイについて説明します。この項の内容は、次のとおりです。



## デプロイメント・ディスクリプタの作成

デプロイメント・ディスクリプタには、サービス・デプロイメント・ディスクリプタとプロバイダ・デプロイメント・ディスクリプタがあります。プロバイダ・デプロイメント・ディスクリプタ・ファイルは、SOAP サーブレットに対してプロバイダの構成情報を記述する XML ファイルです。サービス・デプロイメント・ディスクリプタ・ファイルは、SOAP サーブレットに対してサービスの構成情報を記述する XML ファイルです。

Java で記述されたサービスの場合、必須となるのはサービス・ディスクリプタのみです。すべての Java サービス・ディスクリプタでは、OracleAS SOAP のインストールで提供されるものと同じ Java プロバイダ・ディスクリプタを指すことができます。

PL/SQL ストアド・プロシージャで記述された各サービスには、データベース・ユーザーごとにサービス・ディスクリプタとプロバイダ・ディスクリプタが 1 つずつ必要です。そのメリットは、パスワードまたはユーザーに変更があった場合にも、更新する必要があるのはすべてではなく 1 つのサービス・ディスクリプタのみである点です。

詳細は、「ストアド・プロシージャ」を参照してください。

EJB として記述されたサービスには、EJB コンテナ・ユーザーごとにサービス・ディスクリプタとプロバイダ・ディスクリプタが 1 つずつ必要です。

詳細は、「EJB」を参照してください。

---

**注意：** 独自のプロバイダの記述を必要とする開発者向けに、Apache スタイルのプロバイダ・インタフェースとディスクリプタもサポートされます。Apache ディスクリプタの場合は、1 つのファイルにサービス・プロパティとプロバイダ・プロパティの両方が含まれているため、すべてのサービスに共通のプロバイダ情報を複製する必要があります。

---

サービス・デプロイメント・ディスクリプタ・ファイルでは、次の情報を定義します。

- サービス ID
- サービス・プロバイダ・タイプ（Java など）
- 使用可能なメソッド

ディスクリプタを記述する最善の方法は、まずサンプル・ディレクトリの 1 つから既存のディスクリプタをコピーすることです。

例 A-1 に、Java の `SimpleClock` サービスのディスクリプタ・ファイル `SimpleClockDescriptor.xml` を示します。このディスクリプタ・ファイルは、`samples/simpleclock` ディレクトリにあります。サービス・ディスクリプタ・ファイルは、サービス・ディスクリプタ・スキーマに準拠する必要があります（スキーマ `service.xsd` は、UNIX の場合はディレクトリ `$SOAP_HOME/schemas`、Windows の場合は `%SOAP_HOME%¥schemas` にあります）。

サービス・ディスクリプタ・ファイルでは、`methods` 属性を使用する `isd:provider` 要素内のサービスに関連したメソッドを識別します。`isd:java class` 要素では、SOAP サービスを実装する Java クラスを識別し、そのクラスが静的かどうかを指定します。

**例 A-1 サンプルの単純なクロック・サービス用の Java サービス・ディスクリプタ・ファイル**

```
<isd:service xmlns:isd="http://xmlns.oracle.com/soap/2001/04/deploy/service"
             id="urn:jurassic-clock"
             type="rpc" >
  <isd:provider
      id="java-provider"
      methods="getDate"
      scope="Application" >
      <isd:java class="samples.simpleclock.SimpleClockService"/>
  </isd:provider>
  <!-- includes stack trace in fault -->
  <isd:faultListener class="org.apache.soap.server.DOMFaultListener"/>
</isd:service>
```

**注意：** サービス・ディスクリプタ・ファイルでは、サービス・メソッドのメソッド・シグネチャは定義されません。SOAP では、リフレクションを使用してメソッド・シグネチャが決定されます。

## OC4J への SOAP Web サービスのインストール

Oracle Application Server Containers for J2EE（OC4J）に OracleAS SOAP Web サービスをインストールする手順は、次のとおりです。

1. サービスとプロバイダのデプロイメント・ディスクリプタを作成します。
2. `soap.ear` ファイルを展開します。このファイルは、UNIX の場合は `$SOAP_HOME/lib`、Windows の場合は `%SOAP_HOME¥lib` にあります。
3. サービスを実装する Java クラスおよび Jar を、展開した `soap.ear` のディレクトリの適切な位置にコピーします。

   Java の .class ファイルを `WEB-INF/classes` にコピーします。

   Java の .jar ファイルを `WEB-INF/libs` にコピーします。
4. 更新した `soap.ear` ファイルを再デプロイします。
5. 次のコマンドを実行してプロバイダ・ディスクリプタをデプロイします。

   ```
   providerMgr.sh deploy FileName
   ```

   *FileName* は、プロバイダ・ディスクリプタの xml ファイル名です。
6. 次のコマンドを実行してサービスをデプロイします。

   ```
   serviceMgr.sh deploy FileName
   ```

   *FileName* は、サービス・ディスクリプタの xml ファイル名です。

## インストール済 SOAP Web サービスの無効化

インストール済のサービスを無効化するには、次のコマンドを実行します。

```
serviceMgr.sh undeploy ServiceID
```

*ServiceID* は、サービス・ディスクリプタ内の service 要素の ID 属性です。

## OC4J クラスタへの SOAP Web サービスのインストール

OracleAS SOAP サービスは、クラスタ内の各マシンにインストールする必要があります。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、サービス起動エラーとなる場合があります。

# OracleAS SOAP ハンドラの使用

ハンドラは、`oracle.soap.server.Handler` インタフェースを実装するクラスです。リクエスト、レスポンスまたはエラーという 3 つのコンテキストのいずれかで、ハンドラをチェーンの一部として構成できます。チェーンに含まれるハンドラは、構成ファイルに指定されている順序で起動されることに注意してください。

## リクエスト・ハンドラ

リクエスト・チェーン内のハンドラは、着信するリクエストごとに、SOAP リクエスト・ハンドラ・サーブレットにより SOAP エンベロープが読み取られた直後に起動されます。リクエスト・チェーン内のハンドラによって例外がスローされると、チェーンの処理が即時に終了し、サービスは起動しません。

リクエスト・チェーンの起動中に例外が発生すると、エラー・チェーンが起動します。

## レスポンス・ハンドラ

レスポンス・チェーン内のハンドラは、サービス完了直後にリクエストごとに起動されます。レスポンス・チェーン内のハンドラにより例外がスローされると、チェーンの処理が即時に終了します。レスポンス・チェーンの起動中に例外が発生すると、エラー・チェーンが起動します。

## エラー・ハンドラ

リクエスト・チェーン、サービスまたはレスポンス・チェーンの起動中に例外が発生すると、SOAP リクエスト・ハンドラ・サーブレットによりエラー・チェーン内のハンドラが起動されます。リクエスト・チェーンやレスポンス・チェーンとは異なり、エラー・ハンドラからの例外はロギングされ、エラー・チェーンの処理は継続します。エラー・ハンドラの 1 つが例外をスローしたかどうかに関係なく、エラー・チェーン内のすべてのハンドラが起動されます。

## ハンドラの構成

SOAP 構成ファイル内でハンドラとハンドラ・チェーンを構成します。ハンドラは、サービス・リクエストまたはレスポンスごとに起動するか、エラー発生時に起動できます。ハンドラは各 SOAP リクエストに適用されるという意味でグローバルであり、特定のサービスに対するすべてのリクエストなど、リクエストのサブセットについては構成できません。

ハンドラを構成するには、SOAP 構成ファイル `soap.xml` 内でパラメータを設定します。例 A-2 に、ハンドラ構成を示す SOAP 構成ファイルからのサンプル・セグメントを示します。

**例 A-2 ハンドラ構成**

```
<osc:handlers>
   <osc:handler name="auditor"
      class="oracle.soap.handlers.audit.AuditLogger">
      <osc:option name="auditLogDirectory"
           value="/private1/oracle/app/product/tv02/soap/webapps/soap/WEB-INF"/>
      <osc:option name="filter" value="(!(host=localhost))"/>
   </osc:handler>
</osc:handlers>

<osc:requestHandlers names="auditor"/>
<osc:responseHandlers names="auditor"/>
<osc:errorHandlers names="auditor"/>
```

# OracleAS SOAP の監査ロギングの使用

OracleAS SOAP の監査ロギング機能により、SOAP の使用方法がモニターされ、記録されます。監査ロギングでは、後処理分析とアカウンタビリティのための記録がメンテナンスされます。SOAP の監査ロギング機能は、SOAP リクエスト・ハンドラ・サーブレット（SOAP サーバー）をホスティングする OC4J サーバーで使用可能な監査ロギング機能を補完します。

OracleAS SOAP では、監査証跡は XML 文書として格納されます。OracleAS SOAP では、XML 文書を使用してポータブルな監査証跡が作成され、監査証跡全体または個々の監査レコードを異なるフォーマットに変換できます。

デフォルトでは、OracleAS SOAP の監査ロギングにより、`Handler` インタフェース（`oracle.soap.server` パッケージの一部）を実装する監査ログ出力クラスが使用されます。監査ログ出力クラスは条件付きで起動され、サービス・リクエスト、サービス・レスポンスおよびエラーなどのイベントをモニターします。

この項の内容は、次のとおりです。

- 監査ロギング情報
- 監査可能なイベント
- 監査ログ出力の構成

# 監査ロギング情報

表 A-2 に、監査ログ・レコードごとに使用可能な監査ロギング要素を示します。個々の監査ログ・レコードには、これらの要素の一部のみが含まれている場合があります。ログ・ファイルには、各監査ログ・レコードは SoapAuditRecord 要素として格納されます。

**表 A-2 監査可能な監査レコード要素**

| 監査レコード要素 | 説明 |
|---|---|
| HostName | リクエストを送信したクライアントのホスト名を指定します。 |
| IpAddress | リクエストを送信したクライアントの IP アドレスを指定します。 |
| Method | SOAP リクエスト用のメソッド名を指定します。 |
| Request Envelope | SOAP リクエスト・メッセージ全体を指定します。 |
| Request Envelope Method | SOAP リクエスト・エンベロープ内のメソッドの名前です。 |
| Request Envelope URI | SOAP リクエスト・エンベロープ内のサービスの URI を指定します。 |
| Response Envelope | SOAP レスポンス・メッセージ全体を指定します。 |
| ServiceURI | SOAP リクエスト用のサービス URI を指定します。 |
| SoapAuditRecord | 個々のレコードが含まれています。chainType 属性は、レコードがリクエストの一部として生成されるか、レスポンスの一部として生成されるかを示します。 |
| TimeStamp | SOAP 監査レコードが生成されたシステム時刻を指定します。 |
| User | リクエストに関連するユーザー名を指定します。この要素が提供されるのは、ユーザー・コンテキストがサービス・リクエストまたはサービス・レスポンスに関連付けられている場合のみであることに注意してください。 |

## 監査ロギングの出力

生成される監査ログの XML Schema は、UNIX の場合はディレクトリ $SOAP_HOME/schema、Windows の場合は %SOAP_HOME%¥schema のファイル SoapAuditTrail.xsd にあります。生成される監査ログ・レコードのフォーマットの詳細は、スキーマ・ファイルを参照してください。

# 監査可能なイベント

監査ログ出力クラスは、監査可能なイベントが発生し、SOAP リクエスト・ハンドラ・サーブレットがイベントを監査できるように構成されている場合に起動します。監査可能なイベントには、サービス・リクエストやサービス・レスポンスが含まれます。

## 監査ロギング・フィルタ

SOAP 構成ファイル内で監査ロギング・フィルタを指定し、監査ログに記録される監査可能なイベント・セットを制限できます。SOAP サーバーでは、リクエスト・イベントとレスポンス・イベントにイベント・フィルタが適用されます。表 A-4 に、イベント・フィルタ指定で選択できるフィルタ属性を示します。フィルタを適用すると、監査ログに生成されるレコード数を制限できます。たとえば、特定のホスト用のフィルタを指定すると、そのホストについて生成された監査可能なイベントのみが監査ログに保存されます。

フィルタを使用して監査可能なイベントを定義する構文は、RFC 2254 に由来します。表 A-3 にフィルタの構文、例 A-3 にその例を示します。

**関連項目：**

- A-13 ページの「監査ログ出力の構成」
- RFC 2254 については、ftp://ftp.isi.edu/in-notes/rfc2254.txt を参照してください。

**表 A-3 監査証跡イベントのフィルタ属性**

| 監査イベントのフィルタ属性 | 説明 |
|---|---|
| Host | サービス・リクエストまたはレスポンスのホストの名前を指定します。この属性をフィルタで指定しないと、クライアントのホスト名は監査ログ・レコードのフィルタリングに使用されません。<br>クライアントのホスト名全体を指定するか、ワイルド・カード（「*」）を使用してください。指定したホスト名に埋め込んだワイルド・カードはサポートされません。次に、ワイルド・カードの有効な使用例と無効な使用例を示します。ワイルド・カードを使用する場合は、フィルタの 1 文字目にする必要があります。ホスト名の大 / 小文字区別はありません。この属性を設定する場合は注意する必要があります。DNS 設定に応じて、`explosives.acme.com` のような完全修飾ホスト名が戻される場合と、`explosives` のような非修飾のホスト名が戻される場合があります。一部の IP アドレスの場合は、DNS でホスト名を解決できない場合があります。<br>`Host` フィルタ属性に有効な値は、次のとおりです。<br>`explosives.acme.com, *.acme.com, *.com`<br>`Host` フィルタ属性に無効な値は、次のとおりです。<br>`*, explosives.acme.*, explosives.*, ex*s.acme.com, *ives.acme.com` |
| ip | サービス・リクエストまたはレスポンスのクライアントの IP アドレスを指定します。<br>クライアントの IP アドレスは、4 バイトすべてを使用してドット区切りの 10 進形式で完全指定するか、ワイルド・カード（「*」）を使用して指定する必要があります。埋込みワイルド・カードはサポートされません。ワイルド・カードを使用する場合は、フィルタの最終文字にする必要があります。<br>この属性をフィルタに指定しないと、クライアントの IP アドレスはフィルタリングに使用されません。<br>`ip` フィルタ属性に有効な値は、次のとおりです。<br>`138.2.142.154, 138.2.142.*, 138.2.*, 138.*`<br>`ip` フィルタ属性に無効な値は、次のとおりです。<br>`*, 138.2.*.154, *.2, 138.*.152, 138.2.142, 138.2, 138` |
| urn | サービスの URN を指定します。この属性の場合、ワイルド・カードはサポートされません。 |
| username | クライアントに関連するトランスポート・レベルのユーザー名を指定します。<br>username フィルタ属性の場合、ワイルド・カードはサポートされません。 |

**表 A-4 監査ログ・フィルタの構文**

| フィルタ値 | 説明 |
|---|---|
| attr | 1*（「*」、「(」、「)」、「&」、「\|」、「!」、「*」、「=」を除く任意の US-ASCII 文字） |
| equal | 「=」 |
| filter | 「(」 *filtercomp* 「)」<br>「(」 *filtercomp* と「)」の間には空白は許可されません。 |
| filtercomp | *and* \| *or* \| *not* \| *item*<br>*and* = 「&」 *filterlist*<br>*or* = 「\|」 *filterlist*<br>*not* = 「!」 *filter* |
| filterlist | 2*2 *filter* |
| filtertype | *equal* |
| item | *attr filtertype value*<br>*attr*、*filtertype* および *value* 間の空白は許可されません。 |

**表 A-4 監査ログ・フィルタの構文（続き）**

| フィルタ値 | 説明 |
|---|---|
| value | 1*（ASCII 表現の「)」- 0x29 を除く任意の 8 進数)。<br>文字「*」には特殊な意味があります。<br>「*」文字はワイルド・カードとして参照され、任意の文字と一致します。 |

**例 A-3 サンプル監査ログ・フィルタ**

```
(ip=138.2.142.154)
(!(host=localhost))
(!(host=*.acme.com))
(&(host=*.acme.com)(username=daffy))
(&(ip=138.2.142.*)(|(urn=urn:www-oracle-com:AddressBook)(username=daffy)))
```

# 監査ログ出力の構成

Oracle Application Server に用意されているデフォルトの SOAP 監査ログ出力を構成するには、SOAP 構成ファイル `soap.xml` 内でパラメータを設定します。デフォルトの監査ログ出力を使用可能にして監査ロギングをオンにするには、構成ファイル内で次の操作を実行します。

- 監査ログ・ハンドラの名前とオプションを定義します。デフォルトの SOAP 監査ログ出力は、クラス `oracle.soap.handlers.audit.AuditLogger` に定義されています。デフォルトの監査ログ出力では、構成ファイル内で指定する複数のオプションがサポートされます。使用可能な監査ログ出力オプションについては、表 A-5 を参照してください。
- 監査ログ出力ハンドラ名を `requestHandler`、`responseHandler` または `errorHandler`（あるいはすべてのハンドラ・チェーン）に追加します。

例 A-4 に、監査ロギング構成オプションを含む SOAP 構成ファイルからのサンプル・セグメントを示します。例 A-4 には、すべてのオプションを使用するように設定されている構成オプションを示します。ただし、この構成では極端に大きい監査ログが生成されるため、お薦めしません。

> **注意：** 監査ログ出力を使用してエラーを監査する場合は、`includeRequest` または `includeResponse` を有効化していても、リクエスト・チェーンまたはレスポンス・チェーン内でエラーが発生するタイミングによっては、監査ログ・レコードにリクエスト・メッセージまたはレスポンス・メッセージが含まれない場合があります。

**例 A-4 監査ロギング構成**

```
<osc:handlers>
   <osc:handler name="auditor"
      class="oracle.soap.handlers.audit.AuditLogger">
      <osc:option name="auditLogDirectory"
           value="/private1/oracle/app/product/tv02/soap/webapps/soap/WEB-INF"/>
      <osc:option name="filter" value="(!(host=localhost))"/>
      <osc:option name="includeRequest" value="true"/>
      <osc:option name="includeResponse" value="true"/>
   </osc:handler>
</osc:handlers>
<osc:requestHandlers names="auditor"/>
<osc:responseHandlers names="auditor"/>
<osc:errorHandlers names="auditor"/>
```

**表 A-5 監査ログ出力構成オプション**

| オプション | 説明 |
|---|---|
| auditLogDirectory | 監査ログ・ファイルを保存するディレクトリを指定します。`auditLogDirectory` オプションは必須です。生成される監査ログ・ファイルの名前は `OracleSoapAuditLog.`*`timestamp`* で、*`timestamp`* はファイルの最初の生成日時です。<br>**有効な値 :** 有効なディレクトリを示す任意の文字列 |
| filter | 監査イベント・フィルタを指定します。これはオプションです。`filter` を指定しない場合、SOAP サーバーではすべてのイベントがログに記録されます。<br>**有効な値 :** 有効な任意のフィルタ |
| includeRequest | 監査ログ・レコードに、それを生成したイベントのリクエスト・メッセージを含むように指定します。<br>**有効な値 :** `true`、`false`<br>`true` または `false` 以外の値は、エラーとして処理されます。<br>**デフォルト値 :** `false` |
| includeResponse | 監査ログ・レコードに、それを生成したイベントのレスポンス・メッセージを含むように指定します。<br>**有効な値 :** `true`、`false`<br>`true` または `false` 以外の値は、エラーとして処理されます。<br>**デフォルト値 :** `false` |

**関連項目 :** A-9 ページの「OracleAS SOAP ハンドラの使用」

# OracleAS SOAP の交換可能構成マネージャの使用

OracleAS SOAP では、Apache SOAP 2.3.1 でサポートされているものと同様の交換可能構成マネージャがサポートされます。OracleAS SOAP ではサービス・デプロイメント・ディスクリプタとは異なるプロバイダ・デプロイメント・ディスクリプタがサポートされるため、OracleAS SOAP を使用するインタフェースの詳細は Apache SOAP 2.3.1 とは少し異なります。OracleAS SOAP では、構成マネージャはプロバイダ・マネージャおよびサービス・マネージャとは別個に構成されます。すべての構成マネージャには、`oracle.soap.server.ConfigManager` インタフェースを実装する必要があります。

開発を簡素化するために、構成マネージャの実装を記述するときには、OracleAS SOAP に用意されている抽象クラス（`oracle.soap.server.impl.BaseConfigManager`）を使用できます。この抽象クラスには、永続ストアの読取りと書込みを行う 2 つの抽象メソッドとともに、`ConfigManager` インタフェースのほとんどの標準実装が用意されています。

例 A-5 に、プロバイダ構成マネージャのサンプル実装を示します。

**例 A-5 プロバイダ構成マネージャのサンプル実装**

```
public class MyProviderConfigManager extends BaseConfigManager
{
    public void setOptions(Properties options)
        throws SOAPException
    {
        // handle implementation specific options
    }

    public void readRegistry()
        throws SOAPException
    {
        // read the deployed providers from persistent store
```

```
    }

    public void writeRegistry()
        throws SOAPException
    {
        // write the deployed providers to persistent store
    }
}
```

setOptions メソッドには、<configManager> 要素のすべての <option> 要素に指定されているオプションが渡されます。レジストリの読取り / 書込みは、特定の構成マネージャ実装で同期化する必要があります。

# OracleAS SOAP のトランスポート・セキュリティの操作

Oracle Application Server では、SOAP メッセージを送信する、基礎となるトランスポートのセキュリティ機能が使用されます。また、SOAP メッセージの送信用に HTTP および HTTPS プロトコルがサポートされます。HTTP と HTTPS では、次のセキュリティ機能がサポートされます。

- HTTP プロキシ
- HTTP 認証（Basic RFC 2617）
- プロキシ認証（Basic RFC 2617）

OracleAS SOAP Client のトランスポートでは、Oracle Wallet Manager をサポートするように変更された HTTPClient パッケージが使用されます。OracleAS SOAP のトランスポートでは、これらの機能をサポートするために複数のプロパティが定義されます。Oracle Application Server でサポートされるクライアント・サイドのセキュリティ・プロパティについては、表 A-6 を参照してください。

OracleAS SOAP Client アプリケーションでは、Java コマンドラインで -D フラグを使用して、表 A-6 のセキュリティ・プロパティをシステム・プロパティとして設定できます。また、セキュリティ・プロパティをシステム・プロパティに追加して、これらのプロパティを Java プログラム内で設定する方法もあります（プロパティを追加するには、System.setProperties() を使用します）。

例 A-6 に、Oracle Application Server でトランスポート固有の API を使用して、システム・プロパティに指定した値のオーバーライドがどのようにサポートされるかを示します。クラス OracleSOAPHTTPConnection の setProperties() メソッドには、HTTP 接続専用の設定プロパティが含まれています（このクラスはパッケージ oracle.soap.transport.http にあります）。

**例 A-6 OracleSOAPHHTTPConnection 用のセキュリティ・プロパティの設定**

```
org.apache.soap.rpc.Call call = new org.apache.soap.rpc.Call();
oracle.soap.transport.http.OracleSOAPHTTPConnection conn =
(oracle.soap.transport.http.OracleSOAPHTTPConnection) call.getSOAPTransport();
java.util.Properties prop = new java.util.Properties();
// Use client code to set name-value pairs of properties in prop
.
.
.
conn.setProperties(prop);
```

---

**注意：** プロパティ java.protocol.handler.pkgs はシステム・プロパティとして設定する必要があります。

---

**表 A-6 SOAP HTTP トランスポートのセキュリティ・プロパティ**

| プロパティ | 説明 |
|---|---|
| http.authRealm | HTTP 認証のユーザー名 / パスワードを指定するレルムを指定します。<br>このプロパティは、Basic 認証を使用するときは必須です。 |
| http.authType | HTTP の認証タイプを指定します。指定した値の大 / 小文字区別は無視されます。<br>有効な値 : `basic`、`digest`<br>値 basic は、HTTP の Basic 認証を指定します。<br>`basic` または `digest` 以外の値を指定すると、このプロパティは未設定とみなされます。 |
| http.password | HTTP 認証のパスワードを指定します。 |
| http.proxyAuthRealm | プロキシ認証のユーザー名 / パスワードを指定するレルムを指定します。 |
| http.proxyAuthType | プロキシ認証タイプを指定します。指定した値の大 / 小文字区別は無視されます。<br>有効な値 : `basic`、`digest`<br>`basic` または `digest` 以外の値を指定すると、このプロパティは未設定とみなされます。 |
| http.proxyHost | プロキシ・ホストのホスト名または IP アドレスを指定します。 |
| http.proxyPassword | HTTP のプロキシ認証パスワードを指定します。 |
| http.proxyPort | プロキシ・ポートを指定します。整数値を指定する必要があります。このプロパティが使用されるのは `http.proxyHost` が定義されている場合のみで、他の場合は無視されます。<br>デフォルト値 : `80` |
| http.proxyUsername | HTTP のプロキシ認証のユーザー名を指定します。 |
| http.username | HTTP 認証のユーザー名を指定します。 |
| java.protocol.<br>handler.pkgs | `java.net.URLStreamHandlerFactory` 用のパッケージ接頭辞のリストを指定します。接頭辞は縦線文字（|）で区切る必要があります。<br>この値には `HTTPClient` を含める必要があります。この値は Java プロトコル・ハンドラ・フレームワークに必須です。Oracle Application Server では定義されません。HTTPS を使用する場合は、このプロパティを設定する必要があります。HTTPS を使用する場合にこのプロパティを設定しないと、`java.net.MalformedURLException` がスローされます。<br>**注意 :** このプロパティは、システム・プロパティとして設定する必要があります。<br>たとえば、このプロパティを次のどちらかに設定します。<br>■ `java.protocol.handler.pkgs=HTTPClient`<br>■ `java.protocol.handler.pkgs=sun.net.www.protocol\|HTTPClient` |

**表 A-6 SOAP HTTP トランスポートのセキュリティ・プロパティ（続き）**

| プロパティ | 説明 |
|---|---|
| oracle.soap.transport.1022ContentType | Oracle9*i*AS および Oracle Application Server 10*g* のコンテンツ・タイプ HTTP ヘッダーの値を指定します。このプロパティの値により、Oracle9*i*AS リリース 1.0.2.2、リリース 2（9.0.x）または 10*g*（9.0.4）で稼働中の OracleAS SOAP サーバーがサポートされます。このプロパティは、Oracle9*i*AS リリース 2（9.0.x）の OracleAS SOAP Client または Oracle Application Server 10*g*（9.0.4）と、以前のバージョンのサーバー（Oracle9*i*AS リリース 1.0.2.2 に付属）間の相互運用性を提供します。<br><br>有効な値 : `true`、`false`（大 / 小文字区別なし）<br><br>この値を `true` に設定すると、SOAP メッセージの送信時に Oracle9*i*AS リリース 1.0.2.2 のコンテンツ・タイプ HTTP ヘッダー値が使用されます。その場合は、この値が次のように設定されます。<br><br>`content-type: text/xml`<br><br>この値を `false` に設定すると、SOAP メッセージの送信時に Oracle9*i* Application Server リリース 2（9.0.x）のコンテンツ・タイプ・ヘッダーが使用されます。その場合は、この値が次のように設定されます。<br><br>`content-type: text/xml; charset=utf-8`<br><br>`false` はデフォルト値です。<br><br>**注意 :** 添付ファイル付きの SOAP メッセージの場合、コンテンツ・タイプ HTTP ヘッダーは常に値 `multipart/related` に設定されます。 |
| oracle.soap.transport.allowUserInteraction | allows user interaction パラメータを指定します。指定した値の大 / 小文字区別は無視されます。次のどちらかの場合に、このプロパティを `true` に設定すると、ユーザー名とパスワードの入力を求められます。<br><br>**1.** プロパティ `http.authType`、`http.username` または `http.password` を設定しておらず、HTTP サーバーから `401` HTTP ステータスが戻される場合<br><br>**2.** プロパティ `http.proxyAuthType`、`http.proxyUsername` または `http.proxyPassword` を設定しておらず、HTTP プロキシから `407` HTTP レスポンスが返される場合<br><br>有効な値 : `true`、`false`<br><br>`true` 以外の値を指定すると、`false` とみなされます。 |
| oracle.ssl.ciphers | 有効化する暗号スイートをコロン（`:`）で区切ったリスト形式で指定します。<br><br>デフォルト値 : Oracle SSL でサポートされているすべての暗号スイートのリスト。 |
| oracle.wallet.location | エクスポートする Oracle Wallet またはトラスト・ポイントの位置を指定します。<br><br>**注意 :** 次のように、URL ではなくファイル位置を指定します。<br><br>`/etc/ORACLE/Wallets/system1/exported_wallet`（UNIX）<br><br>`d:¥oracle¥system1¥exported_wallet`（Windows）<br><br>SSL 認証、サーバー認証または相互認証で HTTPS をトランスポートとして使用する場合は、このプロパティを設定する必要があります。 |
| oracle.wallet.password | エクスポートする Wallet のパスワードを指定します。クライアント認証、つまり相互認証で HTTPS をトランスポートとして使用する場合は、このプロパティを設定する必要があります。 |

## SSL 用の Apache リスナーとサーブレット・エンジンの構成

Apache リスナーと mod_ssl（または mod_ossl）を使用する場合は、soap servletlocation/directory 用に次のディレクティブを設定する必要があります。

```
SSLOption +StdEnvVars +ExportCertData
```

このディレクティブは、条件付きで設定できます。詳細は、mod_ssl または mod_ossl のマニュアルを参照してください。デフォルトでは、パフォーマンス上の理由でこのディレクティブは無効化されています。このディレクティブを設定しないと、サーブレット・エンジンは SSL 関連のデータ（暗号スイート、クライアント資格証明書など）にアクセスできません。

## Oracle Application Server SOAP Client での JSSE の使用

この項では、Oracle セキュリティ・インフラストラクチャを使用できない場合に、OracleAS SOAP Client のクライアント・サイドで SSL を使用する方法について説明します。Oracle セキュリティ・インフラストラクチャを使用できる場合は、Oracle クライアント・サイド・ライブラリ（`$ORACLE_HOME/lib/*`、`$ORACLE_HOME/jlib/javax-ssl-1_2.jar` および `$ORACLE_HOME/jlib/jssl-1_2.jar` など）を使用できます。

OracleAS SOAP では、デフォルトのトランスポート・クラスとして次のクラスが使用されます。

```
oracle.soap.transport.http.OracleSOAPHTTPConnection
```

このクラスは、`HTTPClient` パッケージの修正バージョンです。`HTTPClient` の詳細は、次のサイトを参照してください。

```
http://www.innovation.ch/java/HTTPClient/
```

このバージョンの `HTTPClient` パッケージは Oracle Java SSL と統合されており、HTTPS トランスポート用に Oracle Wallet がサポートされます。SOAP のクライアント・サイドで OracleAS SOAP Client を使用できない場合も、OracleAS SOAP Client のクライアント・サイド・ライブラリでトランスポートとして HTTPS を使用できます。

HTTPS を使用する手順は、次のとおりです。

1. 次のトランスポート・クラスを使用します。

   ```
   class org.apache.soap.transport.http.SOAPHTTPConnection
   ```

   RPC を使用する場合は、org.apache.soap.transport.http.SOAPHTTPConnection のインスタンスを引数として渡し、次のメソッドをコールします。

   ```
   method org.apache.soap.rpc.Call#setSOAPTransport
   (org.apache.soap.transport.SOAPTransport)
   ```

   次に例を示します。

   ```
   org.apache.soap.rpc.Call myCallObj = new
   org.apache.soap.rpc.Call();
   myCallObj.setSOAPTransport(new
   org.apache.soap.transport.http.SOAPHTTPConnection());
   ```

   メッセージ機能を使用する場合は、org.apache.soap.transport.http.SOAPHTTPConnection のインスタンスを引数として渡し、次のメソッドをコールします。

   ```
   org.apache.soap.messaging.Message#setSOAPTransport
   (org.apache.soap.transport.SOAPTransport)
   ```

   次に例を示します。

   ```
   org.apache.soap.messaging.Message myMsgObj = new
   org.apache.soap.messaging.Message();
   myMsgObj.setSOAPTransport(new
     org.apache.soap.transport.http.SOAPHTTPConnection());
   ```

2. Java Secure Socket Extension（JSSE）をダウンロードし、指示に従って構成します。JSSE は次のサイトから入手できます。

   `http://java.sun.com/products/jsse/`

   - ファイル `jnet.jar`、`jcert.jar` および `jsse.jar` が、CLASSPATH またはインストール済の拡張機能ディレクトリ（`$JRE_HOME/lib/ext`）にあることを確認します。
   - SunJSSE プロバイダが適切に構成されていることを確認します。この操作は、$JRE_HOME/lib/security/java.security ファイルを編集して次の行を追加すると静的に実行できます。

     `security.provider.`*num*`=com.sun.net.ssl.internal.ssl.Provider`

   *num* は 1 から始まる作業環境番号です。また、次のコード行を追加して、実行時に動的にプロバイダを追加する方法もあります。

   ```
   Security.addProvider(new com.sun.net.ssl.internal.ssl.Provider());
   ```

   セキュリティ・プロバイダを動的に追加するには、適切なパーミッションを設定する必要があります。

   - システム・プロパティ `java.protocol.handler.pkgs` が `com.sun.net.ssl.internal.www.protocol` に設定されていることを確認します。
   - プロキシ・サーバーを使用する場合は、次のシステム・プロパティがそれぞれ適切なプロキシ・ホスト名とプロキシ・ポートに設定されていることを確認します。

     ```
     https.proxyHost
     https.proxyPort
     ```

   - サーバー・サイド認証付きの SSL とデフォルトの `TrustManager` を使用する場合は、サーバー証明書の署名側が次のファイルに含まれていることを確認します。

     `$JRE_HOME/lib/security/jssecacerts`

     または、次の場所に `jssecacerts` が存在しないことを確認します。

     `$JRE_HOME/lib/security/cacerts`

   - KeyManager/TrustManager キーストアのデフォルト位置をオーバーライドするには、次のシステム・プロパティを使用します。

     ```
     javax.net.ssl.keystore
     javax.net.ssl.keyStoreType
     javax.net.ssl.keyStorePassword
     javax.net.ssl.trustStore
     javax.net.ssl.trustStoreType
     javax.net.ssl.trustStorePassword
     ```

     詳細は、JSSE のドキュメントを参照してください。特定のサード・パーティの JSSE 実装を使用する場合は、該当するドキュメントを参照してください。

     **関連項目：** 次のサイトにある `HTTPClient` 情報を参照してください。

     `http://www.innovation.ch/java/HTTPClient/`

# OracleAS SOAP のサンプル・サービスの使用

この項では、OracleAS SOAP に用意されているサンプルについて説明します。すべてのサンプルのクラス・ファイルは、UNIX の場合は `$SOAP_HOME/lib/samples.jar`、Windows の場合は `%SOAP_HOME%¥lib¥samples.jar` にあります。

どのサンプルを実行する場合も、サーブレットの CLASSPATH で `samples.jar` が使用可能になっていることを確認する必要があります。詳細は、各サンプルに付属の README を参照してください。

## Xmethods のサンプル

xmethods サンプル内のクライアントは、インターネット上のシステムでホスティングされる既存のサービスにアクセスするため、SOAP に関する最も簡単なスタート・ガイドとして使用できます。これらのサービスの情報は、次のサイトを参照してください。

`http://www.xmethods.org`

このサンプルは $SOAP_HOME/samples/xmethods にあります。

## AddressBook のサンプル

このサンプルは、Java および複数のクライアントに実装されるサービスです。このサンプルはリテラルの XML エンコーディングを示します。サンプル・ソース・コードについては、`$SOAP_HOME/samples/addressbook` を参照してください。このディレクトリには、トランスポートとして HTTPS を使用するサンプルのアドレス帳クライアントの実行方法を示すスクリプトも含まれています。

## StockQuote のサンプル

このサンプルは、Java および単一のクライアントに実装されるサービスです。このサンプルは $SOAP_HOME/samples/stockquote にあります。

## Company のサンプル

このサンプルには、PL/SQL ストアド・プロシージャと複数のクライアントで構成されるサービスが含まれています。このサンプルは $SOAP_HOME/samples/sp/company にあります。このサンプル・サービスの設定、コンパイルおよびテスト方法の詳細は、このディレクトリにある README ファイルを参照してください。

## Provider のサンプル

このサンプルには、独自プロバイダの作成時に参考になるテンプレート・プロバイダが含まれています。

## AddressBook2 のサンプル

このサンプルは、セッションの有効範囲で Addressbook サービスを使用する方法を示すデモです。SOAP コール間で同じ HTTP セッションをメンテナンスする方法を示します。このサンプルには、WSDL サービス記述ファイルから生成される SOAP クライアント・プロキシの例が含まれています。このサンプルは $SOAP_HOME/samples/addressbook2 にあります。

## Messaging のサンプル

このサンプルは、メッセージ・ベースの SOAP サービスの例です。このサンプルは $SOAP_HOME/samples/messaging にあります。

## Mime のサンプル

このサンプルでは、RPC およびメッセージ・ベースのサービスを使用して添付ファイルの SOAP 処理が実行されます。このサンプルは $SOAP_HOME/samples/mime にあります。

# OracleAS SOAP EJB プロバイダの使用

この項では、OracleAS SOAP の EJB プロバイダと Apache SOAP 2.2 の EJB プロバイダの違いについて説明します。

## ステートレス Session EJB プロバイダ

Apache SOAP の場合、ステートレス EJB プロバイダでは、SOAP リクエストの受信時に EJB のホーム・インタフェースに対する JNDI ルックアップが実行されます。次に、ステートレス EJB の参照を取得するために、EJB のホーム・インタフェース上で create が起動されます。この EJB 参照を使用して、リクエストされたメソッドが起動されます。

OracleAS SOAP では、Apache SOAP と同じメカニズムを使用してステートレス Session EJB がサポートされます。

## Apache SOAP でのステートフル Session EJB プロバイダ

Apache SOAP ステートフル Session EJB プロバイダでは、最初の SOAP リクエストの受信時に、まず JNDI ルックアップを通じてホーム・インタフェースが検索され、後続の create を使用してステートフル Session EJB のオブジェクト参照が取得されます。次に、オブジェクト参照上でリクエストされたメソッドが起動されます。

次のステップでは、プロバイダにより指定された EJB 参照の `EJBHandle` がシリアライズされ、@ デリミタを使用してターゲット URI に追加されます。次に、ステートフル Session EJB プロバイダでは、この変更後のターゲット URI がリクエスト側 SOAP クライアントに戻されます。クライアント・サイドで同じ EJB インスタンスを再利用する必要がある場合は、サービス用にこの変更済ターゲット URI をレスポンスから取得し、次の SOAP コールで設定する必要があります。

このリクエストの受信時に、ステートフル EJB プロバイダでは文字列化された EJB 参照が抽出され、EJBHandle にデシリアライズされます。この EJBHandle から EJB 参照を取得できます。その後、指定した EJB のメソッドを起動できます。

Apache SOAP 実装のデメリットは、クライアントが EJB で認識される必要があり、他の SOAP サーバーでは動作しない場合があることです。

OracleAS SOAP には、クライアントの相互運用性を確保できるように、ステートフル Session EJB 用の代替ソリューションが用意されています。

## OracleAS SOAP のステートフル Session EJB プロバイダ

OracleAS SOAP のステートフル Session EJB プロバイダでは、何もバインドされていない場合は EJB 参照が現行セッションにバインドされ、それ以外の場合は単にセッションから EJB 参照が取得されます。クライアントでは、連続するコール間で現行セッションをメンテナンスするのみで、同じステートフル Session EJB にアクセスできます。

セッション中のいずれかの時点で SOAP クライアントが EJB のホーム・インタフェースの create を起動すると、プロバイダにより create からセッションに EJB 参照がバインディングされ、そのセッション中の他のコール・リクエストに使用されます。

## OracleAS SOAP の Entity EJB プロバイダ

SOAP クライアントでは、Entity EJB でビジネス・メソッドを実行するには、最初にメソッドを実行する新規 EJB を作成するか、なんらかの基準を満たす既存の EJB を検索する必要があります。Entity EJB へのアクセスは、セッション中に発生します。セッションの開始時に、SOAP クライアントでは create または find を（Bean オブジェクトの対象を指定するために）起動する必要があります。同じセッションのメンテナンス中は、他のすべてのビジネス・メソッドはその EJB にダイレクトされます。同一セッションまたは異なるセッション中の後続の find または create では、ビジネス・メソッドの実行リクエストは新規に作成（または検索）された EJB にダイレクトされます。

もう 1 つの問題は、EJB 仕様では、一部の find メソッドで EJB 参照のコレクションまたは単一の EJB 参照を戻すことができる点です。

Entity EJB に関する Oracle のソリューションには、この問題に対して次のソリューションが含まれています。

コレクションを戻す find メソッドを禁止します。これにより、プロバイダは Entity EJB を一意に指定し、後続のビジネス・メソッド・リクエストをターゲットにできます。

## OracleAS SOAP EJB プロバイダのデプロイと使用

EJB プロバイダをインストールし、アプリケーション・サーバーが SOAP サーブレットとデプロイされた EJB の両方をホスティングする OC4J 上のプロバイダに Web サービスをデプロイする手順は、次のとおりです。

1. プロバイダ・ディスクリプタを使用して EJB プロバイダを SOAP にデプロイします。

   プロバイダ・ディスクリプタでは、次の情報を指定します。

   - 中間層による EJB アクセス資格証明
   - JNDI コンテキストのファクトリ・クラス
   - JNDI コンテキストのファクトリ URL
   - プロバイダ・クラス名
   - プロバイダ ID

2. EJB Web サービスを作成します。

   - 関連する EJB クラスを定義し、EJB を J2EE 仕様で定義された EAR ファイルにパッケージします。
   - EJB Web サービスの次の詳細を指定してサービス・ディスクリプタを定義します。
     * JNDI の位置
     * ホーム・インタフェースのクラス名
     * EJB Web サービスの OC4J でのアプリケーションのデプロイ名
     * サービスを関連付けるプロバイダ ID

3. .ear ファイルを OC4J にデプロイします。OC4J 固有の EJB ディスクリプタを変更し、EJB の JNDI の位置を訂正します（サンプルの README を参照）。

## EJB プロバイダの既知の制限

どのサービス・メソッドの場合も、引数として使用できるのは Java のプリミティブ型のみです。現在、Java のユーザー定義型はサポートされていません。

# SP プロバイダと PL/SQL ストアド・プロシージャの使用

OracleAS SOAP のストアド・プロシージャ（SP）・プロバイダでは、PL/SQL ストアド・プロシージャまたはファンクションを SOAP サービスとして公開する操作がサポートされます。Oracle9*i* Database Server では、Java や C/C++ のような他の言語で実装されたプロシージャを、PL/SQL を使用して公開できます。これらのストアド・プロシージャは、PL/SQL インタフェースを通じて SOAP サービスとして公開されます。

SP プロバイダのフレームワークは、PL/SQL プロシージャを Java のラッパー・クラスに変換し、生成された Java クラスを SOAP の Java サービスとしてエクスポートすると機能します。

## SP プロバイダでサポートされる機能

SP プロバイダでサポートされる機能は、次のとおりです。

- PL/SQL ストアド・プロシージャ。プロシージャとファンクションの両方が含まれます（このマニュアルでは、プロシージャという用語を、プロシージャとファンクションの両方の意味で使用しています）。
- IN パラメータ・モード。
- パッケージされたプロシージャのみ（トップレベル・プロシージャをエクスポートするには、パッケージ内でラップする必要があります）。
- オーバーロード・プロシージャ（ただし、変換中に 2 つの異なる PL/SQL 型が同じ Java 型にマップされる場合は、PL/SQL パッケージのエクスポート中にエラーが発生することがあります。このようなエラーを解決するには、オーバーロードを回避する方法と、問題のオーバーロード・プロシージャを含まない新規のダミー・パッケージを記述する方法があります）。
- 単純型。
- オブジェクト型（ユーザー定義型）。

## SP プロバイダでサポートされない機能

SP プロバイダでサポートされない機能は、次のとおりです。

- SP プロバイダのフレームワークでは、PL/SQL から Java への変換に Oracle JPublisher が使用されるため、Oracle JPublisher の制限がすべて継承されます。

## SP プロバイダでサポートされる単純な PL/SQL 型

SOAP SP プロバイダでは、次の単純型がサポートされます。NULL 値は、NATURALN および POSITIVEN を除き、次に示すすべての単純型でサポートされます。

これらの型のマッピングの詳細は、Oracle JPublisher のマニュアルを参照してください。

- VARCHAR2（STRING、VARCHAR）
- LONG
- CHAR（CHARACTER）
- NUMBER（DEC、DECIMAL、DOUBLE PRECISION、FLOAT、INTEGER、INT、NUMERIC、REAL、SMALLINT）
- PLS_INTEGER
- BINARY_INTEGER（NATURAL、NATURALN、POSITIVE、POSITIVEN）

## オブジェクト型の使用

Oracle JPublisher では、ユーザー定義のオブジェクト型の使用がサポートされます。SP プロバイダのフレームワークでは、SOAP でデフォルトの `BeanSerializer` を使用して自動的にシリアライズできるように、`oracle.sql.CustomDatum` スタイルのクラスが生成されます。

オブジェクト型の使用例は、Company のサンプルを参照してください。

## ストアド・プロシージャ・プロバイダのデプロイ

例 A-7 に、ストアド・プロシージャ用のサンプル・プロバイダ・デプロイメント・ディスクリプタを示します。プロバイダ名には任意の一意 ID を使用できます（この例では company-provider を使用しています）。

属性 user、password および url を使用して、データベースに接続する URL が作成されます。この 3 つの属性はすべて必須です。このプロバイダで処理されるサービス用接続の数は、`connections_per_service` を使用して設定します。これはオプションで、デフォルト値は 10 です。

プロバイダ・マネージャを使用して、ローカル構成用に適切に編集された例 A-7 のサンプル・プロバイダ・ディスクリプタをデプロイします。

**例 A-7 サンプル SP プロバイダ・デプロイメント・ディスクリプタ**

```
<isd:provider xmlns:isd="http://xmlns.oracle.com/soap/2001/04/deploy/provider"
    id="company-provider"
    class="oracle.soap.providers.sp.SpProvider">
  <!-- edit the following option "values" as appropriate -->
  <isd:option key="user" value="YOUR-USER-NAME" />
  <isd:option key="password" value="YOUR-PASSWORD" />
  <isd:option key="url" value="jdbc:oracle:thin:@YOUR-HOST:YOUR-PORT:YOUR-SID" />
  <isd:option key="connections_per_service" value="3" />
</isd:provider>
```

## PL/SQL ストアド・プロシージャから Java への変換

シェル・スクリプト `$SOAP_HOME/bin/sp2jar.sh` では、PL/SQL パッケージとそこに含まれるすべてのプロシージャおよびファンクションが、等価のメソッドを持つ Java クラスに変換されます。このパッケージでユーザー定義型が使用されている場合は、その型も対応する Java クラスに変換されます。

samples ディレクトリにある `README` ファイルには、`sp2jar.sh` コマンドを使用して company の例をコンパイル済 Java クラスの jar ファイルに変換する例が含まれています。また、`README` には、PL/SQL パッケージをデータベースにロードする方法も含まれています。

これ以降は、PL/SQL パッケージ company がデータベースにインストールされており、jar ファイル `company.jar` 内で使用可能なコンパイル済 Java クラス・セットにエクスポートされていると仮定します。

生成された `company.jar` は、他の Java サービスの場合と同様、SOAP サーブレットの CLASSPATH で使用可能にする必要があります。

## ストアド・プロシージャ・サービスのデプロイ

例 A-8 に、ストアド・プロシージャ用のサンプル・サービス・デプロイメント・ディスクリプタを示します。プロバイダ要素の ID 属性では、このサービスがデプロイされるプロバイダを識別していることに注意してください。

SP プロバイダのフレームワークでは PL/SQL プロシージャが Java クラスのメソッドに変換されるため、サービス・ディスクリプタは Java サービスの場合と同じです。PL/SQL に固有のすべての情報は、プロバイダ・ディスクリプタの一部になっています。サービス自体は Java サービスと同様です。

プロシージャでオブジェクト型が使用されている場合は、オブジェクト型ごとに型マッピングを定義する必要があります。XML の型名を SQL の型名と同じにして大文字で指定する必要があります（例 A-8 の EMPLOYEE と ADDRESS を参照）。javaType 属性では、Oracle JPublisher で生成された oracle.sql.CustomDatum 型が識別されます。

デフォルトの BeanSerializer を使用すると、型をシリアライズまたはデシリアライズできます。

Oracle JPublisher のデフォルト設定で、生成されるメソッド名は小文字になります。

サービス・マネージャを使用して、例 A-8 のサンプル・サービス・ディスクリプタをデプロイします。

**例 A-8 サンプル・ストアド・プロシージャのサービス・デプロイメント・ディスクリプタ**

```
<isd:service xmlns:isd="http://xmlns.oracle.com/soap/2001/04/deploy/service"
    id="urn:www-oracle-com:company"
    type="rpc" >

   <isd:provider
    id="company-provider"
    methods="addemp getemp getaddress getempinfo changesalary removeemp"
    scope="Application" >
    <isd:java class="samples.sp.company.Company"/>
   </isd:provider>

  <isd:mappings>
    <isd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     xmlns:x="urn:company-sample" qname="x:EMPLOYEE"
     javaType="samples.sp.company.Employee"
         java2XMLClassName="org.apache.soap.encoding.soapenc.BeanSerializer"
             xml2JavaClassName="org.apache.soap.encoding.soapenc.BeanSerializer"/>
    <isd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
             xmlns:x="urn:company-sample" qname="x:ADDRESS"
             javaType="samples.sp.company.Address"
             java2XMLClassName="org.apache.soap.encoding.soapenc.BeanSerializer"
           xml2JavaClassName="org.apache.soap.encoding.soapenc.BeanSerializer"/>
  </isd:mappings>

  <isd:faultListener class="org.apache.soap.server.DOMFaultListener"/>

</isd:service>
```

## ストアド・プロシージャの SOAP サービスの起動

PL/SQL ストアド・プロシージャである SOAP サービスの起動方法は、他の SOAP サービスの場合と同じです。PL/SQL パッケージの変換およびデプロイ中に作成された company.jar ファイルは、クライアント・サイドで SOAP サービスを起動するアプリケーション・プログラムをコンパイルする場合にも必要になります（この jar ファイルが必要になるのは、ストアド・プロシージャの入出力の型がユーザー定義型の場合のみです。プロシージャで組込みタイプのみが使用される場合、生成される jar ファイルはクライアント・サイドでは不要です）。

company のサンプル・ディレクトリにある README ファイルには、サンプル・クライアントをコンパイルしてテストする手順が含まれています。

# SOAP のトラブルシューティングと制限

この項では、Oracle Application Server Web Services のトラブルシューティングに関するいくつかの方法について説明します。この項の内容は、次のとおりです。

- TcpTunnelGui コマンドを使用したトンネリング
- デバッグ用構成オプションの設定
- DMS を使用したランタイム情報の表示
- Java の型の優先順位とオーバーロード・メソッドに関する SOAP の制限

## TcpTunnelGui コマンドを使用したトンネリング

SOAP には、SOAP クライアントと SOAP サーバー間でやりとりされたメッセージを表示できるように、TcpTunnelGui コマンドが用意されています。TcpTunnelGui は SOAP サーバーとは異なる TCP ポートをリスニングし、リクエストを SOAP サーバーに転送します。

次のコマンドを使用して TcpTunnelGui を起動します。

```
java org.apache.soap.util.net.TcpTunnelGui TUNNEL-PORT SOAP-HOST SOAP-PORT
```

表 A-7 に、TcpTunnelGui のコマンドライン・オプションを示します。

**表 A-7 TcpTunnelGui コマンドの引数**

| 引数 | 説明 |
|---|---|
| TUNNEL-PORT | TcpTunnelGui によってクライアントと同じホストでリスニングされるポート |
| SOAP-HOST | SOAP サーバーのホスト |
| SOAP-PORT | SOAP サーバーのポート |

たとえば、SOAP サーバーが次のように稼働中であるとします。

```
http://system1:8080/soap/servlet/soaprouter
```

次のコマンドを使用して TcpTunnelGui をポート 8082 で起動します。

```
java org.apache.soap.util.net.TcpTunnelGui 8082 system1 8080
```

クライアントをテストして SOAP の通信量を表示するには、クライアント・プログラム内で次の SOAP URL を使用します。

```
http://system1:8082/soap/servlet/soaprouter
```

## デバッグ用構成オプションの設定

SOAP リクエスト・ハンドラ・サーブレットのログ・ファイルにデバッグ情報を追加するには、ファイル soap.xml 内で severity オプションの値を変更します。このファイルは、UNIX の場合はディレクトリ $SOAP_HOME/lib、Windows の場合は %SOAP_HOME%¥lib の soap.ear ファイルに格納されています。

デバッグ・オプションを変更するには、soap.ear ファイルを展開し、ファイル soap.xml を変更します。このファイルは、UNIX の場合はディレクトリ webapps/soap/WEB-INF、Windows の場合は webapps¥soap¥WEB-INF にあります。次に、更新した soap.ear ファイルを再デプロイします。

たとえば、次の soap.xml セグメントは、デバッグを使用可能にするために severity に設定する値を示しています。

```
<!-- severity can be: error, status, or debug -->
<osc:logger class="oracle.soap.server.impl.ServletLogger">
    <osc:option name="severity" value="debug" />
</osc:logger>
```

SOAP リクエスト・ハンドラ・サーブレットを停止して再起動すると、ファイル x.log 内のデバッグ情報を表示できます。このファイルは、UNIX の場合はディレクトリ $ORACLE_HOME/Apache/logs、Windows の場合は %ORACLE_HOME%¥Apache¥x¥logs にあります。

## DMS を使用したランタイム情報の表示

Oracle Application Server Web Services には、SOAP リクエスト・ハンドラ・サーブレットの実行、Java プロバイダの実行および個々のサービスに関する情報を収集するために、DMS が装備されています。

DMS 情報には、開始から停止までの次の実行間隔が含まれます。

- SOAP リクエストとレスポンスの（プロバイダ内とサービス内の時間を含む）合計所要時間
- Java プロバイダでの（サービス内の時間を含む）合計所要時間
- サービス（soap/java-provider/*service-URI*）の合計実行時間

DMS 情報については、次のサイトを参照してください。

```
http://hostname:port/soap/servlet/Spy
```

## Java の型の優先順位とオーバーロード・メソッドに関する SOAP の制限

OracleAS SOAP では、Java の組込み（プリミティブ）タイプ、ラッパー型、組込みタイプの 1 次元配列およびラッパー型の 1 次元配列が、SOAP RPC のパラメータとしてサポートされます。

Java プロバイダによるオーバーロード・メソッドの検索時には、常に組込みタイプ・パラメータがラッパー型パラメータより優先されます。オーバーロード・メソッドの場合、どちらが優先されるかが明確でない場合は、該当するメッセージとともにエラーが戻されます。

次に例を示します。

aMethod(int) を含む Java クラスにより、同じクラスの aMethod(Integer) が非表示になります。

aMethod(int[]) を含む Java クラスにより、同じクラスの aMethod(Integer[]) が非表示になります。

SOAP RPC サービスとしてデプロイされた Java クラスは、クライアントがシグネチャ aMethod(int, Float) および aMethod(Integer, float) を含む aMethod() を起動するとエラーを戻します。この場合、オーバーロードされる aMethod() の優先順位を決定するための明確な上位メソッドがありません。

# OracleAS SOAP と Apache SOAP の違い

この項では、Apache SOAP と OracleAS SOAP の違いについて説明します。

## サービスのインストール方法の違い

OracleAS SOAP を OC4J とともに使用する場合は、サービスのインストールに関する追加の手順があります。

## オプションのプロバイダ拡張機能

OracleAS SOAP では、`org.apache.soap.util.Provider` で定義されている Apache プロバイダ・インタフェースと、`oracle.soap.server.Provider` で定義されている拡張プロバイダ・インタフェースの両方がサポートされます。

ネイティブの Apache プロバイダには、`locate()` および `invoke()` の 2 つのメソッドのみが含まれます。Oracle プロバイダ・インタフェースでは locate および invoke メソッドが結合されているため、プロバイダは `locate()` コールと `invoke()` コールの間に入力パラメータを格納する必要がありません。また、Oracle プロバイダ・インタフェースの `init()` および `destroy()` メソッドは、プロバイダがインスタンス化されるときに SOAP サーブレットによって一度のみコールされます。これにより、プロバイダはデータベース接続やネットワーク接続のオープンなど、ワンタイムの初期化を実行し、ワンタイムのクリーン・アップ・アクティビティを実行できます。

Apache プロバイダ・インタフェースを使用する場合は、1 つのデプロイメント・ディスクリプタでサービスとプロバイダのプロパティの両方が提供されます。Oracle プロバイダ・インタフェースを使用する場合、これらのプロパティはサービス・ディスクリプタ・ファイルとプロバイダ・ディスクリプタ・ファイルに分離されます。これにより、共通のプロバイダ・プロパティをサービス間で共有できます。プロバイダ・プロパティに変更があった場合に変更する必要のあるディスクリプタ・ファイルは、1 つのみです。詳細は、「デプロイ」を参照してください。

## Oracle トランスポート・ライブラリ

Oracle トランスポート・ライブラリは、SOAP クライアント用に組み込まれています。これらのライブラリを使用すると、Oracle Wallet Manager を使用して証明書を安全に保管し、HttpClient ライブラリを HTTP 接続の管理に使用できます。HttpClient ライブラリでは、発行元のサーバー以外には Cookie が適切に戻されないという Apache ネイティブ・コードのセキュリティ問題が修正されます。

## Apache EJB プロバイダの変更内容

Apache EJB プロバイダは、OC4J EJB コンテナで動作するように変更されています。また、ステートフル EJB と Entity EJB により提供されるサービスへのクライアント・インタフェースも改善されています。EJB ハンドルは、戻される URL に連結されるのではなく、接続との HttpSession の関連付けに含まれます。HTTPSession の Cookie は SOAP クライアントにより透過的に処理されるため、クライアント・サイドでの特殊なコーディングは不要です。

## ストアド・プロシージャ・プロバイダ

PL/SQL ストアド・プロシージャまたはファンクションを使用してサービスを記述できるように、特殊なプロバイダが追加されています。

## ユーティリティの拡張

`wsdl2java` および `java2wsdl` スクリプトにより、WSDL 記述からクライアント・サイド・コードを作成する作業と、Java サービスの WSDL 記述を生成する操作が簡素化されます。

## サンプル・コードの変更内容

Apache サンプルは、OracleAS SOAP および OC4J で動作するように変更されています。`com`、`calculator` および `weblogic_ejb` の各サンプルは省略されています。Oracle ストアド・プロシージャと OC4J EJB の Web サービスの使用を示す新規のサンプルが追加されています。

## SOAP ヘッダーの mustUnderstand 属性の処理

この項では、SOAP エンベロープのヘッダー・ブロック内で `mustUnderstand` 属性に対して実行されるチェックと、この属性に対する Apache SOAP および OracleAS SOAP の処理の違いについて説明します。

### mustUnderstand チェックの設定

サービスのデプロイメント・ディスクリプタ内で `mustUnderstand` 属性のチェックを使用可能にするには、`checkMustUnderstands` フラグを設定します。このフラグを `true` に設定すると、各ヘッダー・ブロック内で `mustUnderstand` 属性のチェックが実行されます。`checkMustUnderstands` フラグを `false` に設定すると、`mustUnderstand` 属性のチェックは実行されません。`checkMustUnderstands` フラグのデフォルト値は `true` です。

### mustUnderstand チェックの動作

`checkMustUnderstands` フラグを `true` に設定すると、グローバル・リクエスト・ハンドラが処理を完了してからエンベロープを適切なサービスに渡す前に、エンベロープのすべてのヘッダー・エントリがチェックされます。この時点で、`mustUnderstand` 属性が `true` または `1` に設定されているヘッダー・エントリがあると、例外がスローされます。グローバル・ハンドラを使用すると、`mustUnderstand` 属性が `true` に設定されている 1 つ以上のヘッダー・ブロックを処理できることに注意してください。

`checkMustUnderstands` フラグが `false` に設定されている場合、エンベロープのヘッダー・エントリはチェックされず、`true` または `1` に設定されている `mustUnderstand` 属性がエントリに含まれているかどうかは確認されません。サービス実装では、エンベロープの本体を処理する前に、このチェックを実行する必要があります。

### mustUnderstand に関する Apache SOAP と OracleAS SOAP の違い

`mustUnderstand` 属性の処理に関する Apache SOAP と OracleAS SOAP の違いは、次のとおりです。

1. Apache のサービス・デプロイメント・ディスクリプタと Oracle のサービス・デプロイメント・ディスクリプタには、`checkMustUnderstands` 属性を指定できます。`checkMustUnderstands` 属性のデフォルト値は、Apache の場合は `false`、OracleAS SOAP の場合は `true` です。
2. Apache SOAP では、サービス・デプロイメント・ディスクリプタに `checkMustUnderstands='true'` が含まれている場合に、`mustUnderstand='1'` または `mustUnderstand="true"` に設定されたメッセージがサーバーに着信すると、次の障害コード値とともにエラーが戻されます。

   `mustUnderstand`

   この障害コードは、名前空間で修飾されていない不適切なコードです。

OracleAS SOAP では、戻される障害コードは名前空間で修飾されており、SOAP 1.1 で次のように定義されています。

```
SOAP-ENV:MustUnderstand
```

3. Apache SOAP では、`mustUnderstand` 属性はサービス実装で処理する必要があります。OracleAS SOAP では、`mustUnderstand` 属性を SOAP ハンドラまたはサービス実装で処理できます。これは、グローバルに使用される（`mustUnderstand` が 1 に設定されている）ヘッダーを処理する場合に特に役立ちます。このようなヘッダーや機能の例には、暗号化、デジタル署名、認証、ロギングなどがあります。

# Apache Software License, Version 1.1

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。

```
====================================
```

The Apache Software License, Version 1.1

Copyright (c) 2000 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.

# B

# Web サービスのセキュリティ

Web コンテンツへのユーザー・アクセスを制御する機能、およびシステムに侵入しようとする第三者からサイトを保護する機能は非常に重要です。この付録では、Oracle Application Server UDDI Registry も含め、Oracle Application Server Web Services に関するセキュリティのアーキテクチャと構成について説明します。

この章の内容は、次のとおりです。



**関連資料：**

- 『Oracle Application Server 10g セキュリティ・ガイド』
- 『Oracle Identity Management 概要および配置プランニング・ガイド』

# Web サービスのセキュリティ

SOAP は、Oracle Application Server Web Services のメッセージ・プロトコルです。Oracle Application Server Web Services がサポートする SOAP メッセージ用のトランスポート・プロトコルは、HTTP（S）のみです。HTTP（S）に適用される Oracle Application Server のセキュリティは、Oracle Application Server Web Services に活用できます。

Oracle Application Server Web Services では、次のセキュリティ機能がサポートされます。

- 安全な接続 : SSL（HTTPS）を使用して接続を保護することによって、Web サービスを安全に起動できます。
- 認証 : Basic 認証および Digest アクセス認証を HTTP（S）のヘッダーを使用して規定できます。この方式は、認証が SSL と組み合せて指定されないかぎり安全ではありません。
- 認可 : Oracle Application Server Java Authentication and Authorization Service（JAZN）ユーザー・マネージャなどのユーザー・マネージャを使用してプリンシパルを取得することによってサポートされます。

すべての HTTP（S）トランスポート・セキュリティ機能を、すべての種類の Oracle Application Server Web Services 実装（ステートレスおよびステートフルの Java クラス、ステートレス Session Bean およびステートレス・ストアド・プロシージなど）に適用できます。また、ステートレス Session Bean を Web サービスとして公開する場合は、接続がユーザー・マネージャによって認可され、プリンシパル・オブジェクトが取得されると、Bean 上で ACL ポリシーを規定できます。

ストアド・プロシージャを Web サービスとして公開する場合は、data-sources.xml ファイル内の対応するデータ・ソースのパスワードを暗号化すると安全です。

**関連資料 :**

- 『Oracle Application Server Containers for J2EE セキュリティ・ガイド』
- 『Oracle Application Server Containers for J2EE Enterprise JavaBeans 開発者ガイド』の第 11 章「EJB アプリケーションのセキュリティの構成」

# Web サービスのセキュリティの構成

Oracle Application Server Web Services を使用するクライアント・サイド・アプリケーションを実行する場合は、クライアント・アプリケーション内でプロパティを設定して、セキュアな Web サービスにアクセスできます。表 B-1 に、使用可能なプロパティを示します。各プロパティにより、Web サービス・クライアントの証明書と他のセキュリティ情報を提供します。

Web サービスのクライアント・アプリケーションでは、Java コマンドラインで `-D` フラグを使用して、表 B-1 のセキュリティ・プロパティをシステム・プロパティとして設定できます。また、これらのプロパティをシステム・プロパティに追加して、Java プログラム内でセキュリティ・プロパティを設定する方法もあります（プロパティを追加するには `System.setProperties()` を使用します）。さらに、クライアント・サイド・スタブには、クライアント・プロキシ・スタブ内のパブリック・メソッドである `_setTranportProperties` メソッドが含まれています。このメソッドを使用すると、`Properties` 引数を指定してセキュリティ・プロパティに適切な値を設定できます。

**表 B-1 Web サービスの HTTP トランスポートのセキュリティ・プロパティ**

| プロパティ | 説明 |
|---|---|
| http.authRealm | HTTP 認証のユーザー名 / パスワードを指定するレルムを指定します。<br>このプロパティは、Basic 認証を使用するときは必須です。 |
| http.authType | HTTP の認証タイプを指定します。指定した値の大 / 小文字区別は無視されます。<br>有効な値 : basic、digest<br>値 basic は、HTTP の Basic 認証を指定します。<br>basic または digest 以外の値を指定すると、このプロパティは未設定とみなされます。 |
| http.password | HTTP 認証のパスワードを指定します。 |
| http.proxyAuthRealm | プロキシ認証のユーザー名 / パスワードを指定するレルムを指定します。 |
| http.proxyAuthType | プロキシ認証タイプを指定します。指定した値の大 / 小文字区別は無視されます。<br>有効な値 : basic、digest<br>basic または digest 以外の値を指定すると、このプロパティは未設定とみなされます。 |
| http.proxyHost | プロキシ・ホストのホスト名または IP アドレスを指定します。 |
| http.proxyPassword | HTTP のプロキシ認証パスワードを指定します。 |
| http.proxyPort | プロキシ・ポートを指定します。整数値を指定する必要があります。このプロパティが使用されるのは http.proxyHost が定義されている場合のみで、他の場合は無視されます。<br>デフォルト値 : 80 |
| http.proxyUsername | HTTP のプロキシ認証のユーザー名を指定します。 |
| http.username | HTTP 認証のユーザー名を指定します。 |
| java.protocol.handler.pkgs | java.net.URLStreamHandlerFactory 用のパッケージ接頭辞のリストを指定します。接頭辞は縦線文字（|）で区切る必要があります。<br>この値には HTTPClient を含める必要があります。<br>この値は Java プロトコル・ハンドラ・フレームワークに必須です。Oracle Application Server では定義されません。HTTPS を使用する場合は、このプロパティを設定する必要があります。HTTPS を使用する場合にこのプロパティを設定しないと、java.net.MalformedURLException がスローされます。<br>**注意 :** このプロパティは、システム・プロパティとして設定する必要があります。<br>たとえば、このプロパティを次のどちらかに設定します。<br>■ java.protocol.handler.pkgs=HTTPClient<br>■ java.protocol.handler.pkgs=sun.net.www.protocol\|HTTPClient |

**表 B-1 Web サービスの HTTP トランスポートのセキュリティ・プロパティ（続き）**

| プロパティ | 説明 |
|---|---|
| oracle.soap.transport.allowUserInteraction | allows user interaction パラメータを指定します。指定した値の大 / 小文字区別は無視されます。次のどちらかの場合に、このプロパティを true に設定すると、ユーザー名とパスワードの入力を求められます。<br>1. プロパティ http.authType、http.username または http.password を設定しておらず、HTTP サーバーから 401 HTTP ステータスが戻される場合<br>2. プロパティ http.proxyAuthType、http.proxyUsername または http.proxyPassword を設定しておらず、HTTP プロキシから 407 HTTP レスポンスが返される場合<br>有効な値 : true、false<br>true 以外の値を指定すると、false とみなされます。 |
| oracle.ssl.ciphers | 有効化する暗号スイートをコロン（:）で区切ったリスト形式で指定します。<br>デフォルト値 : Oracle SSL でサポートされているすべての暗号スイートのリスト。 |
| oracle.wallet.location | エクスポートする Oracle Wallet またはトラスト・ポイントの位置を指定します。<br>**注意 :** 次のように、URL ではなくファイル位置を指定します。<br>/etc/ORACLE/Wallets/system1/exported_wallet（UNIX）<br>d:¥oracle¥system1¥exported_wallet（Windows）<br>SSL 認証、サーバー認証または相互認証で HTTPS をトランスポートとして使用する場合は、このプロパティを設定する必要があります。 |
| oracle.wallet.password | エクスポートする Wallet のパスワードを指定します。クライアント認証、つまり相互認証で HTTPS をトランスポートとして使用する場合は、このプロパティを設定する必要があります。 |

# Oracle Application Server UDDI Registry のセキュリティ

この項の内容は、次のとおりです。

- Oracle Application Server UDDI Registry リソースの保護
- 保護された UDDI リソースの管理と規定
- Oracle Application Server セキュリティ・サービスの使用

**関連項目 :** 10-33 ページの「OracleAS UDDI Registry の管理」

## Oracle Application Server UDDI Registry リソースの保護

Oracle Application Server の UDDI リソースの保護方法は、次のとおりです。

### Oracle Application Server UDDI Registry

OracleAS UDDI Registry について、次のリソースが保護されます。

- データ - OracleAS UDDI Registry に格納されているデータへの書込みアクセスが保護されます。通常は Web サービスのメタデータです。
- ファンクション - OracleAS UDDI Registry に対する管理操作。
- パスワード - 保護なし。ユーザーのパスワードは JAZN によって保護されます。

### Oracle Application Server Content Subscription Manager アプリケーション

Oracle Application Server UDDI Content Subscription Manager アプリケーションについては、次のリソースが保護されます。

- パスワード - UDDI シンジケーション・サブスクライバ用のパスワードが保護されます。

## 保護された UDDI リソースの管理と規定

OracleAS UDDI Registry リソースの保護は、次のように管理および規定されます。

### Oracle Application Server UDDI Registry

Oracle Application Server Java Authentication and Authorization Service（JAZN）および UDDI アプリケーションによって、OracleAS UDDI Registry に格納されているデータへの書込みアクセスが管理および規定されます。JAZN では、ユーザーの ID およびセキュリティ・ロールが判断されます。データを更新する権限があるのは所有者のみです。

OracleAS UDDI Registry の管理操作についても JAZN によってアクセスが管理および規定されます。また、管理操作を提供するサーブレットも保護されます。

### Oracle Application Server Content Subscription Manager アプリケーション

アプリケーションによって、Oracle Application Server Syndication Services へのアクセスに使用される UDDI シンジケーション・サブスクリプションのパスワードが管理されます。データベースに永続的に格納されるパスワードは、データベース `DBMS_OBFUSCATION PL/SQL` パッケージによってさらに保護されます。

UDDI シンジケーション・サブスクライバのパスワードの更新は、UDDI の Web ベースのツールから実行できます。Web ベースのツールは、JAZN を使用して認証ユーザーのセキュリティ・ロールを問い合せます。パスワードの更新機能は、認証ユーザーに `uddiadmin` セキュリティ・ロールがある場合にのみ使用できます。

## Oracle Application Server セキュリティ・サービスの使用

UDDI は、OracleAS Infrastructure 10*g* のオプションにアクセスするために、JAZN のユーザー・レベルのセキュリティ機能を活用し、サーバー側とクライアント側の両方に SSL 暗号化を使用します。

# UDDI セキュリティの構成

セキュリティに関して UDDI を構成するには、次の分野について考慮します。

- Oracle Application Server UDDI Registry の構成
- UDDI Content Subscription Manager の構成
- UDDI クライアントの構成

## Oracle Application Server UDDI Registry の構成

OracleAS UDDI Registry とクライアント間の通信の機密保護を確保するには、次の手順を実行します。

1. HTTPS アクセスを提供するように Oracle HTTP Server/SSL リスナーを構成します。
2. HTTP アクセスを禁止するように OC4J を構成します。
3. UDDI レプリケーションのエンド・ポイントに対する通信が確実に認可されるには、HTTPS のクライアント証明書ベースの認証を有効にするように Oracle HTTP Server/SSL リスナーを構成します。

公開、管理、レプリケーション Wallet 管理およびサブスクリプション管理など、セキュリティが重要なすべての UDDI エンド・ポイントを構成します（通常、照会エンド・ポイントの機密保護は必要ありません）。

## UDDI Content Subscription Manager の構成

Oracle Application Server Content Subscription Manager が機能するように、UDDI シンジケーション・サブスクライバの適切なパスワードを指定する必要があります。

## UDDI クライアントの構成

UDDI クライアント・ライブラリを使用して OracleAS UDDI Registry と通信するアプリケーションを開発する場合は、Oracle Application Server Web Services のセキュリティ機能を使用して HTTP トランスポート・プロパティを構成できます。

**関連項目：** B-2 ページの「Web サービスのセキュリティの構成」

# C

# OracleAS Web Services の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がある一般的な問題およびその解決方法について説明します。内容は、次のとおりです。

- 問題および解決策
- OracleAS Web Services の問題の診断
- その他の解決策

# 問題および解決策

この項では、一般的な問題およびその解決策について説明します。内容は、次のとおりです。



## Unsupported Response Content Type エラーを受信

Web サービスを起動すると、`unsupported response content type` エラーが発生します。

### 問題

エラーの原因として、スタブまたは動的プロキシのエンド・ポイントの URI アドレスが正しくないことが考えられます。URI アドレスが不適切な場合、クライアントは SOAP ペイロードのないネイティブの HTTP レスポンスを受信し、その結果 unsupported response content type エラーとなります。

### 解決策

URI アドレスが正しいかどうかを確認します。

## JDeveloper で doc/literal を公開できない

JDeveloper リリース 9.0.3、リリース 9.0.4 およびリリース 9.0.5.1 では、Web サービスは doc/literal を公開できません。

### 問題

ステートレス Java クラスおよび JMS 宛先を使用せずに doc/literal を公開しようとしています。

### 解決策

Oracle9*i* Application Server リリース 2（9.0.3）より、Oracle Application Server Web Services では、ステートレス Java クラスおよび JMS 宛先を使用して、doc/literal 操作をサポートしています。このサポートの詳細は、次の URL で Oracle Application Server のマニュアルを参照してください。

```
http://download-west.oracle.com/docs/cd/B10464_02/web.904/b10447/
docservices.htm#sthref259
```

## Web サービスを登録できない

Web サービスをデプロイするとき、Oracle Application Server Control を使用して Web サービスを Oracle Application Server UDDI Registry に登録できません。

**問題**

サービスを登録するためのボタンが表示されていますが、使用可能になっていません。この問題は、多くの場合、Oracle Application Server Portal がインストールされていないことを示しています。Oracle Application Server UDDI Registry は J2EE サーブレット・アプリケーションで、Oracle Application Server インストーラにより Oracle Application Server Portal と同じ J2EE OC4J コンテナに自動的に配置されます。

**解決策**

Oracle Application Server Portal をインストールします。インストール後、サーブレットを最低 1 回は ping してレジストリを初期化し、Oracle Application Server Control の UDDI ユーザー・インタフェースを使用できるようにする必要があります。

## UDDI Registry の画面が表示されない

Oracle Application Server Control に Oracle Application Server UDDI Registry の画面が表示されません。

**問題**

この問題は、Oracle Application Server Portal がインストールされていないことを示しています。Oracle Application Server UDDI Registry の画面を使用できるようにするには、Oracle Application Server Portal をインストールする必要があります。これにより、Oracle Application Server UDDI 管理の前提条件である Oracle Application Server MetaData Repository が確実にインストールされます。

**解決策**

Oracle Application Server Portal をインストールします。インストール後、サーブレットを最低 1 回は ping してレジストリを初期化し、Oracle Application Server Control の UDDI ユーザー・インタフェースを使用できるようにする必要があります。

## UDDI 管理の画面が使用できない

Oracle Application Server Control の UDDI 管理の画面が使用可能になっていません。

**問題**

Oracle Application Server Control の UDDI 管理の画面が使用可能になっていません。Oracle Application Server UDDI Registry が Oracle Application Server の別のインスタンスにインストールされています。

**解決策**

Oracle Application Server Control の UDDI 管理の画面を使用できるようにするには、その画面と同じ Oracle Application Server のインスタンスに Oracle Application Server UDDI Registry をインストールする必要があります。

# OracleAS Web Services の問題の診断

Oracle Application Server は多数のコンポーネントで構成できます。Web Services に固有の問題を診断するため、Web Services に固有の診断メッセージを生成するように Oracle Application Server を構成できます。次の項では、Web Services の診断メッセージを生成するための Oracle Application Server の構成方法について説明します。



## Web Services 診断メッセージの生成

Web Services 固有の診断情報を取得するには、システム・プロパティ `ws.debug` を True に設定します。

- OC4J スタンドアロン・バージョンで ws.debug を設定するには、コマンドラインに次のように入力します。

  ```
  java Dws.debug=true ... -jar oc4j.jar
  ```

- OC4J Java の Standard または Enterprise Edition の場合、`ws.debug` は Oracle Application Server Control 内から設定できます。Web サービスがデプロイされている OC4J インスタンスの「管理」タブから、「サーバー・プロパティ」リンクにナビゲートします。「Java オプション」テキスト・フィールドに、次のように入力します。

  ```
  Dws.debug=true
  ```

診断メッセージは、リダイレクトされた出力 / エラー・ログに記録されます。このログは、Oracle Application Server Control の Log Viewer の画面で確認できます。

# その他の解決策

Oracle MetaLink（`http://metalink.oracle.com`）には、さらに多くの解決策が掲載されています。問題の解決策が見つからない場合は、サービス・リクエストを作成してください。

**関連資料：**

- 『Oracle Application Server リリース・ノート』（Oracle Technology Network（`http://www.oracle.com/technology/documentation/index.html`）で入手可能）

# 用語集

### SOAP

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の名称。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る。

> **関連項目：** SOAP 1.1 仕様の詳細は、`http://www.w3.org/TR/SOAP/` を参照。

### UDDI

Universal Description, Discovery, and Integration（UDDI）仕様は、電子的なイエロー・ページとして機能するオンライン電子レジストリである。各種ビジネス・エンティティが WSDL 定義経由で提供するサービスおよびビジネス・エンティティ自体を登録する情報構造を提供する。

> **関連項目：** Universal Description, Discovery, and Integration（UDDI）仕様の詳細は、`http://www.uddi.org` を参照。

### Web Services Description Language（WSDL）

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る。

> **関連項目：** Web Services Description Language（WSDL）フォーマットの詳細は、`http://www.w3.org/TR/wsdl` を参照。

### Web サービス（Web Service）

次の処理を行う個別のビジネス・プロセス。

- 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る。Web サービスにより、この機能が他のアプリケーションで使用可能になる。
- Web 上での他のサービスによる検索 - Web サービスは電子的なイエロー・ページに登録できるため、アプリケーションで簡単に検索できる。
- 起動 - 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、サービスを起動できる。
- レスポンス処理 - Web サービスを起動すると、サービスの起動に使用されたものと同じインターネット標準プロトコルを使用して、リクエスト側のアプリケーションに結果が戻される。

### ストアド・プロシージャ Web サービス（Stored Procedure Web Service）

ステートレスの PL/SQL ストアド・プロシージャまたはファンクションとして実装された Oracle Application Server Web Services は、**ストアド・プロシージャ Web サービス**と呼ばれる。ストアド・プロシージャ Web サービスを使用すると、Oracle データベース・サーバー上で動作する PL/SQL プロシージャおよびファンクションを、Oracle Application Server Web Services で動作するサービスとしてエクスポートできる。

### 静的 Web サービス・クライアント（Static Web Service Client）

Web サービスを使用する場合は、**静的クライアント**を開発できる。静的クライアントは、OracleAS UDDI Registry 内の Web サービスを参照せずに、そのサービスの位置を認識する。

### 動的 Web サービス・クライアント（Dynamic Web Service Client）

Web サービスを使用する場合は、**動的 Web サービス・クライアント**を開発できる。動的クライアントを使用すると、クライアントでサービスにアクセスする前に参照を実行し、OracleAS UDDI Registry 内で Web サービスの位置を検索できる。

# 索引

## A



## B



## C



## D

## E



## F



## G



## H



## I



## J



## M



## N



## O

## P

## Q



## R



## S



## T



## U

## V



## W

## X