Web サービスのプログラミング

Sun™ ONE Studio 5 プログラミングシリーズ

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No. 817-3293-10
2003 年 7 月, Revision A

原典： *Building Web Services*
Part No: 817-2324-10
Revision A

目次

図目次

はじめに

このマニュアルでは、Sun™ ONE Studio 5, Standard Edition 統合開発環境 (IDE) を使用して、Web サービスおよび Web サービスクライアントを構築し、配備する方法を説明します。

このマニュアルは、Web サービス開発者を対象に書かれていますが、Web サービスの一般的な知識を必要とするユーザーに役立つ概念も記載しています。

このマニュアルで説明しているプログラム例は、実際に作成することができます。作業環境については、以下の Web サイトにあるリリースノートを参照してください。

`http://sun.co.jp/software/sundev/jde/documentation/`

使用するプラットフォームによっては、このマニュアルに掲載している画面イメージと異なることがあります。ほとんどの手順で Sun ONE Studio 5 ソフトウェアのユーザーインタフェースを使用しますが、場合によっては、コマンド行にコマンドを入力する必要があります。その場合は、Microsoft Windows の「コマンドプロンプトウィンドウ」で次の構文を入力します。

```
c:\>cd MyWorkSpace\MyPackage
```

UNIX の場合は、次のように入力します。

```
% cd MyWorkSpace/MyPackage
```

お読みになる前に

このマニュアルを理解するには、次のような知識が必要です。

- Java™ プログラミング言語
- Enterprise JavaBeans™ の概念
- Java サーブレット構文
- HTML 構文
- XML 構文
- J2EE アプリケーションのアセンブリと配備の概念

また、次に示すような J2EE の概念に関する一般的な知識が必要です。

- Java 2 Platform, Enterprise Edition BluePrints
  `http://java.sun.com/j2ee/blueprints`
- Java 2 Platform, Enterprise Edition Specification
  `http://java.sun.com/j2ee/download.html#platformspec`
- The J2EE Tutorial
  `http://java.sun.com/j2ee/tutorial`
- Java Servlet Specification Version 2.3
  `http://java.sun.com/products/servlet/download.html#specs`
- JavaServer Pages Specification Version 1.2
  `http://java.sun.com/products/jsp/download.html#specs`

さらに、Java API for XML-Based RPC (JAX-RPC) の詳細に精通していると役立ちます。詳細は、以下を参照してください。

`http://java.sun.com/xml/jaxrpc`

以下のサイトでは、Web サービスの規格の背景に関する知識などの役に立つ情報を提供しています。

- SOAP 1.1 Specification
  `http://www.w3.org/TR/2000/NOTE-SOAP-20000508`
- Apache SOAP 2.2 Implementation of the SOAP 1.1 Specification
  `http://xml.apache.org/soap/`
- kSOAP 1.0 (a SOAP API suitable for the Java 2 Micro Edition)
  `http://www.ksoap.org/`

- WSDL 1.1 Specification
  `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`
- UDDI 2.0 Specification
  `http://www.uddi.org/specification.html`
- JAXR 1.0_01 API Specification
  `http://jcp.org/jsr/detail/93.jsp`

注 - Sun では、本マニュアルに掲載した第三者の Web サイトのご利用に関しましては責任はなく、保証するものでもありません。また、これらのサイトあるいはリソースに関する、あるいはこれらのサイト、リソースから利用可能であるコンテンツ、広告、製品、あるいは資料に関して一切の責任を負いません。Sun は、これらのサイトあるいはリソースに関する、あるいはこれらのサイトから利用可能であるコンテンツ、製品、サービスのご利用あるいは信頼によって、あるいはそれに関連して発生するいかなる損害、損失、申し立てに対する一切の責任を負いません。

内容の紹介

第 1 章では、Sun ONE Studio 5, Standard Edition IDE の Web サービスの標準規格および Web サービスの機能に関する概要を説明します。

第 2 章では、Web サービスを開発し、テストするためのワークフローの概略を示し、Web サービス開発ツールの使用方法を説明します。また、UDDI レジストリを使用して、他の開発者が Web サービスを利用できるようにする方法についても説明します。

第 3 章では、Web サービスを使用するクライアントの作成方法を説明します。また、Web サービスの UDDI レジストリを検索し、検索された Web サービスを使用するようなクライアントを作成する方法についても説明します。

第 4 章では、Web サービスとクライアントに対し SOAP メッセージハンドラを作成する方法について説明します。

第 5 章では、Web サービスのブロックの構築方法 (非推奨) の 1 つである、XML オペレーションの作成と編集について説明します。また、この作業に使用するツールについても説明します。

付録 A では、HTTP 基本認証または HTTPS/SSL 認証による暗号化を使った Web サービスアプリケーションのセキュリティ保護について説明します。

付録 B では、配備記述子の表示と編集方法について説明します。また、EJB モジュールあるいは Web サービスモジュールの配備記述子に伝播される IDE フィールドも示します。

書体と記号について

書体または記号	意味	例
AaBbCc123	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コーディング例。	.cvspass ファイルを編集します。 DIR を使用してすべてのファイルを表示します。 Search is complete.
AaBbCc123	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して表わします。	> **login** Password:
AaBbCc123 または **ゴシック**	コマンド行の可変部分。実際の名前または実際の値と置き換えてください。	削除するには **DEL** *filename* と入力します。 rm **ファイル名** と入します。
『 』	参照する書名を示します。	『Solaris ユーザーマニュアル』
「 」	参照する章、節、または、強調する語を示します。	第 6 章「データの管理」を参照してください。 これらは、「クラス」オプションと呼ばれます。
\	枠で囲まれたコード例で、テキストがページ行幅を越える場合、バックスラッシュは、継続を示します。	machinename% grep '^#define \ XV_VERSION_STRING'
➤	階層メニューのサブメニューを選択することを示します。	作成: 「返信」➤「送信者へ」

シェルプロンプトについて

シェル	プロンプト
UNIX の C シェル	`machine_name%`
UNIX の Bourne シェルと Korn シェル	`machine_name$`
スーパーユーザー (シェルの種類を問わない)	#

関連マニュアル

Sun ONE Studio 5 のマニュアルは、Acrobat Reader (PDF) ファイル、リリースノート、オンラインヘルプ、サンプルアプリケーションの `readme` ファイル、Javadoc™ 文書の形式で提供しています。

オンラインで入手可能なマニュアル

以下に紹介するマニュアルは、Sun ONE Studio 開発者リソースポータルのドキュメントサイト (http://sun.co.jp/software/sundev/jde/documentation/) および `docs.sun.com`™ (http://docs.sun.com) から入手できます。

`docs.sun.com` Web サイトでは、サンのマニュアルをインターネットを通じて閲覧、印刷、購入することができます。サイト内でマニュアルを見つけられない場合には、製品と一緒にローカルシステムまたはローカルネットワークにインストールされているマニュアルインデックスを参照してください。

- リリースノート (HTML 形式)

  Sun ONE Studio 5 の Edition ごとに用意されています。このリリースでの変更情報と技術上の注意事項を説明しています。

  - 『Sun ONE Studio 5, Standard Edition リリースノート』

- インストールガイド (PDF 形式)

  対応プラットフォームへの Sun ONE Studio 5 統合開発環境 (IDE) のインストール手順を説明しています。さらに、システム要件、アップグレード方法、アプリケーションサーバーの情報、コマンド行での操作、インストールされるサブディレクトリ、Javadoc の設定、データベースの統合、アップデートセンターの使用方法などが含まれます。

  - 『Sun ONE Studio 5, Standard Edition インストールガイド』
  - 『Sun ONE Studio 4, Mobile Edition インストールガイド』

- Sun ONE Studio 5 プログラミングシリーズ (PDF 形式)

  Sun ONE Studio 5 の各機能を使用して、優れた J2EE アプリケーションを開発するための方法を詳細に説明しています。

  - 『Web コンポーネントのプログラミング』

    JSP ページ、サーブレット、タグライブラリを使用し、クラスやファイルをサポートする Web アプリケーションを J2EE Web モジュールとして構築する方法を説明しています。

  - 『J2EE アプリケーションのプログラミング』

    EJB モジュールや Web モジュールを J2EE にアセンブルする方法を説明しています。また、J2EE アプリケーションの配備や実行についても説明しています。

  - 『Enterprise JavaBeans コンポーネントのプログラミング』

    Sun ONE Studio 5 の EJB ビルダーウィザードや、他の IDE コンポーネントを使用し、EJB コンポーネント (コンテナ管理や Bean 管理の持続性の機能を持つセッション Bean やエンティティ Bean、メッセージ駆動型 Bean) を作成する方法を説明しています。

  - 『Web サービスのプログラミング』

    Sun ONE Studio 5 IDE を使用して Web サービスを構築したり、UDDI レジストリを経由して第三者に Web サービスを利用させたり、また、ローカル Web サービスや UDDI レジストリから Web サービスクライアントを生成する方法などを説明しています。

  - 『Java DataBase Connectivity の使用』

    Sun ONE Studio 5 IDE の JDBC 生産性向上ツールを使用し、JDBC アプリケーションを作成する方法について説明しています。

- Sun ONE Studio 5 チュートリアル (PDF 形式)

  Sun ONE Studio 5, Standard Edition の主な機能の活用方法を紹介しています。

  - 『Sun ONE Studio 5 Web アプリケーションチュートリアル』

    簡単な J2EE Web アプリケーションの構築方法を順を追って解説します。

  - 『Sun ONE Studio 5 J2EE アプリケーションチュートリアル』

    EJB コンポーネントと Web サービス技術を使用したアプリケーションの構築方法を順を追って解説します。

  - 『Sun ONE Studio 4, Mobile Edition チュートリアル』

    携帯やPDA 端末などの無線機器を対象とした簡単なアプリケーションの構築方法を順を追って解説します。このアプリケーションは Java 2 Platform, Micro Edition (J2ME™ プラットフォーム) に準拠し、Mobile Information Device Profile (MIDP) と Connected, Limited Device Configuration (CLDC) を満たすものです。

  チュートリアルアプリケーションは、以下のサイトからもアクセスできます。
  `http://forte.sun.com/ffj/documentation/tutorialsandexamples.html`

オンラインヘルプ

オンラインヘルプは、Sun ONE Studio 5 IDE から参照できます。ヘルプを開くには、ヘルプキー (Windows および Linux 環境では F1 キー、Solaris オペレーティング環境では Help キー) を押すか、「ヘルプ」 -> 「内容」を選択します。ヘルプの項目と検索機能が表示されます。

プログラム例

Sun ONE Studio 5 の機能を紹介したプログラム例とチュートリアルアプリケーションを、以下の Sun ONE Studio 開発者リソースのポータルサイトからダウンロードすることができます。

`http://forte.sun.com/ffj/documentation/tutorialsandexamples.html`

このチュートリアルで使用するアプリケーションも上記サイトに収録されています。

Javadoc

Javadoc 形式のマニュアルは、Sun ONE Studio 5 の多くのモジュールに用意されており、IDE の中で参照できます。このマニュアルの使用方法については、リリースノートを参照してください。

技術サポートへの問い合わせ

製品についての技術的なご質問がございましたら、以下のサイトからお問い合わせください (このマニュアルで回答されていないものに限ります)。

`http://sun.co.jp/service/contacting`

第1章

Web サービスの概要

このマニュアルでは、Sun™ ONE Studio 5, Standard Edition 統合開発環境 (IDE) を使用して簡単なあるいは複雑な Web サービスを構築する方法を説明しています。マニュアルの内容は、IDE の使用方法について一般的な知識があることを前提に記述されています。

Web サービスの業界標準は、発展途上です。標準を実装するソフトウェアも進化しています。Web サービスの意味は、新しい技術を使いはじめようとする開発者にとって混乱しやすいものとなっています。この章では、Web サービスの概要について説明します。その知識をもとに、後続の章では IDE の Web サービス機能を使用する方法を説明します。

Web サービスとは

一般に「Web サービス」とは、標準規格に準拠した、外部からの利用が可能な、広範にわたる業界の問題を解決する分散アプリケーションコンポーネントを表します。この節では、Web サービスに関連する問題について紹介し、Web サービスの標準に関する概要を示します。

業界の問題

現代の民間企業または公的機関の基本的な問題の 1 つに、さまざまなソフトウェアおよびハードウェアプラットフォーム上で稼動する個別に開発された多様なコンピュータアプリケーションをどのように統合するかがあります。分散環境の増加によって、企業内のアプリケーションの機能のすべてまたは一部をオープンネットワーク上で別のアプリケーションから利用できることが必要とされるようになってきました。

このような状況では、Web サービスを介して提供されるアプリケーションの機能は、その企業内でだけでなく関連企業でも使用されます。さらに、そのアプリケーションの機能を一般サービスとして提供している場合には、提携関係や契約関係の有無に関係なく、外部の企業がそれらの機能を使用することもあります。

企業内のアプリケーション

従来の企業システムでは、関連する一連のアプリケーション間には、しばしばインタフェースが必要でした。例として、支払い口座、受取り口座、総勘定元帳、および部門予算を管理するソフトウェアがあります。別の例としては、人事データ、賃金台帳、利益用のソフトウェアがあります。

プログラムされたインタフェースや統合を目的とした共有データを従来の方法で使用するには、関連するアプリケーション間で論理およびデータ構造を調整したり、ハードウェアおよびソフトウェアプラットフォームの違いを処理したりすることが必要になります。この手法では、企業内であってもメンテナンスの問題が発生します。1 つのアプリケーション内に加えた変更が、次々に別のアプリケーションでの変更につながることもあります。また、異種コンピュータシステムに依存する企業や業界では、税法および法令報告要件の変更が困難でコストがかかる、という問題もあります。

企業間のアプリケーション共用

企業にとって調整が必要となるのは、処理の効率化、合法性、および外部企業や政府機関との関係といった点です。たとえば、賃金台帳システムは源泉課税や銀行への直接預金だけでなく、その他の企業外のシステムに情報を提供する機能を持つこともあります。近年、インターネットコマース、資金の電子決済、経済および社会制度に必要なその他のネットワークアクティビティが急激に成長しています。生産、供給、取引は広範囲に分散し、複雑な流通経路とその追跡が必要になっています。これらの開発により、異なるプラットフォーム間で相互運用可能であり、ネットワーク間で共用できる、変更可能なコンピュータアプリケーションの必要性が高まりました。

B2B (企業間サービス) アプリケーションの一般的な例として、多くの異なるタイプの部品を多様なベンダーに注文するソフトウェアを考えてみましょう。ここでの取引は、複数の納入業者が関わる可能性があり、決定は動的に変化する価格および納期に依存し、取引に関わる項目には技術仕様の互換性が必要になります。コンピュータベンダーおよび規格グループが Web サービスインフラストラクチャを努力して開発するのは、このような方式での取引を企業が求めるようになってきているためです。このような B2B サービスには、サービス提供側でのオンラインカタログや注文処理、利用者側でのブラウザの入手といった処理が含まれる B2C (企業対顧客) アプリケーションを作成するよりも複雑な要件が求められます。

標準ソリューション

Web サービスは、再利用可能な分散アプリケーションコンポーネントです。このような Web サービスは、EJB コンポーネントといったビジネスサービスの機能を提供し、標準のインターネットプロトコルとメッセージ形式を介して複数のアプリケーションからビジネスメソッドを使用できるようにします。

Web サービスアーキテクチャは、次のような要件を満たすものです。

- **相互運用性。**Web サービスコンポーネントを使用するアプリケーションは、サービスを実行するハードウェアおよびソフトウェアプラットフォームを認識する必要がありません。Web サービスは、標準の Web サービス技術を使用しているかぎり、さまざまな種類のクライアント (Web アプリケーション、ワイヤレスアプリケーション、およびその他のサービス) でアクセス可能です。
- **カプセル化。**Web サービスコンポーネントを使用するアプリケーションは、コンポーネントの内部プログラミングの詳細に関わる必要がありません。
- **可用性。**Web サービスの開発者は、十分な情報を含めて公開し、他の開発者が Web サービスを見つけてそれを使用できるクライアントアプリケーションコンポーネントを作成できるようにします。

Web サービステクノロジは分散アプリケーション開発をサポートします。Web サービスの外部インタフェースの説明に基づき、開発者は標準の Web サービス定義言語を使用して異なる種類のクライアントを作成することができます。このことによって、クライアントの開発者が Web サービスの内部論理に関する知識を持つ必要がなくなります。

Sun ONE Studio 5, Standard Edition IDE には、開発者がこのような要件に見合った Web サービスやクライアントを作成できるユーザーフレンドリーな環境が用意されています。

次の項では、Web サービスを定義する規格、技術、およびアーキテクチャについて説明します。

Web サービスの規格

Web サービスアーキテクチャは、SOAP、WSDL、UDDI の 3 つの関連規格に基づいています。この節では、これらの 3 種類の標準についての概要および詳細説明と仕様全体の参照先を示します。

3 種類の関連した規格は、一般的に次のように定義されています。

- SOAP (Simple Object Access Protocol) は、Web サービスが呼び出されるメカニズムとデータが返される方法を定義します。SOAP クライアントは SOAP サービス上で、メソッドの呼び出し、XML 形式でのオブジェクトのやりとりを行うことができます。

- WSDL (Web Service Definition Language) は、Web サービスの外部インタフェースを記述して開発者がサービスを使用できるクライアントを作成できるようにします。
- UDDI (Universal Discovery, Description, and Integration) は、WSDL ファイルや実行するサービスの保存場所などを含んだ Web サービスに関する情報を登録します。これを利用することで、Web サービス開発者は自分の開発したサービスを公開できます。この結果、クライアント開発者のグループがサービスにアクセスできるようになるほか、開発するクライアントアプリケーションにその機能を組み込むことができるようになります。

Web サービスの実装には、一般に XML (Extensible Markup Language) ドキュメントの HTTP 上での受け渡しが必要です。このマニュアルは、XML および HTTP の知識を持つ読者を対象としています。

注 – Web サービス用語は、技術とともに変化します。たとえば、WSDL という用語は Web Service Definition Language の頭文字をあわせて作られた頭字語ですが、名詞としても使われ、Web サービスの記述を意味します。SOAP は今でも頭字語としても使用されていますが、SOAP プロトコルは「略語」でなくなり、最近の規格草案では、頭字語として使用しないように書かれています。このマニュアルでは一般的な業界での使用方法に従っています。また、用語の意味があいまいな場合には意味を説明しています。

SOAP

SOAP (Simple Object Access Protocol) は、W3C (World Wide Web Consortium) 規格であり、XML を使用したオブジェクト受け渡し用のプロトコルを定義します。SOAP 実行システム (SOAP 規格の実装) を使用すると、クライアントが SOAP 利用可能サービス上でメソッドを呼び出して XML 形式でオブジェクトを渡すことができます。

次の概要は W3C SOAP 1.1 仕様からの抜粋です。詳細については、`http://www.w3.org/TR/2000/NOTE-SOAP-20000508` を参照してください。

「SOAP は、情報を分散環境で交換するための軽量プロトコルです。XML ベースのプロトコルで、次の 3 つで構成されます。メッセージの内容と処理方法を記述するフレームワークを定義するエンベロープ、アプリケーション定義データ型のインスタンスを表すエンコーディング規則セット、およびリモートプロシージャの呼び出しと応答を表す規則の 3 要素です。SOAP は、その他の多様なプロトコルと組み合わせて使用できますが、このマニュアルに定義されている唯一のバインディングでは、SOAP を HTTP および HTTP Extension Framework と組み合わせて使用する方法を説明しています。」

SOAP RPC メッセージのペイロードには、アプリケーションデータ (オブジェクトおよびプリミティブ型) がシリアル化され XML データとして含まれています。

注 – Sun ONE Studio 5, Standard Edition IDE 上で作成された Web サービスは、Java™ API for XML-based Remote Procedure Calls (JAX-RPC) に準拠して SOAP 仕様を実装します。

WSDL

WSDL (Web Service Description Language) は、W3C 規格の Web サービスの外部インタフェースを記述するために使用される XML ベース言語です。

WSDL ファイルへのリンクを提供することにより、他のユーザーが Web サービスを利用できるようにします。この情報によって、開発者はサービスにリモート要求を発行する SOAP クライアントを作成できるようになります。

次の概要は W3C WSDL 1.1 仕様からの抜粋です。詳細については、`http://www.w3.org/TR/2001/NOTE-wsdl-20010315` を参照してください。

「WSDL は、ネットワークサービスを、ドキュメント指向またはプロシージャ指向情報のいずれかを含むメッセージ上で動作するエンドポイントセットとして記述する XML 形式です。操作とメッセージは抽象的に記述され、エンドポイントを定義するために具象ネットワークプロトコルおよびメッセージ形式にバインドされます。関連する具象エンドポイントは抽象エンドポイント (サービス) に組み込まれます。WSDL は、エンドポイントおよびそのメッセージを、通信に使用されているメッセージ形式またはネットワークプロトコルに関わらず、記述できるように拡張できます。」

UDDI

UDDI (Universal Description, Discovery, and Integration) は、Web サービスの外部インタフェースを記述する WSDL ファイルの位置など、Web サービスに関する情報を含んだ Web レジストリ用のプロトコルです。

UDDI レジストリは「public」、「private」、または「hybrid」を使用できます。これらの記述用語は、技術的な区別ではありませんが、レジストリの範囲および使用方法が異なります。

- **public レジストリはインターネット上で広く利用できます。**レジストリの所有者は、他の開発者 (競合相手も含む) にそれぞれのサービスのレジストリ内での公開を許可します。また、一般利用者によるレジストリ内部の検索や、レジストリ内のエントリのダウンロードも許可します。
- **private レジストリは、企業内部での利用に限られています。**この場合、レジストリは企業内全体にわたるビジネスコンポーネントの共有を実現しています。

- **hybrid レジストリは企業外からも利用できますが、利用範囲が限られています。** 使用例として、提携企業や業界内で承認されている開発グループだけからのレジストリの利用を許可する、などが挙げられます。レジストリホストは、レジストリ内でサービスを公開できるユーザーとレジストリを検索できるユーザーを判別します。

Web サービスのレジストリエントリには技術情報を保存し、開発者がサービスにバインドしてメソッドを呼び出すことができるクライアントアプリケーションを作成できるようにします。「クライアント」は相対的な用語で、Web サービスへの呼び出しを発行するコンポーネントを指します。レジストリエントリはサービスを記述する WSDL ファイルの場所 (クライアントの作成に必要) および実行サービスの場所 (クライアントの実行に必要) を含んでいる必要があります。

レジストリエントリには、Web サービスおよびプロバイダに関する追加情報を含めることができます。たとえば、これからそのサービスを利用したいと思うユーザーが、サービスが有料なのかどうかや、サポートの内容の充実度を知りたいとします。

レジストリには複数の Web サービスのエントリがあり、これらは同じ外部インタフェース (同じ WSDL) を実装していても、性能や可用性など実行時特性が異なっていたり、または異なる条件で異なるベンダーによって提供されていたりすることがあります。この場合、レジストリ検索機能によって、クライアント開発者は自分が開発したアプリケーションからアクセスするのに最適な Web サービスを選択することができます。

public レジストリは、数千または数百万のサービスに成長する可能性があります。そのため、UDDI 規格はサービスの分類をサポートしています。業界、地域、製品、サービスなどのカテゴリに基づいて、さまざまな分類がすでに利用可能です。UDDI レジストリ実装により、サービスの公開およびレジストリ検索用の機能が提供されます。

考慮する点の 1 つとして、UDDI レジストリ検索を介して見つけた Web サービスを利用すると、それがそのサービスプロバイダの評価になるという点です。この問題は Web サービスそのものに関連しているわけではありません。プロバイダの信頼性は、デジタル署名の付された証明書を交換するというネットワークセキュリティ技術に関連しています。企業の何社かは、証明書の認証局として広く認知されています。これはクレジットカード会社業界など従来の技術でも同様で、米国の場合は、National Bureau of Standards といった行政機関が保証の役割を果たしていました。

Web サービステクノロジを用いることによって、こういった広く認知されている業界機関や行政機関から、複雑ではあっても標準化された技術情報の一覧を希望者に配布できるようになります。

UDDI 仕様ドキュメント、論文、記事、および参加企業のリストについては、`http://www.uddi.org` を参照してください。Sun Microsystems, Inc. も UDDI プロジェクトに参加している企業の 1 つです。

Web サービス相互運用団体

Web Services Interoperability Organization (WS-I) は、Web サービスに関する業界の連合団体です。WS-I の目的は、プラットフォーム、アプリケーション、プログラミング言語を超えた Web サービスの相互運用性を促進することにあります。相互運用性は、「プロファイル」と呼ばれる仕様に従うことで促進されます。プロファイルには、協賛するサービスが使用する厳密な規則が定義されています。プロファイルには特定のリリースレベルでの Web サービスの仕様や、仕様が定義する機能の利用方法の基準などが含まれています。WS-I プロファイルでは、SOAP、WSDL、および UDDI など下位仕様の利用を通じ、Web サービス開発者の作業の統一が図られています。

WS-I は、World Wide Web コンソーシアム (W3C) やインターネット技術特別調査委員会 (IETF: Internet Engineering Task Force) といった他の標準団体と協調しています。WS-I は、2002 年 2 月に設立され、150 を超える企業会員から構成されています。さらに、Sun Microsystems を含む複数会員で構成される理事会によって運営されています。公開文書など、WS-I についての情報は次の WS-I Web サイトを参照してください。`http://www.ws-i.org`.

Web サービスの公開と使用

図 1-1 は、UDDI レジストリを通じて、Web サービスを利用可能にする方法を示しています。この例では、P 社が Web サービス提供側で、R 社が Web サービス要求側です。また private レジストリを使用して P 社および R 社を 1 つの会社内の部門として考慮することができます。図 1-1 で示している番号は次の作業に対応しています。

1 から 3 までの作業は、開発中に発生する処理です。4 番目の作業は、配備されたクライアントが配備された Web サービスと通信する際の実行時に発生するものです。

1. P 社は Web サービスおよび外部インタフェースを記述する WSDL ファイルを作成します。P 社は Web サービスを UDDI レジストリに公開して、WSDL ファイルの場所や実行 Web サービスの場所を含めた Web サービスに関する情報を提供します。

   P 社の担当は、Web サービスを配備し、ネットワーク上で利用できるようにすることです。

2. R 社は、UDDI レジストリ内で Web サービスを検索して、P 社が作成した Web サービスを選択します。

   この際、多くの検索条件が考えられます。2 社がすでに共同で作業していて、P 社が R 社に特定の Web サービス用の UDDI レジストリ識別子を提供している場合などには、検索が詳細になります。大規模な配備プロジェクトでは、Web サービスにさまざまなバージョンが用意されていることがあり、クライアント開発者はこの中から特定のバージョンの Web サービスを検索することもあります。

UDDI レジストリでは、WSDL ファイル用の URL と実行 Web サービスの URL が提供されます。WSDL ファイルそのものは UDDI レジストリに保存されません。

3. R 社は、WSDL ファイルを取得し、Web サービスと対話処理を実行できるクライアントコンポーネントを作成します。また、R 社はクライアントをアプリケーションに組み込んで、そのアプリケーションを配備します。
4. その後、R 社はアプリケーションを実行します。実行時に、クライアントコンポーネントは Web サービスにバインドして SOAP 要求を実行し、XML データを HTTP 上で渡します。

注 – 実行時には UDDI レジストリは使用されません。これは開発用の機能です。

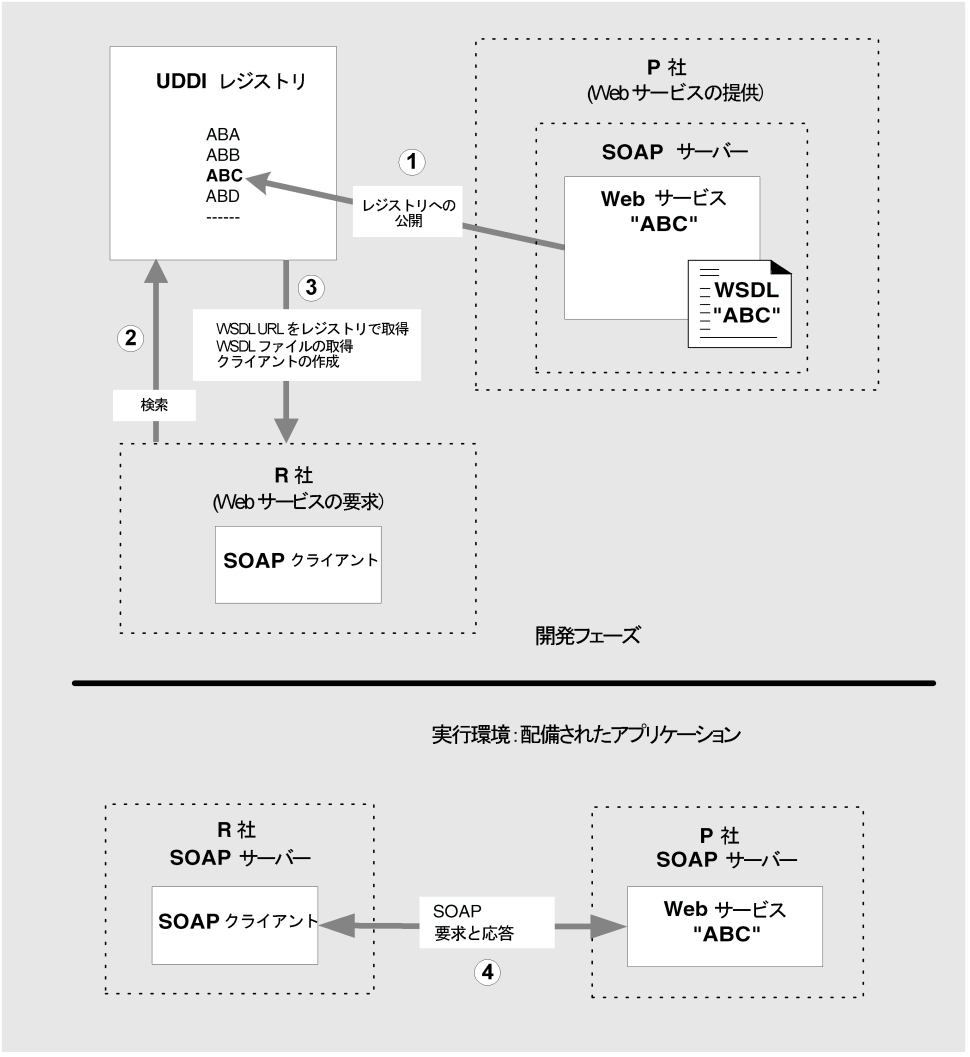

図 1-1 Web サービスの公開と使用

この作業には、次のような技術的な処理が含まれます。

- P 社は、UDDI レジストリホストからレジストリを記述する許可を取得します。P 社は名前、場所情報、およびビジネス分類情報を含むビジネスエントリを追加して、レジストリ内に自身を確立します。
- P 社がレジストリ内に Web サービス用の WSDL をポイントする URL を持つエントリを作成します。このタイプのレジストリエントリは、技術モデル (tModel) と呼ばれます。

- P 社は、レジストリ内に tModel と関連付けられた 1 つ以上のサービスエントリを作成します。各サービスエントリは、Web サービスの実行インスタンスをポイントする URL を持ちます。たとえば、P 社、ビジネスパートナー、一般ユーザー、テストで使用する 4 つの異なる実行インスタンスがあります。すべてのインスタンスは同じ機能、同じ入出力を持ちますが、パフォーマンスおよび可用特性は異なります。
- R 社は、レジストリ内の tModel およびサービスエントリからの情報を使用してクライアントを作成します。tModel は WSDL へのポインタを備え、Web サービスの外部インタフェースに関する構造的な情報を持ちます。サービスエントリは Web サービスの実行インスタンスをポイントし、サポート、パフォーマンス、および特定の実行インスタンスに対して P 社によって提供される可用性に関する追加情報を持ちます。
- R 社は、この情報を使用して指定 Web サービス用のさまざまなクライアントを作成して、カスタマイズします。

Sun ONE Studio 5, Standard Edition Web サービス

IDE は、次の主要な機能をコーディングなしでサポートします (ビジネスコンポーネントが利用可能な場合)。次に示す機能およびその他の機能については、このマニュアルの後続の章で説明しています。

- JAX-RPC Web サービスの作成
- サービスを記述する WSDL の生成
- サービス用の SOAP クライアントの生成
- 生成されたクライアントおよび JSP コンポーネントを使用したサービスの IDE 内でのテスト
- サポートされているアプリケーションサーバーへのサービスのアセンブルと配備
- UDDI レジストリへのサービスの公開
- Web サービス用の UDDI レジストリの検索
- SOAP クライアントの外部 Web サービス記述 (WSDL ファイルまたは UDDI レジストリエントリ) からの生成

今回のリリースでの新機能を次に示します。

- ステートフル Web サービスとクライアント
- セキュリティ -> HTTP 基本認証
- セキュリティ -> HTTP/SSL 認証および暗号化
- SOAP メッセージハンドラ

- 添付書類
- クライアントサポート -> WSDL taglib 生成、MIDP 拡張

Web サービスは、既存のビジネスコンポーネントのメソッドに基づいた作成 (「bottom-up」開発モデル) や、WSDL ファイルに基づいた作成 (「top-down」開発モデル) ができます。WSDL ファイルから Web サービスを作成する場合、IDE はエンタープライズ Bean のスタブも同時に生成します。このスタブは WSDL に沿ったすべてのメソッドを含んでいますが、必要なメソッドコードを追加する必要があります。これらのモデルの詳細については、第 2 章を参照してください。

Web サービスまたはクライアントを構築する際は、IDE を使ってクラスおよび他のオブジェクトの基本構造を生成できます。さらに、生成されたオブジェクトをカスタマイズするのにも IDE を使用できます。このようにして、製品アプリケーションやエンドユーザーのニーズに見合うように、生成されたクライアントをカスタマイズすることができます。

Web サービスアーキテクチャ

IDE で Web サービスを作成すると、「多層型」と「Web 主体型」の 2 つのアーキテクチャを選択できます。デフォルトは、図 1-2 に示す多層アーキテクチャです。もう一方の Web 主体アーキテクチャを図 1-3 に示します。操作手順については、18 ページの「Java メソッドからの JAX-RPC Web サービスの作成」 を参照してください。

- ビジネスロジックが EJB コンポーネント内に実装されている場合は、多層アーキテクチャの使用を推奨します。
- Web 主体アーキテクチャは、ビジネスロジックが Java クラスに実装されていて、アプリケーションに EJB コンテナが不要な場合にお勧めします。

多層アーキテクチャ

図 1-2 は、IDE で多層アーキテクチャを使用して開発した Web サービスアプリケーションの主要コンポーネントを示しています。Web サービスは、Web と J2EE アプリケーションサーバーの EJB コンテナにわたって実行される論理エントリです。

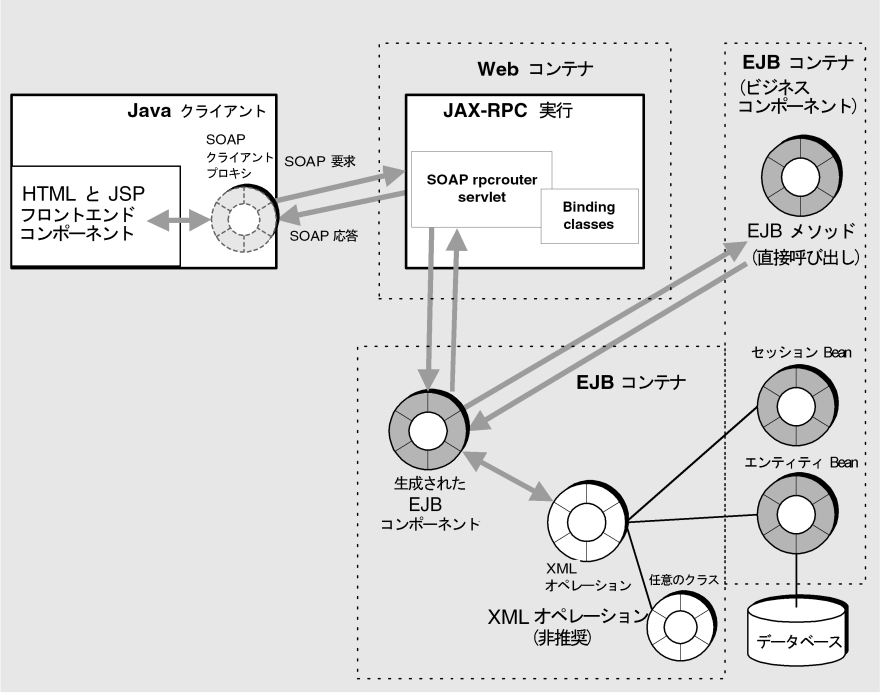

図 1-2　　Web サービスおよびクライアント (多層モデル)

クライアント側に、「SOAP クライアントプロキシ」があります。これは、サーバー上の SOAP 実行システムと通信するためにクライアント上の SOAP 実行システムを使用して SOAP 要求を送信し、SOAP 応答を受信する一連の Java クラスです。SOAP 要求は XML ラッパーで、Web サービス上のメソッド呼び出しおよび入力データをシリアル化形式で含みます。SOAP 応答は XML ラッパーで、メソッドの戻り値をシリアル化形式で含みます。

クライアント側には、SOAP クライアントプロキシへの呼び出しを行い、Web サービスからの戻り値を処理または表示する「フロントエンド」クライアントコンポーネントがあります。このコンポーネントは、HTML および JSP ページで構成されるか、Java Swing コンポーネント、別の種類のクライアント、または別のサービスを使用できます。一般的にクライアントへのアクセスに使用される HTML ブラウザは、ここでは示してありません。

サーバー側では、「実行時 JAX-RPC」(SOAP 1.1 標準の実装) は、Web コンテナ内にあり、クライアントと Web サービス間でやりとりする SOAP メッセージを処理します。

クライアントの SOAP 要求は、生成された EJB コンポーネント (IDE によって Web サービス用のインフラストラクチャとして自動作成されたセッション Bean) 上でメソッド呼び出しに変換されます。生成されたセッション Bean は、EJB コンテナ内のビジネスサービス上でメソッドを呼び出すか、(選択によっては) XML オペレーションを呼び出します。

生成された EJB コンポーネントはアプリケーションサーバーを必要とするため、多層 Web サービスを Web コンテナで実行することはできません。さらに、多層 Web サービスが呼び出せるのは、アプリケーションサーバーの EJB コンテナで実行されているビジネスコンポーネント上のメソッドだけです。

Web 主体アーキテクチャ

図 1-3 は、IDE で Web 主体アーキテクチャを使用して開発した Web サービスアプリケーションの主要コンポーネントを示しています。Web 主体アーキテクチャを選択すると、EJB セッション Bean ではなく Java クラスに基づき、IDE によってサポート構造が生成されます。このため、サポート構造が Web コンテナ内で実行できるようになります。

Web 主体 Web サービスは、Web 層ビジネスコンポーネントと EBJ ビジネスコンポーネント上のメソッドを呼び出すことができるため、多層 Web サービスよりも柔軟性があります。Web 主体の Web サービスが Web 層ビジネスコンポーネントだけを使用する場合は、アプリケーションサーバーは必要ありません。Web サーバーだけで充分です。

アプリケーションが EJB ビジネスコンポーネントだけを使用している場合は、スケーラビリティーを得るために多層アーキテクチャを選択します。ビジネス EJB コンポーネントと同じコンテナで実行される基本構造が、より効率的な形で生成されます。

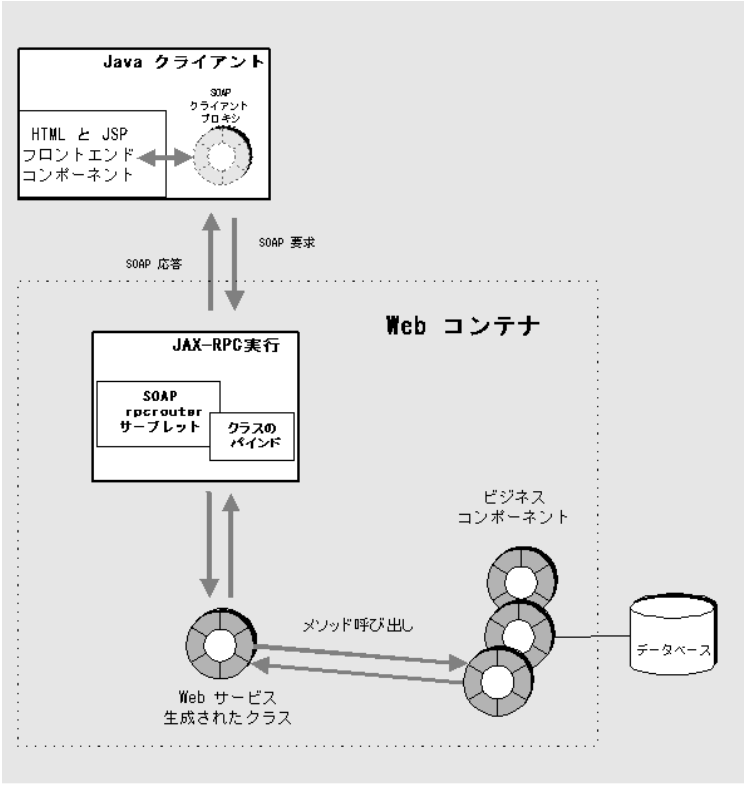

図 1-3　　Web サービスとクライアント (Web 主体モデル)

配備アーキテクチャ

配備モデルは、Web サービスアーキテクチャとビジネスコンポーネントの種類によって異なります。

- 多層アーキテクチャ用には、IDE はアプリケーションサーバーへの配備用の `.ear` ファイルを作成します。
- EJB ビジネスコンポーネントを使用する Web 主体アーキテクチャ用には、アプリケーションサーバーへ配備する `.ear` ファイルが IDE によって作成されます。
- Web ビジネスコンポーネントだけを持つ Web 主体アーキテクチャでは、IDE は Web サーバーに配備する `.war` ファイルを作成します。

アセンブリと配備についての詳細は、30 ページの「Web サービスのアセンブルと配備」 を参照してください。

ソフトウェアバージョン

Web 規格および技術は、発展途上です。Sun ONE Studio 5, Standard Edition IDE は、次に示すバージョンの各規格およびソフトウェア実装を使用します。

- SOAP 1.1 Specification
  `http://www.w3.org/TR/2000/NOTE-SOAP-20000508`
- JAX-RPC 1.0_01 Implementation of the SOAP 1.1 Specification
  `http://java.sun.com/xml/jaxrpc/`
- kSOAP 1.0 (a SOAP API suitable for the J2ME Platform)
  `http://www.ksoap.org/`
- WSDL 1.1 Specification
  `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`
- UDDI 2.0 Specification
  `http://www.uddi.org/specification.html`
- JAXR 1.0_01 API Specification
  `http://jcp.org/jsr/detail/93.jsp`
- Sun™ ONE Application Server 7, Standard Edition
  `http://sun.com/software/products/appsrvr/home_appsrvr.html`
- Java™ 2 Software Development Kit, Standard Edition (J2SE™ SDK)、バージョン 1.4.1_02 以降

注 – IDE は JAX-RPC Web サービスを生成します。IDE では、JAX-RPC クライアントまたは kSOAP クライアントのどちらかを選択できます。以前の Apache SOAP サービスまたはクライアントを使用している場合は、IDE によってこれらのサービスまたはクライアントが JAX-RPC に変換されます。

XML オペレーション (非推奨)

この機能は非推奨となっていて、IDE の今後のリリースではサポートされないことがあります。詳細は、第 5 章を参照してください。

第2章

Web サービスの構築

この章では、Web サービスの開発に使用できるツールおよび手順について説明します。Web サービスの開発には 2 通りの方法が考えられます。

- 「bottom-up」方式では、既存のエンタープライズ bean または Java クラスからメソッド参照を追加することによって、IDE を使った Web サービスを構築する方法です。
- 「top-down」方式は、WSDL ソースから Web サービスと EJB セッションの両方を IDE を使って生成します。続いて、ビジネスロジックを生成されたセッション Bean のダミーのメソッドに追加します。

この章では、bottom-up 方式と top-down 方式の両方を紹介します。

Web サービス開発タスク

次に示す複数のタスクは相互依存していますが、厳密に順序立てられているわけではありません。たとえば、Web サービスを作成しながらメソッド参照を追加することもできますし、後から追加することもできます。Web サービス開発は反復プロセスです。

この章では、次のタスクについて説明しています。

- 既存の Java メソッドからの JAX-RPC Web サービスの作成
- Web サービスへのオペレーションの追加
- 実行時オブジェクトに対する参照の解決
- 環境エントリの追加
- WSDL ソースからの JAX-RPC Web サービスの作成
- 実行時クラスの生成
- Web サービスのアセンブルと配備
- テストクライアントの作成
- Web サービスのテスト
- ステートフル Web サービスとクライアントの作成

- UDDI レジストリの使用
- 配列とコレクションの使用

Java メソッドからの JAX-RPC Web サービスの作成

この方法は、「bottom-up」方式による Web サービスの作成に分類されます。既存のビジネスコンポーネントが作業の開始点となり、Web サービスがビジネスメソッドに対するリモート呼び出しを実装するように構築されます。

Java メソッドから新しい Web サービスを作成する手順は、次のとおりです。

1. **「新規 Web サービス」ウィザードを開くには、エクスプローラで Web サービスを作成する対象となる Java パッケージを右クリックします。その後、「新規」->「すべてのテンプレート」->「Web サービス」->「Web サービス」を選択します。**

   また、IDE のメインウィンドウからウィザードを開くには、「ファイル」->「新規」->「Web サービス」->「Web サービス」を選択します。

   図 2-1 に「新規 Web サービス」ウィザードを示します。

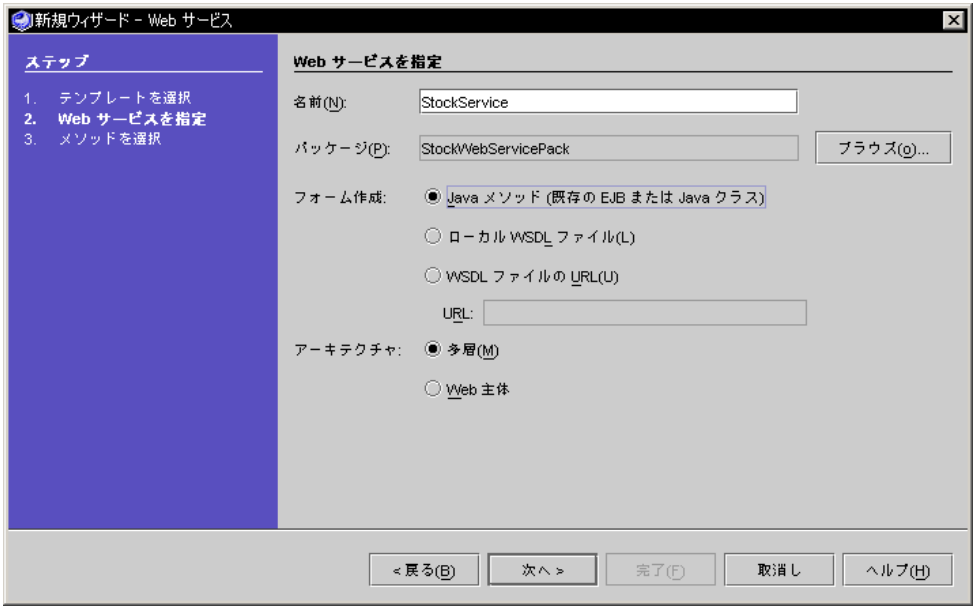

図 2-1 「新規 Web サービス」ウィザード

注 – このウィザードには、Web サービスアーキテクチャを選択できるラジオボタン「多層 (デフォルト)」と「Web 主体」があります。これらのアーキテクチャの違いについては、11 ページの「Web サービスアーキテクチャ」で説明しています。アプリケーションが EJB ビジネスコンポーネントだけを使用している場合は、両方のアーキテクチャを使用できます。ただし、ビジネスコンポーネントと同じ EJB コンテナに生成されたサポート構造が配置されるため、「多層」の方が適しています。

2. **「Java メソッドから作成」ラジオボタンが選択されていることを確認します。**

   「ローカル WSDL ファイルを作成」または「WSDL ファイルの URL から作成」でも可能です。詳しくは、25 ページの「WSDL ソースからの JAX-RPC Web サービスの作成」を参照してください。

3. **「パッケージ」フィールドに Web サービスを作成する場所が正しく指定されていることを確認します。**

   「新規 Web サービス」ウィザードを開いたときに、エクスプローラで Java パッケージが選択されている場合は、パッケージの名前がウィザードの「パッケージ」フィールドに表示されます。目的のパッケージを見つけて選択するには、ウィザードの「ブラウズ」ボタンをクリックします。

注 – Web サービスに実装するファイルの多くは IDE によって生成されるため、Web サービスを独自のパッケージ内に保存しておくと便利です。

4. **「名前」フィールドで、Web サービスの名前を入力します。**

   ウィザードで指定されているパッケージ内の既存の Web サービスの名前を入力すると、強調表示された次のようなメッセージが表示されます。「この名前の付いた Web サービスがパッケージ内に既に存在します。」

5. **「完了」をクリックして、Web サービスを作成するか、「次へ」をクリックしてメソッド参照を追加します。**

   「完了」をクリックすると、新しい Web サービスがエクスプローラに表示されます。サービスは、指定パッケージにノードとして青の球形アイコン () 付きで表示されます。

   「次へ」をクリックすると、「メソッドを選択」のダイアログが表示されます。ダイアログ内で 1 つ以上のメソッドを選択して「完了」をクリックし、Web サービスにメソッド参照を追加します。次に「完了」をクリックして Web サービスを作成します。

   また、Web サービスを作成した後からメソッド参照を追加することができます (21 ページの「Web サービスへのオペレーションの追加」を参照してください)。

6. **エクスプローラで、Web サービスを右クリックし、「プロパティ」を選択します。**

「SOAP RPC URL」プロパティは、次のデフォルト形式になっています。

`http://`*hostname*`:`*portnum*`/`*webserviceName*`/`*webserviceName*

`hostname` および `portnum` の値をサーバーインストールと一致するように設定します。URL の最初のインスタンスである *webserviceName* は「コンテキストルート」または「Web コンテキスト」と呼ばれます。この値は必要に応じて変更できますが、後の手順で作成する J2EE アプリケーション WAR ノードの Web コンテキストプロパティと一致している必要があります (32 ページの「J2EE アプリケーションのアセンブル」を参照してください)。

注 – Web サービスが複数ある場合は、各 SOAP RPC URL が固有のものであることを確認してください。

図 2-2 に SOAP RPC URL が強調表示された Web サービスのプロパティを示します。

図 2-2 コンテキストルートが `StockService` に設定されている Web サービスプロパティ

XML オペレーションの開発 (非推奨)

XML オペレーションは、複数のビジネスメソッド呼び出しを Web サービス用の単一の上位レベルメソッドに統合する手段です。単にビジネスコンポーネントの個々のメソッドを Web サービスとして利用する場合は、XML オペレーションは必要ありません。

XML オペレーションを開発する方法についての詳細は、第 5 章を参照してください。

Web サービスへのオペレーションの追加

エンタープライズ Bean や、メソッドが、Web サービスから利用されるビジネスコンポーネントとして用意されている必要があります。「オペレーション」を Web サービスに追加すると、IDE によってリモートメソッド呼び出しを発行するコードが生成されます。論理的には、オペレーションはメソッド呼び出しを実装するための生成された機構を指します。

注 – Web サービスオペレーションは、EJB と Java クラスメソッド、または XML オペレーション (非推奨) から作成されます。

Web サービスにオペレーションを追加する手順は、次のとおりです。

1. **エクスプローラ中の Web サービスを右クリックし、「新規オペレーション」を選択します。**

   ファイル選択ダイアログが表示されます。

2. **目的のビジネスメソッド (または XML オペレーション) を選択し、「了解」をクリックします。**

   複数の項目を選択するには、Ctrl キーを押しながら項目をクリックします。

   EJB Bean クラス、ホームインタフェース、またはリモートインタフェースを表すノードでなく、論理 EJB ノード (Bean アイコン  付きのノード) の EJB メソッドを選択してください。論理 EJB ノードからメソッドを追加することによって、メソッドを呼び出すために必要な実行時情報を提供できます。図 2-3 は、論理 EJB ノードの例を示しています。

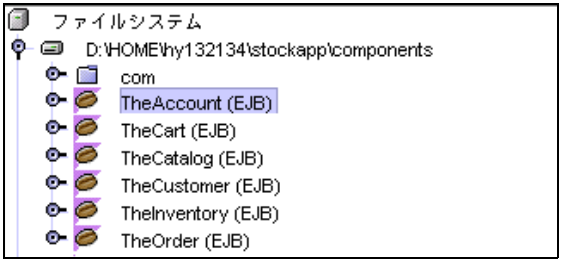

図 2-3　　エクスプローラの論理 EJB ノード

Web サービスに追加されたオペレーションは 図 2-4 に示されているように Web サービスのサブノードに表示されます。

オペレーションと XML オペレーションは次のようにまとめられています。

- EJB または Java メソッドに基づくオペレーションは「Methods」ノードの下に保存されます。
- XML オペレーションは、「XML オペレーション」ノード (非推奨) の下に保存されます。

図 2-4　　Web サービスオペレーション

IDE は、参照オブジェクトを含むパッケージの名前を持つ参照に接頭辞を付けます。XML オペレーション参照の前には矢印 (->) が付けられます。

注 – また、オペレーションは、「コピー」および「ペースト」コマンドを使用して Web サービスに追加することができます。たとえば、EJB メソッドを選択して「編集」->「コピー」を選択し、Web サービスノードを選択して「編集」->「ペースト」を選択します。

さらに、IDE を使用して Web サービスに外部ディレクトリまたはファイルを追加することもできます。この機能では、`.gif`、`.jpg`、`.xml`、`.property`、および `.jar` ファイルといった、添付書類、プロパティファイル、およびその他のリソースを利用できます。

外部ディレクトリまたはファイルを追加する手順は次のとおりです。

1. エクスプローラの Web サービスノードを右クリックし、「プロパティ」を選択します。
2. 図 2-2 に示す「外部ファイル」プロパティの値をクリックし、省略符号ボタン (...) をクリックします。
   IDE によって「外部ファイル」ダイアログが表示されます。
3. 「追加」をクリックします。
   選択ダイアログが表示されます。
4. 選択ダイアログから目的のディレクトリまたはファイルを選択し、「了解」をクリックします。その後「外部ファイル」ダイアログで「了解」をクリックします。

Web サービスからのオペレーションの削除

オペレーションを削除する手順は次のとおりです。

1. エクスプローラで、削除するオペレーションを選択します。
2. オペレーションを右クリックし、「削除」を選択します。
   オペレーションが削除されます。

実行時オブジェクトに対する参照の解決

このタスクについての詳細は、159 ページの「オブジェクトのインスタンス化と参照の解決」を参照してください。この手順は、XML オペレーションを使用している場合や、デフォルト参照で正常に動作しない場合にだけ必要になります。

環境エントリの追加

環境エントリは、Web サービスの EJB モジュール配備記述子に格納するデータです。環境エントリは、Web サービスでターゲットオブジェクトを作成または検出するメソッドに対するパラメータとして使用するために提供されています。

「オブジェクト参照を解決」ダイアログの「メソッド」、「コンストラクタ」、「Static メソッド」フィールドに指定されているメソッドでパラメータを指定できる場合は、パラメータの値を環境エントリにマップできます。環境エントリは配備記述子内に格納されているため、配備時に実行時環境に適した値を設定できます。

環境エントリを追加する手順は、次のとおりです。

1. **Web サービスノードを右クリックし、「プロパティ」を選択します。**

   IDE によって、図 2-2 に示す Web サービスプロパティが表示されます。

2. **「環境エントリ」プロパティをクリックし、表示される省略記号 (...) ボタンをクリックします。**

   図 2-5 に示す「環境エントリ」ダイアログが表示されます。

図 2-5　「環境エントリ」ダイアログ

3. **「追加」をクリックします。**

   図 2-6 に示す「追加 環境エントリ」ダイアログが表示されます。

図 2-6　「追加 環境エントリ」ダイアログ

4. **「追加 環境エントリ」ダイアログ内の各フィールドに次の項目を入力します。**
   - **名前。**環境エントリに名前を割り当てます。
   - **説明。**環境エントリの説明を入力します。
   - **型。**環境エントリをマップするメソッドパラメータの型を入力します。

- **値。**環境エントリの値を割り当てます。

5. **「了解」をクリックします。**

   環境エントリが作成されます。環境エントリは次に Web サービスの実行時クラスを生成するときに、Web サービス EJB モジュール配備記述子に伝達されます。

WSDL ソースからの JAX-RPC Web サービスの作成

この方法は、「top-down」方式による Web サービスの作成に分類されます。Web サービスの外部インタフェースを記述する WSDL ファイルが開始点となります。IDE によって、Web サービスと EJB セッション Bean のスタブとが作成されます。この EJB セッション Bean には、WSDL に記述されている仕様と一致するダミーのビジネスメソッドが含まれています。EJB セッション Bean のビジネスメソッドを実装するためのコードは開発者が記述する必要があります。

top-down 方式では、複数のインタフェースを実装する Web サービスを作成することができます。この方式は、異なるコンピュータ処理や在庫システムを利用している複数の納入業者を持つ企業などで便利です。たとえば、複数の部品納入業者を持つ自動車メーカーが Web サービスを利用したアプリケーションを作成し、各業者のシステムからこのアプリケーションを直接利用させたい場合などに使用できます。

WSDL から新規の Web サービスを作成する手順は次のとおりです。

1. **「新規 Web サービス」ウィザードを開くには、エクスプローラで Web サービスを作成する対象となる Java パッケージを右クリックします。その後、「新規」->「すべてのテンプレート」->「Web サービス」->「Web サービス」を選択します。**

   また、IDE のメインウィンドウからウィザードを開くには、「ファイル」->「新規」->「Web サービス」->「Web サービス」を選択します。

   図 2-7 に「新規 Web サービス」ウィザードを示します。

注 – エクスプローラで目的の WSDL ノードを右クリックし、「新規 Web サービスを作成」を選択することもできます。この場合は手順が簡略化されます。IDE がウィンドウを表示し、ターゲットディレクトリとパッケージの入力を促します。生成された EJB セッション bean は、Web サービスとして同じパッケージの中に格納されます。

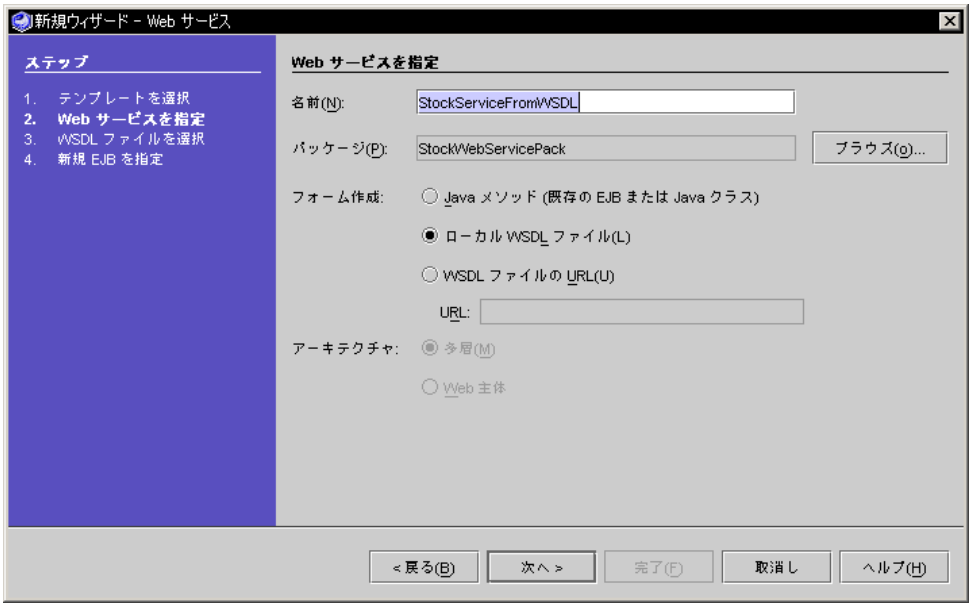

図 2-7　　「新規 Web サービス」ウィザード

2. 「ローカル WSDL ファイルから作成」ラジオボタンを選択して、「次へ」をクリックします。表示されたファイルの一覧 (図 2-8 を参照) から、目的の WSDL ファイルを選択します。

   「WSDL ファイルの URL から作成」ラジオボタンを選択して URL を入力し、「次へ」をクリックすることもできます。

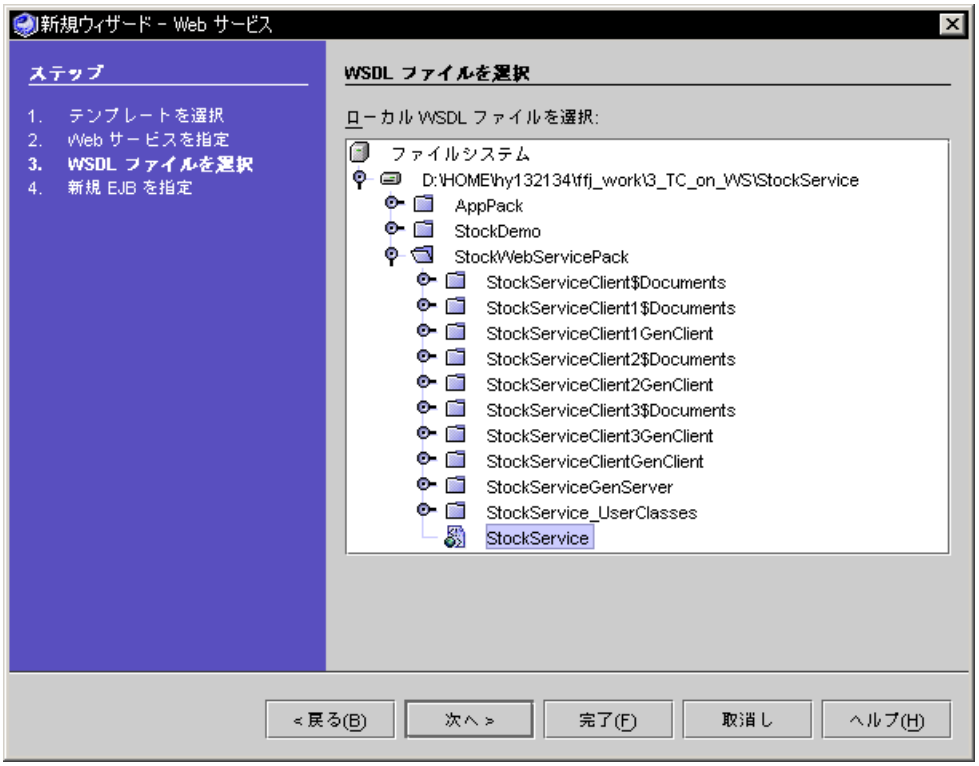

図 2-8　「新規 Web サービス」ウィザード - WSDL ファイルを選択

IDE によって、「新規 EJB を指定」ウィンドウが表示されます (図 2-9 を参照)。

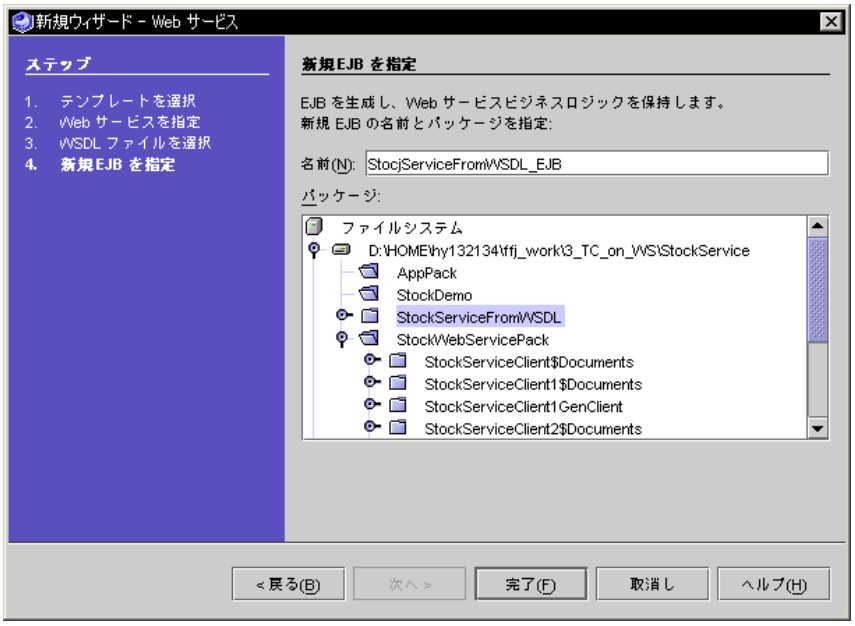

図 2-9　「新規 Web サービス」ウィザード - 新規 EJB を指定

3. 目的の EJB 名を入力し、ターゲットパッケージを選択して「完了」をクリックします。

IDE によって新しい Web サービスと新しい EJB コンポーネントが作成され、ウィザードで指定したパッケージ内に格納されます。WSDL ファイルから作成された Web サービスと EJB コンポーネントが表示されたエクスプローラを図 2-10 に示します。

注 – IDE は Web サービスに実装するファイルを大量に生成するため、EJB セッション Bean と Web サービスを個別のパッケージ内に保存しておくと便利です。

新しい Web サービスと EJB セッション Bean は、WSDL で指定したメソッドを保持します。Web サービスは、生成したセッション Bean において、このメソッドを参照します。

注 – WSDL には Web サービスビジネスロジックは含まれません。したがって、生成された EJB セッション Bean は不完全です。EJB クラスをコンパイルする前に、ビジネスメソッドの記述が必要です。

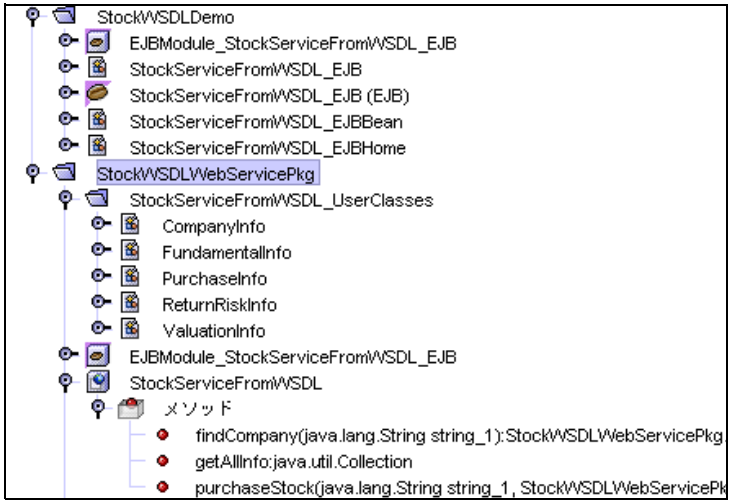

図 2-10　エクスプローラに表示された WSDL から生成された Web サービスと EJB ノード

実行時クラスの生成

Web サービスを J2EE アプリケーションとして組み立て、テスト用に配備するには、Web サービスの実行時クラスを生成する必要があります。

IDE は、生成した実行時クラスを Web サービスと同じパッケージ、つまり *xxx*`GenServer` と名付けられるノードの下に配置します。この *xxx* は Web サービスの名前を表します。生成されるクラスは、アーキテクチャが多層か Web 主体かによって異なります。たとえば、アーキテクチャが多層の場合、生成された EJB コンポーネントに対して複数の実行時クラスが生成されます。

また、XML オペレーションが存在する場合は、IDE は各 XML オペレーションに対して 1 つずつ追加クラスを生成します。詳しくは、第 5 章の XML オペレーションに関する説明を参照してください。

Web サービスの実行クラスを生成する手順は、次のとおりです。

1. **エクスプローラで Web サービスを選択します。**
2. **マウスの右ボタンをクリックして、「Web サービスファイルを生成」を選択します。**

   Web サービスの実行クラスが生成され、コンパイルされます。コンパイルが完了するとエクスプローラは図 2-11 のような表示に変わります。

図 2-11　ノードおよび参照の Web サービス階層

エクスプローラ階層の Web サービスノードと同じパッケージ内の同じ階位に WSDL ノードが表示されます。WSDL ノードは Web サービスと同じ名前を持っていますが、緑の球形 ( ) のアイコンで区別されます。

Web サービスノードの「WSDL を生成」メニューを選択することで、WSDL ファイルとノードを手動で生成することもできます。WSDL ファイルの使用方法についての詳細は、46 ページの「WSDL の生成」を参照してください。

Web サービスのアセンブルと配備

この項の説明は、Web サービスで必要な EJB モジュールが IDE で利用可能であり、Web サービスの実行クラスが生成済みであることを前提としています。

手順は Sun ONE Application Server 7 への配備を例にしています。他のアプリケーションサーバーへの配備手順もここでの手順と同様ですが、ホスト名、ポート番号、およびその他の特別な要件についてはアプリケーションサーバーの管理者に問い合わせる必要があります。

Web サービスが Web 主体アーキテクチャで EJB ビジネスメソッドを参照しない場合は、アセンブルおよび配備手順は簡略化されます。次の説明は、簡単な Web 主体ケースから始まり、J2EE アプリケーションとしてアセンブルおよび配備される Web サービス用の手順へと続きます。

Web 主体アプリケーション

Web サービスを Web 主体アーキテクチャで作成する場合、IDE は WAR ファイルだけを作成します。この場合、J2EE アプリケーションを作成する必要はありません。

Web サービスが Web 主体アーキテクチャで作成され、EJB コンポーネントを参照しない場合、Web サービスノードにはエクスプローラから実行できる「配備して実行」メニューが表示されます。

- **Web サービスノードを右クリックし、「配備」を選択します。**

  IDE によって WAR ファイルがアセンブルされ、IDE のデフォルト Web 層アプリケーションサーバーに配備されます。Web サービスのテストに関しては、35 ページの「Web サービスのテスト」を参照してください。

  「コンテキストルート」プロパティを Web サービスの「SOAP RPC URL」プロパティに一致させるためなど、デフォルトの構成を上書きする目的で WAR ファイルの編集が必要になることがあります。また、組織立っている製品環境などでは配備担当者に対して WAR ファイルを提供することがあります。このような場合、次の手順で処理することができます (詳細は状況に応じて変更してください)。

1. **Web サービスノードを右クリックし、「WAR ファイルをエクスポート」を選択します。**

   IDE によって、Web サービスと同じパッケージ内に WAR ファイルが作成され、エクスプローラに WAR ノードが表示されます。

2. **WAR ファイルを右クリックし、「Web モジュールとしてアンパック」を選択します。**

   IDE によって、WAR ファイルが選択したディレクトリに配置され、そのディレクトリが IDE ファイルシステムにマウントされます。

3. **エクスプローラで、パッケージ解除した Web モジュールに移動し、`WEB-INF` ノードを右クリックし、「プロパティ」を選択します。**

4. **「コンテキストルート」プロパティを、Web サービスの「SOAP RPC URL」プロパティと一致するように変更します (図 2-2 を参照してください)。**

   「SOAP RPC URL」にコンテキストルート値が次の形式で表示されます。
   `http://`*hostname*`:`*portnum*`/`*contextRoot*`/webservicename.`

   注 – 「コンテキストルート」プロパティ値 `WEB-INF` の前には、「*/contextRoot*」のようにスラッシュ (/) を付ける必要があります。

   ---

5. **Web モジュールを Web サーバーに配備します。**

   この方法についての詳細は、Sun ONE Studio 5 プログラミングシリーズの『Web コンポーネントのプログラミング』を参照してください。

次の項では、J2EE アプリケーションとしてアセンブルおよび配備する Web サービスについて説明しています。

J2EE アプリケーションのアセンブル

この項では EJB ビジネスコンポーネントを使用する Web サービスに関する操作を説明しています。

Web サービスおよび参照されるコンポーネントを含む J2EE アプリケーションを作成する手順は、次のとおりです。

1. **J2EE アプリケーションを作成するパッケージを右クリックし、「新規」->「すべてのテンプレート」->「J2EE」->「アプリケーション」を選択します。**

   新規アプリケーションウィザードが表示されます。

2. **「名前」フィールドにアプリケーションの名前を入力し、「完了」をクリックします。**

   J2EE アプリケーションノードがパッケージに追加されます。

3. **アプリケーションノードを右クリックし、「モジュールの追加」を選択します。**

   「アプリケーションにモジュールを追加」ダイアログが表示されます。

4. **Web サービスおよび Web サービスによって参照されるすべての EJB モジュールを選択して、「了解」をクリックします。**

   図 2-12 に、次の下位ノードを持つ J2EE アプリケーションノードを示します。

   - *webserviceName*_EjbJar と名付けられた EJB JAR ノード。*webserviceName* には、Web サービスの名前が表示されます。このノードは、生成された Web サービスの EJB モジュールに対応します。
   - *webserviceName*_War と名付けられた WAR ノード。*webserviceName* には、Web サービスの名前が表示されます。このノードは、生成された Web サービスのWeb モジュールに対応します。
   - アプリケーションに追加した各 EJB モジュール用の EJB JAR ノード。

図 2-12 Web サービス用 J2EE アプリケーション

5. **必要に応じて Web サービスの WAR ノードの 「Web コンテキスト」プロパティを編集します。**

   ほとんどの場合、Web サービス WAR ファイルと Web サービスそのもののデフォルトの「Web コンテキスト」プロパティ (Web サービスの名前) は IDE によって設定されるため、この手順は不要です。このプロパティの値は、Web サービスにアクセス

するための URL の一部となります。WAR ファイル値は、Web サービスの「SOAP RPC URL」プロパティに組み込まれている「コンテキストルート」値に一致する必要があります (図 2-2 を参照)。

デフォルト設定を編集する手順は、次のとおりです。

a. Web サービスの WAR ノードを右クリックし、「プロパティ」を選択します。

b. 「Web コンテキスト」プロパティをクリックし、新しい値を入力して Enter キーを押します。

図 2-13 は、Web サービス WAR ファイルのプロパティを示しています。この WAR ファイルの「Web コンテキスト」値は `StockService` となっていて、Web サービスの「SOAP RPC URL」プロパティのコンテキストルート値と一致しています。

図 2-13　Web コンテキストを `StockService` に設定した Web サービス WAR ファイルプロパティ

6. 必要に応じて J2EE アプリケーションの「アプリケーションサーバー」プロパティを編集します。

このプロパティでは、配備用のアプリケーションサーバーインスタンスを指定する必要があります。プロパティは、IDE のデフォルトアプリケーションサーバーにデフォルト設定されています。この値は「エクスプローラ」ウィンドウの「実行」タブで設定できます。

プロパティを編集する手順は次のとおりです。

a. J2EE アプリケーションノードを右クリックし、「プロパティ」を選択します。

b. 「アプリケーションサーバー」プロパティ値をクリックし、省略記号 (...) ボタンをクリックします。

図 2-14 に示すダイアログが表示されます。

c. 表示されているアプリケーションサーバーインスタンスのどれかを選択します。

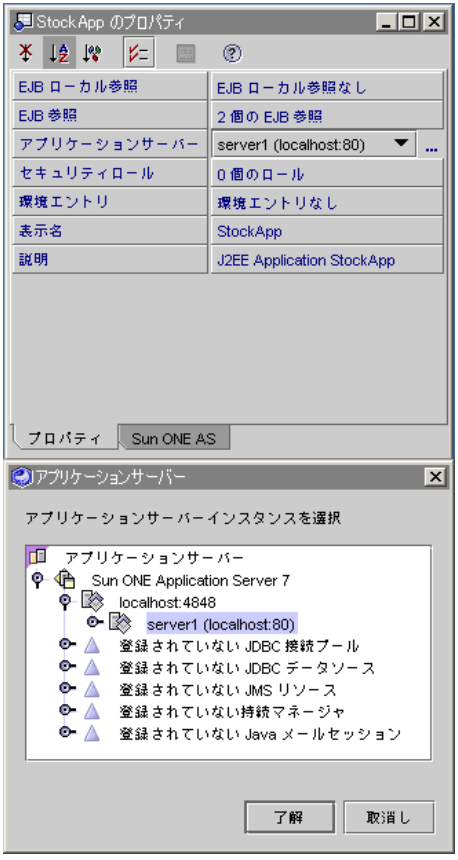

図 2-14　配備用のアプリケーションサーバーインスタンス

J2EE アプリケーションの配備

Web サービスアプリケーションをアプリケーションサーバーに配備する手順は、次のとおりです。

1. **J2EE アプリケーションノードを右クリックし、「配備」を選択します。**

   この操作により、アプリケーションのパッケージ処理が開始されます。処理が完了すると、アプリケーションはサーバーに配備されます。

2. **パッケージ処理および配備のステータスメッセージを確認します。**

   この処理には数分間かかります。IDE ステータス行には、パッケージの進捗状況を示すメッセージが表示されます。

   配備が完了すると、Web サービスがテストできる状態になります。

Web サービスのテスト

IDE では、次のソースからの Web サービスクライアントの作成方法をわかりやすくサポートしています。詳細は、第 3 章を参照してください。

- ローカル IDE Web サービス
- WSDL ファイル
- UDDI レジストリで公開されている Web サービス

操作については、67 ページの「ローカル Web サービスからのクライアントの作成」を参照してください。「デフォルトテストクライアント」を持つ Web サービスのテストといった、重要な特殊操作についても記載されています。

Web サービス用のデフォルトテストクライアントの設定

IDE を使用して Web サービスからクライアントを生成する場合、「新規クライアント」ダイアログにクライアントをサービス用のデフォルトのテストクライアントに設定できるチェックボックスが表示されます。

デフォルトテストクライアントを変更する手順は次のとおりです。

- エクスプローラの Web サービスノードを右クリックし、「 テストクライアント」プロパティを編集します。

注 – Web サービスが J2EE アプリケーションにアセンブルされるときに、IDE によってデフォルトのテストクライアントが Web サービス WAR ファイルに自動的にパッケージ化されます。

Web サービスのテスト

デフォルトのテストクライアントを使用して Web サービスをテストできます。Web サービスのアーキテクチャによって、2 つの手順があります。

J2EE アプリケーションでのテスト

Web サービスが多層アーキテクチャの場合、または Web 主体アーキテクチャで EJB ビジネスメソッドを使用している場合の手順は次のとおりです。

1. **クライアントを生成して、Web サービスのデフォルトのテストクライアントに設定します。**
2. **Web サービスを J2EE アプリケーションにアセンブルします。**

   操作については、32 ページの「J2EE アプリケーションのアセンブル」を参照してください。

3. アプリケーションノードを右クリックして「実行」を選択します。または、テストクライアントノードを右クリックし、「実行」を選択します。

   IDE は Web サービスアプリケーションを配備して、そのアプリケーションを J2EE アプリケーション内で指定されている J2EE アプリケーションサーバー内で実行します。その後、デフォルトのテストクライアントを実行します。Web サービスがデフォルトのテストクライアントを持たない場合は、IDE によってテストクライアントを作成するかどうかが確認されます。

   または、Web サービスアプリケーションを配備してからテストクライアントノードを右クリックし、「実行」を選択することもできます。IDE はテストクライアントのアセンブル後、そのクライアントの WAR ファイルを IDE のデフォルト Tomcat Web コンテナに配備し、クライアントを実行します。

Web 主体アプリケーションのテスト

Web サービスが Web 主体アーキテクチャで EJB ビジネスメソッドを使用していない場合は、J2EE アプリケーションはありません。この場合は、次の手順でテストを実行します。

1. クライアントを生成して、Web サービスのデフォルトのテストクライアントに設定します。
2. Web サービスの「SOAP RPC URL」プロパティが Web サーバーのポートを参照していることを確認します。
   - 「SOAP RPC URL」プロパティを表示または変更するには、エクスプローラの Web サービスノードを右クリックして「プロパティ」を選択します。
   - サーバーのポート番号を調べるには、エクスプローラの「実行時」タブから「インストールされているサーバー」ノードを展開します。
3. Web サービスノードを右クリックし、「配備」を選択します。

   IDE によってアセンブルされた WAR ファイルがデフォルトの Web コンテナに配備され、Web サービスを実行します。
4. Web サービスノードを右クリックして、「実行」を選択します。

   IDE によってデフォルトのテストクライアントが開始され、ブラウザに開始ページが表示されます。

テストの反復

テストは反復プロセスです。Web サービスを編集して実行時クラスおよびクライアントを再生成し、条件を満たすまでテストします。

ステートフル Web サービスの作成

IDE では、アプリケーション内部でステートフル Web サービスがステートフルクライアントを持つセッションを維持し、そのセッション内で一連の「対話処理」が実行できるようなアプリケーションを作成することができます。

ここでの対話処理とは、単一セッション内で実行される一連の Web サービスオペレーションです。対話処理は、「対話イニシエータ」としてマークされているメソッドが実行されると開始されます。対話処理は「対話ターミネータ」メソッドが実行されると終了します。その対話処理のバインド内で実行されるその他のメソッドは、そのセッションのメンバーであり、セッションデータを共有します。

対話コンポーネントは、対話イニシエータメソッドが最初に実行され、その他のメソッドがその後に実行されるように設計してください。対話イニシエータメソッドを使用して、セッション用の資源を作成することもできます。対話ターミネータメソッドは、資源漏れを防ぐためにセッションの処理中で解放されなかった資源をすべて解放します。

1 つのステートフル Web サービスに対する同じクライアントからの複数の要求は、そのビジネスコンポーネントの同じインスタンスを介して発行される必要があります。クライアントは、複数の Web サービスメソッドに対して複数の呼び出しを実行することができ、データは同一セッション内で保持されます。

多層アーキテクチャを持つ Web サービスでは、ビジネスコンポーネントはステートフル EJB セッション Bean です。Web 主体アーキテクチャを持つ Web サービスでは、ビジネスコンポーネントが Java オブジェクトになっていることもあります。

この節で説明する手順は、ビジネスコンポーネントがステートフル EJB セッション Bean で作成されていることを前提としています。

ビジネスコンポーネントの設定

セッション Bean をステートフルにするには次の 2 つの方法があります。

Bean の作成時に次の操作を実行します。

- 図 2-15 に示すように、ウィザードで「ステートフル」ラジオボタンをクリックします。

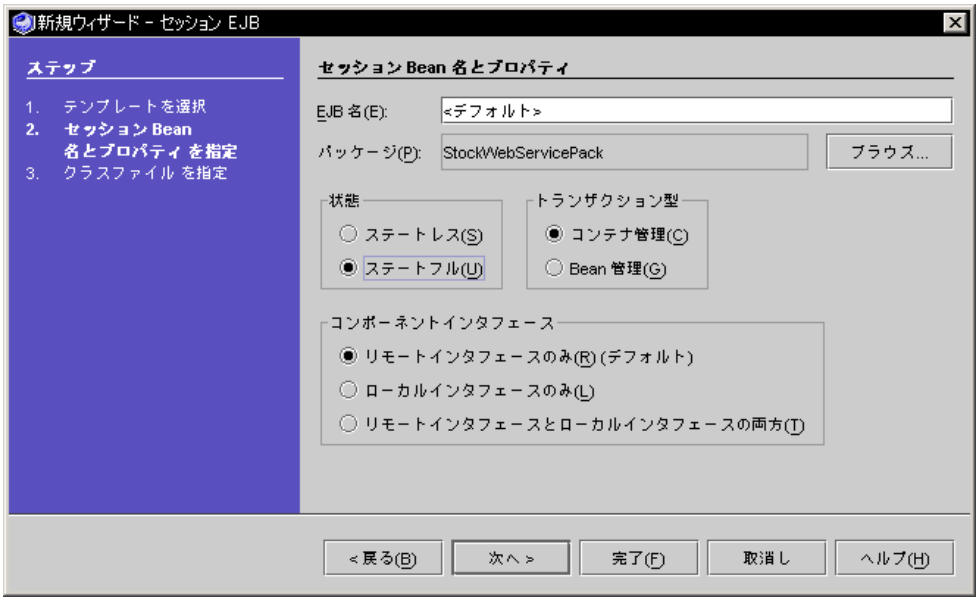

図 2-15　　ウィザードに表示されたステートフルセッション Bean

Bean がすでにある場合は次のように実行します。

1. 論理 EJB ノードを右クリックし、「プロパティ」を選択します。
2. 図 2-16 に示すように、「Bean 型」プロパティを ステートフル に設定します。

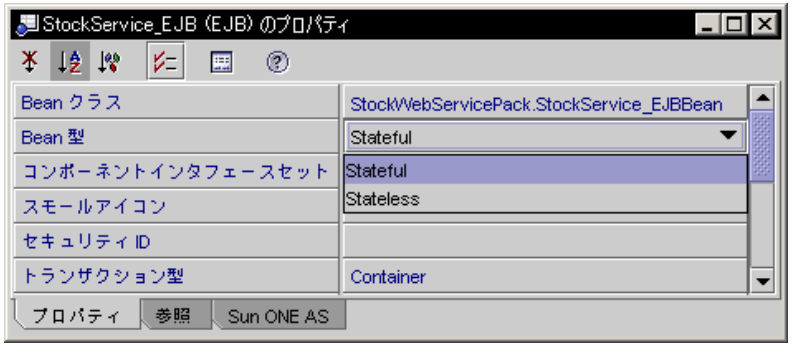

図 2-16　　ステートフルセッション Bean - Bean 型プロパティ

Web サービスクライアントが対話処理の開始と終了に使用するセッション Bean 上のビジネスメソッドは、開発者が指定する必要があります。対話処理に必要な資源を作成し、対話処理を開始するメソッドと、不要になった資源を廃棄し、対話処理を終了するメソッドを別に用意しておくと便利です。

注 – メソッドには任意の名前を付けられます。この項の例では、メソッド名として `startOrder` と `submitOrder` を使用しています。

次に、対話処理を開始、終了する 2 つのメソッドを作成する手順を示します。

- **セッション Bean クラスを変更して、`startOrder` と `submitOrder` というメソッドを追加します。**

  IDE での操作手順は次のとおりです。

1. **図 2-17 に示すように、エクスプローラで論理 EJB ノードを展開します。**

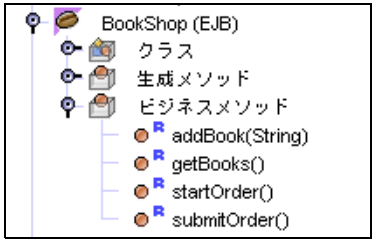

図 2-17　エクスプローラに表示したビジネスメソッド

2. **ビジネスメソッドノードを右クリックし、「ビジネスメソッドを追加」を選択します。**

   「ビジネスメソッドを追加」ダイアログが表示されます (図 2-18 を参照)。

図 2-18　「新規ビジネスメソッドを追加」ダイアログ

3. **名前の値を入力し、「了解」をクリックしてデフォルトプロパティを持つメソッドを追加します。**

   ここで挙げているメソッド名は例です。メソッドには任意の名前を付けることができます。

   この手順を実行すると、セッション Bean には次の 2 つのメソッドが作成されます。

   ```
   public void startOrder() {}
   public void submitOrder() {}
   ```

   各対話処理の目的は、本の注文を作成、発行するというものです。`startOrder` メソッドは、顧客情報 (クレジットカード情報を含む) が関連付けられたショッピングカートオブジェクトと、本のオブジェクトを入れる Vector を初期化します。`submitOrder` メソッドは、購入要求を発行することで顧客情報と注文する本のデータを渡し、ショッピングカートオブジェクトを解放してから対話処理を終了します。

Web サービスとテストクライアントの設定

ここでの例では、Web サービスがすでに作成されていて、対話イニシエータメソッドと対話ターミネータメソッドを含むビジネスメソッドも追加済みであると仮定します。実際の操作でメソッド名が異なっている場合は、適宜変更してください。

Web サービスを対話式にするには、Web サービスのプロパティと、対話処理を開始・終了するメソッドのプロパティを設定する必要があります。

1. Web サービスノードを右クリックし、「プロパティ」を選択します。
2. 対話式プロパティの値を `True` に設定します (図 2-19 を参照)。

図 2-19　Web サービス対話式のプロパティ

対話処理を開始・終了するメソッドのプロパティを設定します。

1. エクスプローラで Web サービスノードを展開し、次にメソッドノードを展開します。`startOrder` メソッドを右クリックし、「プロパティ」を選択します。
2. 「対話イニシエータ」プロパティを `True` に設定します (図 2-20 を参照)。

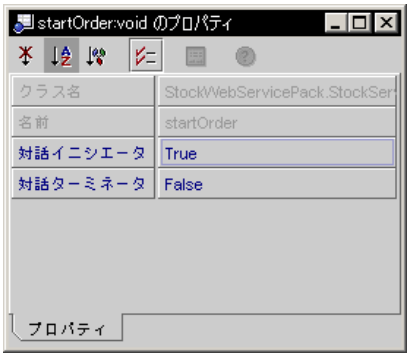

図 2-20　Web サービスの対話イニシエータメソッドおよび対話ターミネータメソッドのプロパティ

「対話ターミネータ」プロパティの値が `False` になっていることに注意してください (デフォルト)。

3. **エクスプローラで Web サービスノードを展開し、次にメソッドノードを展開します。`submitOrder` メソッドを右クリックし、「プロパティ」を選択します。**
4. **「対話ターミネータ」プロパティを `True` に設定します。**

   「対話イニシエータ」プロパティの値が `False` になっていることに注意してください (デフォルト)。

   この時点で、35 ページの「Web サービスのテスト」に説明されているように、Web サービスを生成し、テストクライアントを作成することができます。

   IDE は Web サービスと同じ対話式プロパティ値でテストクライアントを作成します。既存のテストクライアントを使用している場合は、対話式プロパティ値が `True` になっていることを確認してください。

   対話式プロパティを設定する手順は次のとおりです。

1. **テストクライアントノードを右クリックし、「プロパティ」を選択します。**
2. **対話式プロパティの値を `True` に設定します (図 2-21 を参照)。**

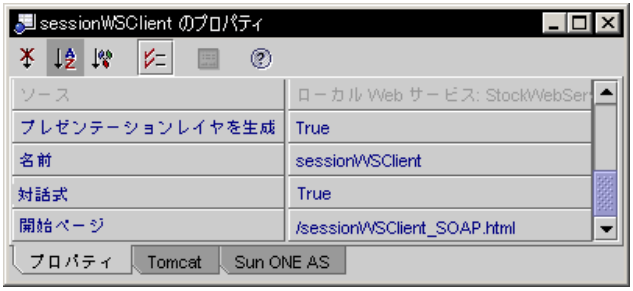

図 2-21　Web サービステストクライアント対話式のプロパティ

テストクライアントを IDE で生成すると、次のものも同時に作成されます。

- 対話イニシエータメソッドと対話ターミネータメソッドを起動する JSP タグ
- 対話イニシエータメソッドと対話ターミネータメソッドを含む、Web サービスのメソッドに対する「呼び出し」ボタンを持つクライアント開始ページ用のHTML

HTML 開始ページと JSP ノードを参照する手順は次のとおりです。

1. **エクスプローラでテストクライアントノードを展開してから、「生成されたドキュメント」ノードを展開します。**
2. **ノードを右クリックし、「開く」を選択します。**

IDE が生成したテストクライアントについての詳細は、第 3 章を参照してください。

注 – WSDL 標準ではステートフルかどうかは指定されません。したがって、UDDI レジストリまたは WSDL からクライアントを生成した場合は、クライアントの「対話式」プロパティは、デフォルト値である `False` に設定されます。Web サービスがステートフルであることがわかっている場合は、「対話式」プロパティを手動で `True` に設定できます。

ステートフル Web サービスのテスト

ここでの例は、Web サービスとテストクライアントが生成済みで、アプリケーションをアセンブル、配備してあることを前提としています。(30 ページの「Web サービスのアセンブルと配備」と 35 ページの「Web サービスのテスト」を参照してください。)

アプリケーションをテストクライアントと共に実行する手順については 35 ページの「Web サービスのテスト」を参照してください。

アプリケーションを実行すると、開始画面が表示されます。対話イニシエータメソッドおよび対話ターミネータメソッドを含む、Web サービスのビジネスメソッド用の「呼び出し」ボタンが表示されます (図 2-22 を参照)。

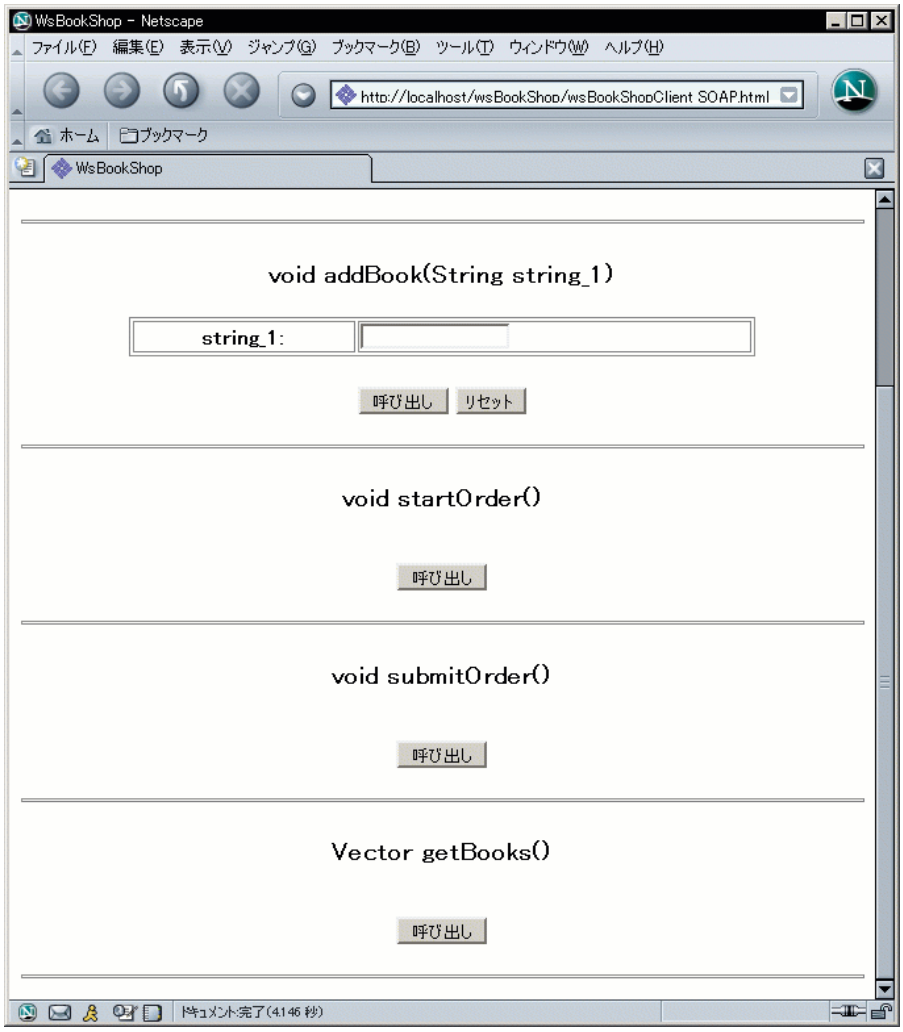

図 2-22　　ステートフルテストクライアントの開始画面

アプリケーションのステートフルの状態をテストする手順は、使用されているビジネスメソッドの目的によって異なります。図 2-22 に示した例では、ステートフルの状態を次のようにテストできます。

1. `startOrder` メソッドの「呼び出し」ボタンをクリックします。
2. `addBook` メソッドを使用して複数の本を追加し、追加する度に `getBooks` メソッドで新しい本が Vector に付加されたかどうか確認します。
3. `submitOrder` メソッドの「呼び出し」ボタンをクリックします。

今度は同じ手順を `startOrder` メソッドを使わないで実行します。この場合、追加した本は表示されません。

注 – 対話ターミネータメソッドの起動ボタンをエンドユーザーがクリックすると、対話処理の終了時に資源を解放することができます。しかし、HTML と JSP ページに基づいて IDE が生成した簡単なテストクライアントでは、エンドユーザーにこの操作を毎回確実に実行させる手段はありません。ただし、製品利用を目的として設計された堅固なクライアントであれば、エンドユーザーがどのような方法でアプリケーションを終了してもクリーンアップ処理を強制的に実行することができます。

UDDI レジストリの使用

この節では、IDE を使用して UDDI レジストリを介して Web サービスを他の開発者が利用できるようにする方法を説明します。UDDI レジストリに Web サービスを公開する前に、次の処理を完了させておく必要があります。

- 他の開発者およびエンドユーザーがネットワーク上でアクセス可能なアプリケーションサーバーまたは Web サーバーに実行する Web サービスを配備します。
- Web サービスから WSDL を生成して、他の開発者がネットワーク上でアクセス可能な Web サーバーに公開します。

  Web サービスを配備すると、IDE によって WSDL が自動的に公開されます。
- IDE のレジストリオプションを構成します。

  この処理によって、デフォルト値が決定します。これで UDDI レジストリに Web サービスを公開する際に入力する必要のある情報量を大量に減らすことができます。

この節では、次の項目について説明します。

- WSDL の生成
- UDDI レジストリオプションの管理
- 外部 UDDI レジストリへのアクセス
- UDDI レジストリへの Web サービスの公開
- 内部 UDDI レジストリの使用

注 – クライアントの開発については、第 3 章で説明します。UDDI レジストリ内で Web サービスを見つける方法および Web サービスを使用できるクライアントを生成する方法については、90 ページの「UDDI レジストリからのクライアントの作成」を参照してください。

WSDL の生成

Web サービス用の実行クラスを生成すると、IDE は自動的に WSDL ファイルを作成し、エクスプローラ上にノードとして表示します。また、次の手順で WSDL ファイルを生成し、表示することもできます。

1. **エクスプローラで、Web サービスノードを右クリックし、「WSDLを生成」を選択します。**

   エクスプローラ階層の Web サービスノードと同じパッケージ内の同じ階位に WSDL ノードが表示されます。WSDL ノードは Web サービスと同じ名前を持っていますが、緑の球形 () のアイコンで区別されます。

2. **エクスプローラで、WSDL ノードを右クリックして「開く」を選択します。**

   「ソースエディタ」ウィンドウは読み取り専用モードで表示されます。WSDL は XML ドキュメントであり、読み取りとコピーができます。WSDL ファイルはディレクトリ階層でも見つけることができます。Web サービスパッケージのディレクトリの下にあり、ファイル名は *webserviceName*`.wsdl` になります。

Web サービスの生成と配備、および UDDI レジストリへの Web サービスの公開に IDE を使用している場合は、WSDL は IDE によって自動的に管理されます。

注 – WSDL ファイルは、UDDI レジストリを使用しなくても他の開発者と共有することができます。共有するには、共有ドライブに配置したり、電子メールの添付ファイルとして送信する方法があります。UDDI レジストリを使用することなく WSDL からクライアントを生成する方法についての詳細は、89 ページの「WSDL ファイルからのクライアントの作成」を参照してください。

UDDI レジストリオプションの管理

IDE は既知の UDDI レジストリのリストおよび各レジストリにアクセスするために必要な情報 (URL、ユーザー ID、およびパスワードなど) を保持します。また、Web サービスクライアントを作成するとき、または Web サービスをレジストリに公開するときに使用されるデフォルト情報 (ビジネス、カテゴリ、および識別子など) も保持します。

注 – IDE ウィザードを使用して UDDI レジストリにアクセスする前に、レジストリ情報のデフォルト値を設定します。デフォルト値を設定することによって、ウィザードで値を繰り返し入力する手間を省くことができます。

UDDI レジストリオプションを管理する手順は次のとおりです。

- IDE のメインウィンドウから「ツール」->「オプション」->「分散アプリケーションサポート」->「Web サービス設定」->「UDDI レジストリ」を選択します。

  図 2-23 に示されている「オプション」ダイアログが表示されます。

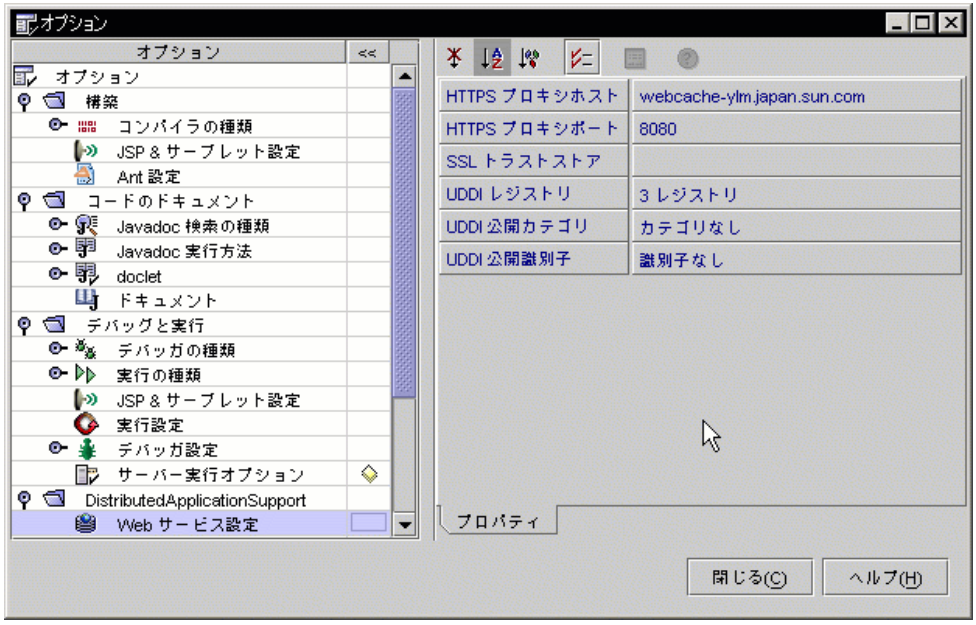

図 2-23　　UDDI レジストリのオプション

デフォルトの公開カテゴリおよび識別子の設定

Web サービスに関連付けられている公開カテゴリおよび公開識別子は、Web サービスを公開するときに UDDI レジストリに保存されます。

識別子はビジネスまたはテクニカルモデル (tModel) に固有のデータ要素です。カテゴリはビジネス、サービスまたは tModel を分類するデータ要素です。カテゴリまたは識別子をユーザークエリに指定して、ビジネス、サービスまたはレジストリ内の tModel を見つけることができます。

デフォルトの公開カテゴリを設定する手順は次のとおりです。

1. UDDI レジストリのオプションダイアログから「公開カテゴリ値」をクリックします (図 2-23 を参照)。
2. 表示される省略符号ボタン (...) をクリックします。

   図 2-24 に示す「公開カテゴリ」ダイアログが表示されます。

図 2-24　「UDDI 公開カテゴリ」ダイアログ

デフォルトの公開カテゴリは次のように編集できます。

- 「サービスに使用」チェックボックスまたは「tModel に使用」チェックボックスをクリックして、レジストリに公開するときのカテゴリをサービスまたは tModels のデフォルトとして設定します。
- カテゴリを削除するには、目的のカテゴリを選択してから「削除」をクリックします。
- カテゴリを追加する場合は、「追加」をクリックします。

  図 2-25 に示すように「UDDI カテゴリ (分類)」選択ダイアログが表示されます。カテゴリは業界で定められた標準です。

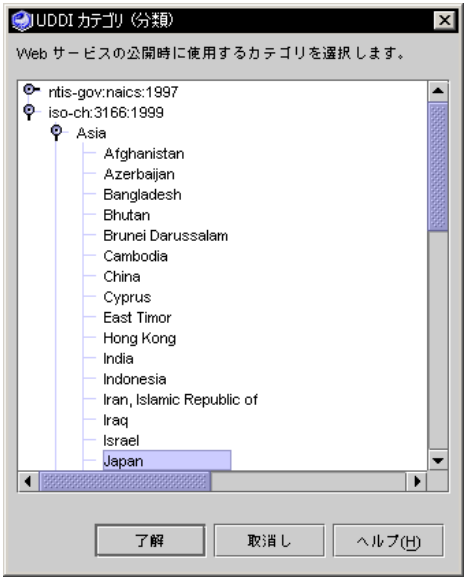

図 2-25　UDDI カテゴリ (分類)

- ノードを展開して、目的のカテゴリを選択し、「了解」をクリックします。複数の項目を選択するには、Ctrl キーを押しながら項目をクリックします。連続する複数の項目を選択するには、Shift キーを押しながら項目をクリックします。

  選択したカテゴリは「公開カテゴリ」ダイアログに表示されます。

注 – Web サービスを UDDI レジストリに公開するとき、IDE では tModel に関連付けられる隠しカテゴリを自動的に追加します。カテゴリの名前は `uddi-org:types`、値は `wsdlSpec` です。このカテゴリは tModel が Web サービスの WSDL 記述を持っていることを示します。これは tModels にだけ適用され、Web サービスの分類には使用されません。このカテゴリは IDE では表示されませんが、レジストリツールの検索の機能を使用して検出することができます。

デフォルトの公開識別子を設定する手順は次のとおりです。

1. **UDDI レジストリのオプションダイアログから「公開識別子値」をクリックします (図 2-23 を参照)。**
2. **表示される省略符号ボタン (...) をクリックします。**

   図 2-26 に示す「公開識別子」ダイアログが表示されます。

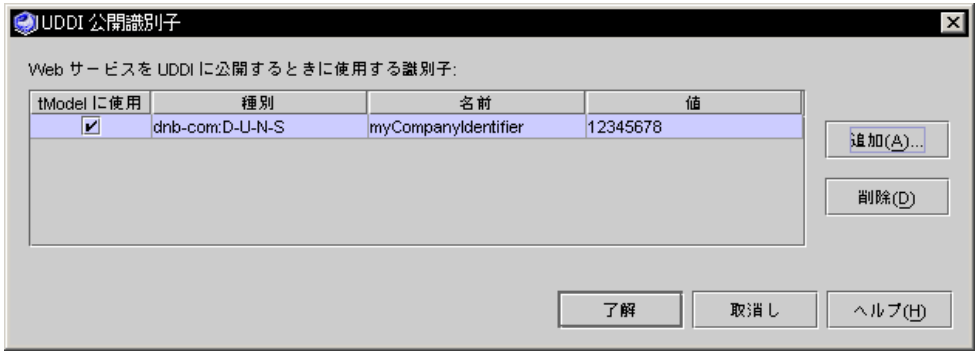

図 2-26 「UDDI 公開識別子」ダイアログ

デフォルトの公開識別子は次のように編集できます。

- 識別子をレジストリに公開する場合に tModel のデフォルト識別子として設定したい場合は、「tModel に使用」チェックボックスを選択します。
- 識別子を削除するには、識別子を選択して「削除」をクリックします。
- 識別子を追加する場合は、「追加」をクリックします。

  「UDDI 識別子を追加」ダイアログが表示されます (図 2-27 を参照)。識別子タイプと tModel UUID は業界で定められた標準です。

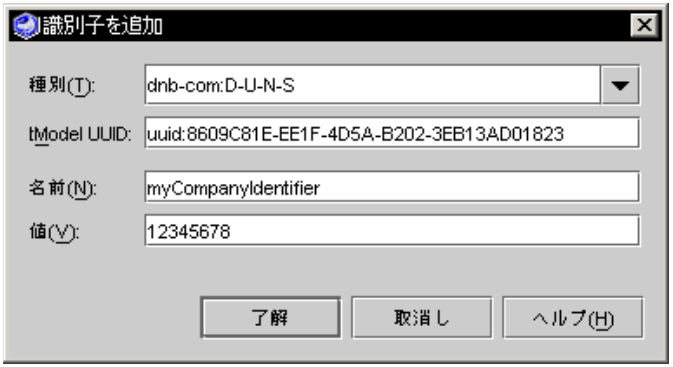

図 2-27　「識別子を追加」ダイアログ

1. 型の選択肢の中から 1 つ選択します。

   IDE によって選択した型に対する tModel UUID が表示されます。ユーザー定義による型を指定する場合は、名前と tModel UUID (レジストリキー) を入力します。

2. 名前と説明を入力して、「了解」をクリックします。

   IDE によって公開識別子ダイアログが再び表示されます。

3. 「了解」をクリックします。

   IDE によって、「カテゴリと識別子の UDDI 設定」ダイアログが表示されます。

IDE 内でのレジストリ情報の編集

IDE で既知のレジストリに関する情報を編集するには、「オプション」ダイアログの「UDDI レジストリ」の「レジストリ」プロパティの省略符号ボタン (...) をクリックします (図 2-23 を参照)。図 2-28 に示す「プロパティエディタ」ダイアログが表示されます。

プロパティエディタには、選択したレジストリに関する既知のレジストリおよび詳細情報が表示されます。画面の最上位部分には既知のレジストリの表が表示されます。レジストリ名をクリックすると、画面の下部に詳細情報が表示されます。

レジストリ情報を編集する方法は、次のとおりです。

- レジストリをリストに追加するには、「追加」をクリックします。「照会 URL」、「公開 URL」、および「ブラウザツール URL」に内容を説明するレジストリ名および値を指定します。
- レジストリを削除するには、選択してから「削除」をクリックします。
- デフォルトレジストリを変更するには、希望するレジストリを選択して「デフォルト」列のチェックボックスをクリックします。

- レジストリ名を編集するには、現在の名前をダブルクリックして新しい名前を入力します。
- デフォルトのログインを設定するには、「名前」および「パスワード」に値を入力します。

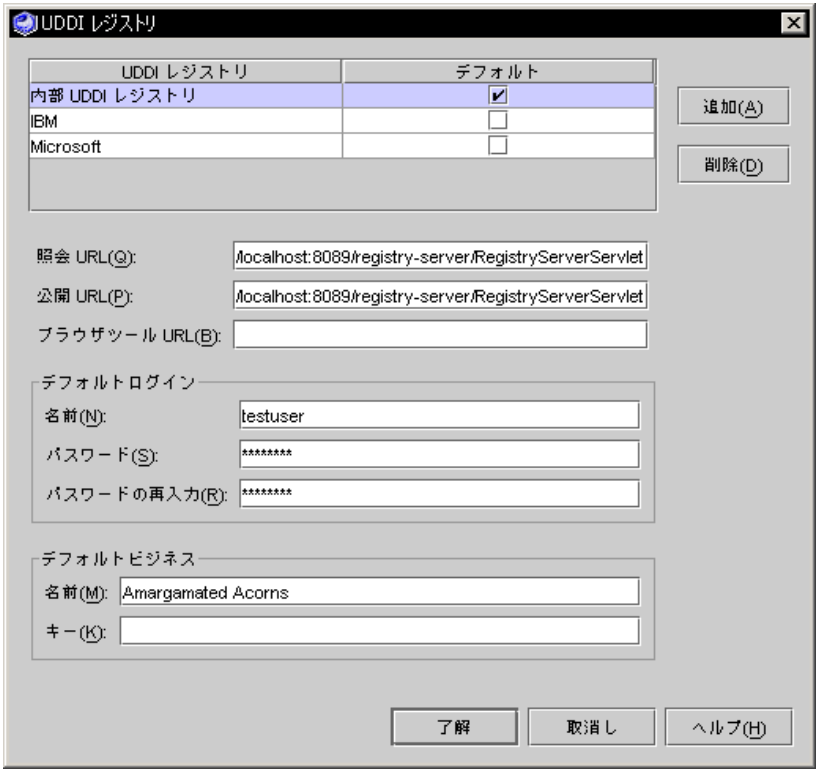

図 2-28　UDDI レジストリプロパティエディタ

IDE が UDDI ウィザードでデフォルトとして使用するビジネスを設定することもできます。デフォルトのビジネスを設定するには、「名前」および「キー」の値を指定します。

注 – デフォルトのログイン名とパスワードを設定することができますが、環境によってはセキュリティ上危険な場合もあります。パスワードは IDE のファイルシステムに保存され、暗号化されません。パスワードを毎回入力する方が安全です。

Web サービスを公開するときや、クライアントを生成するためにレジストリを検索しているときに、UDDI レジストリプロパティを編集することができます。91 ページの「クライアントの作成 - 手順」を参照してください。

外部 UDDI レジストリへのアクセス

UDDI レジストリにアクセスするには、IDE に特定の情報を提供する必要があります(50 ページの「IDE 内でのレジストリ情報の編集」を参照してください)。

レジストリを検索して Web サービスクライアントを生成するには、照会 URL が必要です。Web サービスをレジストリに公開するには、公開 URLが必要です。この情報を、レジストリを管理する「レジストリのオペレータ」から取得します。

セキュリティはレジストリによって異なります。レジストリへの公開にはログイン名とパスワードが必要になることもあります。これらのログイン名とパスワードはレジストリのオペレータに確保してください。パブリックレジストリでは、Web ブラウザから独自のアクティビティを設定できるツールが提供されている場合があります。

Web サービスの UDDI レジストリへの公開

この項では、公開に必要な作業と用語、および手順について説明します。

公開タスクと用語

UDDI レジストリ内の Web サービスエントリは、「ビジネス」エントリ、「テクニカルモデル (tModel)」エントリ、オプションのカテゴリ、および識別子と関連付けられています。Web サービスを UDDI レジストリに公開するとき、IDE ウィザードで次のタスクを実行できます。

- 新しいビジネスエントリを追加するか、または既存のビジネスエントリを検出してビジネスエントリを Web サービスエントリに関連付けます。(このウィザードで追加したビジネスには、カテゴリおよび識別子は関連付けられません。)
- 新しい tModel エントリを追加するか、または既存の tModel エントリを検出して tModel エントリを Web サービスエントリに関連付けます。
- (省略可能) 1 つ以上の標準業界カテゴリおよび識別子を Web サービスエントリまたは tModel エントリに関連付けます。

ビジネスとは Web サービスを所有する組織のエンティティです。レジストリにエントリを追加すると、そのエントリに対して固有のキーが自動的に生成されます。

tModel はレジストリエントリで、オーバービュー URL と呼ばれる WSDL ファイルを指す URL を含んでいます。WSDL ファイルは Web サービスの外部インタフェースを記述します。この情報は IDE または他の開発者ソフトウェアで実行時に、Web サービス上でメソッドを呼び出すことができるクライアントを作成するために使用されます。

UDDI レジストリ内の Web サービスエントリは、「エントリポイント」(「エンドポイント URL」または「サービス URL」ともいいます) を指定します。この URL は、クライアントが Web サービスの実行インスタンスを検出するために使用します。公開中に、IDE ではデフォルトでこの URL を Web サービスの「SOAP RPC URL」プロパティの値に設定します。この値は編集できます。

標準業界カテゴリおよび識別子は、UDDI レジストリ内の Web サービスエントリおよび tModel エントリと関連付けて後に続く検索を簡略化することができます。UDDI レジストリ検索の詳細については、90 ページの「UDDI レジストリからのクライアントの作成」を参照してください。

ビジネスは、複数の異なるサービスを UDDI レジストリに公開できます。各サービスは、tModel エントリに関連付けられている WSDL によって構造的に記述されます。1 つの tModel を複数のサービスエントリから参照することができます。たとえば、サービスプロバイダとして企業内で使用するサービスの 1 つのインスタンス、ビジネスパートナー用の別のインスタンス、および一般ユーザー用の別のインスタンスをサポートする場合があります。多様なインスタンスは tModel によって参照される WSDL 内の記述に従って同じ外部インタフェースを持ちますが、パフォーマンスおよび可用特性は異なります。

Web サービスを公開すると、tModel への参照が自動的に含まれます。

公開手順

この手順を実行する前に、Web サービスアプリケーションを配備しておく必要があります。IDE は、サービスの WSDL ファイルのコピーを配備したサービス内に配置します。サービスを公開すると、tModel がこの WSDL ファイルを参照します。

Web サービスを UDDI レジストリに公開する手順は、次のとおりです。

1. **エクスプローラ中の Web サービスノードを右クリックし、「UDDI に公開」を選択します。**

   図 2-29 に示されている「新規 Web サービスを UDDI に公開」ダイアログが表示されます。

   a. **「サービス名」を UDDI レジストリに入力します。**

   b. **http (デフォルト) か、ネットワークアクセスポイント型の複数の http を選択します。**

   c. **「ネットワークアクセスポイント」のアドレス URL は、デフォルトで Web サービスのSOAP RPC URL プロパティの値に設定されています。この URL を編集します。**

   d. **(省略可能) サービスの説明を入力します。**

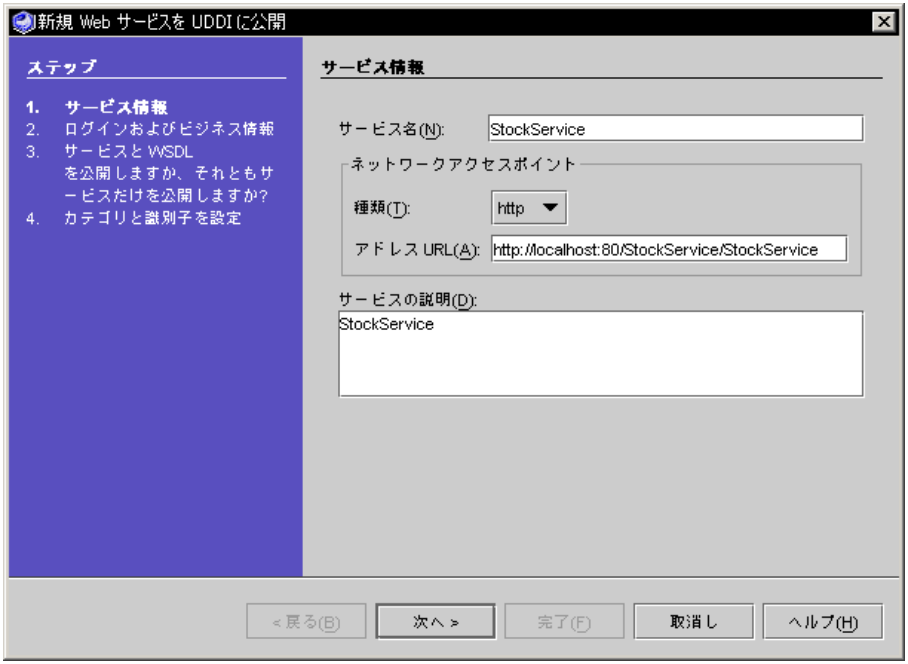

図 2-29　「新規 Web サービスを UDDI に公開」ダイアログ

注 – 「アドレス URL」のホスト名のデフォルト設定は `localhost` です。この設定をホストのネットワークマシン名に変更し、ポート番号が正しいことを確認します。

---

2. **「次へ」をクリックすると、「ログインおよびビジネス情報」ダイアログが表示されます (図 2-30 を参照)。**

   選択した UDDI レジストリ用のユーザー ID、パスワード、およびビジネスのデフォルト値をすでに設定している場合は、デフォルト値が表示されます。デフォルト値は、各フィールドに別の値を入力ことで変更できます。

   a. **公開用のターゲット UDDI レジストリを選択します。**

   b. **選択したレジストリに公開できるように、登録用のユーザー ID およびパスワードを入力します。**

c. **ビジネス名となる値を入力します。**

「検索」、「追加」ボタンが使用可能な状態になります。

ビジネスには、レジストリにすでにエントリを持つビジネスを指定できます。また、レジストリに新規ビジネスを追加して、そのビジネスを Web サービスに関連付けることもできます。

既存のビジネスを利用する場合は、「検索」をクリックします。「ビジネスキー」の既存の値が表示されます。レジストリ内の複数のビジネスが「ビジネスキー」の値に一致する場合は、レジストリサーバーによって最初に返された「ビジネスキー」値が表示されます。

新規ビジネスを使用する場合は、「追加」をクリックします。IDE はビジネスエントリを UDDI レジストリに追加して、「ビジネスキー」の生成値を表示します。この方法で追加したビジネスには、カテゴリおよび識別子は関連付けられていません。

レジストリのデフォルト値を、「ログインおよびビジネス情報」ダイアログから変更することもできます。

注 – 公開作業中にレジストリのデフォルト値を変更する場合は、「編集」をクリックします。IDE によって UDDI レジストリプロパティエディタが表示されます。詳細については、50 ページの「IDE 内でのレジストリ情報の編集」を参照してください。

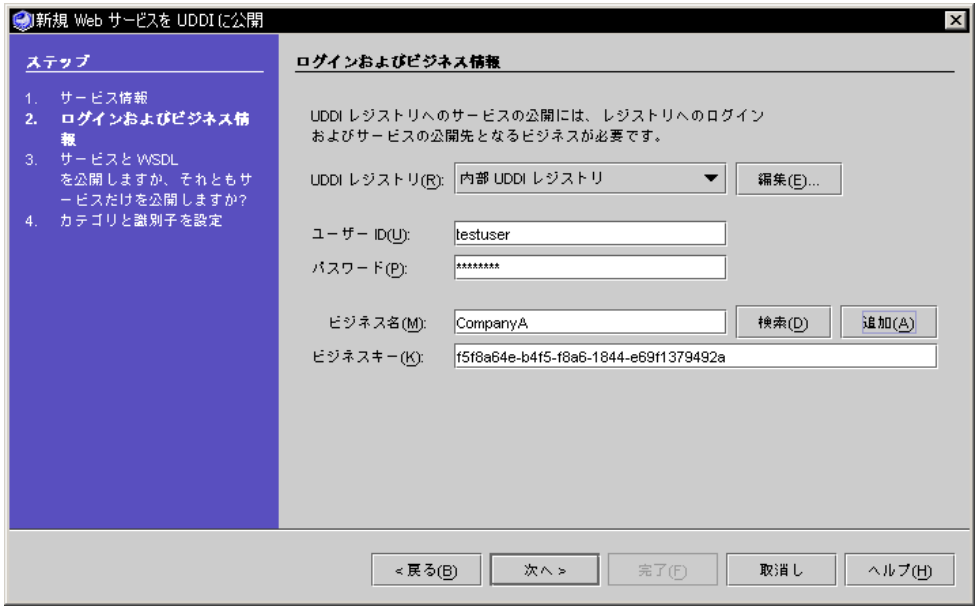

図 2-30　「UDDI ログインおよびビジネス情報」ダイアログ

d. 「次へ」をクリックします。

IDE によって tModel 選択ダイアログが表示されます (図 2-31 を参照)。

3. tModel 選択ダイアログに Web サービスの情報を入力します。

公開したい内容と詳細を指定します。

Web サービスには、新規または既存の tModel を関連付けることもできます。ダイアログでは次の操作が実行できます。

- レジストリ内に新規 tModel を作成する
- 既存の tModel を検索する

新しい tModel を作成すると、IDE 命名規則に基づいて、配備した Web サービスの Address URL から WSDL File URL が派生されます (図 2-29 を参照してください)。Address URL を変更した場合、または Web サービスを IDE の外部に配備した場合は、WSDL File URL の値は空白のままになります。

UDDI レジストリエントリからのクライアントの作成には、WSDL ファイル URL が使用されます。図 2-31 に示すデフォルトの URL では、WSDL ファイルが実行中の Web サービスから配布されます。この WSDL ファイルは、UDDI レジストリからクライアントを作成する際にダウンロードするものです。WSDL ファイルは別の場所に保存しておくと便利です。WSDL ファイル用の静的 URL を用意しておくと、Web サービスそのものが使用できなくても WSDL の URL にアクセスしてクライアントを作成できます。

注 – 認証プロパティとして HTTP 基本認証を持つWeb サービスを公開する場合は、WSDL ファイルの URL の変更が必要なこともあります。詳細については、184 ページの「基本認証と UDDI 公開」を参照してください。

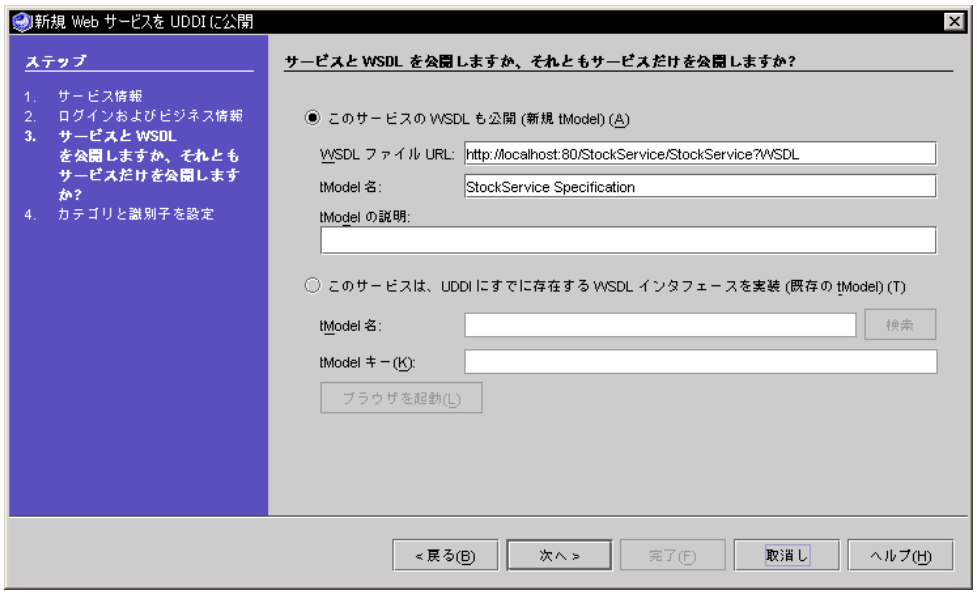

図 2-31 UDDI tModel 選択ダイアログ

UDDI 内の既存の tModel を使用する場合は、tModel 名を入力して「検索」をクリックします。tModel キー が表示されます。レジストリ内の複数の tModel が「tModel 名」の値に一致する場合は、レジストリサーバーによって最初に返された「tModel キー」値が表示されます。キー値がすでに分かっている場合は、「tModel キー」フィールドに値を入力します。

注 – ブラウザツールを提供している UDDI レジストリに Web サービスを公開している場合は、それらのツールを使用して tModel キーを検索できます。「ブラウザを起動」をクリックすると、UDDI レジストリのブラウザツール URL によって指定された Web ページにデフォルトの Web ブラウザが表示されます (図 2-28 を参照)。

---

4. **「次へ」をクリックします。**

図 2-32 に示す「UDDI カテゴリと識別子を設定」ダイアログが表示されます。ダイアログには、デフォルトのカテゴリおよび識別子が表示されます。

「編集」をクリックして、カテゴリまたは識別子を追加または削除します。編集手順の詳細な説明については、47 ページの「デフォルトの公開カテゴリおよび識別子の設定」を参照してください。

図 2-32　　UDDI カテゴリと識別子を設定

5. 「完了」をクリックします。

UDDI レジストリに Web サービスが公開されます。これには数分間かかります。IDE の最上位ウィンドウステータス行に正常に完了したことを示すメッセージが表示されます。

内部 UDDI レジストリの使用

Sun の Java™ Web Services Developer Pack (Java WSDP) は、単一ユーザー UDDI レジストリサーバーとして IDE に統合されています。内部レジストリは開発過程での完全なテストを簡便に実行できるように提供されています。UDDI レジストリは専用の Web サーバー (アプリケーション開発に使用されるサーバーインスタンスとは異なります) 内で実行されるサーブレットで構成され、XML データベースに格納されます。IDE は、レジストリサーバーを開始または停止したときに、専用サーバーおよびデータベースサーバーも自動的に開始または停止します。

注 – 内部 UDDI レジストリは単一のユーザー用に事前に設定されています。名前は `testuser` で、パスワードは `testuser` です。

内部 UDDI レジストリサーバーの開始と停止

内部 UDDI レジストリサーバーを開始する手順は、次のとおりです。

1. **「エクスプローラ実行時」タブ付き区画で UDDI サーバーレジストリノードを展開します。**

   「内部 UDDI レジストリ」ノードが表示されます。

2. **「内部 UDDI レジストリ」ノードを右クリックし、「サーバーを起動」を選択します (図 2-33 を参照)。**

   IDE 出力ウィンドウにサーバー開始メッセージが表示されます。IDE が以前のサーバープロセスを停止させたことを示すメッセージが表示されることもあります。

   内部 UDDI レジストリサーバーがすでに実行されている場合は、「サーバーを起動」メニュー項目はアクティブになりません。

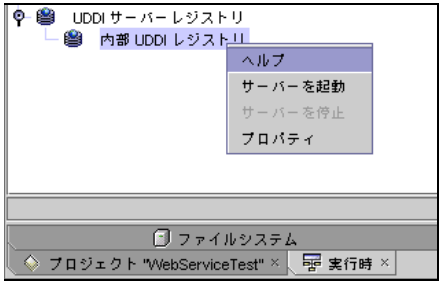

図 2-33　内部 UDDI レジストリサーバーの開始

内部 UDDI レジストリサーバーを停止する手順は、次のとおりです。

1. **「エクスプローラ実行時」タブ付き区画で UDDI サーバーレジストリノードを展開します。**

   「内部 UDDI レジストリ」ノードが表示されます。

2. **「内部 UDDI レジストリ」ノードを右クリックして、「サーバーを停止」を選択します。**

   IDE 出力ウィンドウにサーバー停止のメッセージが表示されます。IDE が以前のサーバープロセスを停止させたことを示すメッセージが表示されることもあります。

   内部 UDDI レジストリサーバーが実行されていない場合は、「サーバーを停止」メニュー項目はアクティブになりません。

サンプルレジストリブラウザの使用

Java WSDP では、「サンプルレジストリブラウザ」が提供されます。サンプルレジストリブラウザは、IDE の UDDI ウィザードがサポートしていない、いくつかの UDDI 保守作業を実行するために使用します。

サンプルレジストリブラウザを使用すると、内部レジストリからサービスを削除したり、名前およびキー以外のビジネス情報を追加したりなど、IDE ウィザードで提供されていないその他のアクションを実行できます。ただし、その名前が示すとおり、サンプルレジストリブラウザは完全な機能をすべて備えたツールではありません。たとえば、tModel を表示または削除することはできません。

Java WSDP ソフトウェアおよびサンプルプログラムは IDE のホームディレクトリの下の `jwsdp` ディレクトリにあります。サンプルプログラムソースは `wsdp/samples` ディレクトリにあります。

詳細およびダウンロード可能なチュートリアルについては、Java WSDP Web サイト `http://java.sun.com/webservices/webservicespack.html`を参照してください。

この節の目的は、サンプルレジストリブラウザののための参考となる一部の情報を説明するもので、その機能のすべてを説明することではありません。

注 – サンプルレジストリブラウザは、Sun ONE Studio 5, Standard Edition IDE 外の独自のウィンドウで実行します。ただし、主な用途は内部レジストリサーバーにアクセスすることです。外部レジストリは一般に検索および公開用の独自のツールセットを備えています。

サンプルレジストリブラウザを開始する前に、58 ページの「内部 UDDI レジストリサーバーの開始と停止」 に説明されているように IDE を使用して内部 UDDI レジストリを開始します。

サンプルレジストリブラウザを開始する手順は、次のとおりです。

1. **コマンドウィンドウを開いて現在のディレクトリを *s1studio-install-directory*`/jwsdp/bin` に変更します。**
2. **次のコマンドを入力します。**
   - Microsoft Windows システムの場合は `jaxr-browser`
   - Solaris 環境の場合は `./jaxr-browser.sh`

   図 2-34 に示すサンプルレジストリブラウザが表示されます。

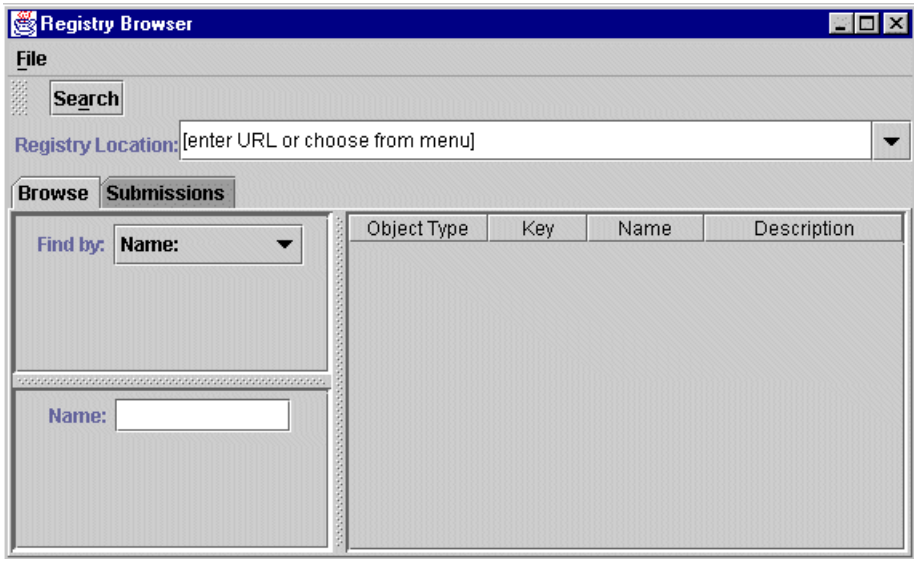

図 2-34　Java WSDP サンプルレジストリブラウザ

内部レジストリ用にサンプルレジストリブラウザを構成する手順は、次のとおりです。

1. 図 2-35 に示す「Registry Location」リストで、最後の URL を選択します。

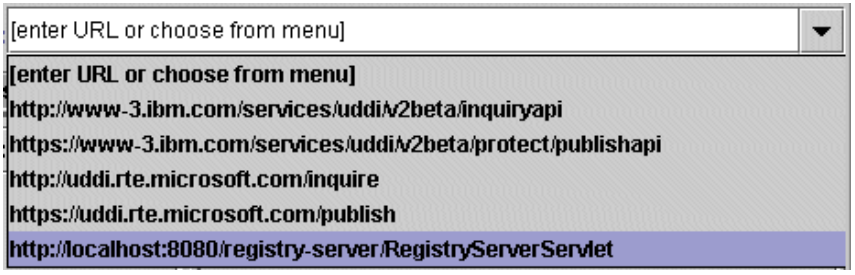

図 2-35　Java WSDP サンプルレジストリブラウザでの URL の選択

2. 図 2-36 に示すように URL のポート番号を 8080 から専用レジストリ Web サーバー用の正しいポート番号に変更します。

   IDE では、このサーバー用にデフォルトのポート番号として 8089 が設定されています。

   ポート番号を確認する手順は次のとおりです。

   a. エクスプローラの「実行時」タブで UDDI サーバーレジストリノードを展開します。

      「内部 UDDI レジストリ」ノードが表示されます。

b. 「内部 UDDI レジストリ」ノードを右クリックし、「プロパティ」を選択します。

サーバー URL プロパティ値にポート番号が表示されます。

ポート番号は、UDDI レジストリプロパティエディタの照会 URLおよび公開 URL にも表示されます (図 2-28 を参照)。

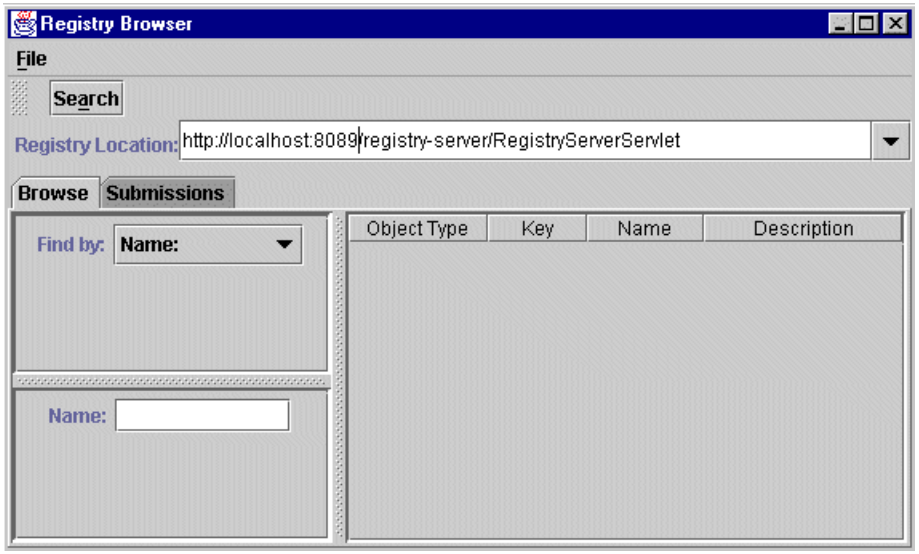

図 2-36　内部レジストリ URL を備えた Java WSDP サンプルレジストリブラウザ URL

図 2-37 に、会社名による検索の結果を示した例を示します。

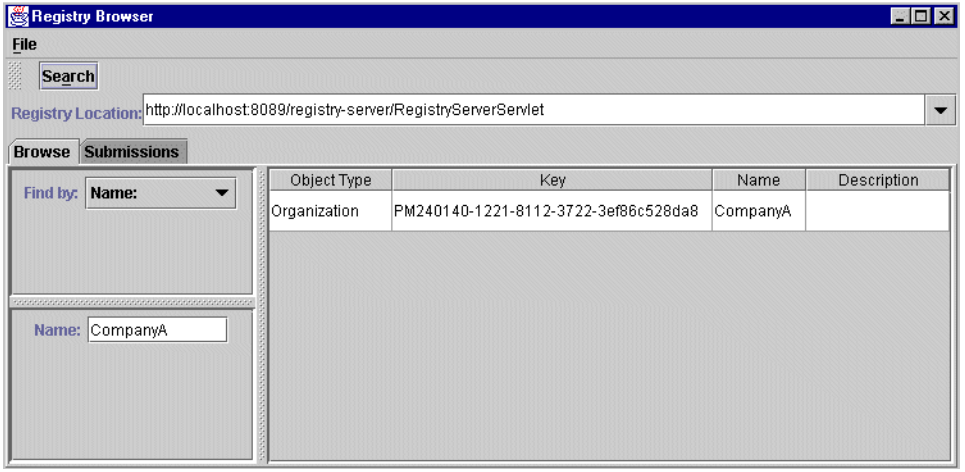

図 2-37　選択したビジネスを表示した Java WSDP サンプルレジストリブラウザ

図 2-38 は、「Submission」タブ付き区画を使用した例を示しています。

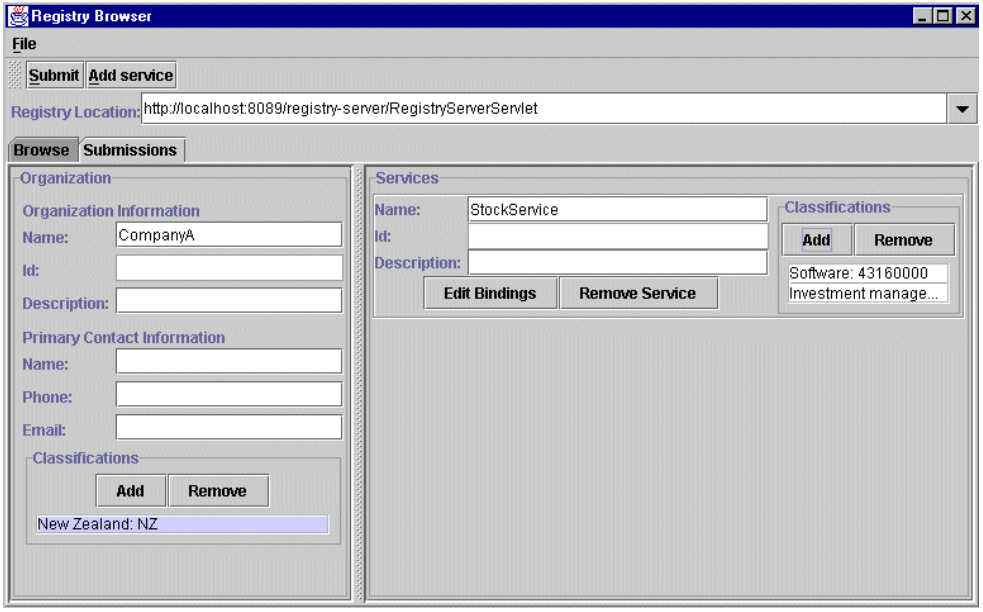

図 2-38 「Submission」タブ付き区画を表示した Java WSDP サンプルレジストリブラウザ

サンプルレジストリブラウザの追加機能を確認するために、次に示す Java WSDP Web サイトにあるチュートリアルをダウンロードして作業することができます。
`http://java.sun.com/webservices/webservicespack.html`

配列とコレクションの使用

この節では、`Array` および `Collection` 型の入力および出力データのサポートについて概説します。この項目の詳細については、第 5 章を参照してください。

配列

IDE の配列のサポートの概要は、次のとおりです。

- 配列出力は直接メソッド呼び出しおよび XML オペレーションでサポートされています。
- 配列入力は直接メソッド呼び出しでサポートされています。

- 配列入力は XML オペレーションでサポートされていません。

コレクション

コレクションは、`java.util.Collection` を実装する Java クラスです。IDE のコレクションのサポートの概要は、次のとおりです。

- コレクション出力は直接メソッド呼び出しおよび XML オペレーションでサポートされています。
- コレクション入力は、サーバー上およびクライアントプロキシ内での直接メソッド呼び出しでサポートされていますが、生成された JSP ページ内での呼び出しはサポートされていません。
- コレクション入力は XML オペレーションではサポートされていません。

コレクションは、ユーザークラスに定義された型のオブジェクトを含むことがあります。これらは Web サービスに参照されていない場合もあります。ユーザーが定義するオブジェクトに対しては、必ずシリアライズクラスを提供し、JAX-RPC タイプレジストリにシリアライズデータ型を追加してください。これで、JAX-RPC 実行時に適切に処理されるようになります。IDE はシリアライズクラスを追加する Web サービスプロパティを提供します。

シリアライズクラスを追加する手順は次のとおりです。

1. **Web サービスノードを右クリックし、「プロパティ」を選択します。その後、「シリアライズクラス」プロパティの値をクリックします。省略符号ボタン (...) をクリックします。**

   IDE によって既存のシリアライズクラス、および「追加」ボタンと「削除」ボタンが表示されます (図 2-39 を参照)。

2. **「追加」をクリックします。**

   「クラスの選択」画面が表示されます。ここから 1 つ以上のマウント済みパッケージを選択します。クラスとして、必ず JavaBean コンポーネントを選択してください。

   「1 つ追加」をクリックしても同じ操作ができます。IDE によって「クラスを 1 つ追加」ダイアログが表示されます。このダイアログではシリアライズクラス (`Java.lang.String` など) を名前で入力できます。

3. **目的のクラスを選択します。Ctrl キーを押しながらクリックすると、1 つ以上の項目を選択することができます。「了解」をクリックします。**

   IDE によってシリアライズクラスが表示されます。

図 2-39　シリアライズクラスプロパティエディタ

添付書類

添付書類の使用については、100 ページの「添付ファイル」 を参照してください。

SOAP メッセージハンドラ

SOAP メッセージハンドラを Web サービスやクライアントと使用する方法については、第 4 章 を参照してください。

セキュリティ保護

Web サービスとクライアントに対するセキュリティ保護の実装については、付録 A を参照してください。

配備記述子

Web サービス用の実行時クラスを生成すると、Web モジュールおよび EJB モジュール配備記述子も生成されます。Web サービス J2EE アプリケーションを組み立てると、これらの配備記述子がアプリケーションに含まれます。配備記述子は、アプリケーションの実行時プロパティを設定するために使用される XML ファイルです。J2EE アプリケーションはこれらの記述子のフォーマットを定義します。

Web サービスの配備記述子は、開発中はいつでもソースエディタ内で参照できます。配備記述子は配備処理の最後に編集することもできます。配備記述子の表示と編集方法については、付録 B を参照してください。

第3章

Web サービスクライアントの作成

Sun ONE Studio 5, Standard Edition IDE では、クライアントコードを記述することなく Web サービスクライアントを作成する方法を提供します。この章では、次の各タスクを実行するために使用する IDE ツールおよび手順について説明します。

- ローカル IDE Web サービスからのクライアントの作成
- WSDL からのクライアントの作成
- UDDI レジストリエントリからのクライアントの作成

ローカル Web サービスからのクライアントの作成

ここでの説明は、IDE で開発された既存の Web サービスが稼働中であることを前提としています。

Web サービスを開発中にテストする場合は、簡単なクライアントを自動的に生成することができます。(35 ページの「Web サービス用のデフォルトテストクライアントの設定」を参照してください。)

生成されたクライアントはカスタマイズしたり、商用のより洗練されたクライアントに置換したりすることができます。

クライアントの作成

クライアントは、Web サービスノード、パッケージノード、または IDE のメインウィンドウメニューから作成できます。

Web サービスノードから開始する場合

1. エクスプローラで「新規 Web サービスクライアント」ウィザードを開くには、Web サービスノード (青い球のアイコンが付いています) を右クリックし、「新規 Web サービスのテストクライアントを作成」を選択します。

   図 3-1 に示すダイアログが表示されます。

図 3-1　　ローカル Web サービスから作成した新規クライアントのダイアログ

2. 「パッケージ」フィールドに Web サービスのクライアントを作成する位置を指定するか、目的のパッケージがある位置までブラウズし、選択します。

3. 「名前」フィールドで、新しい Web サービスクライアントの名前を入力し、「了解」をクリックします。

   新しい Web サービスクライアントがエクスプローラに表示されます。クライアントノード ( ) が指定したパッケージの下に表示されます。

   すでにパッケージ内に存在する Web サービスクライアントの名前を入力すると、IDE には情報エラーメッセージが表示されます。

   Java パッケージノードから開始する場合

1. Java パッケージノードを右クリックし、「新規」->「すべてのテンプレート」->「Web サービス」->「Web サービスクライアント」を選択します。

   または、IDE のメインウィンドウから「ファイル」->「新規」->「すべてのテンプレート」->「Web サービス」->「Web サービスクライアント」を選択します。

   図 3-2 に示す「Web サービスクライアント」ダイアログが表示されます。

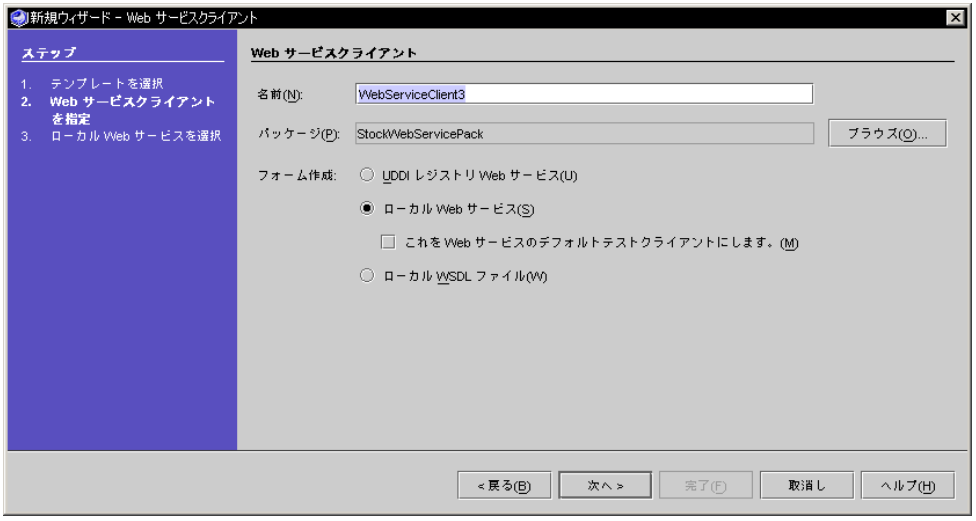

図 3-2　「新規 Web サービスクライアント」ダイアログ

2. 「名前」および「パッケージ」を設定します。
3. 「ソース」として、「ローカル Web サービス」を選択します。
4. 「次へ」をクリックして、選択ダイアログを開きます。目的の Web サービスを選択し、「完了」をクリックします。

   新しい Web サービスクライアントがエクスプローラに表示されます。クライアントノード ( ) が指定したパッケージの下に表示されます。

クライアントタイプの JAXRPC または kSoap への設定

IDE では、JAX-RPC 実行環境 (デフォルト) または kSOAP 実行環境を使用するクライアントを生成できます。JAX-RPC クライアントは、プレゼンテーション層に JSP ページを持つ Web アプリケーションとして、そしてフロントエンドの Web ブラウザとして実装されます。

注 – 旧 Apache SOAP クライアントがある場合、IDE はクライアントの生成中に、自動的にそのクライアントを JAX-RPC クライアントに変換します。

クライアント型を変更する手順は次のとおりです。

1. クライアントを作成します。

2. クライアントノードを右クリックし、「プロパティ」を選択します。「SOAP 実行」プロパティのデフォルト値は JAXRPC です。

3. プロパティ値をクリックします。

   選択値 JAXRPC および kSoap が表示されます (図 3-3 を参照)。

4. kSoap を選択するか、デフォルト値の JAXRPC のままにしておきます。

   クライアントを生成すると、kSOAP または JAX-RPC クライアントが作成されます。

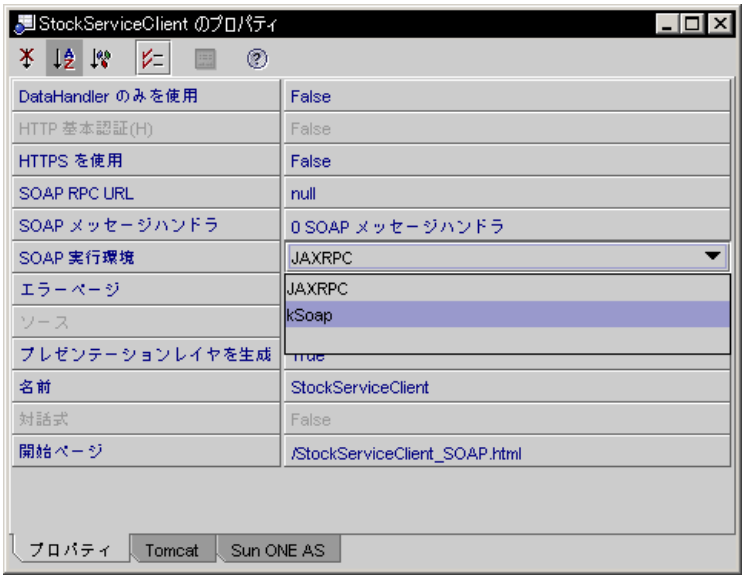

図 3-3　　クライアント SOAP 実行プロパティ

kSOAP 実行を使用するクライアントは、携帯電話や PDA など、一般的にユーザーインタフェースが表示領域の制約を受ける小さい J2ME デバイスで実行します。開発者はしばしば Java AWT、WML、およびその他のプレゼンテーション技術を使用してユーザーインタフェースを作成します。

注 – kSOAP クライアントを IDE で生成するには、kjava モジュールがインストールされている必要があります。また、kSOAP クライアントを IDE でテストするには、kjava-emulator モジュールがインストールされている必要があります。これらの 2 つのモジュールは IDE には自動的にインストールされません。アップデートセンターからダウンロードする必要があります。ダウンロードする場合は、「ツール」->「アップデートセンター」を選択して「アップデートセンター」ウィザードを使用してください。アップデートセンターの詳細については、『Sun ONE Studio 5, Standard Edition インストールガイド』を参照するか、IDE のオンラインヘルプまたはリリースノートを参照してください。

ワイヤレスアプリケーションについては、次を参照してください。

- J2ME サイト `http://wireless.java.sun.com`
- ワイヤレスデベロッパーサイト `http://java.sun.com/j2me`
- xxii ページの「オンラインで入手可能なマニュアル」に含まれる「Mobile Edition」のマニュアル

以降の節では、JAX-RPC クライアントと kSOAP クライアントについて説明します。

JAX-RPC クライアントの生成

JAX-RPC クライアントを生成する手順は次のとおりです。

1. **SOAP 実行プロパティ値を `JAXRPC` に設定します。**
2. **エクスプローラで、クライアントノードを右クリックし、「クライアントファイルを生成」を選択します。**

   IDE によって次のものが生成されます。

   - 「クライアントプロキシ」と呼ばれるクラスのセット、または実行時に Web サービスと SOAP メッセージをやり取りする「クライアントスタブ」
   - Web サービス操作を開始するためのカスタムタグ (各 Web サービスに対し 1 つのタグ) を複数持つ JSP タグライブラリ
   - HTML ページと前述のカスタムタグを使用する一連の JSP ページ (各 Web サービスに対し 1 ページ) 「開始ページ」と呼ばれるHTML ページは、エンドユーザーがアプリケーションを使用する際の開始位置となります。

   これらの要素については、以降で説明します。

生成ファイル

IDE では、クライアントと同じパッケージ内にある次の 2 つのノードの下に生成したクライアントファイルが表示されます。

- *xxx*`GenClient` というノード (*xxx* はクライアント名を表します) にはクライアントプロキシクラスが表示されます。
- (クライアントノードの下にある) `Generated Documents` というノードには、生成された HTML 開始ページ、HTML エラーページ、および JSP ページが表示されます。

JAX-RPC クライアントのエクスプローラ階層を図 3-4 に示します。

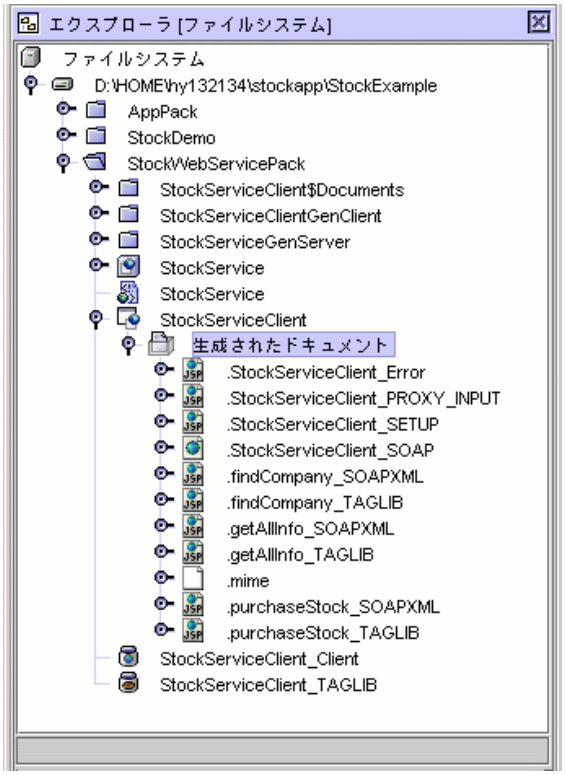

図 3-4　　エクスプローラでのクライアントドキュメントのノードと GenClient ノード

JSP カスタムタグ

このマニュアルの説明は、JSP テクノロジの一般的な知識があることを前提に記述されています。JSP テクノロジに関してはさまざまな資料が入手できます。JSP 仕様については次のサイトを参照してください。

`http://java.sun.com/products/jsp/download.html#specs`

IDE は、JSP タグライブラリを生成します。これには、クライアントが所有する JSP ページへの参照を簡単にするためのカスタムタグが含まれています。Web デザイナを介して Macromedia 社の Dreamweaver といったデザインツールを使用すると、Java プログラミング言語の知識がなくても、クライアントの表示形式をカスタマイズすることできます。

カスタムタグの例を次に示します。

```
<ws:sayHello string_1=":?" var="resp">
</ws:sayHello>
```

このタグを使用して、次のようなメソッドを使用しているリモート Web サービスを起動できます。

```
String resp = sayHello(String string_1);
```

各 Web サービスのオペレーション名と操作用のパラメータは、対応するタグ名とタグ属性にマップされています。

タグ属性値 `":?"` は、タグハンドラに対し、HTML 形式の要求変数からタグ属性値にバインドするように指定します。

Web サービスに対する呼び出しの結果は、`var` 属性内で指定されている JSP スクリプト変数として JSP ページに返されます。JSP ページは、デフォルトの結果プレゼンテーションを作成するための Java 標準タグライブラリ (JSTL: Java Standard Tag Library) を使用して、返されたスクリプト変数の内容を表現します。

クライアント HTML ページおよび JSP ページ

生成されるクライアントドキュメントは次のとおりです。

- **各 Web サービスオペレーション用に 1 つの JSP ページ。**各 JSP ページはクライアントプロキシを呼び出し、ビジネスメソッド (または XML オペレーション) を実行してから結果オブジェクトを返す Web サービスに対して SOAP メッセージを送信します。生成された JSP ページは、Web ページデザイナを使ってカスタマイズすることができます。カスタマイズした JSP ページを元に、ユーザーフレンドリーで洗練されたクライアントを作成できます。
- **1つの HTML 開始ページ。**開始ページには、「起動」ボタン、「リセット」ボタン、および生成された JSP ページそれぞれに対する入力フィールド (入力パラメータが必要な部分) が表示されます (図 3-12 を参照)。
- **1つの HTML エラーページ。**

開始ページおよび JSP エラーページの名前は、IDE 内でクライアントノードのプロパティとして表示されます (図 3-3 を参照)。

HTML ページまたは JSP ページのコードを参照する手順は次のとおりです。

- **目的のドキュメントのノードを右クリックし、「開く」を選択します。**

生成された開始ページのコード例を図 3-5 に示します。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
<title>
StockService
</title>
</head>
<center>
<h2>StockService</h2>
<h4>http://localhost:80/StockService/StockService</h4>
<hr> </hr>
<h4><a href="StockServiceClient_PROXY_INPUT.jsp">Input proxy server</a> :
<hr> </hr>
<h3>CompanyInfo findCompany(String string_1)</h3>
<form method="post" action="findCompany_TAGLIB.jsp">
<table width="75%">
<tr>
<td>
<table width="100%" border="1">
    <tr>
    <th>string_1:</th>
    <td>
    <input type="text" name="string_1"/>
    </td>
    </tr>
</table>
</td>
</tr>
</table>
<br>
<input type="submit" value="Invoke"/>
<input type="reset" value="Reset"/>
</form>
```

図 3-5　　クライアント HTML 開始ページ

生成された JSP ページのコード例を図 3-6 に示します。

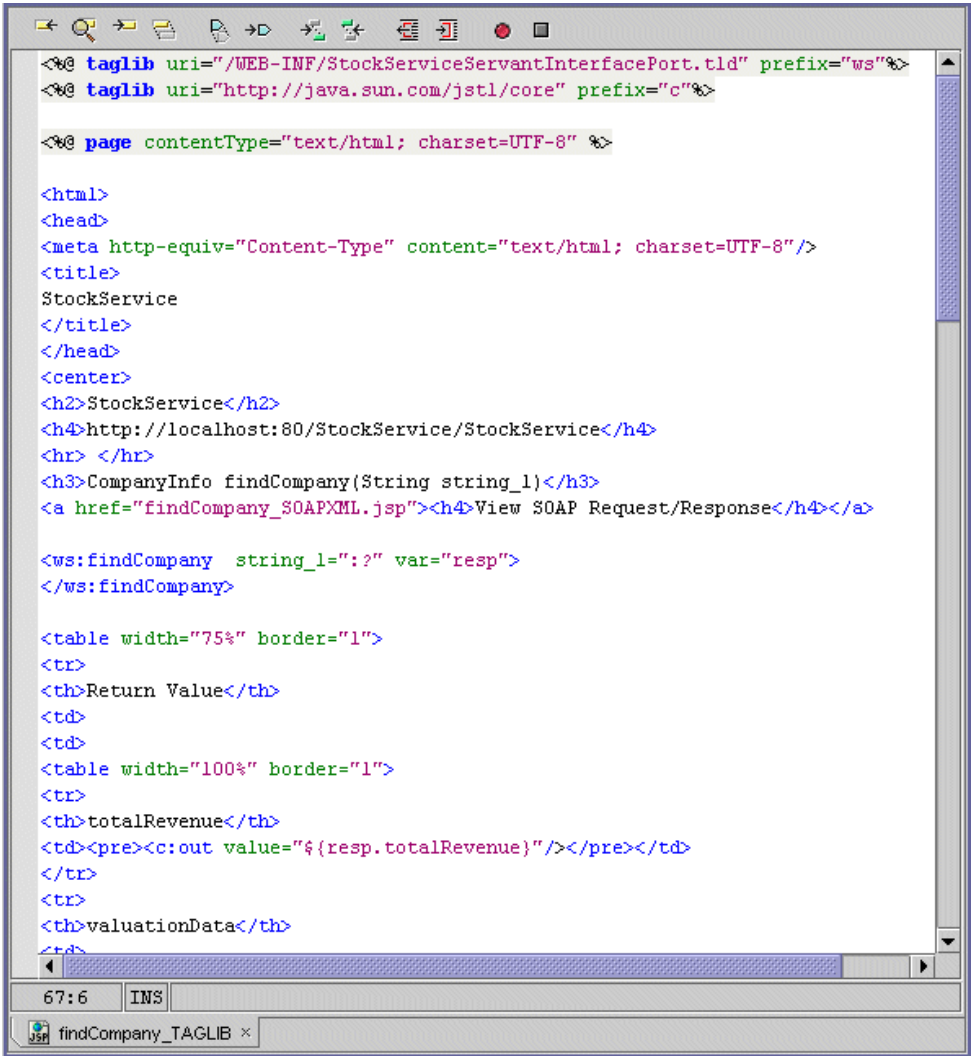

図 3-6　　クライアントサンプル JSP ページ

クライアントプロキシクラス

図 3-7 に、JAX-RPC クライアント用に生成された一部のクラスを表示しているエクスプローラを示します。

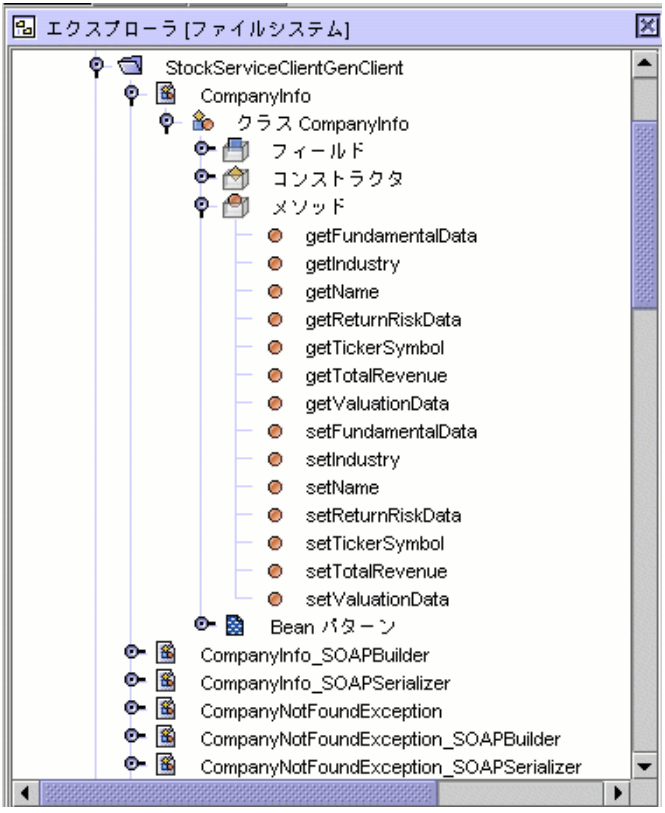

図 3-7　　クライアント SOAP プロキシ GenClient ノード

Java ソースコードを表示する手順は次のとおりです。

- **クラスまたはメソッドノードを右クリックして、「開く」を選択します。**

  IDE を使用すると、プロキシソースを編集できますが、IDE 内でクライアントを生成し直した場合は、プロキシソースの変更は保持されません。

  生成されたクライアントプロキシは、 Java Swing クライアントや洗練された HTML および JSP ページとともに使用できます。

独自のフロントエンドクライアントコンポーネントの使用

ビジネスメソッド用のフロントエンドとして、Swing コンポーネントや IDE が生成した HTML ページや JSP ページよりも高度なページがすでに揃っている場合は、IDE にクライアントプロキシだけを作成させることができます。

クライアントプロキシクラスだけを作成する手順は次のとおりです。

1. クライアントを作成します。
2. クライアントノードを右クリックし、「プロパティ」を選択します。「プレゼンテーションレイヤを生成」プロパティのデフォルト値は `True` です。
3. 「プレゼンテーションレイヤを生成」プロパティの値を `False` に変更します。

   クライアントを生成すると、クライアントプロキシクラスだけが作成されます。この操作は一般的に、フロントエンドプレゼンテーションが IDE で自動生成されたものとは別に用意されているときに実行します。この場合、クライアントの WAR ファイルに、自分で用意したドキュメント、ライブラリ、またはクラスを追加する必要があります。

   クライアントの WAR ファイルに、自分で用意したドキュメント、ライブラリ、またはクラスを追加する手順は次のとおりです。

1. IDE を使わないでファイルを作成した場合 (Web デザインツールを使って作成した HTML ページや JSP ページなど) は、それらのファイルが含まれるディレクトリをマウントし、IDE のエクスプローラからそれらのファイルを認識できるようにします。
2. クライアントノードを右クリックし、「WAR ファイルに追加」->「オブジェクトの種類」を選択します。「オブジェクトの種類」はドキュメント、ライブラリ、またはクラス のどれかです。

   IDE によって選択ダイアログが表示されます。
3. ドキュメント、ライブラリ、またはクラスまでフォルダまでナビゲートし、目的のオブジェクトを選択して「了解」をクリックします。

   クライアントノードの下に**ドキュメント**、**ライブラリ**、または**クラス**フォルダが、追加されたオブジェクトへの参照とともに IDE によって作成されます (図 3-8 を参照)。

図 3-8　　ドキュメント、ライブラリ、およびクラスに対するクライアント参照

この IDE の操作によって、ユーザーの用意したクライアントの WAR ファイルにドキュメント、ライブラリ、またはクラスが追加されます。以降、これらのドキュメント、ライブラリ、またはクラスを使用して、カスタマイズしたクライアントを構築できるようになります。

注意 – ファイルまたはフォルダへの参照を削除すると、参照だけが削除され、ファイルまたはフォルダはそのまま残ります。参照されるフォルダの「中にある」ノードは、「実際」のファイルやフォルダを表すリンクです。参照されるフォルダ内のノードを削除すると、ノードがリンクされているファイルまたはフォルダが削除されます。実際のファイルまたはフォルダを削除しない場合は、参照されるフォルダ内のノードを削除しないでください。

クライアント生成中の検査

生成のプロセスにおいて、IDE はWeb サービスの WSDL を分析します。ここで無効なデータ型 (JavaBean コンポーネントに対して無効なユーザーオブジェクトなど) を見つけると、検査警告やエラーメッセージを発行します。たとえば、UDDI レジストリを経由してアクセスする Web サービスなど、第三者の Web サービスを元にクライアントを作成する場合などに、この検査は非常に重要になります。IDE 検査は JAX-RPC リファレンス実装に基づいて実施されます。

Web サービスからのクライアントの参照

「Web サービスから再表示」は、「クライアントファイルを生成」の代替メニュー項目です。「クライアントファイルを生成」を選択すると、IDE は、Web サービスプロパティとクライアントプロパティの組み合わせに応じて、クライアントプロキシクラスとドキュメントを生成します。

たとえば、ローカル Web サービスがステートフルだった場合、IDE はデフォルトでステートフルクライアントを作成します。また、Web サービスとクライアント両方の対話式プロパティは `True` に設定されます。クライアントの対話式プロパティを手動で `False` に設定して「クライアントファイルを生成」を指定すると、IDE はその変更を保持したままステートレスクライアントを生成します。

「クライアントファイルを生成」の代わりに「Web サービスから再表示」を選択することもできます。このメニュー項目を選択した場合は、IDE によって Web サービスのプロパティに基づいたクライアントプロキシクラスとドキュメントが生成されます。この場合、ローカル Web サービスの対話式プロパティには `True` が設定され、IDE はクライアントの対話式プロパティを Web サービスに合わせて `True` に設定します。

Web サービスがクライアントの作成後に拡張された場合は、メソッドが追加されていたり、メソッドシグニチャが変更されていることがあります。「Web サービスから再表示」を実行すると、新規 WSDL が使用され、拡張された Web サービスに適切なクライアントファイルが生成されます。

クライアントが WSDL ファイル、または UDDI レジストリ中の WSDL 参照から作成されている場合は、「Web サービスから再表示」の実行時に WSDL 内にある Web サービスプロパティだけが使用されます。特に、セキュリティと対話式プロパティは WSDL 内で指定されていないので、IDE によってこれらのクライアント値が変更されることはありません。

Web サービスに基づいてクライアントを再生成する手順は次のとおりです。

- **エクスプローラ中のクライアントを右クリックし、「Web サービスから再表示」を選択します。**

  これによりクライアントプロキシおよびドキュメントが生成し直されます。

注 – クライアントがローカル Web サービスから生成されている場合、メニュー項目は「Web サービスから再表示」となります。クライアントが WSDL ファイルから生成されている場合、メニュー項目は「WSDL から再表示」となります。クライアントが UDDI レジストリから生成されている場合、メニュー項目は「UDDI から再表示」となります。すべての場合で、IDE はクライアントを元の場所にある WSDL に基づいて生成します。IDE は WSDL ファイルの URL をクライアントのソースプロパティの値として保持します。

サービスエンドポイント URL

クライアントの実行時にサービスインスタンスにアクセスするには、クライアントプロキシがサービスの「エンドポイント URL」を持っている必要があります。この URL は、IDE がクライアントプロキシ内にデフォルト URL として設定することができます。また、クライアントプロキシの実行時に、プロキシ内の `serviceURL` パラメータに URL を渡すこともできます。こうして渡された URL はデフォルトの URL に優先されます。

IDE によって生成されたクライアント JSP ページは、実行時に URL を渡しませんが、クライアントプロキシが `serviceURL` のデフォルト値を持っているとみなします。デフォルト URL は、クライアントプロキシを生成するために使用する WSDL から取得されます。

Web サービスが製品システムに再配備された場合などには、クライアントにエンドポイント URL の WSDL 値を上書きさせることもできます。

クライアントの実行時にエンドポイント URL を渡す手順は次のとおりです。

1. **エクスプローラ中のクライアントノードを右クリックし、「プロパティ」を選択します。**
2. **URL として SOAP RPC URL プロパティを設定します。**

   この URL は、WSDL ファイルから派生したすべてのデフォルト URL に優先されます。

JAX-RPC クライアントの配備

IDE は、アプリケーションサーバーの Web コンテナに含まれる Web アプリケーションとしてクライアントを配備します。

クライアントを配備する手順は次のとおりです。

- **エクスプローラ中のクライアントノードを右クリックし、「プロパティ」を選択します。**

  クライアントの配備済み WAR ファイルがエクスプローラの「実行時」タブの付いた区画に表示されます (図 3-9 を参照)。

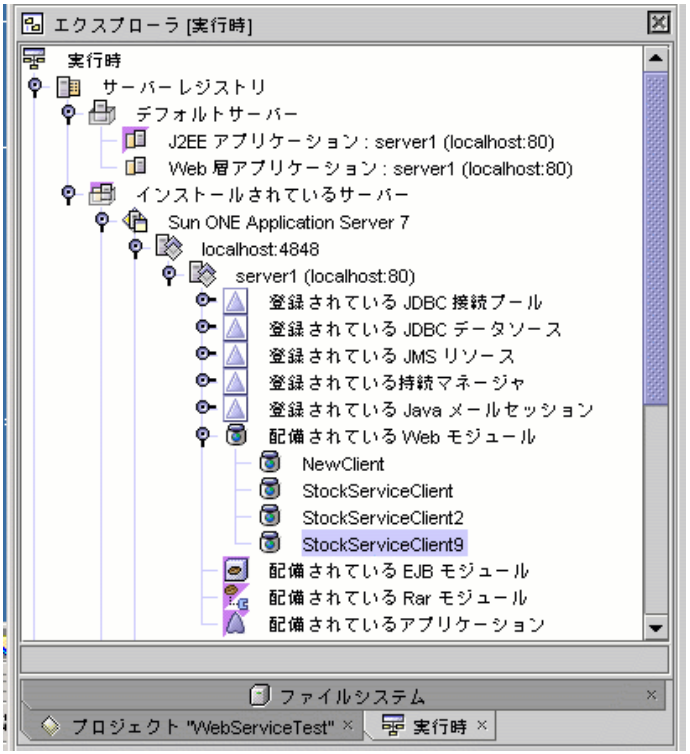

図 3-9　エクスプローラに表示されたクライアントの WAR ファイル (配備済み Web モジュールノード)

クライアントの WAR ファイルノードが、エクスプローラの「ファイルシステム」タブの付いた区画に表示されます (図 3-10 を参照)。

図 3-10　エクスプローラに表示されたクライアントの WAR ファイル (クライアントノード)

WAR ファイルの内容を表示する手順は次のとおりです。

- **エクスプローラの「ファイルシステム」タブ内に表示されたクライアントの WAR ノードを右クリックし、「WAR の内容を表示」を選択します。**

  IDE によって WAR ファイルの内容が表示されます (図 3-11 を参照)。

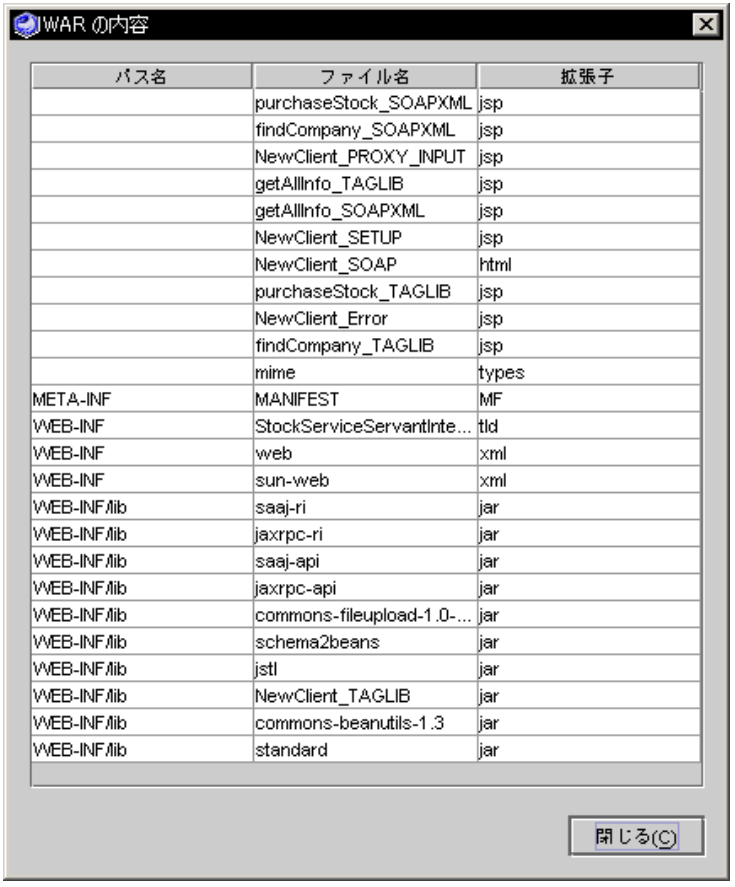

図 3-11　クライアントの WAR ファイルの内容

JAX-RPC クライアントの実行

ここでの説明は、クライアントとそのクライアントが参照する Web サービスが配備済みで、それらの Web サービスとクライアントのサーバーが稼動中であることを前提としています。

クライアントを実行する手順は次のとおりです。

- **エクスプローラの「クライアント」ノードを右クリックし、「実行」を選択します。**

  IDE によってクライアントが実行され、デフォルトの Web ブラウザに開始ページが表示されます (図 3-12 を参照)。

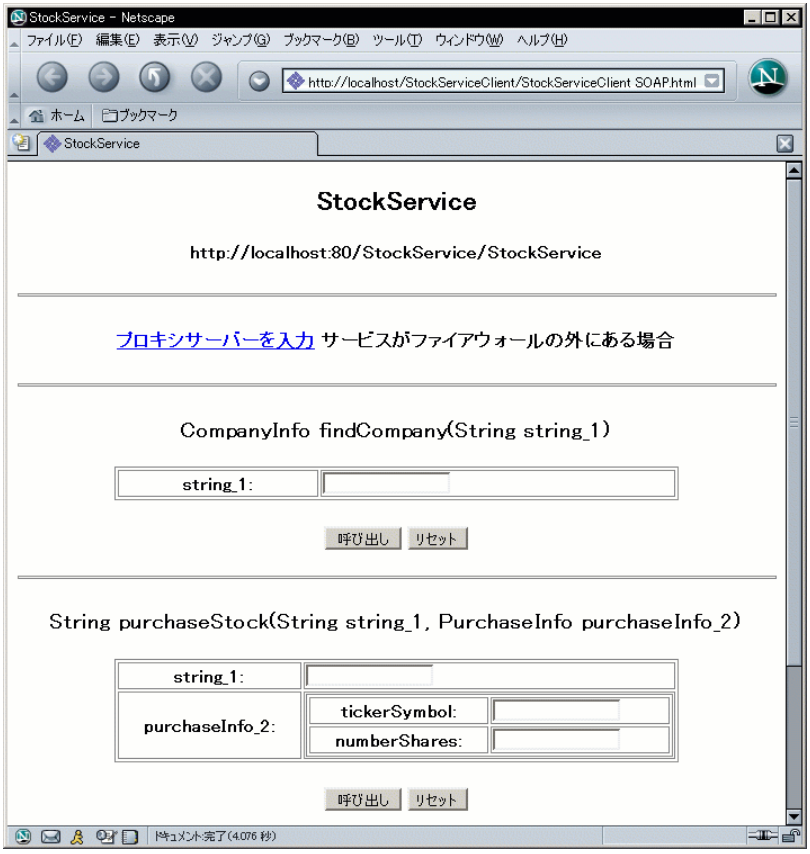

図 3-12　　クライアント開始ページ

ここでは、Web サービスが J2EE アプリケーションに配備され、35 ページの「Web サービス用のデフォルトテストクライアントの設定」で説明したように、その Web サービスのクライアントをテストクライアントにした場合の例を説明します。このとき、クライアントは IDE によって Web サービスとともに配備され、J2EE アプリケーションノードからそのクライアントを実行することができます。

デフォルトのテストクライアントを J2EE アプリケーションノードから実行する手順は次のとおりです。

- **エクスプローラの「J2EE アプリケーション」ノードを右クリックし、「実行」を選択します。**

  IDE によって生成されたクライアントを介し、各 Web サービスのメソッド呼び出しの結果を 2 通りの方法で参照することができます。1 つはユーザーフレンドリーな形式で参照する方法で、もう 1 つは XML 形式の SOAP 要求および応答メッセージとして参照する方法です。

  図 3-12 に示した例と `findCompany` メソッドを合わせて考えてみましょう。`string_1` に有効な株式取引名を入力し、「呼び出し」をクリックしたとします。クライアントによって、図 3-13 に示したような会社情報がブラウザに表示されます。この時点の表示から、ユーザーが参照する表示を Web デザイナがカスタマイズすることができます。

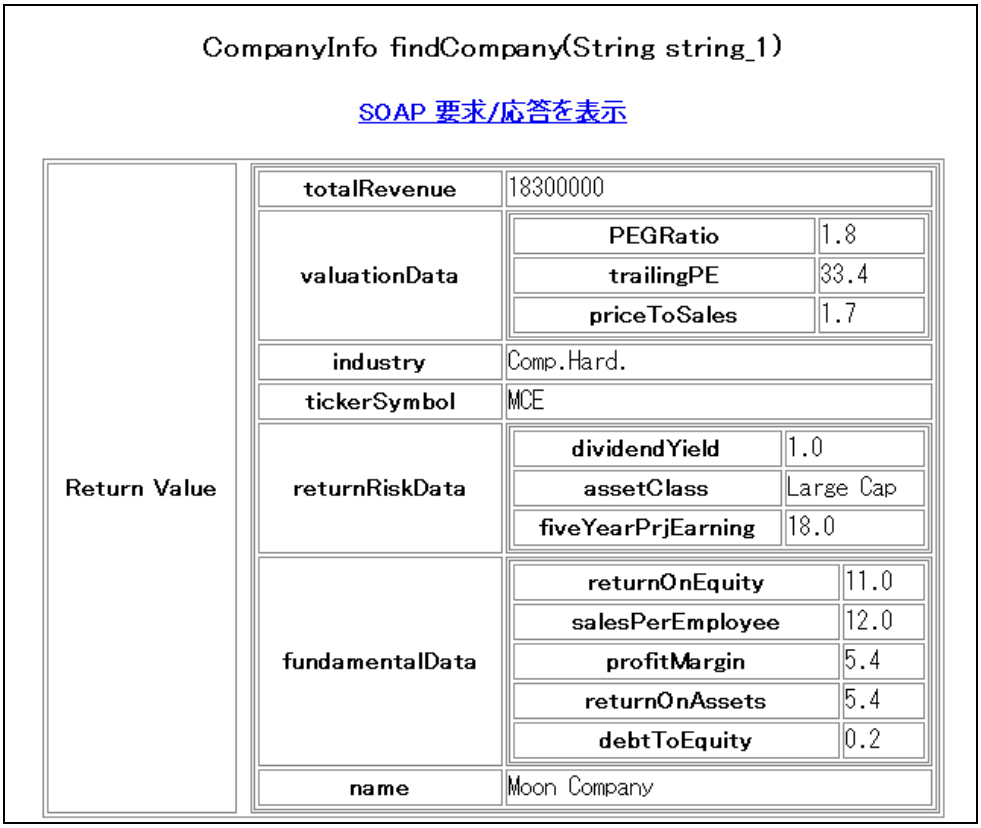

図 3-13　クライアントによる会社情報の表示

「SOAP 要求/応答を表示」をクリックすると、クライアントによって SOAP 要求メッセージ (クライアントから Web サービスに送信されたもの) と SOAP 応答メッセージ (Web サービスからクライアントに送信されたもの) が表示されます。図 3-14 の表示例では SOAP エンベロープとそのデータが含まれていて、この情報はクライアントまたは Web サービスのデバッグに利用できます。

図 3-14 クライアントによる会社情報の表示 (SOAP 要求/応答による形式)

注 – IDE は JSP ページを生成する際に、そのページが対応する Web サービスオペレーションと一致する名前を付けます。クライアントのテスト中はブラウザの URL を参照して、現在表示されている部分がどの JSP ページによって実装されているのかを確認できます。

ファイアウォール

クライアントがファイアウォールの内側で実行されていて、Web サービスがファイアウォールの外側にある場合は、プロキシホストとプロキシポートを指定する必要があります。

プロキシホストとプロキシポートを実行時に指定する手順は次のとおりです。

- **開始ページにある「プロキシサーバーを入力」をクリックします。**

  クライアントによって、図 3-15 に示すページが表示されます。このページでは、プロキシホストとプロキシポートの値を入力できます。

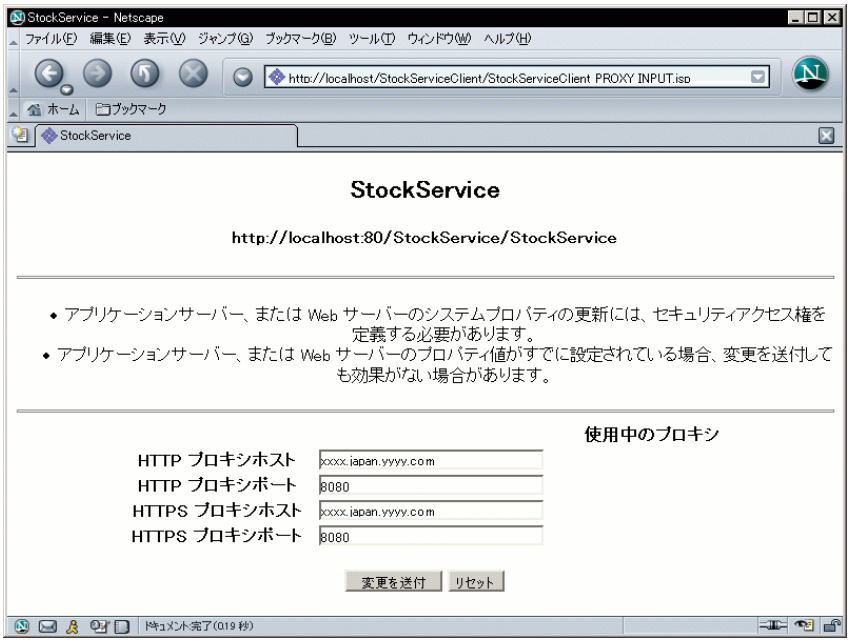

図 3-15　クライアント入力プロキシサーバーのページ

IDE で、プロキシホストとプロキシポートのデフォルト値を指定する手順は次のとおりです。

- **IDE のメインメニューから「ツール」->「オプション」->「IDE 構成」->「システム」->「システム設定」->「システム」を選択し、プロキシホストとプロキシポートの値を設定します。**

  ここで指定した値が IDE で生成されるクライアントのデフォルト値となります。

注 – クライアントのユーザーが実行時にプロキシホストとプロキシポートの値を入力した場合は、これらの実行時の値が IDE で設定したすべてのデフォルト値よりも優先されます。

分散アプリケーション

クライアントと Web サービスが異なるサーバーで実行されている場合、さまざまな問題が発生します。開発環境では、サーバーは通常 1 台だけで動作の統制を取ることは容易です。しかし、製品環境ではサーバーが複数あることが多く、それぞれは別のシステム管理者によって管理されています。こういった場合、サーバーでのアクセス権の設定を要求する java.security.AccessControlException メッセージなど、実行時メッセージが出力されることがあります。

このような問題については、『Sun ONE Studio 5, Standard Edition リリースノート』を参照してください。

kSOAP クライアントの生成

kSOAP クライアントを IDE で生成するには、`kjava` モジュールがインストールされている必要があります。このモジュールのインストール方法については、69 ページの「クライアントタイプの `JAXRPC` または `kSoap` への設定」を参照してください。

kSOAP クライアントを生成する手順は次のとおりです。

1. **69 ページの「クライアントタイプの `JAXRPC` または `kSoap` への設定」 の手順に従い、クライアントの「SOAP 実行時」プロパティ値を `kSoap` に設定します。**
2. **エクスプローラ中のクライアントノードを右クリックし、「クライアントファイルを生成」を選択します。**

   IDE によって次のものが生成されます。

   - *xxx*kSOAPProxy という名前のクライアントプロキシクラス。*xxx* はクライアント名となります。クライアントプロキシクラスは、実行時に Web サービスと SOAP メッセージをやり取りします。
   - *xxx*MIDlet という名前の MIDlet。*xxx* はクライアント名となります。MIDlet は、要求と応答に対するグラフィカルユーザーインタフェースを提供します。

   生成されたクライアントノードを図 3-16 に示します。

図 3-16 エクスプローラに表示された kSOAP クライアントノード

kSOAP 実行を使用するクライアントは、携帯電話や PDA など、一般的にユーザーインタフェースが表示領域の制約を受ける小型の J2ME デバイスで実行します。開発者はしばしば Java AWT、WML、およびその他のプレゼンテーション技術を使用してユーザーインタフェースを作成します。

注 – IDE は、kSOAP クライアントに対しては生成ドキュメントノードを作成しません。また、HTML ページや JSP ページも作成しません。

---

kSOAP クライアントの実行

kSOAP クライアントを IDE で実行するには、`kjava-emulator` モジュールがインストールされている必要があります。このモジュールのインストール方法については、69 ページの「クライアントタイプの JAXRPC または kSoap への設定」を参照してください。

IDE で kSOAP クライアントを実行する手順は次のとおりです。

- **エクスプローラ中のクライアントノードを右クリックし、「実行」を選択します。**

  IDE によってエミュレータが開かれ、クライアントメニューが表示されます (図 3-17 を参照)。

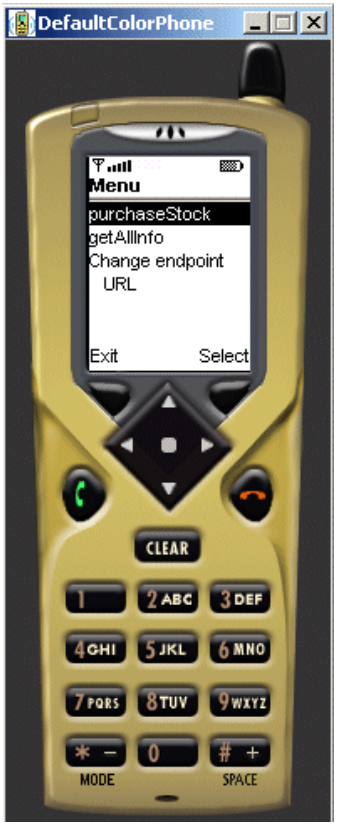

図 3-17　携帯電話エミュレータでの kSOAP クライアントの実行

WSDL ファイルからのクライアントの作成

WSDL は、Web サービスの外部インタフェースの簡易表示を提供します。Web サービスの WSDL 記述を作成する手順については、46 ページの「WSDL の生成」を参照してください。WSDL および UDDI レジストリに関するその他の詳細については、5 ページの「WSDL」および 45 ページの「UDDI レジストリの使用」を参照してください。

WSDL は、UDDI レジストリを使用していない場合でも役立ちます。プロジェクト内の何人かが別の場所または別のスケジュールで異なるいくつかのクライアントを開発している場合があります。彼らに、クライアントを生成するための WSDL のコピーを渡すことができます。WSDL ファイルは XML ドキュメントであり、共有ネットワークドライブに配置したり、電子メール添付ファイルとして送信したり、ディスクで配布したりすることができます。開発者が、Web クライアントをテストするためには、サービスの実行インスタンスにネットワークでアクセスできる必要があります。

次の手順は、WSDL ファイルがあること、ファイル名拡張子が `.wsdl` であること、ファイルを含んでいるディレクトリが IDE にマウントされていることを前提としています。`.wsdl` ファイルはエクスプローラにノードとして緑の球形アイコン () 付きで表示されます。この命名規則に従わない WSDL ファイルを受け取った場合は、名前を変更するか、またはファイル名拡張子 `.wsdl` を付けてコピーを作成できます。

WSDL からクライアントを生成する手順は、次のとおりです。

1. **IDE のメインウィンドウから「ファイル」->「新規」->「Web サービス」->「Web サービスクライアント」を選択し、「次へ」をクリックします。**

   図 3-2 に示す「Web サービスクライアント」ダイアログが表示されます。

2. **「名前」および「パッケージ」を設定し、「ソース」で「ローカル WSDL ファイル」を選択します。**

3. **「次へ」をクリックして選択ウィンドウを開きます。**

4. **選択ウィンドウで WSDL ノードを選択し、「完了」をクリックします。**

   IDE によって、デフォルトの「SOAP 実行」プロパティに `JAXRPC` を持つクライアントノードが作成されます。ワイヤレスクライアントが必要な場合は、このプロパティを `kSoap` に変更できます。

5. **エクスプローラ中のクライアントノードを右クリックし、「クライアントファイルを生成」を選択します。**

   IDE によってクライアントファイルが作成されます。

注 – 「クライアントファイルを生成」の代わりに「WSDL から再表示」を選択することもできます。このコンテキストメニューに関する説明は、78 ページの「Web サービスからのクライアントの参照」 を参照してください。

6. **配備および実行など、その後の手順は 67 ページの「ローカル Web サービスからのクライアントの作成」と同じです。**

   (JAX-RPC 用の) 生成ドキュメントや (kSOAP クライアント用の) MIDlet を生成しないで、クライアントのサポートするクラスだけを生成する手順は、次のとおりです。

1. **エクスプローラ中のクライアントノードを右クリックし、「プロパティ」を選択します。次に、「プレゼンテーションレイヤを生成」プロパティの値を `False` に設定します。**
2. **エクスプローラ中のクライアントノードを右クリックし、「クライアントファイルを生成」を選択します。**

   IDE によってクライアントプロキシクラスだけが生成されます。

3. **クライアントプロキシへの呼び出しを行う独自のクライアントコンポーネントを作成します。**

   これは、クライアントプロキシを使用するために簡単にカスタマイズできる開発済みのクライアントがすでにある場合、または Java Swing クライアントなど、IDE によって生成された HTML および JSP ページとは異なるクライアントを使用する場合に適しています。

UDDI レジストリからのクライアントの作成

UDDI レジストリからクライアントを作成する必要があるビジネスニーズおよびシナリオについては、第 1 章を参照してください。この節では、計画と作業フローの概要、さらに UDDI レジストリの検索、Web サービスの選択、およびクライアント作成までの手順を説明しています。

クライアントの企画と作業フローの作成

作業フローは、次の操作で構成されます。

- Web サービスを選択するための基準の決定
- 検索する UDDI レジストリの選択
- Web サービス用の UDDI レジストリの検索

- Web サービス用の WSDL のダウンロード
- クライアントの生成
- クライアントのカスタマイズ

UDDI レジストリ内で Web サービスを検索する基準を決定するには、事前の計画が必要です。また、これはアプリケーションの設定手順の一部であり、Web サービスの特性を考えるだけでは不十分です。サービスを提供するビジネスについて考え、そのビジネスがどのようなサポートを提供するか、サービスの使用コストはどのくらいなのかについても考える必要があります。プライベートレジストリを使用していて、所属するプロジェクト企業、またはプロジェクトを共同計画しているビジネスパートナーがサービスプロバイダとなる場合には、分析は簡単になります。

クライアントの作成 - 手順

この項では、IDE を使用してレジストリを選択する方法、サービス用のレジストリを検索する方法、およびサービスの実行インスタンスにアクセス可能なクライアントの生成方法について説明します。

1. **Java パッケージノードを右クリックし、「新規」->「すべてのテンプレート」->「Web サービス」->「Web サービスクライアント」を選択します。**

   または、IDE のメインウィンドウから「ファイル」->「新規」->「すべてのテンプレート」->「Web サービス」->「Web サービスクライアント」を選択します。

   図 3-18 に示す「Web サービスクライアント」ダイアログが表示されます。

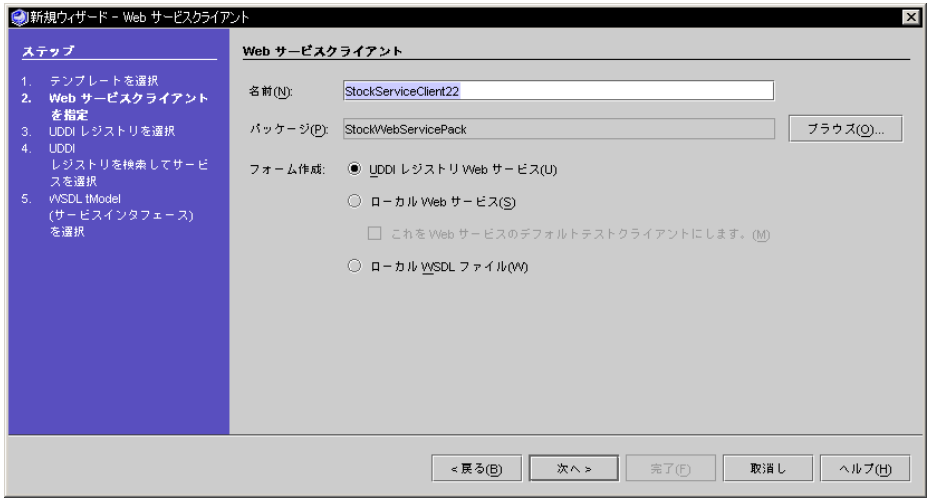

図 3-18　「新規 Web サービスクライアント」ウィザード

2. **「名前」および「パッケージ」を設定し、「ソース」の「UDDI レジストリ Web サービス」ラジオボタンを選択します。**

3. 「次へ」をクリックすると、「UDDI レジストリを選択」ダイアログが表示されます (図 3-19 を参照)。

ダイアログに UDDI レジストリのリストが表示されます。

この時点で目的のレジストリを選択できます。

また、「編集」をクリックして、デフォルトのレジストリを変更するか、レジストリ情報を編集することもできます (50 ページの「IDE 内でのレジストリ情報の編集」を参照してください)。

注 – IDE 内の最初のデフォルトレジストリは、IDE にバンドルされている内部 UDDI レジストリです。このレジストリの詳細については、58 ページの「内部 UDDI レジストリの使用」を参照してください。

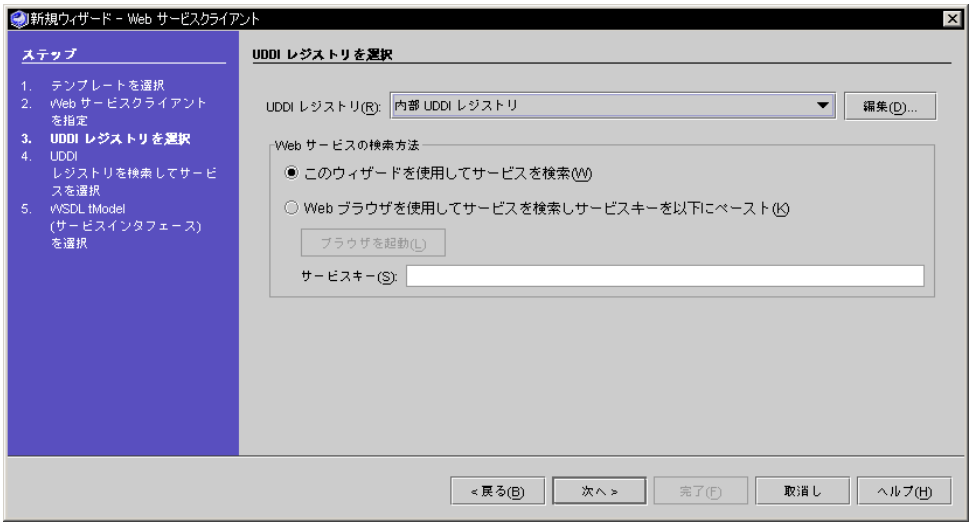

図 3-19　「UDDI レジストリを選択」ダイアログ

IDE では 2 つの方法で Web サービスを検索できます。2 つあるラジオボタンのうち、どちらかをクリックして検索方法を選択します。デフォルトの検索方法は、図 3-19 に示す IDE ウィザードを使用する方法です。もう 1 つの方法は、レジストリの所有者によって提供されている Web ブラウザとブラウザに基づくソフトウェアを使用する方法です。

ほとんどの UDDI レジストリは、レジストリ検索およびレジストリエントリの追加、編集、削除用のブラウザベースのツールを提供します。レジストリのブラウザツールURL は、IDE 内で編集できるプロパティの 1 つです (50 ページの「IDE 内でのレジストリ情報の編集」を参照してください)。

選択したレジストリを独自のブラウザツールで検索する手順は、次のとおりです。

a. **「Web ブラウザを使用してサービスを検索しサービスキーを以下にペースト」のラジオボタンを選択します。**

「ブラウザを起動」ボタンがアクティブになります。

Web サービスのレジストリキーを前の検索で取得している場合は、その値を「サービスキー」フィールドにペーストして「次へ」をクリックします。「完了」をクリックして、中間 UDDI レジストリ検索手順をバイパスしてクライアントを作成できる最後のウィザード手順が表示されます。

b. **「ブラウザを起動」をクリックします。**

デフォルトの Web ブラウザがブラウザツール URL Web ページに表示されます。

c. **ブラウザツールを使用して、クライアントでアクセスする Web サービスの場所を見つけます。**

d. **Web サービスレジストリキーをコピーして「UDDI レジストリを選択」ダイアログの「サービスキー」フィールドにペーストします。**

e. **「次へ」をクリックします。**

「完了」をクリックして、中間 UDDI レジストリ検索手順をバイパスしてクライアントを作成できる最後のウィザード手順が表示されます。

注 – この後の手順では、IDE ウィザードを使用して UDDI レジストリを検索することを前提としています。これは「Web サービスの検索方法」の下のデフォルトのラジオボタンです。

---

4. **「UDDI レジストリ」リストからレジストリを選択し、「次へ」をクリックして、図 3-20 に示す「UDDI レジストリを検索してサービスを選択」ダイアログを表示します。**

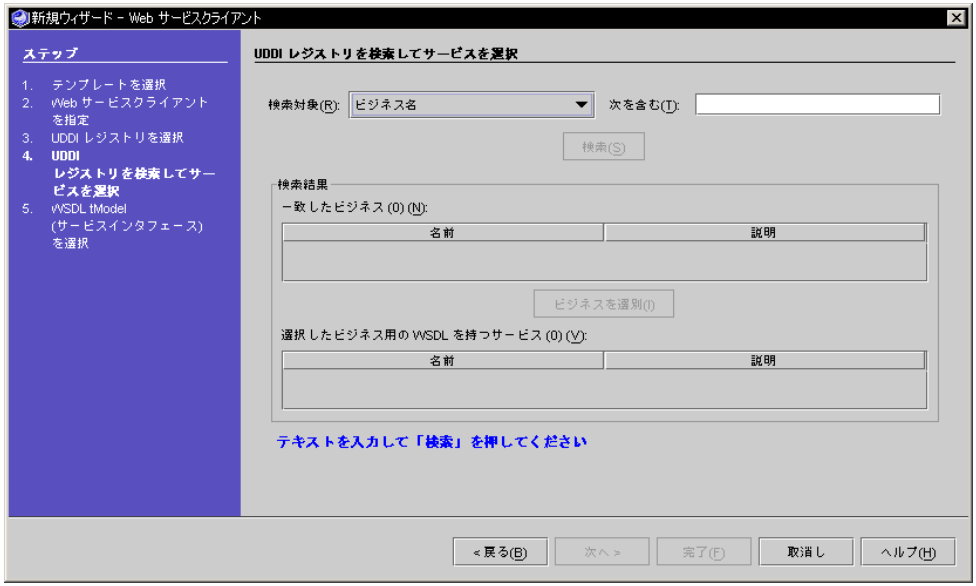

図 3-20　「UDDI レジストリを検索してサービスを選択」ダイアログ

名前に指定した文字列を含むビジネスを検索できます。ビジネス名ではなく、型を基準にして検索することもできます。検索の型のリストを図 3-21 に示します。

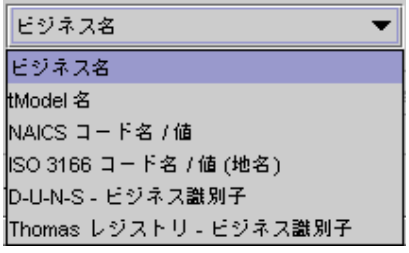

図 3-21　UDDI レジストリ検索タイプ

UDDI レジストリの検索には tModel 名、NAICS コード名、または ISO 3166 コード名を使用できます。

D-U-N-S ビジネス識別子または Thomas レジストリビジネス識別子を検索することもできます。この場合、値「次を含む」ではなく「次と等しい」を検索します。前の検索による識別子が分かっている場合があります。非公開 UDDI レジストリを使用している場合は、プロジェクトまたは企業内の誰かに識別子を聞いてください。

注 – 検索タイプに正確な値が必要な場合は、「UDDI レジストリを検索してサービスを選択」ダイアログ内のラベル (図 3-20) が「次を含む」から「次と等しい」に変わります。

NAICS または ISO 3166 コード名検索の場合は、「次を含む」フィールドに名前またはコードを入力できます。たとえば次のような検索ができます。

- ISO 3166 を選択して `United Kingdom` または `GB` の検索文字列を入力します。
- NAICS を選択して `Other Financial Vehicles` または `52599` の検索文字列を入力します。

tModel 名に対する検索が、検索文字列に合致する tModels に基づくサービスを含むすべてのビジネスを返します。

NAICS Code Name または ISO 3166 Code Name に対する検索が、検索文字列に合致するカテゴリを持つすべてのビジネスを返します。

注 – 「次を含む」フィールドに文字列を指定すると、IDE によって次のように文字列がワイルドカードにラップされます。`%`*yourstring*`%` オプションでさらにワイルドカードを入力することができます。たとえば、名前に文字列 *bio*`%`*tech* を含むビジネスを検索できます。`%` は任意の文字列に一致します。文字列検索は大文字と小文字を区別します。

次の手順では、ビジネス名で検索することを前提としています。

5. **「次を含む」フィールドに文字列を入力して、「検索」をクリックします。**

   Matching Businesses 表には、図 3-22 に示すようにレジストリ名に検索文字列を含むすべてのビジネスが表示されます。各ビジネスは名前とオプションの説明とともに一覧表示されます。一致するビジネスの数も表示されます。

   表からビジネスを選択して、この手順の手順 7 に進みます。

   検索文字列およびレジストリのサイズによっては、検索で多数のビジネスが返される場合があります。これは、次第に数十万から数百万のエントリに増加することが予測される公開レジストリで発生する可能性があります。検索によって返されたリストが大きすぎる場合は、詳細な検索文字列を入力するか、またはレジストリブラウザツールを使用して、より詳細な検索を実行します。

注 – 実際のシナリオでは、開発チームまたはプランナーはどのようなビジネスを対象とするかを理解していると考えられます。レジストリ内のビジネスを検出するときに大切なことは、正しいスペルを取得すること、または (多くの部門、部署およびプロジェクトを持つ巨大企業の場合は) 名前のどの部分がレジストリで使用されているかを判別することです。レジストリ検索機能で情報を検索することもできますが、大抵の場合、こうした情報は適切なプロジェクトリーダーによって管理されています。

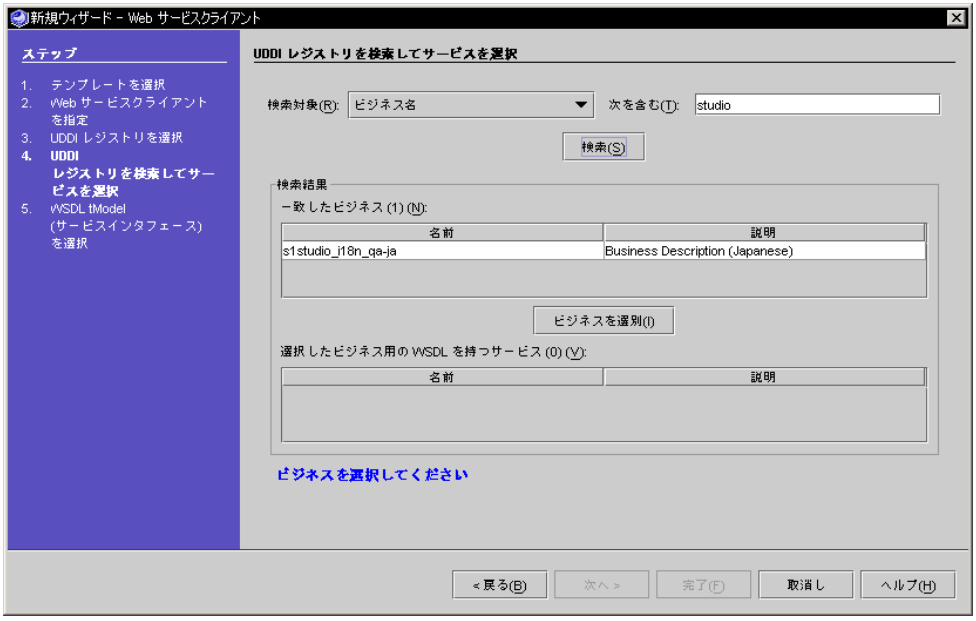

図 3-22　一致するビジネスが表示された「UDDI レジストリを検索してサービスを選択」ダイアログ

6. 「ビジネスを選別」をクリックして検索を絞ります。

レジストリの中には WSDL tModel エントリを持たないサービスもあります。これらのサービスを利用するクライアントを作成することができないため、IDE では「一致したビジネス」表からこれらのサービスを除外することができます。「ビジネスを選別」をクリックすると、IDE は結果セット内のビジネスに関連付けられているサービス用の UDDI レジストリを検索します。IDE は、`http://` で始まり、`wsdl` または `asmx` で終わる「オーバービューURL」フィールドを備えた `OverviewDoc` を持つ tModels 用のサービスを調べます。このテストで不一致だったビジネスは「一致したビジネス」表から除外されます。

IDE は、図 3-23 に示されているように「取消し」ボタンを備えた進捗モニターウィンドウが表示されます。フィルタ処理には多少時間がかかります。時間がかかりすぎる場合は、「取消し」をクリックして、より詳細な検索文字列を使用して検索をやり直すことができます。

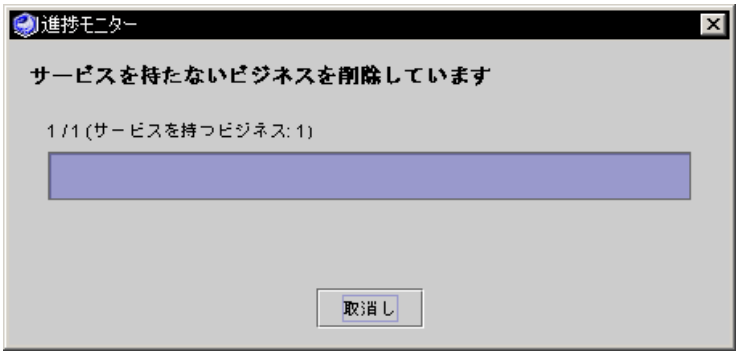

図 3-23　UDDI レジストリフィルタビジネス進捗モニター

7. **「検索結果」リストからビジネスを選択します。**

図 3-24 に示すように、選択したビジネスに関連付けられているサービスが表示され、「次へ」ボタンがアクティブになります。WSDL tModel エントリがあるサービスだけが表示されます。

注 – レジストリには、決定的な助けとなるビジネスまたはサービスに関する十分な情報がない場合があります。公的レジストリは、まだ初期段階で、使用パターンおよび実践が時間経過につれて明らかになる広範囲な Web サービスインフラストラクチャの要素として表示できます。非公開レジストリでは、レジストリの所有者が、サービスを公開するための基準をより詳細に管理することができます。たとえば、非公開レジストリでは、エントリをドキュメント化する基準およびレジストリに公開されるサービスの実行可用性に関する要件がある場合があります。開発およびテスト用のサービスと生産アプリケーションで使用するサービスには区別がある場合があります。

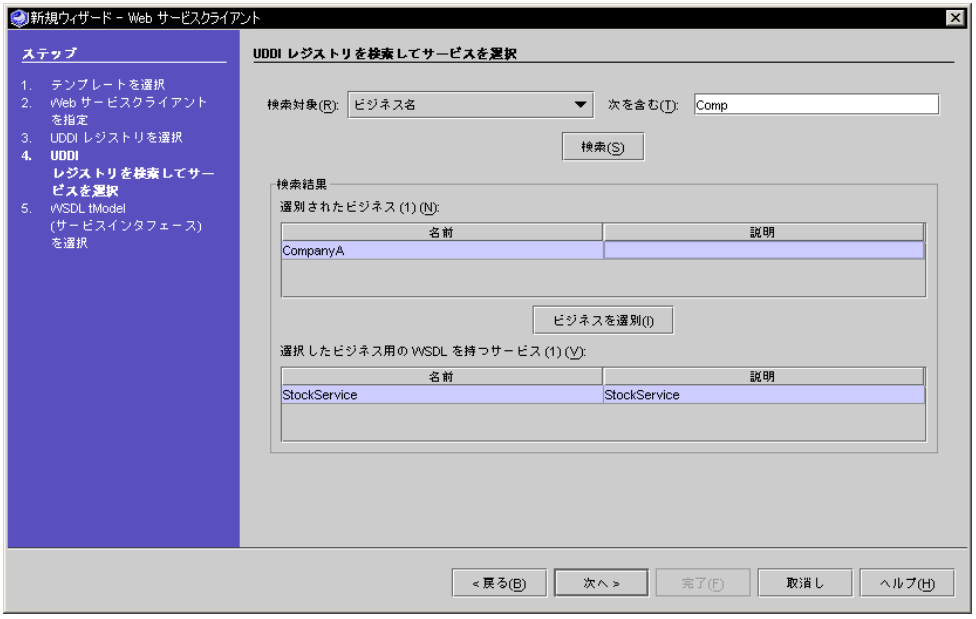

図 3-24　　UDDI レジストリを検索してサービスを選択

8. **サービスを選択して「次へ」をクリックすると、「サービスインタフェース」ダイアログが表示されます (図 3-25 を参照)。**

   ダイアログには、サービスとそれが参照する tModels および WSDL に関する詳細情報が表示されます。

   サービスに対して、名前、ネットワークエンドポイント、およびキーが表示されます。ネットワークエンドポイントは、クライアントが実行サービスインスタンスにアクセスするために使用できる URL です。

   各 WSDL tModel に対して、名前およびオーバービュー URL が表示されます。オーバービュー URL は、IDE がクライアントの作成に使用する WSDL 用のロケータです。

注 – Web サービスプロバイダは、指定サービスインスタンス用の tModels を公開する場合があります。たとえば、1 つの tModel はサービスメソッドを完全に使用できるようにしたり、別の tModel はサービスメソッドのサブセットへのアクセスを提供する場合があります。これは、2 つの異なる tModels から生成されたクライアントに反映されます。

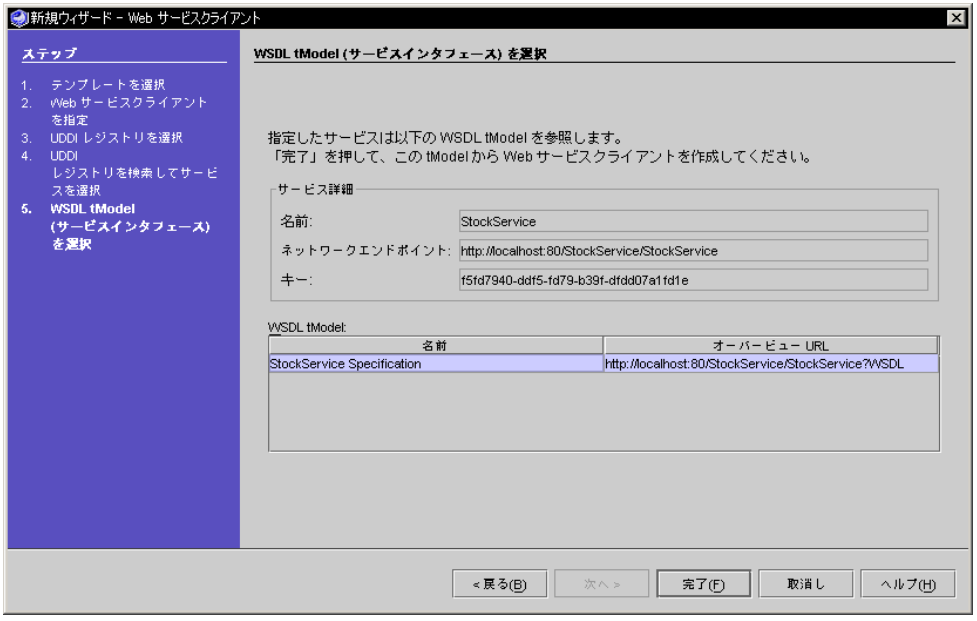

図 3-25　UDDI レジストリ表示サービス詳細と tModel

9. **WSDL tModel を 1 つ選択し、「完了」をクリックします。**

   IDE によって Web サービスクライアントが作成されます。クライアントは、エクスプローラの指定したパッケージの下に新しい「クライアント」ノードとして表示されます。

10. **エクスプローラの「クライアント」ノードを右クリックし、「クライアントファイルを生成」を選択します。**

    IDE によってクライアントファイルが作成されます。

注 – 「クライアントファイルを生成」の代わりに「UDDI から再表示」を選択することもできます。このコンテキストメニューに関する説明は、78 ページの「Web サービスからのクライアントの参照」 を参照してください。

---

11. **配備および実行など、その後の手順は 67 ページの「ローカル Web サービスからのクライアントの作成」と同じです。**

添付ファイル

Web サービスオペレーションの入力パラメータおよび戻り値は、次のどちらかの方法で転送されます。メッセージの SOAP 本体の属性か XML タグとして転送する、またはメッセージに添付ファイルとして転送する、のどちらかです。添付ファイルは、画像など効率的に XML 形式にシリアル化できないものを転送する際に使用します。添付ファイルは、メッセージ内で明確に規定されている、MIME で符号化されている部分にパッケージ化されます。IDE では SOAP の添付ファイル用に WWW コンソーシアム (W3C: World Wide Web Consortium) 標準をサポートしています。

最も一般的に利用される JAX-RPC 形式の添付ファイルとしては、`gif` および `jpeg` データといった画像ファイルが挙げられます。IDE は、画像 (`Java.awt.Image` クラスのオブジェクト) を返すように設計されている「Web 主体」Web サービス用に、画像を認識する JAX-RPC テストクライアントを生成できます。

添付ファイルの例

次の例は、画像と文字列を処理する複数のメソッドを持つ `attTest` と名付けられている Java クラスを利用したものです。`attWS` と名付けられたWeb 主体 Web サービスは、Java クラスのビジネスクラスを持っています。また、`attWSClient` と名付けられた JAX-RPC テストクライアントは、この Web サービスと通信します。

次の例は、Web サービスが配備済みで、サーバーが稼働中であることを前提としています。

コード例 3-1 に Java クラスである `attTest` のソースコードを示します。ビジネスメソッドは次の 4 メソッドです。

- `echoImage`
- `echoMessage`
- `echoXML`
- `getImage`

コード例 3-1 画像を扱う Java クラス

```
/* attTest.java */

package attPack;

     import javax.xml.transform.Source;
     import javax.xml.parsers.*;
     import org.w3c.dom.*;
     import javax.xml.transform.dom.*;
```

コード例 3-1　　画像を扱う Java クラス (続き)

```
     import javax.xml.transform.stream.*;
     import org.xml.sax.*;
     import java.io.*;
     import javax.activation.*;
     import javax.mail.internet.*;

public class attTest {

    /* Creates a new instance of attTest */
    public attTest() {
          }
    // input and output java.awt.Image
     public java.awt.Image echoImage(java.awt.Image image){
     java.awt.Image im = image;
     return im;
     }

     // mixed input
    public String echoMessage(java.awt.Image image, String msg) {
     return msg;
     }

     // input and output xml
     public javax.xml.transform.Source
echoXML(javax.xml.transform.Source src) {
     return src;
     }

     // output java.awt.Image
     public java.awt.Image getImage(){
     //Point to a gif file in your own directory.
     String filename = "C:\\test\\microscope.gif";
     java.awt.Image image =
java.awt.Toolkit.getDefaultToolkit().getImage(filename);
     return image;
     }

}
```

図 3-26 に示したエクスプローラの例では、attWS と名付けられた Web 主体 Web サービスと attWSClient という名前のテストクライアントが表示されています。このテストクライアントは、attTest Java クラスを呼び出すようになっています。

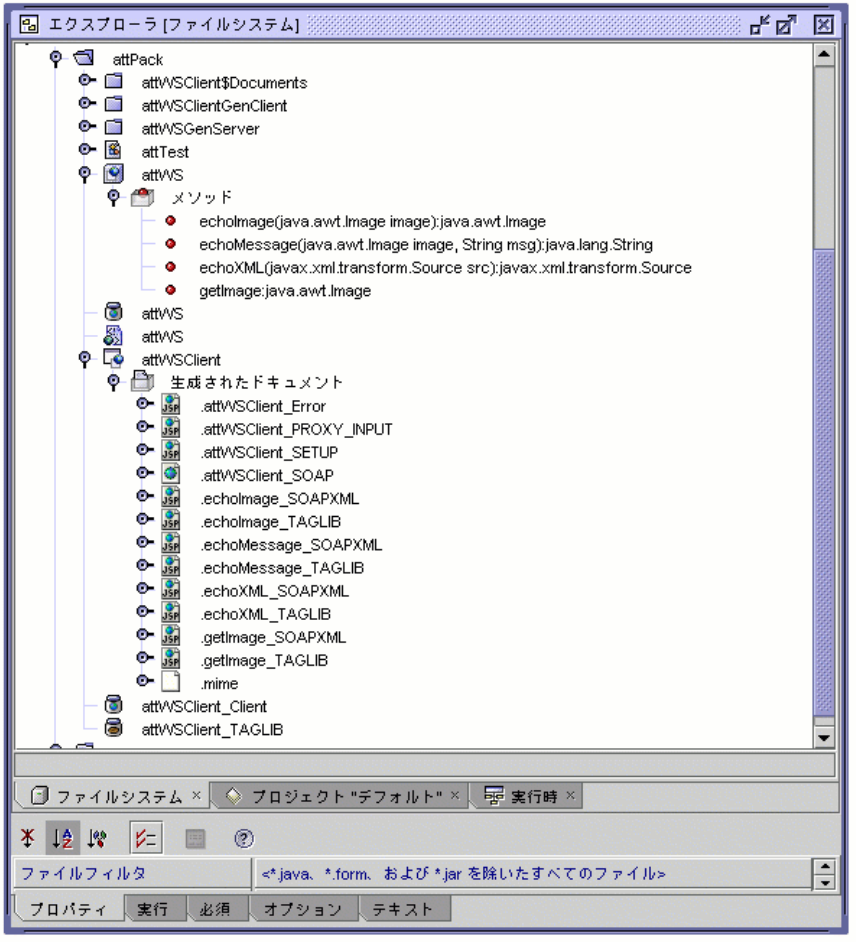

図 3-26　画像と文字列を処理する Web サービスとクライアントを表示したエクスプローラ

Web サービスを右クリックして「実行」を選択すると、IDE によってブラウザが開かれ、クライアントの開始ページが表示されます (図 3-27 を参照)。

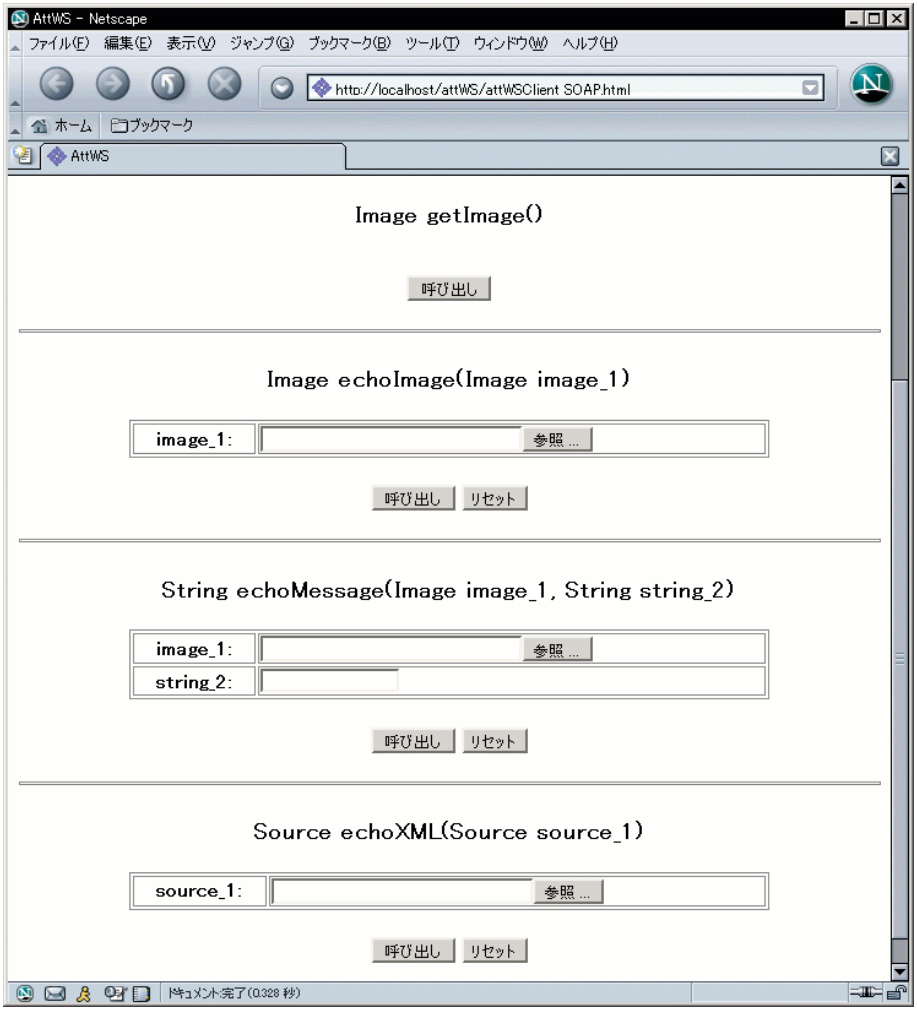

図 3-27　画像と文字列を処理するためのクライアントの開始ページ

`getImage` メソッドのウィンドウから「呼び出し」をクリックすると、クライアントによって図 3-28 に示したページが表示されます。

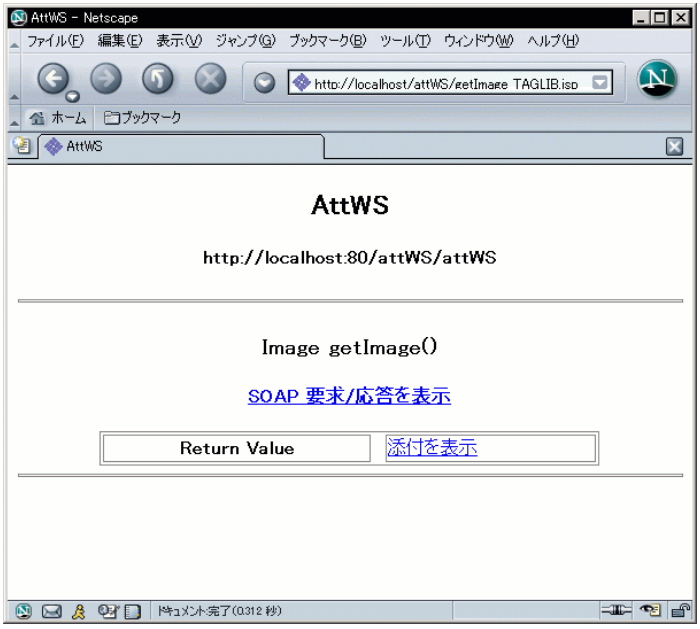

図 3-28　　getImage メソッド用のクライアント JSP ページ

「添付を表示」リンクをクリックすると、クライアントによって新規ウィンドウが開かれ、画像が表示されます (図 3-29 を参照)。

図 3-29　　クライアントによる画像表示

DataHandler プロパティ

JAX-RPC 仕様の 7.2 節では次のように記述されています。

> Java サービスエンドポイントインタフェースのリモートメソッドは、MIME で符号化された内容を次の Java タイプを使って表現することがあります。
>
> - MIME タイプをマップした標準 Java に基づいた Java クラス。この詳細については「7.5 MIME タイプと Java タイプのマッピング」を参照してください。
> - 任意の MIME タイプの内容に対する Java `class javax.activation.DataHandler`。
>
> JAX-RPC 仕様では、さまざまな MIME タイプの内容をサポートするために JavaBean Activation Framework を採用しています。`DataHandler` クラスは、さまざまな MIME タイプで表現されるデータに対応して一貫したインタフェースを提供します。

(JAX-RPC 仕様の URL については 15 ページの「ソフトウェアバージョン」 を参照してください。)

IDE では、生成したクライアントが MIME で符号化された内容を DataHandler クラスを使って表現できるように指定するためのクライアントプロパティが提供されています。これを行うには次の手順に従ってください。

1. **「クライアント」ノードを右クリックし、「プロパティ」を選択します。その後、「DataHandler のみを使用」プロパティを `False` (デフォルト) から`True` に変更します。**
2. **「クライアント」ノードを右クリックし、「クライアントファイルを生成」を選択します。**

図 3-30 に、「DataHandler のみを使用」プロパティが `False` のときに IDE が生成するクライアントクラス attWSRPC を示します。

```
// Helper class generated by xrpcc, do not edit.
// Contents subject to change without notice.

package attPack.attWSClientGenClient;

public interface AttWSRPC extends java.rmi.Remote {
    public java.awt.Image getImage() throws
          java.rmi.RemoteException;
    public java.awt.Image echoImage(java.awt.Image image_1) throws
          java.rmi.RemoteException;
    public java.lang.String echoMessage(java.awt.Image image_1, java.lang.String string_2) throws
          java.rmi.RemoteException;
    public javax.xml.transform.Source echoXML(javax.xml.transform.Source source_1) throws
          java.rmi.RemoteException;
}
```

図 3-30 「DataHandler のみを使用」プロパティが `False` のときのクライアントクラス `attWSRPC`

図 3-31 に、「DataHandler のみを使用」プロパティが True のときに IDE が生成するクライアントクラス attWSRPC を示します。

```
// Helper class generated by xrpcc, do not edit.
// Contents subject to change without notice.

package attPack.attWSClientGenClient;

public interface AttWSRPC extends java.rmi.Remote {
    public javax.activation.DataHandler getImage() throws
        java.rmi.RemoteException;
    public javax.activation.DataHandler echoImage(javax.activation.DataHandler image_1) throws
        java.rmi.RemoteException;
    public java.lang.String echoMessage(javax.activation.DataHandler image_1, java.lang.String string_2) throws
        java.rmi.RemoteException;
    public javax.activation.DataHandler echoXML(javax.activation.DataHandler source_1) throws
        java.rmi.RemoteException;
}
```

図 3-31 「DataHandler のみを使用」プロパティが True のときのクライアントクラス attWSRPC

DataHandler クラスの使用方法については、http://java.sun.com/j2ee/sdk_1.2.1/techdocs/api/javax/activation/DataHandler.html を参照してください。

ステートフル Web サービスとクライアント

ステートフル Web サービスとクライアントの作成については、37 ページの「ステートフル Web サービスの作成」を参照してください。

SOAP メッセージハンドラ

SOAP メッセージハンドラを Web サービスやクライアントと使用する方法については、第 4 章を参照してください。

セキュリティ保護

Web サービスとクライアントに対するセキュリティ保護の実装については、付録 A を参照してください。

第4章

メッセージハンドラの使用

この章では、ハンドラの概要と、IDE を使用して Web サービスとクライアントでハンドラを利用する方法について説明します。

「JAX-RPC メッセージハンドラ」はユーザーによって記述されるクラスで、このハンドラを使用することで RPC 要求または RPC 応答を表す SOAP メッセージを変更できます。ハンドラは、Web サービスまたは Web サービスクライアントと関連付けることができます。ハンドラは、メッセージのヘッダーブロックと本体に対してアクセスできますが、クライアントまたはサーバーアプリケーションのコードにはアクセスできません。ハンドラは通常、ヘッダーブロックを処理し、Web サービスまたはクライアントアプリケーションコードが実行した処理を補足するように設計されます。

「ハンドラチェーン」を使ってハンドラのクラスをまとめると、指定した順序でのハンドラの実行やハンドラ間のデータの共用ができるようになります。最初のハンドラを Web サービスやクライアントに追加すると、IDE によってハンドラチェーンが自動的に作成されます。その後追加するハンドラは、同じハンドラチェーンに組み込まれます。Web サービスまたはクライアント 1 つに対し、最大で 1 つのハンドラチェーンが作成されます。

ハンドラの一般的な利用方法は次のとおりです。

- ログと監査
- 暗号化と復号化
- データのキャッシング

ハンドラのコーディング方法に関する詳細は、`http://java.sun.com/xml/jaxrpc` にある「Java API for XML-Based RPC (JAX-RPC) Specification 1.0」を参照してください。

SOAP メッセージおよびハンドラに関する詳細は、`http://www.w3.org/TR/2000/NOTE-SOAP-20000508` にある「SOAP 1.1 Specification」を参照してください。

注 – この章に記載されているのは高度な操作です。通常の Web サービスおよびクライアントでは、ハンドラを使用する必要はありません。

SOAP メッセージハンドラ

この節では、IDE でのハンドラ機能を理解するためのハンドラの基本的な情報と、より高度な情報を理解するための導入説明を記述しています。ハンドラクラスを実際に記述する場合は、JAX-RPC および SOAP の仕様書に記載されているハンドラに関連する内容も参照してください。

SOAP メッセージハンドラとヘッダーブロック

SOAP ヘッダーは、SOAP メッセージの拡張部分にあたります。ヘッダーは通常、セキュリティ保護や監査といった上位機能のサポートに使用されます。

SOAP メッセージの XML ドキュメントに `Header` 要素がある場合は、それが SOAP `Envelope` 要素の直接の子要素となります。そのヘッダーの直接の子要素がある場合は、その子要素が「ヘッダーエントリ」または「ヘッダーブロック」となります。

コード例 4-1 ヘッダーブロックを 1 つだけ持つSOAP ヘッダー

```
<SOAP-ENV:Header>
    <t:Transaction xmlns:t="some-URI"
    SOAP-ENV:mustUnderstand="1"
    SOAP:actor="www.xyz.com/actor1">
        5
    </t:Transaction>
</SOAP-ENV:Header>
```

コード例 4-1 では、ヘッダーブロックを 1 つだけ持つ SOAP ヘッダーを示しています。`actor` および `mustUnderstand` 属性については、この章で後述します。

要求ハンドラと応答ハンドラ

ハンドラクラスは RPC 要求と RPC 応答を表す SOAP メッセージを処理できます。主な処理は次の 2 つのメソッドが実行します。

```
public boolean handleRequest (MessageContext context) {...}
```

```
public boolean handleResponse (MessageContext context) {...}
```

handleRequest または handleResponse メソッドは、SOAP メッセージの一部を MessageContext パラメタから取得します。これには、エンベロープ、ヘッダー、ヘッダー要素、本体、および本体要素が含まれています。例題コードについてはコード例 4-2 を参照してください。

コード例 4-2　　SOAP メッセージを部分的に抽出するハンドラコードの例

```
public boolean handleRequest (MessageContext context) {

    try {
        SOAPMessageContext smc = (SOAPMessageContext) context;
        SOAPMessage msg = smc.getMessage();
        SOAPPart sp = msg.getSOAPPart();
        SOAPEnvelope se = sp.getEnvelope();
        SOAPHeader shd = se.getHeader();

        SOAPBody sb = se.getBody();
        java.util.Iterator childElems = sb.getChildElements();
        SOAPElement child;
        StringBuffer message = new StringBuffer();
        while (childElems.hasNext()) {
            child = (SOAPElement) childElems.next();
            message.append(new Date().toString() + "--");
            formLogMessage(child, message);

        } ...
```

handleRequest または handleResponse メソッドは、メッセージのヘッダーまたは本体を変更することができます。また、これらのメソッドはヘッダーブロックを「消費」(メッセージからの削除) したり、「追加」したりできます。

例えば、handleRequest または handleResponse メソッドは、外部ログまたは監査ファイルを記述できます。さらに、これらのメソッドは、メッセージの本体を暗号化または復号化するためのルーチンを呼び出すこともできます。ハンドラは通常、メッセージの本体に含まれる要素の解析や変更のためには使用されません。

handleRequest または handleResponse メソッドが実行する処理は、メッセージのヘッダーの内容またはメッセージ本体に含まれるデータによって変わります。

ハンドラは、クライアントとサービスの対話処理中に 4 個所で実行できます。

クライアント側のハンドラ

- クライアントの RPC 要求がサービスに送信される以前に要求ハンドラを起動する場合
- サービスの RPC 応答がクライアントに配信される以前に応答ハンドラを起動する場合

サービス側のハンドラ

- クライアントの RPC 要求がサービスに配信される以前に要求ハンドラを起動する場合
- サービスの RPC 応答がクライアントに送信される以前に応答ハンドラを起動する場合

ハンドラクラスは 1 つだけでも `handleRequest` と `handleResponse` メソッドの両方を持つことができます。

`handleRequest` メソッドのハンドラの例については、126 ページの「ハンドラのコード例」 を参照してください。

注 – JAX-RPC ハンドラクラスは、`javax.xml.rpc.handler.Handler` インタフェースを実装する必要があります。

ハンドラチェーン

コード例 4-3 に Web サービスまたはクライアントが持つハンドラチェーンのスキーマの例を示します。

コード例 4-3 ハンドラチェーンのスキーマの記述

```
handler chain
    roles
    "http://acme.org/auditing"
    "http://acme.org/morphing"
    handler class "acme.MyHandler1"
    headers
        "ns1:foo ns1:bar"
        "ns2:foo ns2:bar"
    properties
        name="property1" value="xyz1"
        name="property2" value="xyz2"
        name="property3" value="xyz3"
    handler class "acme.MyHandler2"
```

この例でのハンドラチェーンには、`acme.MyHandler1` と `acme.MyHandler2` の 2 つのハンドラクラスが含まれています。`acme.MyHandler1` は、3 つのプロパティと 2 つのヘッダーブロックで構成されています。2 つの SOAP actor のロールは、ハンドラチェーンに関連付けられています。

ハンドラプロパティ、ヘッダーブロック、および SOAP actor のロールの利用については、この章で後述します。ハンドラチェーンは IDE が自動的に管理します。特別なコーディングは必要ありません。

SOAP actor のロール

SOAP actor ロールを使用すると、特定のヘッダーブロックがハンドラやメッセージの最終的な受信者 (Web サービスやクライアントなど) によって確実に処理されるように指定することができます。

SOAP `actor` は SOAP ヘッダーブロックの属性であり、コード例 4-1 に示したように、一般的に `mustUnderstand` 属性と組み合わせて使用されます。

あるヘッダーブロックに、次の属性があるとします。

```
SOAP-ENV:mustUnderstand="1"
```

```
SOAP:actor="www.xyz.com/actor1"
```

このヘッダーブロックを持つ SOAP メッセージが actor のロールである `"www.xyz.com/actor1"` を持つハンドラチェーンによって処理された場合、ヘッダーブロックはこのハンドラチェーンの中に含まれるハンドラのどれかか、メッセージの受信先である Web サービスまたはクライアントによって消費されます。ヘッダーブロックが消費されない場合は、受信先 Web サービスまたはクライアントによってビジネスメソッドが呼び出される前に実行時 SOAP が例外をスローします。

`mustUnderstand` 属性が省略された場合、またはこの属性に「`0`」が指定された場合は、ヘッダーブロック中の `actor` 属性の有無に関係なく、SOAP 実行環境はこのヘッダーブロックの消費や利用を強制しません。

SOAP 1.1 Specification では、要素に属性 `mustUnderstand="1"` をタグ付けするのは、要素の目的を知らない開発者が誤ってそれらの要素を無視したりしないため、と説明されています。(ハンドラを記述する開発者と SOAP メッセージとそのヘッダーブロックを設計する開発者が異なっていることもあるためです。)

ハンドラプロパティ

「ハンドラ」プロパティは、ハンドラを構成するために使用できます。プロパティとその値は、初期化時に HandleInfo パラメータを介してハンドラインスタンスに渡されます。コード例 4-6 に「ハンドラ」プロパティの使用例を示します。ハンドラの開発中にプロパティを追加したい場合は、IDE を使用します。(119 ページの「「ハンドラ」プロパティを追加する」を参照してください。)

ログファイルが異なる、または暗号化・復号化のルーチンが異なるなど、プロパティ値だけが異なっている 1 つのハンドラを 2 つの異なる Web サービスまたはクライアントが使用することもあります。

ハンドラの初期化にヘッダーブロックを取得する

ハンドラが実行する処理は RPC 要求または RPC 応答で渡されるヘッダーブロックによって異なります。このヘッダーブロックには、同じハンドラチェーン内で過去に実行されたハンドラがメッセージに追加したヘッダーブロックも含まれます。ハンドラは、複数あるヘッダーブロックのうち、一部のものだけを使用することもあります。

注 – ヘッダーブロックは、SOAP メッセージにヘッダー要素がある場合はこの要素の直接の子要素となります。110 ページの「SOAP メッセージハンドラとヘッダーブロック」を参照してください。

ハンドラを Web サービスまたはクライアントに追加する場合、ハンドラに特定のヘッダーブロックを処理させることができます。IDE では、Web サービスまたはクライアントの開発中にヘッダーブロック名のリストを表示させることができます。(122 ページの「ハンドラへの SOAP ヘッダーブロックの追加」を参照してください。)

IDE に設定するヘッダーブロック名は、 `HandlerInfo` 型のパラメータにあるハンドラに渡されます。コード例 4-4 のコーディングでは、ハンドラが処理対象とするヘッダーの配列を取得する例を示しています QName 配列を使って `requestHandler` または `responseHandler` メソッドを繰り返すことで、ヘッダーブロック名を取得できます。

コード例 4-4 ヘッダーブロックのリストを初期化するハンドラコードの例

```
public class StockServerMailHandler
        implements javax.xml.rpc.handler.Handler
{
    private QName[] headers;

    public StockServerMailHandler () {
    }

    public void init(HandlerInfo config) {
        headers = config.getHeaders();
    }
```

RPC 要求または RPC 応答中にある SOAP 実行時メッセージには、複数のヘッダーブロックが含まれていることもあります。ハンドラは、`handleRequest` メソッドおよび `handleResponse` メソッド中の `Message Context` 型のパラメタを介してこれらのヘッダーブロックをすべて取得できます (コード例 4-7 を参照)。逆に、初期化中に Handler Info パラメタから収集した名前を持つヘッダーブロックだけを処理するように、ハンドラを記述することもできます。

SOAP メッセージへのヘッダーブロックの追加

ハンドラは、新しいヘッダーに主だった情報を提供するわけではありません。しかし、SOAP メッセージへのヘッダーブロックの追加が欠かせない場合がいくつかあります。たとえば、複数のハンドラによる特定のヘッダーブロックの処理、またはハンドラ 1 つと受信先 Web サービスまたはクライアントによる特定のヘッダーブロックの処理が必要だったとします。ここで問題となるのは、あるヘッダーブロックがハンドラによって消費されてしまうと、その時点でその SOAP メッセージからは削除されてしまうという点です。これを解決するために、ヘッダーブロックをコピーしておいてから消費し、ヘッダーブロックのコピーを SOAP メッセージに追加して、その後メッセージをハンドラチェーンに渡すというようにハンドラを記述することができます。

ヘッダーブロックをメッセージに追加する例をコード例 4-5 に示します。ヘッダーブロックは必ずヘッダー要素の子要素とします。これは、メッセージにヘッダー要素がない場合に、コードによって要素を追加できるようにするためです。

コード例 4-5　　SOAP メッセージにヘッダーを追加するハンドラの記述名

```
package stock;

import java.util.Hashtable;
import javax.xml.rpc.handler.MessageContext;
import javax.xml.rpc.handler.HandlerInfo;
import javax.xml.rpc.handler.soap.SOAPMessageContext;
import javax.xml.soap.*;
import java.util.*;

public class AddHeaderHandler
        implements javax.xml.rpc.handler.Handler {

    private Map props;

    public AddHeaderHandler () {
    }
    public void init(HandlerInfo config) {
        props = config.getHandlerConfig();
    }
    public boolean handleFault(MessageContext context) {
    return true;
    }
        public boolean handleRequest (MessageContext context) {

     try {
        SOAPMessageContext smc = (SOAPMessageContext) context;
        SOAPMessage msg = smc.getMessage();
        SOAPPart sp = msg.getSOAPPart();
        SOAPEnvelope se = sp.getEnvelope();
```

コード例 4-5　　SOAP メッセージにヘッダーを追加するハンドラの記述名 (続き)

```
        SOAPHeader sh = se.getHeader();
        if(sh == null){
          sh = se.addHeader();
        }
        //add header elements
                          Name headerName =
se.createName("myNewHeader");
               SOAPHeaderElement she =
sh.addHeaderElement(headerName);
     }
    catch (Exception ex) {
        //ex.printStackTrace();
    }
        return true;
    }
    public boolean handleResponse (MessageContext context) {
    return true;
    }


    public javax.xml.namespace.QName[] getHeaders() {
      return null;
    }

    public void destroy() {
    }
}
```

SOAP メッセージには、WSDL コードを介してヘッダー要素を追加することもできます。次の URL にある WSDL 1.1 Specification を参照してください。
`http://www.w3.org/TR/wsdl#_soap:header.`

IDE でのハンドラの使用

この節での手順は、Web サービス、クライアント、およびハンドラクラスが IDE で作成済みになっていることを前提に説明されています。

Web サービスまたはクライアントへのハンドラの追加

Web サービスまたはクライアントにハンドラクラスを追加する手順は次のとおりです。

1. 「Web サービス」ノードまたは「クライアント」ノードを右クリックし、「プロパティ」を選択します。

   「SOAP メッセージハンドラ」プロパティが、現在 Web サービスに関連付けられているハンドラの数と共に表示されます (図 4-1 を参照)。

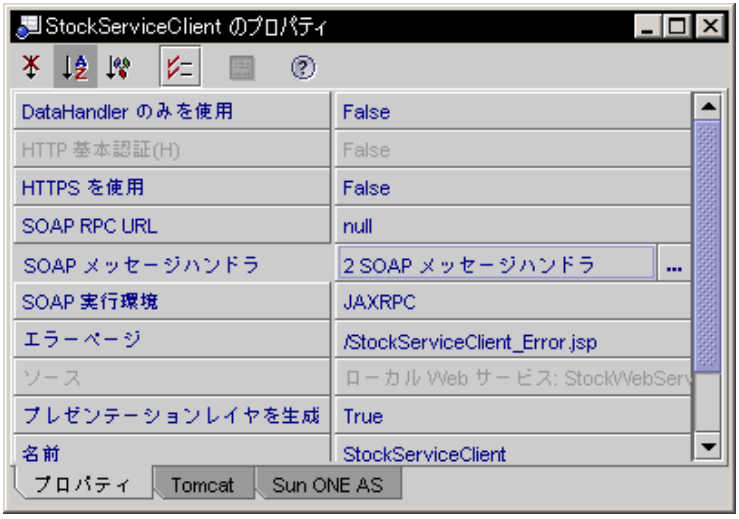

図 4-1　「SOAP メッセージハンドラ」プロパティ

2. 「SOAP メッセージハンドラ」プロパティの値をクリックし、省略符号ボタン (...) をクリックします。

   IDE によって「SOAP メッセージハンドラ」ダイアログが表示されます (図 4-2 を参照)。この例では、Web サービスに対し 2 つのハンドラクラスがすでに関連付けられています。

図 4-2　「SOAP メッセージハンドラ」ダイアログ

「SOAP メッセージハンドラ」ダイアログは、ハンドラとハンドラチェーンを管理するための機能を提供します。

- 「追加」ボタンと「削除」ボタンを使うと、Web サービスまたはクライアントに関連付けられているハンドラチェーンからハンドラクラスを追加したり、削除したりできます。
- 「上へ移動」ボタンおよび「下へ移動」ボタンを使うと、ハンドラチェーン内でのハンドラクラスの実行順序を変更できます。表示されているリストの最上部にあるハンドラが最初に実行されます。
- 「変更」プロパティでは、ハンドラにプロパティ名と値を追加できます。
- 「SOAP ヘッダーを変更」ボタンを使うと、ハンドラに処理させたい SOAP ヘッダーブロックを指定することができます。
- 「SOAP actor のロール」ボタンを使うと、1 つ以上の SOAP actor のロールをハンドラチェーンに関連付けることができます。

これらの各機能については、この章で説明しています。

3. **「追加」をクリックし、選択ダイアログで必要なハンドラを 1 つ以上選択して「了解」をクリックします。**

選択したハンドラが「SOAP メッセージハンドラ」ダイアログに表示され、指定した Web サービスまたはクライアントのハンドラチェーンの一部となります。

注 – JAX-RPC ハンドラクラスは、`javax.xml.rpc.handler.Handler` 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を持たないハンドラを追加しようとすると、IDE によってエラーメッセージが表示されます。

「ハンドラ」プロパティを追加する

プロパティとその値をハンドラに追加する手順は次のとおりです。

1. **Web サービスノードまたはクライアントノードを右クリックし、「プロパティ」を選択します。**

   「SOAP メッセージハンドラ」プロパティが、現在 Web サービスに関連付けられているハンドラの数と共に表示されます (図 4-1 を参照)。

2. **「SOAP メッセージハンドラ」プロパティの値をクリックし、省略符号ボタン (...) をクリックします。**

   IDE によって「SOAP メッセージハンドラ」ダイアログが表示されます (図 4-3 を参照)。この例では、Web サービスにはハンドラクラスが 1 つだけあります。

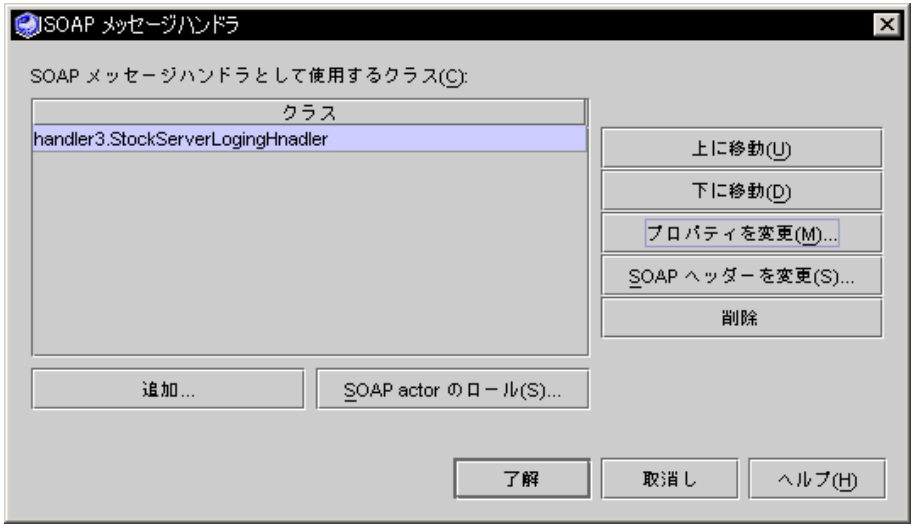

図 4-3　「SOAP メッセージハンドラ」ダイアログ

3. **ハンドラを強調表示させて、「変更」プロパティをクリックします。**

   「プロパティを設定」ダイアログが表示されます (図 4-4 を参照)。この例では、ハンドラには `logFile` というプロパティが設定されていて、プロパティ値はハンドラがログ情報を書き込むファイルの完全限定名となっています。

図 4-4　SOAP メッセージハンドラの「プロパティを設定」ダイアログ

ダイアログには、選択したハンドラのプロパティとプロパティ値が表示されます。ここでプロパティを削除または追加できます。

4. 「追加」をクリックします。

「プロパティを追加」ダイアログが表示されます (図 4-5 を参照)。この例では、ハンドラには `audiFile` というプロパティが追加され、プロパティ値はハンドラが監査情報を書き込むファイルの完全限定名となっています。

図 4-5　SOAP メッセージハンドラの「プロパティを追加」ダイアログ

5. プロパティ名と値を入力して、「了解」をクリックします。

「プロパティを設定」ダイアログに新規プロパティとその値が表示されます (図 4-6 を参照)。

図 4-6　　SOAP メッセージハンドラの「プロパティを設定」ダイアログ

`logFile` プロパティの値が設定されたハンドラを初期化するハンドラのコード例についてはコード例 4-6 を参照してください。

コード例 4-6　　`logFile` プロパティ値を使用するハンドラのコード例

```
public class StockServerLoggingHandler
        implements javax.xml.rpc.handler.Handler
{
    private PrintStream pout;
    private QName[] headers;

    public StockServerLoggingHandler () {
    }
    public void init(HandlerInfo config) {
        Map props = config.getHandlerConfig();
        headers = config.getHeaders();
//Get the value of the property "logFile"
        String logFileName = null;
        if(props != null)
          logFileName = (String)props.get("logFile");

        File logFile = new File(logFileName);
        if(!logFile.exists()){
            try{
                logFile.createNewFile();
            }
            catch(Exception e){
                e.printStackTrace();
            }
        }
        try{
```

コード例 4-6　　logFile プロパティ値を使用するハンドラのコード例 (続き)

```
            FileOutputStream out = new FileOutputStream(logFile,
true);
            pout = new PrintStream(out);
          }catch(IOException e){
              e.printStackTrace();
          }
      }
```

ハンドラへの SOAP ヘッダーブロックの追加

ハンドラに SOAP ヘッダーブロックを追加する手順は次のとおりです。

1. **Web サービスノードまたはクライアントノードを右クリックし、「プロパティ」を選択します。**

   「SOAP メッセージハンドラ」プロパティが、現在 Web サービスに関連付けられているハンドラの数と共に表示されます (図 4-1 を参照)。

2. **「SOAP メッセージハンドラ」プロパティの値をクリックし、省略符号ボタン (...) をクリックします。**

   IDE によって「SOAP メッセージハンドラ」ダイアログが表示されます (図 4-3 を参照)。

3. **ハンドラを強調表示させて、「SOAP ヘッダーを変更」をクリックします。**

   「SOAP ヘッダーを設定」ダイアログが表示されます (図 4-7 を参照)。

図 4-7　　「SOAP ヘッダーを設定」ダイアログ

ダイアログに、選択したハンドラの SOAP ヘッダーブロック (SOAP ヘッダー QName) が表示されます。ここで QName を削除または追加できます。

4. 「追加」をクリックします。

図 4-8 に示すダイアログが表示されます。

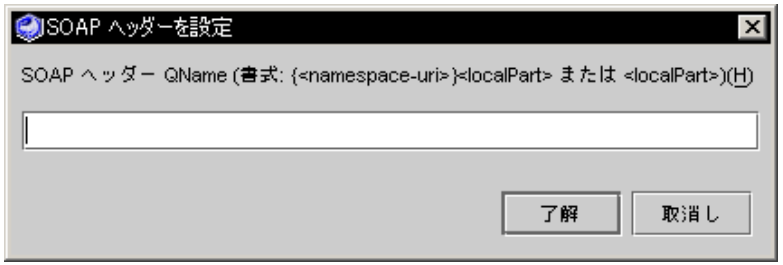

図 4-8 「SOAP ヘッダーを設定」ダイアログ

5. SOAP ヘッダー QName を入力し、「了解」をクリックします。

新規 SOAP ヘッダーブロック名が「SOAP ヘッダーを設定」ダイアログに表示されます (図 4-7 を参照)。

注 – ヘッダーブロックが名前空間の修飾子である場合は、{*namespace-uri*}*localpart* の構文に従って入力してください。この構文は図 4-8 に示した「SOAP ヘッダーを設定」ダイアログに表示されます。{`foo/myNS`}`header1` などがそうです。

これらのヘッダー名は、ハンドラインスタンスが実行時に構成されるときに使用できます。この機能に関する詳細と使用方法を示したコード例については、114 ページの「ハンドラの初期化にヘッダーブロックを取得する」を参照してください。

注 – この SOAP ヘッダーブロック名のリストは、開発時に Web サービスまたはクライアントに対して設定する静的な情報で、ハンドラが実行時に初期化される際にハンドラに渡されます。ハンドラの実行時にヘッダーブロックを SOAP メッセージに動的に追加させる方法については、115 ページの「SOAP メッセージへのヘッダーブロックの追加」を参照してください。

SOAP actor のロールをハンドラチェーンに設定する

IDE を使用すると、ハンドラチェーンに 1 つ以上の SOAP actor のロールを関連付けることができます。SOAP actor の利用方法については、112 ページの「ハンドラチェーン」, 113 ページの「SOAP actor のロール」および 125 ページの「ヘッダーブロックの処理を強制する」を参照してください。

1. **「Web サービス」ノードまたは「クライアント」ノードを右クリックし、「プロパティ」を選択します。**

   「SOAP メッセージハンドラ」プロパティが、現在 Web サービスに関連付けられているハンドラの数と共に表示されます (図 4-1 を参照)。

2. **「SOAP メッセージハンドラ」プロパティの値をクリックし、省略符号ボタン (...) をクリックします。**

   IDE によって「SOAP メッセージハンドラ」ダイアログが表示されます (図 4-3 を参照)。

   SOAP actor ロールは、個々のハンドラではなく、ハンドラチェーンに設定します。

3. **「SOAP actor のロール」をクリックします。**

   IDE によって「SOAP actor のロールを構成」ダイアログが表示されます (図 4-9 を参照)。

   ダイアログに、ハンドラチェーンに設定されている SOAP actor のロールが表示されます。ここで、actor のロールを追加または削除できます。

図 4-9　「SOAP actor のロールを構成」ダイアログ

4. **「追加」をクリックします。**

   図 4-10 に示すダイアログが表示されます。

図 4-10　「SOAP actor のロールを構成」ダイアログ

5. 設定したい SOAP actor のロールを入力し、「了解」をクリックします。

   SOAP actor のロールは、有効な URI で指定してください。

   新規 SOAP actor のロールが「SOAP actor のロールを構成」ダイアログに表示されます
   (図 4-11 を参照)。

図 4-11　「SOAP actor のロールを構成」ダイアログ

ヘッダーブロックの処理を強制する

この節では、特定の SOAP actor が必要とするヘッダーをハンドラチェーンに確実に処理させる方法について説明します。

ハンドラチェーンは、1 つ以上の SOAP actor のロール (有効な URI として表現されます) で動作するように構成できます。ハンドラチェーン内のハンドラを構成して、指定した限定名 (QName) を持つヘッダーブロックを処理させ、そのヘッダーブロックをハンドラの getHeaders() メソッドに返させるようにできます。

SOAP メッセージの処理時には、ハンドラチェーンは次の操作を実行します。

1. ハンドラチェーンが実行する actor のロール (構成済みの actor から選択) を決定します。また、ハンドラチェーン内の各ハンドラが処理するすべてのヘッダーブロックを追跡します。
2. SOAP メッセージにヘッダーブロックがある場合は、そのヘッダーブロックが構成済みの actor に必須であるかどうかを調べます。必須である場合は、ハンドラチェーンの処理対象とします。必須であるかどうかは、ヘッダーブロックにある actor および mustUnderstand 属性の値を参照することで確認します。
3. 各必須ヘッダーブロックがハンドラの処理対象となっているリスト内に含まれているかを確認します。リストに入っているヘッダーブロックは、ハンドラチェーンによって処理対象であると判断されます。必須ヘッダーブロックが、ハンドラチェーンによって処理対象でないと判断された場合は、SOAPFaultException がスローされます。
4. 個々のハンドラを起動して、該当するハンドラチェーンに対応するハンドラブロックを処理します。

ハンドラチェーンに必須ヘッダーブロックを強制的に処理させる手順は次のとおりです。

1. **ハンドラチェーンに適切な actor のロールを追加します。**

   この手順は、123 ページの「SOAP actor のロールをハンドラチェーンに設定する」で説明します。

2. **特定のヘッダーブロックに対して各ハンドラを構成し、ハンドラの処理対象となるヘッダーを指定します。**

   この手順は、122 ページの「ハンドラへの SOAP ヘッダーブロックの追加」で説明します。

3. **ハンドラのコードに、指定するヘッダーブロックを getHeaders() メソッドに返すよう記述します。**

   ヘッダーブロックは、ハンドラの `init()` メソッドに渡される `HandlerInfo` から取得します。

4. **ハンドラブロックをハンドラコード内で処理します。**

ハンドラのコード例

コード例 4-7 に記述されている Web サービスハンドラは、RPC 要求メッセージを次のように処理します。

1. ハンドラの初期化中に init メソッドが、ログメッセージの出力先ファイルの名前を指定するプロパティを取得します。
2. handleRequest メソッドは、日付、時刻、および要求に含まれるデータで構成されるログメッセージを作成します。
3. その後、handleRequest メソッドがログメッセージを出力先ログファイルに書き込みます。

コード例 4-7 handleRequest メソッドを持つハンドラの例

```
package stock;

import java.util.Hashtable;
import javax.xml.rpc.handler.MessageContext;
import javax.xml.rpc.handler.HandlerInfo;
import javax.xml.rpc.handler.soap.SOAPMessageContext;
import javax.xml.soap.*;
import java.io.*;
import javax.swing.*;
import java.util.*;
import javax.xml.namespace.*;

public class StockServerLoggingHandler
        implements javax.xml.rpc.handler.Handler
{
    private PrintStream pout;
    private QName[] headers;

    public StockServerLoggingHandler () {
    }

    public void init(HandlerInfo config) {
        Map props = config.getHandlerConfig();
        headers = config.getHeaders();

//Get the value of the property "logFile"

        String logFileName = null;
        if(props != null)
          logFileName = (String)props.get("logFile");

//if logFile property is not defined, create default directory and log file name

     if(props == null || !props.containsKey("logFile") || logFileName == null
        || logFileName.length() == 0){
     logFileName = System.getProperty("user.home") + File.separator
        + "stockhandler.log";
        }
```

コード例 4-7　　handleRequest メソッドを持つハンドラの例 (続き)

```
// create log file if it does not already exist

        File logFile = new File(logFileName);
        if(!logFile.exists()){
            try{
                logFile.createNewFile();
            }
            catch(Exception e){
                e.printStackTrace();
            }
        }

// associate output print stream with log file

        try{
          FileOutputStream out = new FileOutputStream(logFile, true);
          pout = new PrintStream(out);
        }catch(IOException e){
            e.printStackTrace();
        }
    }

    public boolean handleFault(MessageContext context) {
      return true;
    }

  public boolean handleRequest (MessageContext context) {

// get SOAP message and extract the envelope, header, body, and body elements

    try {
        SOAPMessageContext smc = (SOAPMessageContext) context;
        SOAPMessage msg = smc.getMessage();
        SOAPPart sp = msg.getSOAPPart();
        SOAPEnvelope se = sp.getEnvelope();
        SOAPHeader shd = se.getHeader();

        SOAPBody sb = se.getBody();
        java.util.Iterator childElems = sb.getChildElements();
        SOAPElement child;
        StringBuffer message = new StringBuffer();

// iterate through body elements, formatting a log message for each element

        while (childElems.hasNext()) {
            child = (SOAPElement) childElems.next();
```

コード例 4-7　　handleRequest メソッドを持つハンドラの例 (続き)

```
            message.append(new Date().toString() + "--");
            formLogMessage(child, message);
        }

// extract headers

        if(shd != null){
            java.util.Iterator iter =
                shd.extractHeaderElements(SOAPConstants.URI_SOAP_ACTOR_NEXT);
            while(iter.hasNext()){
                SOAPHeaderElement el = (SOAPHeaderElement)iter.next();
                Name n = el.getElementName();

                for(int i = 0; i < headers.length; i++)
           {
           QName header = headers[i];
                    String qNameSpace = header.getNamespaceURI();
                    String nNameSpace = n.getURI();
           //System.out.println("Name.getURI(): " + nNameSpace);
                    String qLocalPart = header.getLocalPart();
                    String nLocalPart = n.getLocalName();

// print headers

                    if(qNameSpace != null &&
           nNameSpace != null &&
           qNameSpace.equals(nNameSpace) &&
                        qLocalPart.equals(nLocalPart) )
           {
                        message.append(" " + n.getQualifiedName() + ":");
               Iterator iter2 = el.getChildElements();
                        while(iter2.hasNext())
               {
               Object sel = iter2.next();
                            if(sel instanceof SOAPElement)
               {
                   message.append(" " + ((SOAPElement)sel).
                       getElementName().getLocalName()+ ": ");
                            }
               else if(sel instanceof Text)
                            {
                                message.append(((Text)sel).getValue());
               }
               }
           }
           }
            }
```

コード例 4-7　　handleRequest メソッドを持つハンドラの例 (続き)

```
        }
        pout.println(message.toString());
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
    return true;
  }

// format a log message

    private void formLogMessage (SOAPElement child, StringBuffer message) {
        message.append(child.getElementName().getLocalName());
        message.append(child.getValue() != null ? ":" + child.getValue() + " "
    : " ");

    try{
    java.util.Iterator childElems = child.getChildElements();
    while (childElems.hasNext()) {
    Object c = childElems.next();
    if(c instanceof SOAPElement)
    formLogMessage((SOAPElement)c, message);
        }
       }catch(Exception e){
           e.printStackTrace();
       }
    }

    public boolean handleResponse (MessageContext context) {
        return true;
    }

    public javax.xml.namespace.QName[] getHeaders() {

// return headers configured on the node

      return headers;
    }

    public void destroy() {
          pout.close();
    }
}
```

第5章

XML オペレーションの開発 (非推奨)

XML オペレーションは、Web サービスアクセス用に設計されていない既存のビジネスコンポーネント用の Web サービスインタフェースを作成する効率的な方法を提供します。

たとえば、顧客からの注文を受け付けるために、ビジネスコンポーネントとして在庫のチェック、顧客信用チェック、オーダーの出荷、および請求書の処理を行うメソッドを持っている Web サービスを作成しているとします。これらのメソッドを単一の XML オペレーションとして組み合わせて注文を受け付けるようにすることができます。

注 – XML オペレーションは IDE の本リリースでは非推奨となっていて、今後のリリースではサポートされないことがあります。状況に応じて、直接メソッドを呼び出す、ステートフル Web サービスやクライアントを使うなどして、XML オペレーションなしでもアプリケーションは作成できます。Web サービス機能の進化に伴い、XML オペレーションの必要性は減少してきています。

この章では、XML オペレーションの概念、ツールの説明、およびオペレーションの作成と編集について説明します。

XML オペレーションの概要

IDE を使用すると、コーディングを追加することなく既存の J2EE コンポーネントに基づいて複雑なアプリケーションを作成できます。次のようなことができます。

- 異なるタイプの複数ビジネスコンポーネントの機能を組み合わせるWeb サービスを作成します。

- ビジネスコンポーネント上で複数メソッドを希望の順序で選択して呼び出し、戻り値を 1 つのメソッドから別のメソッドに渡す Web サービスを作成します。

EJB コンポーネントは、コンポーネントの内部論理を理解して管理できるクライアントによって使用されることを想定して設計されています。このメソッドは、Web サービスとして直接利用されるには論理設計上のレベルが下位であるか、または目的のオブジェクトを目的の形式で返すには、異なるコンポーネント上のいくつかのメソッド呼び出しを必要とする場合があります。

Web サービスモデルで設計された新しいビジネスコンポーネントは、適切な上位レベル機能のみを提供するメソッドを持っている場合があります。ただし、既存のビジネスコンポーネントまたは Web サービスの基本として設計されていないコンポーネントは、必要な上位レベルメソッドを持っていない場合があります。

IDE は、この問題を仲介の役割を果たす「XML オペレーション」を提供することによって解決します。XML オペレーションは複数の EJB メソッドを上位ビジネス関数として 1 つにまとめ、特定の Web サービスに提供することができます。コードを直接操作しないエディタを使用して、XML オペレーションを定義する方法をこの章で後述します。

XML オペレーションとは

XML オペレーションは、いくつかのビジネスメソッドをカプセル化することができます。外部クライアントからは、オペレーションは Web サービス内への単一の RPC 呼び出しのように見えます。

XML オペレーションは、特定の Web サービス要求が処理される方法を指定する論理エンティティです。すべての XML オペレーションを作成して、他のコンポーネントのビジネスメソッドとともに Web サービスに追加します。次に Web サービスの実行クラスを IDE 内に生成します。この操作により、EJB セッション Bean および各 XML オペレーションに対して 1 つのクラスが作成されます。クライアントが Web サービスに要求を送信すると、要求はセッション Bean 上でメソッド呼び出しに変換されます。要求はビジネスメソッドへの単一の直接呼び出しとして、または XML オペレーションによって定義されるメソッド呼び出しの複雑なセットとして処理されます。

単一の XML オペレーション内のすべてのメソッドは同じ状態にあり、オペレーションが継続している間、データを共有できます。

要約すると次のようになります。

- Sun Web サービス内の XML オペレーションは、上位ビジネスメソッドの役割を果たす
- RPC 呼び出しは、Web サービスの一部となる生成された EJB コンポーネント内のメソッドとして実装される
- 生成された EJB コンポーネントは、ビジネスコンポーネントのさらに下位レベルコンポーネントメソッドに対するクライアントとして機能する

- 各 XML オペレーション要求は単一のトランザクション内で実行され、単一のユニット内で複数 EJB メソッド呼び出しのグループ化を可能にする

XML オペレーション機能は、手動コーディングと比較して主に次のような利点があります。

- コーディングなしのエディタを使用することで、Java 言語の経験が少なくても XML オペレーションを構築できます。アプリケーションのコンポーネントに対する理解、および Java メソッドの仕組みに関する知識があれば充分です。
- 手動でステートレスセッション Bean を設計、コーディング、テストする必要がなく、既存のビジネスコンポーネントに対するサービスインタフェースを簡単に変更できます。
- コードレスエディタで XML オペレーションを編集し、サービスコンポーネント用の実行クラスを再生成することによってサービスインタフェースを変更できます。

要求応答メカニズム

Sun Web サービスは、2 つの機能を提供します。これらは、簡易な機能と複雑な機能として理解できます。ビジネスコンポーネントへの直接メソッド呼び出しと XML オペレーションの 2 つが、それに相当します。Web サービスを使用しているクライアント側から見ると、XML オペレーションがいくつかのメソッド呼び出しをカプセル化しているため、この 2 つの機能は類似しています。各 XML オペレーションは、特定のクライアント要求メッセージに対する応答を定義します。Web サービス開発者は、XML オペレーションを定義して既存のコンポーネントから生成します。

Web サービスがクライアント要求を受信すると、要求を XML ドキュメント (XML入力ドキュメント) の形式で適切な XML オペレーションに転送します。XML オペレーションは、ビジネスコンポーネント上で 1 つまたは複数のメソッドを呼び出し、これらのメソッド呼び出しの戻り値を XML ドキュメント (XML 出力ドキュメント) に変換してドキュメントをクライアントに返します。

XML オペレーションが実行されると、Web サービスは次を実行します。

1. XML 入力ドキュメントを解析し、XML オペレーションが呼び出すように定義されているメソッドのパラメータにドキュメントの要素をマッピングする
2. XML オペレーションに定義されているメソッドを指定された順序で呼び出す

   メソッドからの戻り値は、別のメソッドへの入力として渡すことができ、高度なオペレーションを構築できます。
3. メソッドの戻り値を XML オペレーションの定義に従って XML 出力ドキュメントにフォーマットする
4. XML 出力ドキュメントを返す

たとえば、図 5-1 は、3 つの異なるオブジェクト上でメソッドを呼び出す ProductName という XML オペレーションを示しています。

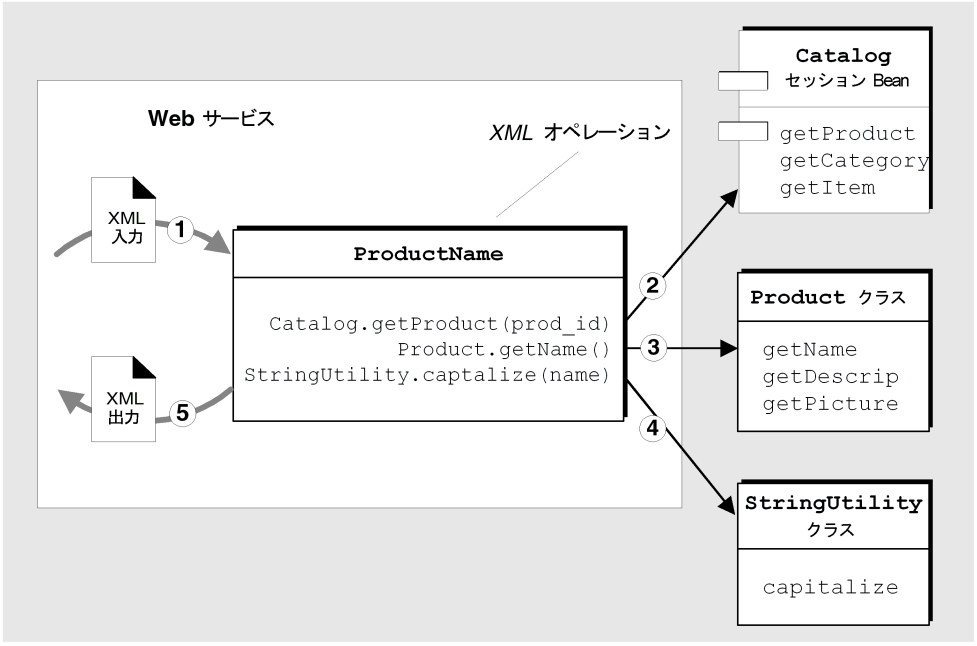

図 5-1 クライアント要求を満たすために複数のメソッドを呼び出す XML オペレーション

ProductName XML オペレーションは、製品 ID を要求パラメータとして受け取り、対応する製品名を大文字で返します。実行時には次の処理が行われます。

1. ドキュメントの「prod_id 要素」の値を Catalog.getProduct メソッドへの入力パラメータとして使用し、XML 入力ドキュメントを解析する
2. Catalog.getProduct メソッドを呼び出し、メソッドは Product クラスのインスタンスを返す
3. getName メソッドを Product オブジェクト上で呼び出し、メソッドは製品名を含む String オブジェクトを返す
4. Static (静的) メソッド StringUtility.capitalize を呼び出し、製品名をパラメータとして含む String オブジェクトを渡す。このメソッドは、最初に大文字でフォーマットされた製品名を含む String オブジェクトを返す。
5. 大文字で示された製品名を XML ドキュメントとして含む String オブジェクトを、フォーマットして返す

ツールの概要

XML オペレーションの開発に使用するツールは、エクスプローラおよびソースエディタから利用できます。

XML オペレーション定義を作成する最初の手順として、オペレーションを作成する場所となるノードをエクスプローラで選択します。ノードを右クリックし、「テンプレートから新規作成」を選択します。この手順は、エクスプローラでのクラスまたはその他のオブジェクトの作成手順と同様です。(実行する手順の詳細は、141 ページの「XML オペレーションの作成」を参照してください。

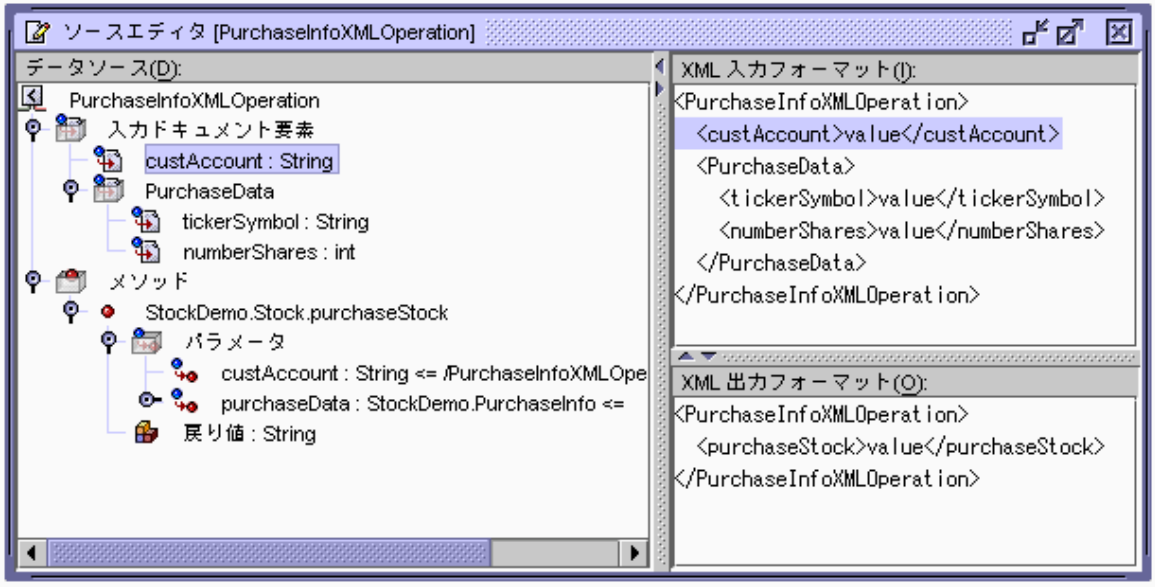

図 5-2　　XML オペレーションソースエディタ - 複雑な入力を表示

XML オペレーションをさらに開発するには、エクスプローラまたはソースエディタからツールを利用します。どちらのウィンドウからも同じ一連のコマンドが使用できます。ソースエディタの方が、コマンドを実行したり、コマンドの実行結果を同じウィンドウで参照したりできるため、一般的に便利です。図 5-2 および図 5-3 は、ソースエディタで表示された XML オペレーションを示しています。

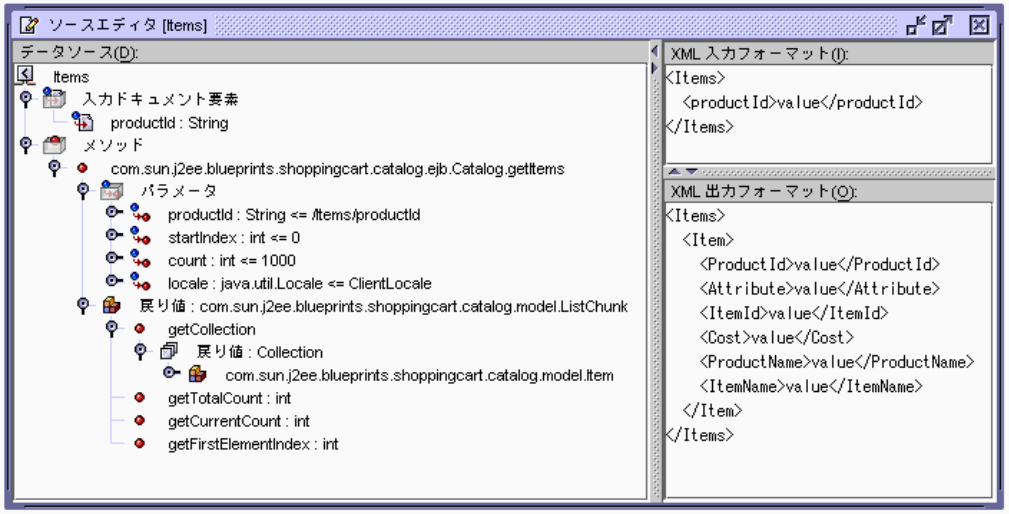

図 5-3　　XML オペレーションソースエディタ - 複雑な出力を表示

ソースエディタは、XML オペレーションを次の 3 つの区画に表示します。

- **「データソース」区画。**この区画では、XML オペレーションに対する編集コマンドを参照したり実行したりできます。
- **「XML 入力フォーマット」区画。**この表示専用区画には、XML 入力ドキュメントのフォーマットが表示されます。
- **「XML 出力フォーマット」区画。**この表示専用区画には、XML 出力ドキュメントのフォーマットが表示されます。

「データソース」区画

「データソース」区画には、XML オペレーションがツリー形式で表示されます。構造上の各ノードは、XML 入力ドキュメント要素、XML オペレーションによって呼び出されるメソッド、これらのメソッドに対するパラメータ、メソッドの戻り値、および体系化されたノードを表します。各ノードの型は、独自のコマンドとプロパティを持ちます。

次の方法で XML オペレーションを編集できます。

- ノードを選択してからメニューコマンドを選択します。
- ノードをダブルクリックしてプロパティシートを表示して、そのプロパティを編集します。

「データソース」区画の最上位には、「入力ドキュメント要素」 ノードと「メソッド」ノードの 2 つの組織ノードが含まれます。

入力ドキュメント要素ノード

XML 入力ドキュメントには、クライアント要求を定義するデータが含まれます。このドキュメントは、「入力ドキュメント要素」ノードとして表されます。このノードを展開すると、XML 入力ドキュメント要素を参照して編集できます。これらの要素はサブノードとして表されます。図 5-4 は、ソースエディタで「入力ドキュメント要素」ノードを展開したときの表示を抜粋したものです。

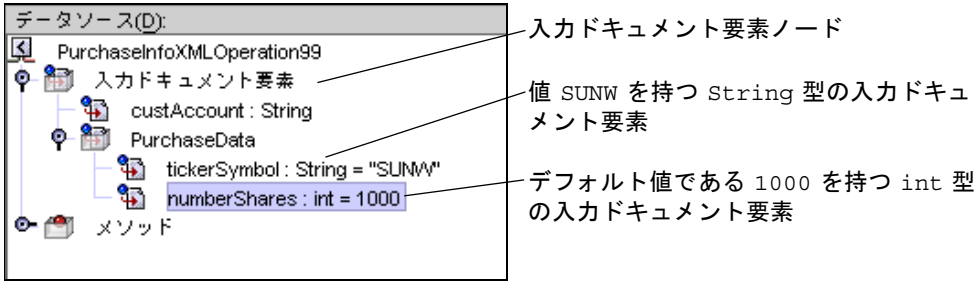

図 5-4 入力ドキュメント要素ノード

これらのノードから、XML 入力ドキュメントに対して次の編集操作を実行できます。

- 要素の追加 (146 ページの「入力ドキュメント要素の追加」を参照)
- 要素の削除 (150 ページの「メソッドまたは入力ドキュメント要素の削除」を参照)
- 要素の名前変更 (147 ページの「入力ドキュメント要素の名前変更」を参照)
- 要素の並べ替え (150 ページの「メソッドまたは入力ドキュメント要素の並び替え」を参照)
- 要素のデフォルトの値の指定 (148 ページの「入力ドキュメント要素にデフォルト値を指定する」を参照)
- 要素の固定 (149 ページの「入力ドキュメント要素を常時に設定する」を参照)

メソッドノード

メソッドノードからは、次のことが実行できます。

- XML オペレーションによって呼び出されるメソッド (多重定義されたメソッドを含む) を指定する。
- データソース上で呼び出しをメソッドに追加することによって、XML オペレーションでどのデータを取得するかを指定する
- メソッドパラメータをソースにマップすることによって、メソッドパラメータ用の値を指定する

- XML オペレーションのメソッド呼び出しによって返されるクラスを展開または縮小して、詳細データを取得する
- XML 出力ドキュメントから返されたクラスのフィールドを選択して除外し、クライアントに送信するデータの量を減らす
- メソッドによって返されたオブジェクトを Web サービス内で他の XML オペレーションで利用できるようにする
- メソッド戻り値を配置する
- 継承メソッドを表示して選択する

メソッドの実行と返されるデータ

XML オペレーションは、他の実行時オブジェクト上でメソッドを実行します。メソッドノードにメソッド呼び出しを追加することによって XML オペレーションが実行するメソッドを指定します。メソッド呼び出しを追加することにより、XML オペレーションをプログラムしてデータを返したり、その他のタイプの処理を実行することができます。

メソッドが呼び出される順序を一旦削除して、配列し直すこともできます。XML オペレーションは、上から下に、順番にメソッドを実行します。

メソッドを追加、削除、または並び替えると、戻り値に対応する要素が追加、順序変更、および削除され、XML 出力ドキュメントに影響します。そのような変更は、XML 出力区画に表示されます。これらのトピックについての詳細は、144 ページの「XML オペレーションへのメソッドの追加」、150 ページの「メソッドまたは入力ドキュメント要素の並び替え」、および 150 ページの「メソッドまたは入力ドキュメント要素の削除」を参照してください。

パラメータの指定

メソッドがパラメータをとる場合は、パラメータはそのメソッドのパラメータノードの下に一覧表示されます。デフォルトでは、XML オペレーションは XML 入力ドキュメントの要素を類似した名前のパラメータにマップすることで、各パラメータの値を取得します。

ただし、XML 入力要素は、任意の方法でメソッドパラメータにマップし直すことができます。XML オペレーション内の 2 つのメソッドが、類似した名前のパラメータを取る場合に再マッピングが必要になることがあります。その場合には、XML オペレーションはデフォルトで両方のパラメータを同じ XML 入力要素にマップします。これが適切でない場合は、新しい入力要素を作成してパラメータの 1 つをそれにマップし直します。

また、入力要素以外のソースの種類にもパラメータをマップできます。たとえば、次のものに対してパラメータをマップできます。

- 固定値

- XML オペレーション内の別のメソッド呼び出しの戻り値
- Web サービスに定義されていて要求に応じてインスタンス化されるターゲットオブジェクト
- 別の XML オペレーションでメソッド呼び出しによって返され、明示的に共有されているオブジェクト
- システム共有オブジェクト

メソッドパラメータをソースにマップする方法についての詳細は、150 ページの「メソッドパラメータのソースへのマッピング」を参照してください。ターゲットオブジェクトについての詳細は、159 ページの「オブジェクトのインスタンス化と参照の解決」を参照してください。システム共有オブジェクトについての詳細は、156 ページの「システム共有オブジェクト」を参照してください。

展開または縮小したデータの取得

オブジェクト (またはオブジェクトの配列またはコレクション) を返すメソッド呼び出しを追加する場合、オブジェクトのクラスタイプがメソッドのサブノードとして表示されます。このクラスが文字列 `get` で始まるメソッドを含んでいる場合は、メソッドはこのクラスのサブノードとして表示されます。これらの「取得」メソッドによって返されたオブジェクトも、デフォルトでクラスノードとして表示されますが、そのメソッドを表すサブノードは表示されません。

そのようなクラスノードを選択して展開することができます。クラスノードを展開すると、クラス内の取得メソッドのすべてのノードが追加され、同様にこれらのメソッドの戻り値に対応する要素が XML 出力に追加されます。

反対に、クラスを縮小することで、取得メソッドが「データソース」区画に表示されないようにすることもできます。また、メソッドを個別に削除することもできます。どちらの場合も、XML オペレーションが実行されたときにはメソッドは呼び出されません。これらのメソッドに対応する XML 出力の要素は、自動的に削除されます。

クラスを展開または縮小するたびに、XML 出力に対応する変更は「XML 出力フォーマット」区画に表示されます。このトピックについての詳細は、155 ページの「クラスを展開する」を参照してください

クライアントに返されるデータの調整

返されたクラスが XML 出力に取り込みたくないデータを保持している場合があります。たとえば、顧客アカウントクラスが「アカウント ID」フィールドと、内部目的にだけ使用される対応する取得メソッドを持っている場合があります。そのような場合は、XML 出力から、このメソッドに対応する要素を選択して除外することができます。

XML 出力から要素を除外しても、XML オペレーションによって呼び出されるメソッドまたはこれらのメソッドにより、XML オペレーションに返されるデータセットに影響はありません。実行により影響を受けるのは、XML 出力ドキュメントからクライアントに返されるデータセットだけです。不必要な要素を除外することによって、アプリケーションコンテナ間で渡されるデータを最小限におさえることができ、パフォーマンスが最適化されます。このトピックについての詳細は、154 ページの「XML 出力ドキュメントから要素を除外する」を参照してください

開発ワークフロー

XML オペレーションを開発する手順の概要は、次のとおりです。

1. XML オペレーションを作成します。

   この手順では、次を持つ XML オペレーションが作成されます。

   - 1 つのメソッド呼び出し
   - メソッドパラメータに基づいたデフォルト XML 入力ドキュメント
   - メソッドの戻り値に基づいたデフォルト XML 出力ドキュメント
   - XML 入力要素のメソッドパラメータに対するデフォルトマッピング

2. (省略可能) 次の手順のいくつかまたはすべてを実行して XML オペレーションを編集します。

   - メソッド呼び出しを追加または削除します。
   - 入力ドキュメント要素を追加、削除、または名前変更します。
   - メソッドパラメータをソースにマップします。
   - 返されたオブジェクトを共有します。
   - 返されたクラスを展開または縮小します。
   - 要素の名前を変更するか、XML 出力ドキュメントから削除します。

3. XML オペレーションを Web サービスに追加します。

   Web サービスがない場合は、作成する必要があります。詳細は、18 ページの「Java メソッドからの JAX-RPC Web サービスの作成」および 21 ページの「Web サービスへのオペレーションの追加」を参照してください。

4. XML オペレーションを Web サービスでテストします。

   このトピックについての詳細は、35 ページの「Web サービスのテスト」を参照してください

5. XML オペレーションを編集して Web サービスの実行時クラスを生成し直して、条件を満たすまでテストします。

XML オペレーションの作成

XML オペレーションは単一で作成するか、またはエンタープライズ Bean に基づいて XML オペレーションのグループを生成することができます。

XML オペレーションの作成

XML オペレーションを作成する手順は、次のとおりです。

1. エクスプローラで、XML オペレーションを作成するフォルダを右クリックし、「新規」->「Web サービス」->「XML オペレーション」を選択します。

   図 5-5 に示す「新規ウィザード - XML オペレーション」ダイアログが表示されます。

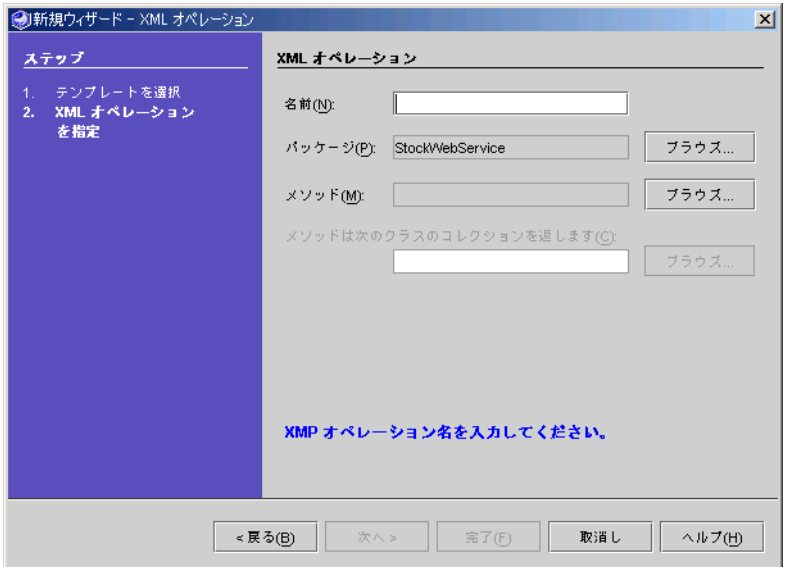

図 5-5　「新規ウィザード - XML オペレーション」ダイアログ

2. 「名前」フィールドで、XML オペレーションの名前を入力します。
3. 「パッケージ」フィールドで XML オペレーションを作成する場所が正しく指定していることを確認します。
4. 「メソッド」フィールドの横にある「ブラウズ」ボタンをクリックします。

   図 5-6 に示す「メソッドの選択」ダイアログが表示されます。

図 5-6　「メソッドの選択」ダイアログ

5. **XML オペレーションに取り込むメソッドを指定して、「了解」ボタンをクリックします。**

   メソッドは、クラス、Bean、インタフェース、EJB コンポーネントから追加することができます。

   EJB のホームおよびリモートインタフェース内に定義されているメソッドを含めることができますが、EJB ローカルインタフェース内にだけ定義されているメソッドを含めることはできません。

   EJB メソッドを追加している場合は、EJB Bean クラスまたはホームインタフェースやリモートインタフェースを表すノードでなく、論理 EJB ノード (Bean アイコン付きのノード) にブラウズしてください。論理 EJB ノードからメソッドを追加することによって、メソッドを呼び出すために必要な実行時情報を提供できます。XML オペレーションを作成するときに、取り込めるメソッドは 1 つです。XML オペレーションでさらにメソッドが必要な場合は、後から追加することができます。XML オペレーションへのメソッドの追加については、144 ページの「XML オペレーションへのメソッドの追加」を参照してください。

6. **選択したメソッドが配列またはコレクションを返す場合は、その中に含まれるオブジェクトのクラス、親クラス、またはインタフェースを選択します。**

   a. **「XML オペレーション」ダイアログの「メソッドはこのクラスのコレクションを戻します」フィールドの横にある「ブラウズ」ボタンをクリックします。**

      ファイル選択のダイアログが開きます。

   b. **ファイル選択のダイアログを使用して、クラスまたはインタフェースを選択します。**

7. **「完了」をクリックします。**

   XML オペレーションが作成され、ソースエディタ内で表示されて、編集することができます (図 5-3 を参照)。XML オペレーションの編集についての詳細は、144 ページの「XML オペレーションの編集」を参照してください。

エンタープライズ Bean からの XML オペレーションの生成

XML オペレーションを単一で作成する他に、エンタープライズ Bean、EJB モジュール、あるいは 1 つまたは複数のエンタープライズ Bean を含むパッケージに基づいて XML オペレーションのグループを生成することができます。これにより、各 エンタープライズ Bean のホームインタフェースおよびリモートインタフェース上の各メソッドに対して 1 つの XML オペレーションが生成されます。生成された XML オペレーションへの参照は、自動的に Web サービスに追加されます。

エンタープライズ Bean から XML オペレーションを生成するには、先に Web サービスを作成している必要があります。新規 Web サービスの作成手順についての詳細は、18 ページの「Java メソッドからの JAX-RPC Web サービスの作成」を参照してください。

エンタープライズ Bean から XML オペレーションを作成する手順は、次のとおりです。

1. **Web サービスを右クリックし、「EJB からオペレーションを作成」を選択します。**
   ファイル選択のダイアログが表示されます。
2. **エンタープライズ Bean、EJB モジュール、またはエンタープライズ Bean を含むパッケージにブラウズして「了解」をクリックします。**
3. **「完了」をクリックします。**
4. **クラス、親クラス、またはエンタープライズ Bean 内のメソッドによって返された配列またはコレクションに含まれているオブジェクトのインタフェースを指定します。**
   メソッドが配列または Collection を返した場合は、図 5-7 に示されているように「何の Collection ですか ?」ダイアログが表示されます。

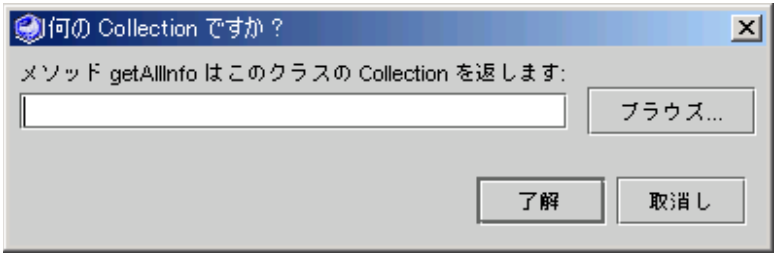

図 5-7　「何の Collection ですか ?」ダイアログ

コレクションを返すメソッドの名前がダイアログに示されます。

オブジェクト型を指定する手順は次のとおりです。

a. 「ブラウズ」ボタンをクリックします。

ファイル選択のダイアログが表示されます。

b. クラス、親クラス、あるいは、配列またはコレクションに含まれているオブジェクトのインタフェースに移動して「了解」をクリックします。

注 – このダイアログは、配列またはコレクションを返す各メソッドに対して 1 回ずつ、自動的に表示されます。

XML オペレーションが生成され、それらへの参照が Web サービスに追加されます。必要のない XML オペレーションを削除して、その他のオペレーションを要件に応じて編集することができます。

XML オペレーションの編集

ここでは、XML オペレーションを編集する方法を説明します。

XML オペレーションへのメソッドの追加

XML オペレーションにメソッドを追加する手順は、次のとおりです。

1. ソースエディタで XML オペレーションを開きます。
2. 「データソース」区画でメソッドノードを右クリックし、「メソッドを追加」を選択します。

図 5-8 に示す「メソッドを追加」ダイアログが表示されます。

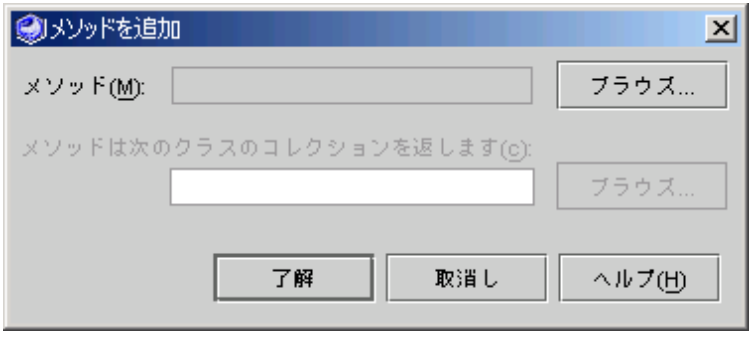

図 5-8 「メソッドを追加」ダイアログ

3. 「ブラウズ」をクリックしてメソッドを選択します。

   図 5-9 に示す「メソッドの選択」ダイアログが表示されます。

図 5-9　　「メソッドの選択」ダイアログ

4. XML オペレーションに取り込むメソッドを指定して、「了解」ボタンをクリックします。

   メソッドは、クラス、Bean、インタフェース、EJB コンポーネントから追加することができます。

   EJB のホームおよびリモートインタフェース内に定義されているメソッドを含めることができますが、EJB ローカルインタフェース内にのみ定義されているメソッドを含めることはできません。

   EJB メソッドを追加している場合は、EJB Bean クラスまたはホームインタフェースやリモートインタフェースを表すノードでなく、論理 EJB ノード (Bean アイコン付きのノード) にブラウズしてください。論理 EJB ノードからメソッドを追加することによって、メソッドを呼び出すために必要な実行時情報を提供できます。

5. 選択したメソッドが配列またはコレクションを返す場合は、その中に含まれるオブジェクトのクラス、親クラス、またはインタフェースを選択します。

   a. 「メソッドを追加」ダイアログの「メソッドは次のクラスのコレクションを返します」フィールドの横にある「ブラウズ」ボタンをクリックします。

      ファイル選択のダイアログが開きます。

   b. ファイル選択のダイアログを使用して、クラスまたはインタフェースを選択します。

6. 「了解」をクリックします。

   このアクションにより、次の結果が得られます。

   - メソッドがメソッドノードに追加される
   - メソッドに対するパラメータがパラメータノードに追加される

- パラメータに対応する要素が入力ドキュメント要素ノードと「XML 入力フォーマット」区画に追加されまる。これらの要素は、パラメータに値を指定するためにマップされる。
- メソッドの戻り値に対応する要素は、「XML 出力フォーマット」区画に追加される。

注 – 後からメソッドを別のパッケージに移動するか、またはメソッドのクラスをメソッドのシグニチャーが変更されるような方法で編集する場合は、XML オペレーションからメソッド呼び出しを削除して追加し直す必要があります。メソッドシグニチャーを変更すると、名前、パラメータ、戻り値のクラス、および例外のリストへの変更が含まれます。

入力ドキュメント要素の追加

入力ドキュメント要素を追加する手順は、次のとおりです。

1. **ソースエディタで XML オペレーションを開きます。**
2. **「データソース」区画で「入力ドキュメント要素」ノードを右クリックし、「入力ドキュメント要素を追加」を選択します。**

   図 5-10 に示されている「入力ドキュメント要素を追加」ダイアログが表示されます。

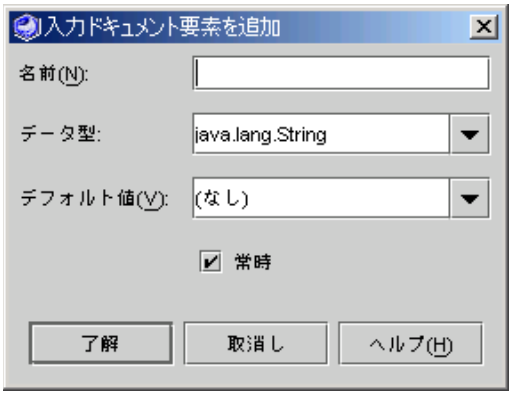

図 5-10 「入力ドキュメント要素を追加」ダイアログ

3. **「名前」フィールドに要素の名前を入力します。**

4. **「データ型」コンボボックスをクリックして、要素のデータ型を選択します。**

   この入力ドキュメント要素をターゲットオブジェクトをインスタンス化するためのパラメータとして使用する場合は、(`String` のように) クラスをデータ型として選択する必要があります。プリミティブ (`int` または `double` のように) は動作しません。

5. **(省略可能) 要素のデフォルトの値を指定します。**

   要素のデフォルトの値を指定する場合は、「デフォルト値」フィールドに入力します。

   この値は、クライアント要求がこの要素を提供しない場合に使用されます。

6. **この要素がメソッドパラメータにそれ以上マップされなくなったときに自動的に削除されるようにする場合は、「常時」チェックボックスの選択を解除します。**

   メソッドパラメータのソースをマップするたびに、メソッドパラメータに現在マップされていない入力要素が削除されます。

7. **「了解」をクリックします。**

   新規入力要素が「データソース」区画の入力ドキュメント要素ノードに追加されます。「XML 入力フォーマット」区画には、更新された XML 入力ドキュメントが表示されます。

入力ドキュメント要素の名前変更

入力ドキュメント要素の名前を変更する手順は、次のとおりです。

1. **図 5-11 に示されているように、入力ドキュメント要素を選択して「プロパティ」ウィンドウを開きます。**
2. **「名前」プロパティをクリックします。**

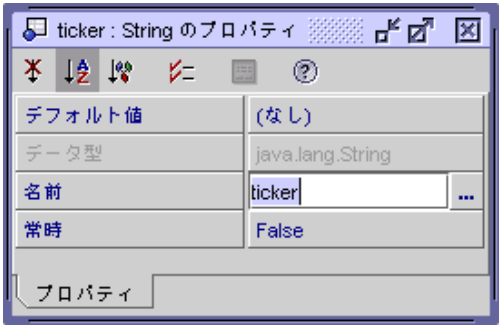

図 5-11 入力ドキュメント要素のプロパティダイアログ

3. **入力ドキュメント要素の名前を入力して Enter キーを押します。**

   入力ドキュメント要素の名前が変更されます。新しい名前は入力ドキュメント要素内のノードと「XML 入力フォーマット」区画の両方に追加されます。

出力ドキュメント要素の名前変更

XML 出力ドキュメントの各要素は、デフォルトで要素の値を返すメソッドにちなんだ名前が付けられます。たとえば、`append` という名前のメソッドに呼び出しを追加すると、`append` という名前の XML 出力ドキュメントに要素が追加されます。`getName` という名前のメソッドに呼び出しを追加すると、`Name` という名前の要素が追加されます。これらの要素は、メソッド呼び出しの「タグ名」プロパティの値を変更することによって名前を変更することができます。

出力ドキュメント要素の名前を変更する手順は、次のとおりです。

1. **図 5-12 に示されているようにメソッドフォルダ内で、要素の値を返すメソッドを選択して、メソッドの「戻り値」ノードを選択し、「プロパティ」ウィンドウを表示します。**
2. **「タグ名」プロパティをクリックします。**

図 5-12　出力ドキュメント要素のプロパティダイアログ

3. **要素の名前を入力して Enter キーを押します。**

   出力ドキュメント要素の名前が変更されます。新しい名前が「XML 出力フォーマット」区画に表示されます。

入力ドキュメント要素にデフォルト値を指定する

入力ドキュメント要素にデフォルト値を指定する手順は、次のとおりです。

1. 図 5-13 に示されているように入力ドキュメント要素のプロパティシートを開きます。
2. 「デフォルト値」プロパティをクリックします。

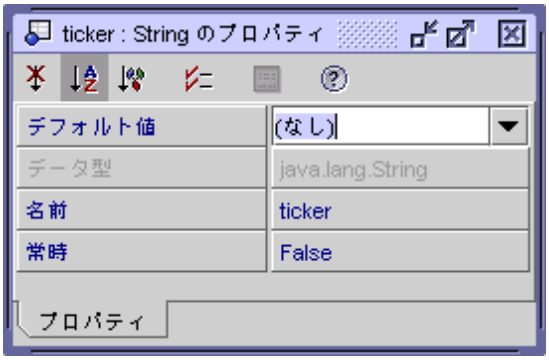

図 5-13　入力ドキュメント要素プロパティ (デフォルト値)

3. 値を入力するか、null 値を指定する場合は「(なし)」を選択します。
   この値は、クライアント要求がこの要素を提供しない場合に使用されます。

入力ドキュメント要素を常時に設定する

IDE では、入力ドキュメント要素の「常時」プロパティが有効になっていない場合は、メソッドパラメータにマップされない入力ドキュメント要素を自動的に削除します。

たとえば、XML オペレーションに対するパラメータをとるメソッドを追加すると、IDE では入力ドキュメント要素を自動的に追加してメソッドパラメータにマップします。次にそのメソッドパラメータを別のソース (たとえば、別のメソッドの戻り値) にマップし直すと、その入力ドキュメント要素はそれ以上メソッドパラメータにマップされないため、IDE によって削除されます。

入力ドキュメント要素がマップされない場合に自動的に削除されないようにするには、「常時」プロパティを有効にします。

入力ドキュメントの「常時」プロパティを有効にする手順は、次のとおりです。

1. 図 5-13 に示されているように入力ドキュメント要素のプロパティシートを開きます。
2. 「常時」プロパティをクリックします。
3. コンボボックスで「True」を選択して Enter キーを押します。
   入力ドキュメント要素が常時に設定されます。

メソッドまたは入力ドキュメント要素の並び替え

XML オペレーション内の入力ドキュメント要素とメソッドを並び替えることができます。

メソッドを並び替えると、メソッドが呼び出される順序が変更され、XML 出力ドキュメント内の要素の順序が変更されます。メソッドはソースエディタ内に一覧されている順序で上から下へ呼び出されます。

XML オペレーション内のメソッドの戻り値は、XML オペレーション内の別のメソッドへのパラメータとして使用できます (詳細は、150 ページの「メソッドパラメータのソースへのマッピング」を参照してください)。XML オペレーションにそのような依存性がある場合は、パラメータを指定しているメソッドがパラメータを要求するメソッドの前に呼び出されていることを確認する必要があります。

入力ドキュメント要素を並び替える機能は、開発の簡便性のためのものです。実行時には、入力ドキュメント要素の順序は Web サービスに大きな影響を与えません。

メソッドまたは入力ドキュメント要素を並び替える手順は、次のとおりです。

1. **ソースエディタ内で XML オペレーションを開いて、並び替えるメソッドまたは入力ドキュメント要素を選択します。**
2. **メソッドまたは入力ドキュメント要素を右クリックし、「上へ移動」または「下へ移動」を選択します。**

メソッドまたは入力ドキュメント要素の削除

XML オペレーションから入力ドキュメント要素とメソッドを削除することができます。メソッドを削除すると、XML オペレーションが実行されたときにメソッドは呼び出されません。削除されたメソッドに対応する XML 出力の要素も、削除されます。

メソッドまたは入力ドキュメント要素を削除する手順は、次のとおりです。

1. **ソースエディタで XML オペレーションを開いて、削除するメソッドまたは入力ドキュメント要素を選択します。**
2. **メソッドまたは入力ドキュメント要素を右クリックし、「削除」を選択します。**

メソッドパラメータのソースへのマッピング

メソッドパラメータをソースにマップする手順は、次のとおりです。

1. **ソースエディタ内で XML オペレーションを開いて、マップするメソッドパラメータを選択します。**

2. パラメータを右クリックし、「ソースの変更」を選択します。

図 5-14 に示す「メソッドパラメータソース」ダイアログが表示されます。

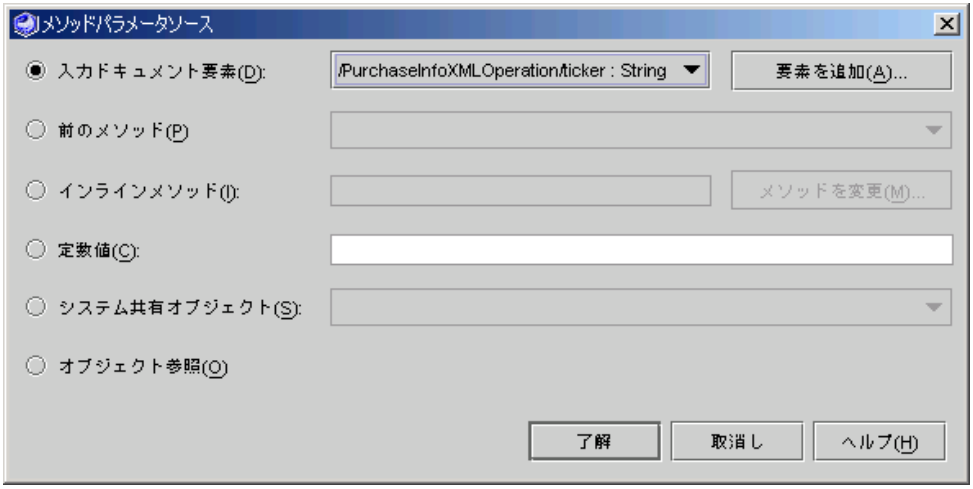

図 5-14　「メソッドパラメータソース」ダイアログ

3. ソースの種類を選択します。

次の表は、利用可能なソースの種類の説明です。

ソースの種類	説明
入力ドキュメント要素	パラメータを XML 入力ドキュメントの要素にマップする場合に、このラジオボタンを選択します
前のメソッド	パラメータをこの XML オペレーションの前のメソッド呼び出しの戻り値にマップする場合に、このラジオボタンを選択します
インライン メソッド	パラメータをインラインメソッド呼び出しの戻り値にマップする場合に、このラジオボタンを選択します。「メソッドを変更」をクリックして目的のメソッドを選択します

ソースの種類	説明
定数値	パラメータを定数値にマップする場合に、このラジオボタンを選択します
システム共有オブジェクト	パラメータをこれらの種類のデータの 1 つにマップする場合に、このラジオボタンを選択します。 Web サービスは、Web サービスにアクセスする Web ブラウザクライアントに関するデータを含む「システム共有オブジェクト」を実行時に保持します。システム共有オブジェクトは、Web サービスによってインスタンス化され、HTTP 要求および J2EE セキュリティメカニズムから取得されたデータが入力されます。 システム共有オブジェクトは、ユーザー名に関するデータを保持します。データは `String` オブジェクトおよび `java.security.Principal` オブジェクトとして提供されます
オブジェクト参照	パラメータを Web サービスに定義されているターゲットオブジェクトにマップする場合に、このラジオボタンを選択します。このオプションは、パラメータのタイプがクラス (`String` を含む) の場合にだけ利用可能です このラジオボタンを選択すると、要求されたクラスの新しいオブジェクトにパラメータをマップする Web サービス内に、デフォルトオブジェクト参照が追加されます。この参照を構成し直すことで、選択したオブジェクトを解決できます。詳細は、159 ページの「オブジェクトのインスタンス化と参照の解決」を参照してください。

4. **パラメータの値のソースを指定します。**

ソースの種類が「入力ドキュメント要素」、「メソッドの戻り値」、「定数値」、または「システム共有オブジェクト」に設定されている場合は、有効になっているフィールドを使用してソースを指定します。次の表は、指定されたソースの種類に依存して実行される動作について説明しています。

フィールド名	動作
入力ドキュメント要素	有効になっているコンボボックスから入力ドキュメント要素を選択します。コンボボックスには、パラメータで必要な型の XML オペレーションに含まれるすべての入力ドキュメント要素が一覧表示されます。入力ドキュメント要素を選択すると、その値がパラメータにマップされます

フィールド名	動作
前のメソッドまたはインラインメソッド	有効になっているコンボボックスからメソッドを選択します。コンボボックスには、パラメータに適切した型を返す現在のメソッドの前に呼び出されるメソッドが一覧表示されます。メソッドを選択すると、その戻り値がパラメータにマップされます
定数値	有効になっているフィールドで、値を示す文字列を入力します。`String` または `char` 型の値の場合は、値の一部である場合以外は引用符を入力しないでください
システム共有オブジェクト	リストからオブジェクトを選択します。利用できるオブジェクトのリストは、パラメータの型によって異なります。 `java.security.Principal` 型のパラメータの場合は、コンボボックスに 1 つのオブジェクト `UserPrincipal` が表示されます `java.lang.String` 型のパラメータの場合は、コンボボックスに 1 つのオブジェクト `UserName` が表示されます

5. 「了解」をクリックします。

メソッド戻り値の配置

メソッド戻り値を配置する手順は、次のとおりです。

1. ソースエディタ内で XML オペレーションを開いて、ノードの下の「戻り値」ノードを見つけます。
2. ノードを右クリックし、「プロパティ」を選択します。

   プロパティの 1 つは「キャスト」で、値はデフォルトの「なし」です。もう 1 つのプロパティは「データ型」です。「キャスト」の値は、Java 規則に準拠している任意の型に変更できます。
3. 「了解」をクリックします。

継承メソッドの表示および選択

継承メソッドを表示する手順は、次のとおりです。

1. ソースエディタ内で XML オペレーションを開いて、ノードの下の「戻り値」ノードを見つけます。

   または、エクスプローラで、「戻り値」ノードを探すこともできます。
2. ノードを右クリックし、「展開」を選択します。

   「展開」ウィンドウが表示され、取得メソッドが自動的に選択されます。

3. 「継承されたメソッドを表示」チェックボックスを選択します。
   継承されたメソッドが表示されます。
4. 目的のメソッドを選択して「了解」をクリックします。
   選択したメソッドがエクスプローラおよび「データソース」区画に表示されます。

XML 出力ドキュメントから要素を除外する

XML 出力ドキュメントから要素を除外する手順は、次のとおりです。

1. ソースエディタで XML オペレーションを開きます。
2. 「データソース」区画で、メソッドノードを開きます。
3. 除外する要素に対応する「データソース」区画のノードを識別します。
   「データソース」区画でノードを選択すると、対応する要素が「XML 出力フォーマット」区画で強調表示されます。図 5-15 の例では、「データソース」区画内で選択されたクラスノードおよび「XML 出力フォーマット」区画でそれに対応する強調表示されている要素を示しています。

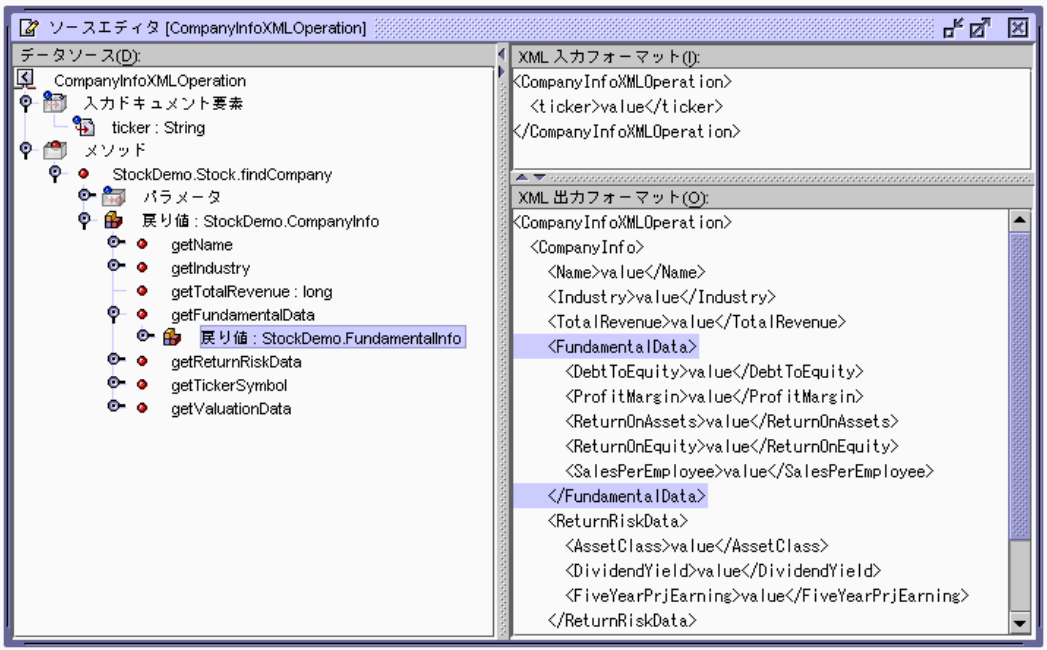

図 5-15　ソースエディタ - 出力要素を除外する

4. **ノードを右クリックし、「出力からタグを省略」を選択します。**
   対応する要素が「XML 出力フォーマット」区画から削除されます。

XML 出力ドキュメントに要素を含める

デフォルトで、XML オペレーションのメソッド呼び出しの戻り値はすべて、XML 出力ドキュメントに含まれます。これらの戻り値を表す要素のどれかを除外した場合は、XML 出力ドキュメントに追加し直すことができます。

XML 出力ドキュメントに要素を含める手順は、次のとおりです。

1. **ソースエディタで XML オペレーションを開きます。**
2. **「データソース」区画で、メソッドノードを開きます。**
3. **含める要素に対応するノードを識別します。**
   含めることができるのは、配列またはコレクション、クラス、またはプリミティブを返すメソッドを表す型のノードです。
4. **ノードを右クリックし、「出力にタグを取り込み」を選択します。**
   選択したノードに対応する要素が「XML 出力フォーマット」区画に追加されます。

クラスを展開する

クラスを展開する手順は、次のとおりです。

1. **ソースエディタで XML オペレーションを開きます。**
2. **「データソース」区画で、メソッドノードを開きます。**
3. **クラスを右クリックして「展開」を選択します。**
   クラス上のすべての取得メソッドはサブノードとし、クラスノードに追加されます。クラスおよび取得メソッドの戻り値に対応する要素は、「XML 出力フォーマット」区画に追加されます。

クラスを縮小する

クラスを縮小する手順は、次のとおりです。

1. **ソースエディタで XML オペレーションを開きます。**
2. **「データソース」区画で、メソッドノードを開きます。**

3. クラスを右クリックし、「縮小」を選択します。

クラスの取得メソッドを表すノードが「データソース」区画から削除されます。対応する要素およびクラスに対応する要素は、「XML 出力フォーマット」区画から削除されます。

システム共有オブジェクト

Web サービスは、Web サービスにアクセスする Web ブラウザクライアントに関するデータを含む「システム共有オブジェクト」を実行時に保持します。システム共有オブジェクトは、Web サービスによってインスタンス化され、HTTP 要求および J2EE セキュリティメカニズムから取得されたデータが入力されます。システム共有オブジェクトの使用方法についての詳細は、150 ページの「メソッドパラメータのソースへのマッピング」 および図 5-14 を参照してください。

システム共有オブジェクトはユーザー名に制限され、J2EE セキュリティメカニズムから取得され、`String` オブジェクトおよび `java.security.Principal` オブジェクトとして提供されます。

`java.security.Principal` 型のパラメータの場合は、「メソッドパラメータソース」ダイアログ内の「システム共有オブジェクト」選択肢に 1 つのオブジェクト `UserPrincipal` が表示されます。

`java.lang.String` 型のパラメータの場合は、「システム共有オブジェクト」選択肢に 1 つのオブジェクト `UserName` が表示されます。

Static ユーティリティメソッド

ビジネスメソッドによって返されたデータを、Web ページに表示するために、何らかの処理が必要になる場合があります。たとえば、EJB メソッドが `double` 型の値を返すときに、その値を通貨形式で表示したいことがあります。最適な方法は、ビジネスコンポーネントに新しいメソッドを追加するのではなく、Static ユーティリティメソッドを記述するか、再利用することです。

Static ユーティリティメソッドの構成

作成する Static ユーティリティメソッドを IDE でいくつかのクラスに分類して、使いやすくします。

サービス固有の場所に配備される Web サービスに固有のユーティリティクラスは、Web サービスを含むパッケージなどに配置します。汎用ユーティリティクラスは、他のサービスや Web サービス開発者が簡単に再利用できるパッケージに配置します。

Static ユーティリティメソッドの使用

XML オペレーションで Static ユーティリティメソッドを使用する手順は、次のとおりです

1. **エクスプローラで Static ユーティリティクラスをマウントします。**
2. **ソースエディタで XML オペレーションを開きます。**
3. **XML オペレーションにユーティリティメソッドへの呼び出しを追加します。**

   この方法についての詳細は、144 ページの「XML オペレーションへのメソッドの追加」を参照してください。

   ユーティリティメソッド呼び出しは、処理するデータを返すメソッド呼び出しの後に配置する必要があります。メソッドを配置し直すには、右クリックして「上へ移動」または「下へ移動」を選択します。

   ユーティリティメソッドを追加すると、メソッドの戻り値に対応する要素が XML 出力ドキュメントに追加されます。この要素は「XML 出力フォーマット」区画に表示されます。

   メソッドの入力パラメータに対応する新しい入力ドキュメント要素も追加されます。この入力ドキュメント要素は無視することができます。次の手順でメソッドパラメータソースをマップし直すと削除されます。

4. **ユーティリティメソッドの入力パラメータを、データコンポーネントに対するメソッド呼び出しによって返された値にマップします。**

   データコンポーネントに対するメソッド呼び出しでは、ユーティリティメソッドで処理する値が返されます。そのため、データコンポーネントからの出力を、ユーティリティメソッドの入力にマップする必要があります。

   a. **ユーティリティメソッドノードを展開して、パラメータノードを展開します。**

   b. **パラメータを右クリックし、「ソースの変更」を選択します。**

      図 5-16 に示す「メソッドパラメータソース」ダイアログが表示されます。

   c. **「前のメソッド」を選択します。**

図 5-16　「メソッドパラメータソース」ダイアログ

d. 有効になっているコンボボックスで、データコンポーネントからデータを返すメソッドを選択して「了解」をクリックします。

パラメータのマッピングについての詳細は、図 5-14 および 150 ページの「メソッドパラメータのソースへのマッピング」を参照してください。

5. データコンポーネントのメソッド呼び出しの戻り値を XML 出力ドキュメントから除外するには、データコンポーネントを呼び出すメソッドを右クリックし、「出力からタグを省略」を選択します。

要素が「XML 出力フォーマット」区画から削除されます。クライアントでデータを処理する必要がありますが、多くの場合、データコンポーネントによって返される raw データではありません。

6. (省略可能) ユーティリティメソッドの「タグ名」プロパティを適切な名前に設定します。

7. (省略可能) 出力ドキュメント要素の名前を変更します。

ユーティリティメソッドを XML オペレーションに追加すると、対応する要素が XML 出力ドキュメントに追加されます。デフォルトで、この要素にはメソッドにちなんだ名前 (`formatAsDollars` など) が付けられます。多くの場合、別の名前 (`Price` など) のほうが適しています。要素の名前を変更するには、ユーティリティメソッドの「タグ名」プロパティの値を変更します。この方法についての詳細は、148 ページの「出力ドキュメント要素の名前変更」を参照してください。

オブジェクトのインスタンス化と参照の解決

XML オペレーションを開発するときに、XML オペレーションが呼び出すメソッドを指定します。これらのメソッドを実行時に呼び出すために、Web サービスは特定のオブジェクトを必要とします。各メソッド呼び出しに対して、次のものを見つけるか、またはインスタンス化します。

- メソッドが定義されているクラスのインスタンス
- メソッドによってパラメータとして要求される各クラスのインスタンス

このタスクを実行するために、Web サービスでは、これらの各ターゲットオブジェクトへの参照とし、ターゲットオブジェクトが存在しない適切なクラスのオブジェクトをインスタンス化する方法の定義を維持します。XML オペレーションにメソッド呼び出しを追加すると、デフォルトオブジェクト参照とターゲットオブジェクト定義が自動的に Web サービスに追加されます。これらのデフォルトは一般的に適切な設定であり、編集する必要はありません。

ただし、オブジェクト参照のターゲットを手動で指定して、要件に合うように新しいターゲットオブジェクト定義を編集して作成できます。IDE 外で作成されたエンタープライズ Bean へのオブジェクト参照を手動で解決することが必要な場合があります。

この節では、次の操作手順について説明します。

- オブジェクト参照のターゲットを指定する
- 新規ターゲットオブジェクトを定義する
- ターゲットオブジェクト定義を編集する

オブジェクト参照のターゲットを指定する

オブジェクト参照のターゲットを指定する手順は、次のとおりです。

1. **「オブジェクト参照を解決」ダイアログを開きます。**

   エクスプローラで、Web サービスを右クリックし、「オブジェクト参照を解決」を選択します。「オブジェクト参照を解決」ダイアログが表示されます (図 5-17 を参照)。

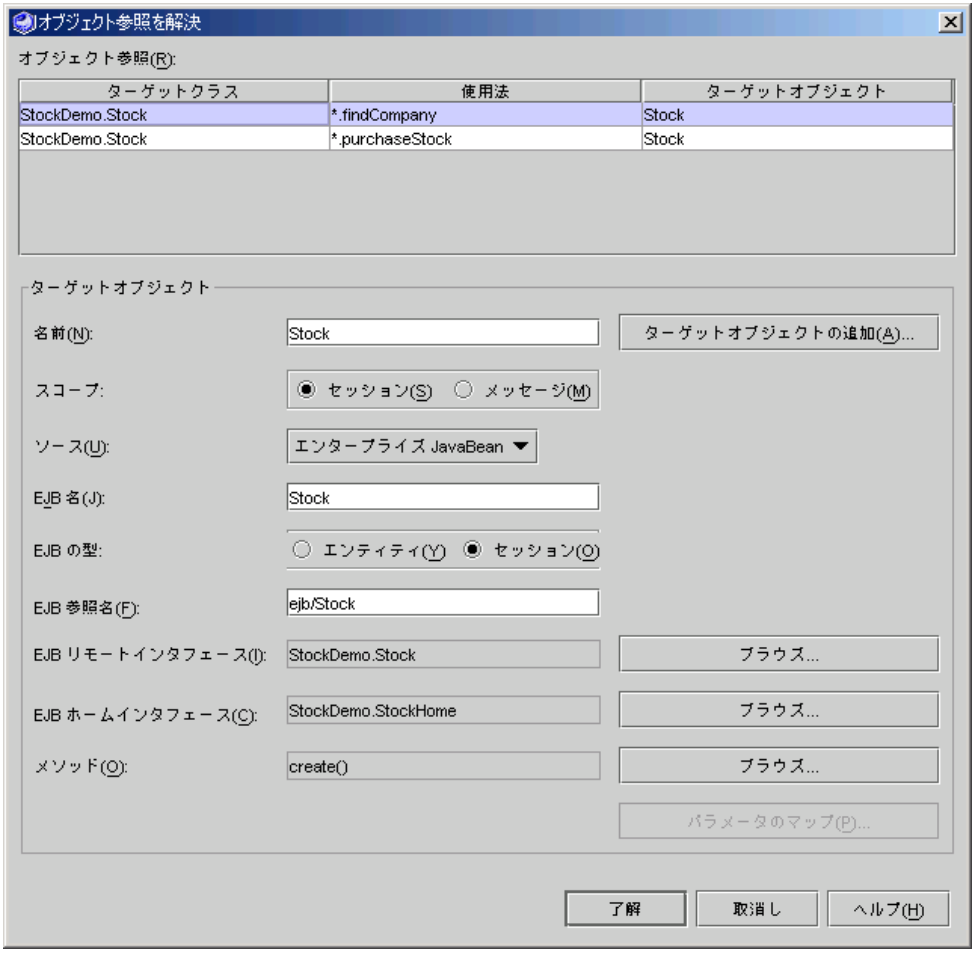

図 5-17　「オブジェクト参照を解決」ダイアログ

Web サービスが XML オペレーションを参照している場合は、図 5-18 のようにダイアログ内の表に XML オペレーション名が記載された追加の列が表示されます。

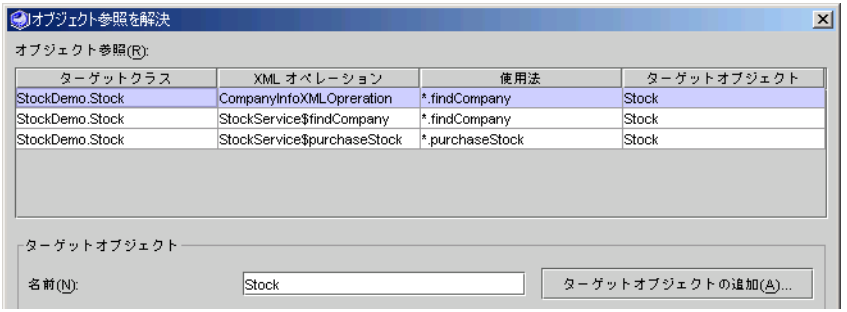

図 5-18　XML オペレーションがある「オブジェクト参照を解決」ダイアログ

2. **編集するオブジェクト参照を選択します。**

   オブジェクト参照は表形式でダイアログの上部に一覧表示されます。各行は、1 つの参照を表しています。次の表は、列の説明です。

列名	説明
ターゲットクラス	参照で要求されるオブジェクトのクラス
XML オペレーション	ターゲットオブジェクトへの参照を保持する XML オペレーションの名前
使用法	XML オペレーションがターゲットオブジェクトを使用する方法を示す。使用方法は 2 つあり、XML オペレーションは次の処理を行う。 • ターゲットオブジェクト上でメソッドを呼び出す • ターゲットオブジェクトをメソッド呼び出しへのパラメータとして渡す この列では、呼び出されたメソッドの名前を提供し、メソッド呼び出しを簡単にするためのターゲットオブジェクトの使用方法を示す(アスタリスクを付ける)。 たとえば、`*.getCustomer` の値は `getCustomer` というメソッドがターゲットオブジェクト上で呼び出されることを示す。`updateCustomer(customerInfo:*)` の値は、ターゲットオブジェクトが `customerInfo` パラメータとして `updateCustomer` メソッドに渡されることを示す。
ターゲットオブジェクト	参照を解決するオブジェクトの名前

3. **参照を解決するオブジェクトを選択します。**

   編集しているオブジェクト参照を示している行で、「ターゲットオブジェクト」列をクリックしてドロップダウンリストからオブジェクトを選択します。このリストは、必要なクラスの利用可能なオブジェクトをすべて表示し、このダイアログ内に定義済みのターゲットオブジェクトを含んでいます。

新規ターゲットオブジェクトの定義

すでに定義されているターゲットオブジェクトがオブジェクト参照に適していない場合は、新しいターゲットオブジェクトを定義できます。

新規ターゲットオブジェクトを定義する手順は、次のとおりです。

1. **「オブジェクト参照を解決」ダイアログを開きます。**

   エクスプローラの Web サービスを右クリックし、「オブジェクト参照を解決」を選択します。「オブジェクト参照を解決」ダイアログが表示されます (図 5-17 を参照)。

2. **新規ターゲットオブジェクトを定義する参照を選択します。**

   ダイアログに現在のターゲットオブジェクトの定義が表示されます。

3. **「ターゲットオブジェクトの追加」をクリックします。**

   「ターゲットオブジェクトの追加」ダイアログが表示されます。

4. **「ターゲットオブジェクト名」フィールドにターゲットオブジェクトの名前を入力します。**

5. **「了解」をクリックします。**

   ダイアログで新しく定義されたターゲットオブジェクトへの参照が自動的に解決されます。新しいターゲットオブジェクトは既存のものと同じ定義を使用します。参照される名前だけが変更されます。定義の編集手順については、162 ページの「ターゲットオブジェクト定義を編集する」を参照してください。

ターゲットオブジェクト定義を編集する

ターゲットオブジェクト定義は、Web サービスによる次の処理の実行方法を指定します。

- すでにインスタンス化されているターゲットオブジェクトを指定する
- 新規ターゲットオブジェクトをインスタンス化する

Web サービスは、同じターゲットオブジェクトに対して解決する複数の参照を持つことができます。たとえば、同じセッション Bean が Web サービス内の多くの XML オペレーションで使用されます。そのようなシナリオでは、ターゲットオブジェクト定義を編集すると、これらの参照すべてに影響します。これは、同じオブジェクトに対して解決するためです。この動作がアプリケーションに適していない場合は、1 つまたは複数の参照に対して新しいターゲットオブジェクト定義を作成する必要があります。新規定義の作成手順については、162 ページの「新規ターゲットオブジェクトの定義」を参照してください。

ターゲットオブジェクト定義を編集する手順は、次のとおりです。

1. 「オブジェクト参照を解決」ダイアログを開きます。

   エクスプローラ中の Web サービスを右クリックし、「オブジェクト参照を解決」を選択します。「オブジェクト参照を解決」ダイアログが表示されます (図 5-17 を参照)。

2. 必要に応じて「名前」フィールドを編集します。

   このフィールドでは、ターゲットオブジェクトが参照される名前を指定します。ターゲットオブジェクトの値もダイアログの上部にあるオブジェクト参照リストの「ターゲットオブジェクト」列に表示されます。このフィールドは必須です。

3. 必要に応じて「スコープ」フィールドを編集します。

   この必須フィールドでは、ターゲットオブジェクトが参照される範囲を定義します。オプションは次のとおりです。

   - **セッション。**ターゲットオブジェクトはセッションが継続している間、セッション内のすべての XML オペレーションで参照できます。
   - **メッセージ。**ターゲットオブジェクトはそれをインスタンス化したクライアント要求の実行中に限り、参照できます。1 つの要求に対して 1 つの XML オペレーションだけが実行されるため、ターゲットオブジェクトへのアクセスはその 1 つの XML オペレーションに制限されます。XML オペレーションが実行されるたびに、新しいターゲットオブジェクトが作成されます。

   注 – 範囲を選択しても現在のリリースの Sun ONE Studio 5, Standard Edition IDE で作成した Web サービスの実行特性に影響はありません。

   ---

4. 必要に応じて「ソース」フィールドを編集します。

   このフィールドでは、ターゲットオブジェクトが取得またはインスタンス化されるメカニズムの種類を指定します。オプションは次のとおりです。

   - **エンタープライズ JavaBean。**ターゲットオブジェクトは、エンタープライズ Bean のリモートまたはホストインタフェースです。ホームインタフェースは JNDI 検索から取得されます。リモートインタフェースは、対応するホームインタフェースへのメソッド呼び出しによって取得されます。
   - **コンストラクタ。**ターゲットオブジェクトは、ターゲットクラス上のコンストラクタメソッドへの呼び出しによって返されます。
   - **static メソッド。**ターゲットオブジェクトは、`static` メソッドへの呼び出しによって返されます。

5. **必要に応じて残りのフィールドを編集します。**

「ソース」フィールドは、ダイアログ内の他のどのフィールドが表示されるか、および入力用に有効になるかに影響します。ターゲットオブジェクトのソースに適している次の表のどれかの説明を参照してダイアログ内の残りのフィールドを編集します。

ソースがエンタープライズ Bean の場合は、表 5-1 を参照してください。

表 5-1 「ソース」フィールドに「エンタープライズ JavaBean」が設定されている場合に有効になるフィールド

フィールド	説明
EJB 名	EJB モジュール配備記述子内に定義されているエンタープライズ Bean の名前。
EJB の型	「エンティティ」または「セッション」のどちらかのエンタープライズ Bean の型。必須。
EJB 参照名	ターゲットオブジェクトを指定する JNDI 検索内の文字列。デフォルト値は、「名前」フィールドの値に、接頭辞として文字列 `ejb/` を付けたものです。必須。
EJB リモートインタフェース	Enterprise Bean のリモートインタフェース。ターゲットオブジェクトがホームインタフェースの場合でも必須。
EJB ホームインタフェース	エンタープライズ Bean のホームインタフェース。必須。
メソッド	リモートインタフェースを返すためにホームインタフェースで呼び出される検索メソッドまたは生成メソッド。検索メソッドは、エンティティ Bean で使用され、生成メソッドはセッション Bean で使用される。ターゲットオブジェクトがリモートインタフェースの場合にだけ必須。

ソースがコンストラクタの場合は、表 5-2 を参照してください。

表 5-2 「ソース」フィールドが「コンストラクタ」に設定されている場合に有効になるフィールド

フィールド	説明
クラス	ターゲットオブジェクトのコンストラクタが定義されているクラス。このフィールドは、「コンストラクタ」フィールド用のコンストラクタを選択したときに自動的に記入される。読み取り専用。
コンストラクタ	ターゲットオブジェクトをインスタンス化するために使用されるコンストラクタメソッド。コンストラクタのクラスは、「クラス」フィールドに指定される。必須。

ソースが static メソッドの場合は、表 5-3 を参照してください。

表 5-3　「ソース」フィールドが「static メソッド」に設定されている場合に有効になるフィールド

フィールド	説明
クラス	ターゲットオブジェクトを返す静的 (Static) メソッドが定義されているクラス。このフィールドは、「静的 (Static) メソッド」フィールドで静的 (Static) メソッドを選択すると自動的に記入されます。読み取り専用。
static メソッド	ターゲットオブジェクトをインスタンス化するために使用される静的 (Static) メソッド。必須。

6. 必要に応じてメソッドパラメータをソースにマップします。

「パラメータのマップ」ボタンが有効になっている場合は、手順に従って「メソッド」、「コンストラクタ」、または「static メソッド」フィールドに指定されているメソッドのパラメータに値を指定します。

a. 「パラメータのマップ」をクリックします。

「パラメータのマップ」ダイアログが表示されます。

図 5-19　「パラメータのマップ」ダイアログ

b. マップするパラメータを表す行を指定します。

各行は、1 つのメソッドパラメータを表します。メソッドで複数のパラメータを指定できる場合は、「パラメータ名」列を使用してパラメータを識別します。メソッドがソースコードとして提供されている場合、この列はパラメータの名前を示します。メソッドが JAR ファイルで提供されている場合、「パラメータ名」列には、メソッドが必要とするパラメータの順序を param1、param2、などの名前に表示します。

c. パラメータの値の型を指定します。

「値の型」列をクリックし、リストからオプションを選択します。次の表は、オプションの説明です。

オプション	説明
定数	パラメータを「値」フィールドに指定されている値にマップします。
環境エントリ	パラメータを「値」フィールドに指定されている環境エントリの値にマップします。環境エントリの設定に関して詳細は、23 ページの「環境エントリの追加」を参照してください。
ターゲットオブジェクト	パラメータを「値」フィールドに指定されているオブジェクトにマップします。オブジェクトは、Web サービスの「オブジェクト参照を解決」ダイアログ内で定義されているターゲットオブジェクト。
入力ドキュメント要素	パラメータを「値」フィールドに指定されている入力ドキュメント要素にマップします。入力ドキュメント要素のデータ型はオブジェクト (たとえば、`String`) である必要があります。基本型 (たとえば、`int`) は使用できません。入力ドキュメント要素のデータ型の宣言については、146 ページの「入力ドキュメント要素の追加」を参照してください。

d. 「値」フィールドにパラメータの値を指定します。

次の表は、パラメータの値の型に依存して値を指定する方法を説明しています。

値の型	ユーザー操作
定数	「値」フィールドにリテラル値を入力します。
環境エントリ	「値」フィールドに環境エントリの名前を入力します。 環境エントリの設定に関して詳細は、23 ページの「環境エントリの追加」を参照してください。
ターゲットオブジェクト	「値」フィールドをクリックし、リストからオブジェクトを選択します。 このリストには、Web サービスに定義されている「データ型」フィールド内に指定されているクラスのすべてのターゲットオブジェクトが含まれます。

値の型	ユーザー操作
入力ドキュメント要素	「値」フィールドをクリックし、リストから入力ドキュメント要素を選択します。 このリストには、次の両方の入力ドキュメント要素が含まれます。 指定したデータ型 ターゲットオブジェクトを参照する各 XML オペレーションに定義済み 2 番目の要件により、マッピングしているパラメータがすべての状況において値に対するソースを持っていることが保証されます。たとえば、2 つの XML オペレーションは同じターゲットオブジェクトを参照できますが、どちらの XML オペレーションが先に実行されるかはわかりません。したがって、それぞれの XML オペレーションでパラメータの値を提供して、どちらが先に実行されてもパラメータ値が利用できるようにする必要があります。入力ドキュメント要素についての詳細は、137 ページの「入力ドキュメント要素ノード」を参照してください。

付録 A

Web サービスのセキュリティ保護

この付録では、Web サービスのセキュリティ保護について説明します。ここでは、IDE の一般的な知識があること、このマニュアルのここまでのページを読んでいること、特に Web サービスおよび Web サービスクライアントの作成方法を説明している第 2 章および第 3 章を読んでいることを前提としています。

また、ここでの操作は、Sun™ ONE Application Server 7 サーバーがインストールおよび稼働中で、同サーバーが IDE のデフォルトのアプリケーションサーバーに指定されていることを前提に記述されています。異なるアプリケーションサーバーを使用している場合は、それに応じて操作方法を変えてください。

セキュリティの概要

JAX-RPC および IDE では、HTTP 基本認証および HTTP/SSL 認証と暗号化をサポートしています。この付録では、セキュリティ保護のためのテクノロジについての概要と 2 種類のセキュリティを実装するための IDE の使用方法について説明しています。

注 – SSLは「Secure Sockets Layer」の略で、当初は Netscape Web ブラウザ用に開発され、現在は事実上の業界標準となっています。

HTTP 基本認証

基本認証では、Web サーバーは Web クライアントから取得したユーザー名およびパスワードを使用して、クライアントユーザー (「主体」ともいいます) を認証します (図 A-1 を参照してください)。パスワードは、ネットワーク上での傍受からは保護さ

れていません。この制限を克服するために、この章で後述するように SSL 保護されたセッション内で基本認証を実行させることができます。すべてのメッセージの内容は、この基本認証の実行によって暗号化されます。

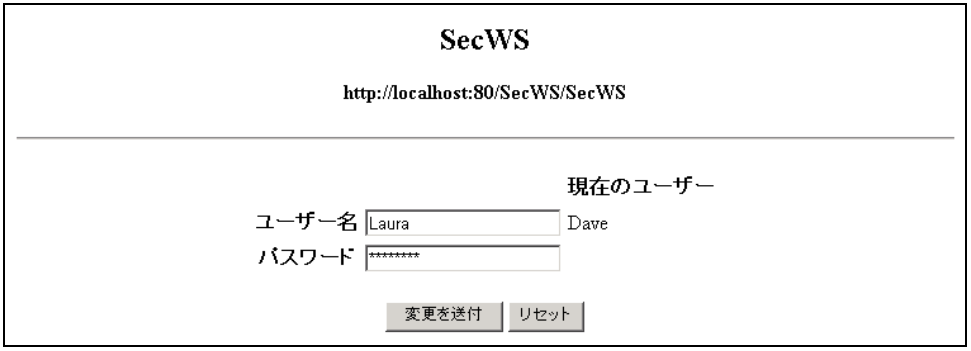

図 A-1　　HTTP 基本認証ログインページ

HTTP 基本認証を実装するために、次の操作のうちのどれかを実行します。

- IDE に Web サービスの認証プロパティを設定する
- IDE での Web サービスとロールの関連付け
- アプリケーションサーバーまたは Web サーバーでユーザー (主体) とユーザーグループを定義する
- アプリケーションサーバーまたは Web サーバーでユーザーまたはグループをロールにマップする

これらの操作の実行方法については 174 ページの「HTTP 基本認証の利用」で説明しています。

このセキュリティ機構は Web サービス全体とその中のメソッドすべてに適用されます。

注 – IDE はさまざまな HTTP 基本認証を使用します。この認証内で、生成されたクライアントがログイン画面を作成したり、ユーザーパスワードがクライアントプロキシを介して Web サーバーのアプリケーションサーバーに渡されたりします。

HTTP/SSL 認証および暗号化

SSL (Secure Sockets Layer) では、データ暗号化、サーバー認証、メッセージの完全性、および任意のクライアント認証が提供されます。SSL とその関連するテクノロジを利用することで、ネットワーク上のアプリケーションでのセキュリティに関する 2 つの基本的な問題を解決できます。

- パスワード、鍵、およびデータの傍受
- 送信者または受信者の識別情報の偽装

公開鍵暗号化

SSL では「公開鍵暗号化」テクノロジが使用されています。このテクノロジでは、サーバーまたはクライアントのそれぞれが 2 つの要素で構成される鍵を持ちます。1 つを公開鍵、もう 1 つを非公開鍵といいます。非公開鍵は他者には公開されませんが、公開鍵は関連する相手に公開されます。これらの 2 つの要素は数学的な組み合わせになっていて、次の特性を持っています。

- 公開鍵によって暗号化されたメッセージは、非公開鍵によって複号化できる。
- 非公開鍵によって暗号化されたメッセージは、公開鍵によって複号化できる。
- 公開鍵から非公開鍵を数学的に求めることはできない。

このマニュアルでは非公開鍵の暗号化に関する数学については述べませんが、インターネットで情報を得ることができます。

たとえば、ユーザー A からユーザー B に対して公開鍵の添付されたメッセージがネットワークを介して送信された場合、そのメッセージはユーザー B だけが復号化することができます。ユーザー A の送信したメッセージを第三者がネットワーク上で傍受したとしても、読むことはできません。また、第三者が途中でメッセージを改ざんしたとしても、ユーザー B を欺くことはできません。これは、ユーザー B がメッセージを復号化したときにその部分が不正な表示となるためです。このように、公開鍵の暗号化はデータの暗号化とメッセージの完全性をサポートします。

公開鍵による暗号化では、デジタル署名を使用することもできます。ユーザー A が非公開鍵を使ってメッセージを暗号化すると、有効な署名が書き込まれたことになります。これは、ユーザー A の公開鍵を持つ受信者であれば、その暗号化を実行できるのがユーザー A だけであることを確認できるからです。このデジタル署名の実現過程は実際にはもっと複雑ですが、このマニュアルでは詳細を省きます。

ユーザー A の公開鍵はネットワークを介して配布されますが、受信者には通知された公開鍵が本当にユーザー A のものであるかを確認する手段が必要となります。これは第三者がユーザー A の名前をかたって公開鍵を配布することもできるためです。これを確認する方法として、信頼できる「認証局」から発行された「公開鍵証明書」を使用する方法があります。

公開鍵証明書

サーバーおよびクライアントは「公開鍵証明書」を交換することで、互いを認証しあいます。公開鍵証明書は、ID カードのデジタル版ともいえるものです。これは、認証局 (CA: Certificate Authority) と呼ばれる信頼できる機構から発行されています。

証明書は、所有者の名前と、証明書に含まれている公開鍵がその所有者に属していることを証明します。証明書にはまた、有効期限、証明書を発行した認証局の名前、およびその認証局のデジタル署名も含まれています。SSL テクノロジには、クライアントまたはサーバーがデジタル署名に基づき証明書とその内容を確認できるソフトウェアが含まれています。

SSL が有効になっているクライアントまたはサーバーには、「トラストストア」と呼ばれるデータベース (またはファイル) があります。これには信頼証明書が含まれていて、管理者によって維持されています。クライアントまたはサーバーは、実行中に受信した証明書がトラストストアにある証明書の所有者によって署名されているときにだけ受け入れます。SSL は、受信した証明書のデジタル署名を復号化するために信頼証明書の公開鍵を使います。

証明書は「X.509 公開鍵インフラストラクチャ (PKI: Public Key Infrastructure)」標準に準拠しています。PKI 標準およびそのテクノロジに関する詳細については、次の Web サイトを参照してください。

インターネット技術標準化委員会 (IETF: Internet Engineering Task Force) —
`http://www.ietf.org/html.charters/pkix-charter.html`

Sun Microsystems —
`http://docs.sun.com/source/816-6156-10/index.html`

団体によっては、有償の証明書を発行している商用認証局もあります。そういった商用の認証局からは、複数種類の証明書が発行されていることがあります。証明書が高額になるほど、所有者の身元確認が詳細になります。

企業や公的機関では、証明書発行用の証明書サーバーソフトウェアを使用して自身の発行するアプリケーションの認証局となることがあります。この場合、証明書の信頼性は発行元の企業や提携企業のドメインの中に限られることがあります。

Sun Microsystems の証明書サーバーソフトウェアに関しては、次の Sun の Web サイトに文書やダウンロード可能な関連情報が含まれています。

Sun™ ONE Certificate Server 4.7 —
`http://docs.sun.com/db/prod/sunone#hic`

Sun ONE アプリケーションサーバーでの証明書管理の方法については、次のマニュアルで説明しています。

『Sun ONE Application Server 7 セキュリティ管理者ガイド』
`http://docs.sun.com/db/doc/816-6482-10`

SSL ハンドシェイク

ブラウザが SSL で保護されている Web サイトにアクセスしたとします。目的の URL が HTTP ではなく HTTPS で開始されている場合、ブラウザはセキュリティ保護されたポートからサーバーに接続しようとします。一般的にクライアントがサーバーとの SSL セッションを開始する場合、接続されるのは SSL リスナーが関連付けられているポートです。

さて、この接続時にサーバーには有効な証明書があっても、ブラウザには証明書がなかったとします。これは、インターネット上での買い物を目的として、顧客がセキュリティ保護されたサーバーに接続するときに多く発生する状況です。言い換えれば、クライアントはサーバーに自身を証明する必要があるのです。クライアントの方にも証明書があれば、サーバーはクライアントを認証できます。この手順を「SSL 相互認証」といいます。

サーバーが SSL 対応に構成されている場合は、「ハンドシェイク」プロセスが開始されます。次にハンドシェイクプロセスの概要を説明します。

1. クライアントは、初期パケットをサーバーに送信します。これには、自身が利用できる暗号化アルゴリズムなどの情報が含まれています。
2. サーバーも、自身が利用できる暗号化アルゴリズムや証明書のコピーといった情報の含まれるパケットをクライアントに送信します。
3. クライアントはサーバーの証明書を確認します。

   確認の過程ではクライアントのトラストストアが使用されます。詳細は、171 ページの「公開鍵証明書」を参照してください。
4. 次に、クライアントはランダムセッション鍵を生成し、サーバーの証明書から得たサーバーの公開鍵でそのセッション鍵を暗号化します。

   セッション鍵は SSL 接続の間だけ有効な一時鍵です。セッション鍵は従来の 2 方向暗号化・復号化アルゴリズムで処理され、その鍵を交換する両方が知っている必要があります。
5. クライアントが暗号化したセッション鍵をサーバーに送信します。

   受信されたセッション鍵は、そのサーバーだけが自身の非公開鍵を使って復号化することができます。これで、クライアントとサーバーの両方がセッション鍵を持ちました。このセッションの間は、クライアントとサーバーの両方がこの鍵を使ってデータの暗号化と復号化を実行します。

この方法によって 2 つの問題を解消できます。

- この例では、クライアントが証明書を持っていなかったため、両方向のデータ受け渡しに使用する公開鍵をクライアントが暗号化してもサーバーは使用することができません。これを解消するためにはセッション鍵が必要です。

- 公開鍵の暗号化は大量の計算処理を必要とします。それよりは、証明書のデジタル署名を確認し、サーバーに対して生成されたセッション鍵の安全な転送を確保するために、公開鍵の暗号化を初期ハンドシェイクの間だけ適用する方が効率的です。セッション鍵を使うことでこの問題も解消されました。

HTTP 基本認証の利用

ここでの説明は、IDE で Web サービスが作成済みであることを前提としています。作成した Web サービスをセキュリティ保護するためには、次の作業が必要です。

- HTTP 基本認証に Web サービスの 認証プロパティを設定する
- 認証ロール名をWeb サービスへ割り当てる
- アプリケーションサーバーでユーザー (主体) とユーザーグループを定義する
- アプリケーションサーバーで、ロール名をユーザーとユーザーグループにマップする
- セキュリティ保護された Web サービスにアクセスするためのクライアントを作成・生成する

アプリケーションを実行すると、ユーザー名とパスワードを入力するための Web ページへのリンクがブラウザの開始画面に表示されます。

ユーザーとユーザーグループへのロールのマッピング

Web サービスに HTTP 基本認証ロールを割り当てる手順は次のとおりです。

1. **Web サービスノードを右クリックし、「プロパティ」を選択します。**
2. **「認証」プロパティを 「HTTP 基本 認証」に設定します。**

   「認証」ダイアログが表示されます (図 A-2 を参照)。
3. **「HTTP 基本認証」のラジオボタンを選択します。**

   「追加」ボタンと「削除」ボタンが使用可能な状態になります。

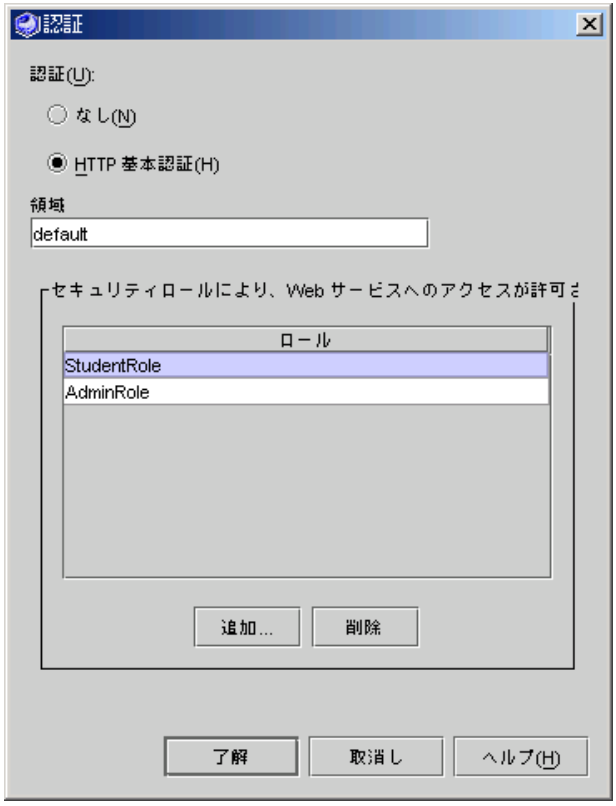

図 A-2　「認証」ダイアログ

4. 「追加」をクリックしてロールを追加します。

「ロールを追加」ダイアログが表示されます (図 A-3 を参照)。

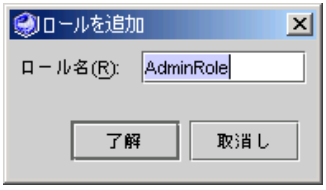

図 A-3　「ロールを追加」ダイアログ

5. ロール名の値を入力して「了解」をクリックします。

新規ロール名が「認証」ダイアログに表示されます。

ロール名をグループ名とユーザー名にマップします。マッピングは、Web サービスのレベルか、その Web サービスを含む J2EE アプリケーションのレベルで実行できます。

Web サービスレベルでのロールのマッピング

Web サービスレベルでロール名をグループ名およびユーザー名にマップする手順は次のとおりです。

1. Web サービスノードを右クリックし、「プロパティ」を選択します。
2. 「Sun ONE AS」タブをクリックします。

   「マップされたセキュリティロール」プロパティに Web サービスにマップしたロールの数が表示されます (図 A-4 を参照)。

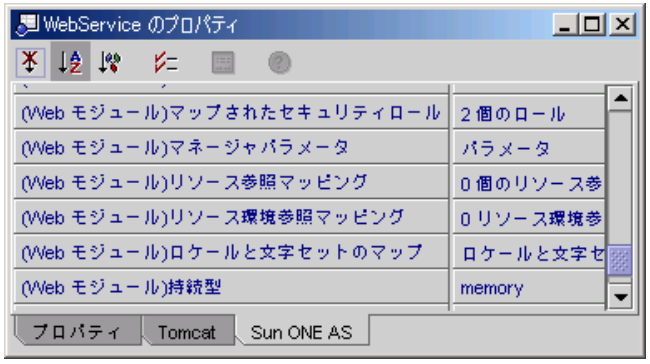

図 A-4　　「Sun ONE AS」プロパティ

3. 「マップされたセキュリティロール」プロパティをクリックします。

   省略符号ボタン (...) が表示されます。

4. 省略符号ボタン (...) をクリックします。

   「マップされたセキュリティロール」ダイアログが表示されます (図 A-5 を参照)。

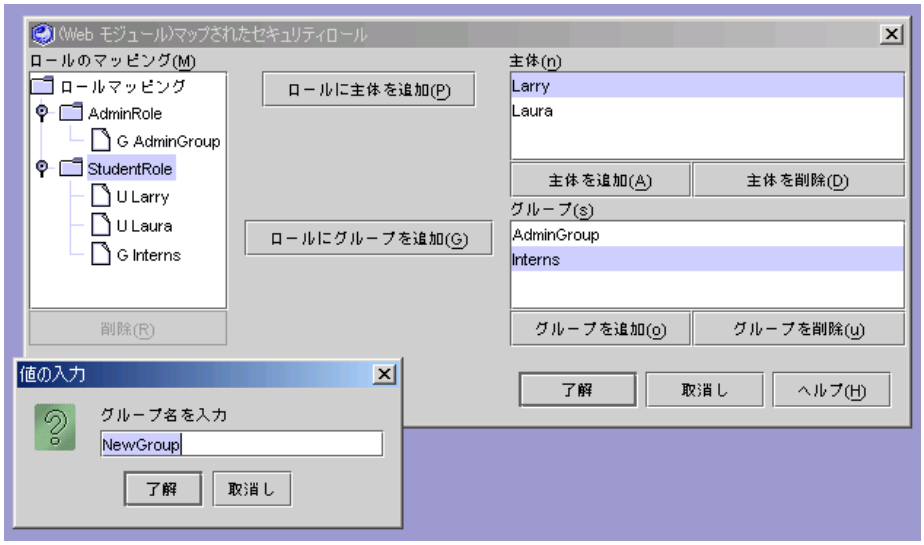

図 A-5　　「マップされたセキュリティロール」ダイアログ

「マップされたセキュリティロール」ダイアログには、主体、グループ、ロール、およびそれぞれのマップが表示されます。次のようなことができます。

- 主体 (ユーザー) の追加と削除
- グループの追加と削除
- ロールに対する主体とグループのマップの追加
- ロールに対する主体とグループのマップの削除

a. 主体またはグループを追加するには、「主体を追加」または「グループを追加」をクリックします。

「値の入力」ダイアログが表示されます (図 A-5 を参照)。

i. 名前となる値を入力します。

ii. 「了解」をクリックします。

b. ロールに対して主体またはグループをマップする手順は次のとおりです。

i. 主体またはグループを選択します。

複数の主体または複数のグループを選択する場合は、Ctrl キーを押しながら目的の主体またはグループをクリックします。

ii. ロールを選択します。

主体またはロールをのどちらかを選択することで、「ロールに主体を追加」ボタンまたは「ロールにグループを追加」ボタンが有効になります。

iii. 「ロールに主体を追加」ボタンまたは「ロールにグループを追加」ボタンをクリックします。

選択した主体またはグループがマップされ、選択したロールの下に表示されます。

J2EE アプリケーションレベルでのロールのマッピング

ここでの説明は、Web サービスが J2EE アプリケーション内に表示されている状態から進めます。また、Web サービスの 認証プロパティは HTTP 基本認証 で、Web サービスには必要なロールが割り当て済みになっているものとします。J2EE アプリケーションレベルでロール名をグループ名およびユーザー名にマップする手順は次のとおりです。

これを行うには次の手順に従ってください。

1. **Web サービス用に J2EE アプリケーションを作成します** (32 ページの「J2EE アプリケーションのアセンブル」 を参照)。
2. **アプリケーションノードを右クリックして「プロパティ」を選択します。**
3. **「Sun ONE AS」タブをクリックします。**
   「マップされたセキュリティロール」プロパティに Web サービスにマップしたロールの数が表示されます。
4. **「マップされたセキュリティロール」プロパティをクリックします。**
   省略符号ボタン (...) が表示されます。
5. **省略符号ボタン (...) をクリックします。**
   「マップされたセキュリティロール」ダイアログが表示されます (図 A-5 を参照)。

以降の手順は、Web サービスレベルでのマッピングで説明した手順と同じです (176 ページの「Web サービスレベルでのロールのマッピング」を参照してください)。

複数ロールの使用

HTTP 基本認証は、Web サービス全体に適用されます。技術的には、認証は Web サービスを実装する JAXRPC サーブレットに適用されます。ある 1 つの Web サービスに対するすべてのロールは、その Web サービスとそのメソッドに対するアクセスの許可または拒否を決定するという同じ目的を持ちます。

目的が 1 つなら、なぜ複数のロールが必要になるのでしょうか。たとえば、Web サービスにロールを 1 つだけ割り当てて、すべての認証済みユーザーとユーザーグループをそのロールにマップすることもできます。

しかし、製品としてのアプリケーションサーバー 1 台において複数の Web サービスが実行されている環境を考えてみてください。そしてその環境には、複雑な構成を持つ組織から複数のユーザーがアクセスしているとします。セキュリティ保護を目的として、これらのユーザーとユーザーグループを Web サービスにマップするのはかな

り高度になります。複数のロール名を Web サービスへ割り当てる機能、また、逆に、与えられた単一のロール名を複数の Web サービスに割り当てる機能があれば、こういった複雑なセキュリティ保護の関係の設定・維持が簡単になります。

さらに、複数ロールを積極的にサポートするのは、メソッドレベルでのセキュリティ保護が Web サービステクノロジが進化していく 1 つの方向であるためです。Web サービス内でユーザーを異なるロールにマップする機能は、メソッドレベルでのセキュリティ保護を構成する 1 つの要素です。

ロールのマッピングとその優先度

Web サービスレベルおよび J2EE アプリケーションレベルでロールを主体とグループにマップした場合、Web サービスレベルでのマッピングが優先されます。

アプリケーションサーバーでのロールのマッピング

ユーザーおよびユーザーグループに対するロール名のマッピングは IDE によって実行され、そのマップはアプリケーションサーバーに保存されます。Web サービスを他のアプリケーションサーバーに配備した場合は、アプリケーションサーバーによって提供されている管理ツールを使用してマッピングをやり直す必要があります。製品環境では、アプリケーションサーバー管理者がマッピングを実行することもあります。

注 – この付録で説明している手順は、IDE にバンドルされている Sun ONE Application Server を使った操作です。

ロール名のリダイレクト

特定の状況では、Web サービスに対応付けられているロール名がアプリケーションサーバーで使用できないことがあります。例えば、ロール名が、販売する商品名の命名規則に矛盾したり、すでに使用されている他のロール名と衝突したりすることがあります。このとき、他社から購入した Web サービスを使用していたり、Web サービスが複数のアプリケーションによって共有されていたりすると、ロール名を直接変更するのは難しいでしょう。

この問題を解決するために、IDE ではロール名を J2EE アプリケーションレベルでリダイレクトできる機能を提供しています。この機能では J2EE アプリケーションで定義したロール名を、アプリケーションサーバーで使用される異なるロール名にリンクすることができます。

ロール名をリダイレクトする手順は次のとおりです。

1. **アプリケーションノードを右クリックして「プロパティ」を選択します。**
   「プロパティ」タグに「セキュリティロール」プロパティが割り当てられたロールの数と共に表示されます。
2. **「セキュリティロール」プロパティをクリックします。**
   省略符号ボタン (...) が表示されます。
3. **省略符号ボタン (...) をクリックします。**
   「セキュリティロール」ダイアログが表示されます (図 A-6 を参照)。

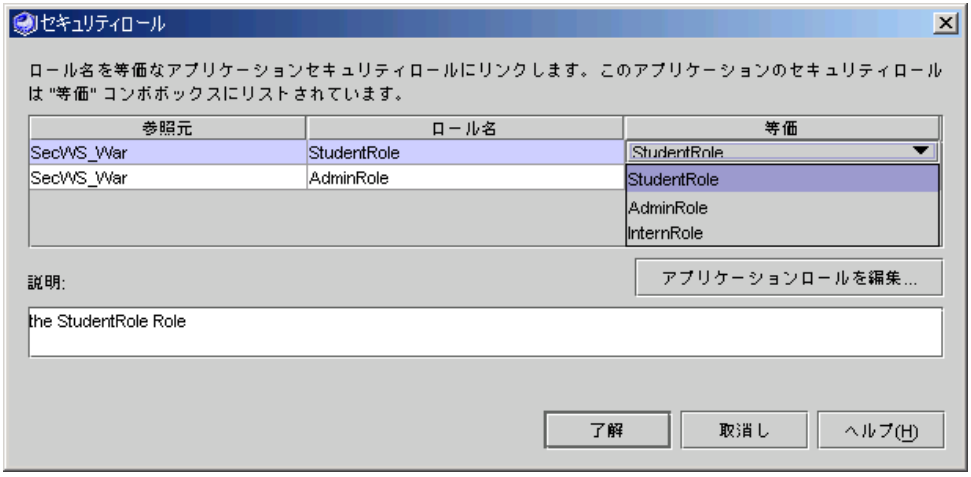

図 A-6　「セキュリティロール」ダイアログ

「セキュリティロール」ダイアログに WAR ファイル、ロール名、およびリンクされているロール名が表示されます。「等価」と表示された列には、リンクされたターゲットロール名が、ロール名に等しいデフォルト値と共に表示されます。

ロール名をターゲットロール名にリンクする手順は次のとおりです。

a. **「等価」列中の値をクリックします。**
   指定できるターゲットロール名のリストが表示されます。
b. **目的のターゲットロール名を選択します。**
   新規名が「等価」列に表示されます。

「セキュリティロール」ダイアログに表示される指定できるターゲットロール名のリストでは、エントリやエントリの説明を追加または削除できます。

エントリを追加する手順は、次のとおりです。

1. 「アプリケーションロールの編集」をクリックします。

   「アプリケーションロールの編集」ダイアログが表示されます (図 A-7 を参照)。

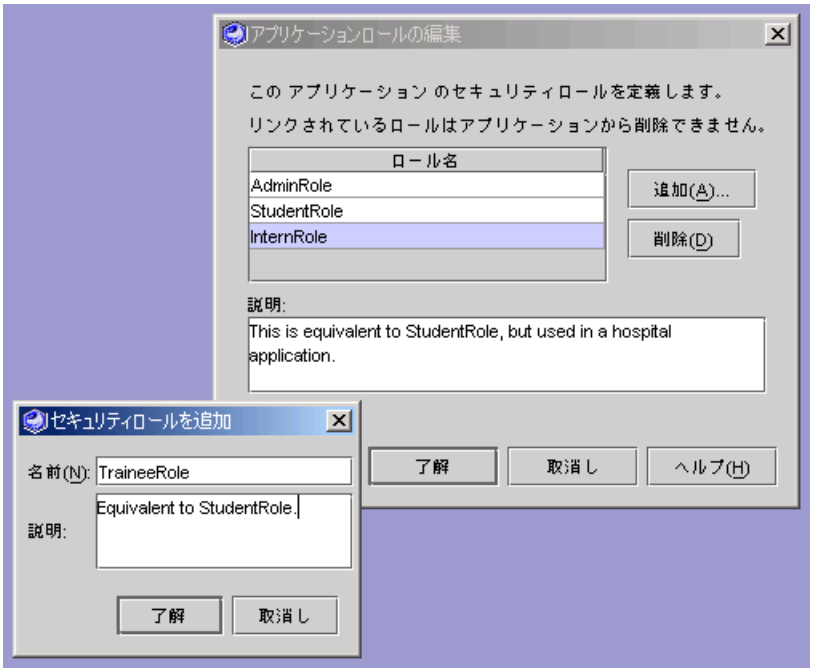

図 A-7　　「アプリケーションロールの編集」ダイアログ

ロール名に説明を付けて追加できます。現在リンクされていないロール名は、削除することもできます。

2. 「追加」をクリックします。

   「セキュリティロールの追加」ダイアログが表示されます。

3. 名前と説明を入力して、「了解」をクリックします。

   「等価」列の値をクリックすると、指定できるロールのリストに新規名が表示されます。

セキュリティ保護の検証と HTTP 基本認証

ここでの説明は、クライアントが作成済みで、そのクライアントがテストする Web サービスのデフォルトのテストクライアントになっていることを前提としています。テストクライアントの作成と利用についての手順は、35 ページの「Web サービスのテスト」を参照してください。

注 – Web サービスの 「認証」プロパティの値が `HTTP` **基本認証**となっている場合、IDE は「HTTP 基本認証」プロパティに `True` の値を持つクライアントを作成します。生成されたクライアントは、「生成されたドキュメント」フォルダに JSP ログインページを持っています。

HTTP 基本認証をテストするには、アプリケーションサーバーで提供されている管理ツールを使って、そのサーバー内でのテストユーザーを定義する必要があります。操作方法については、アプリケーションサーバーのマニュアルを参照してください。次の例では、Sun ONE Application Server 7 での管理ツールの開始方法を説明しています。

アプリケーションサーバー管理ツールを開始する手順は次のとおりです。

1. IDE の「実行時」タブでアプリケーションサーバーインスタンスを右クリックし、「Sun ONE 管理 GUI を起動」を選択します。

   IDE によってデフォルトの Web ブラウザが開始され、ブラウザに管理ツールが表示されます (図 A-8 を参照)。

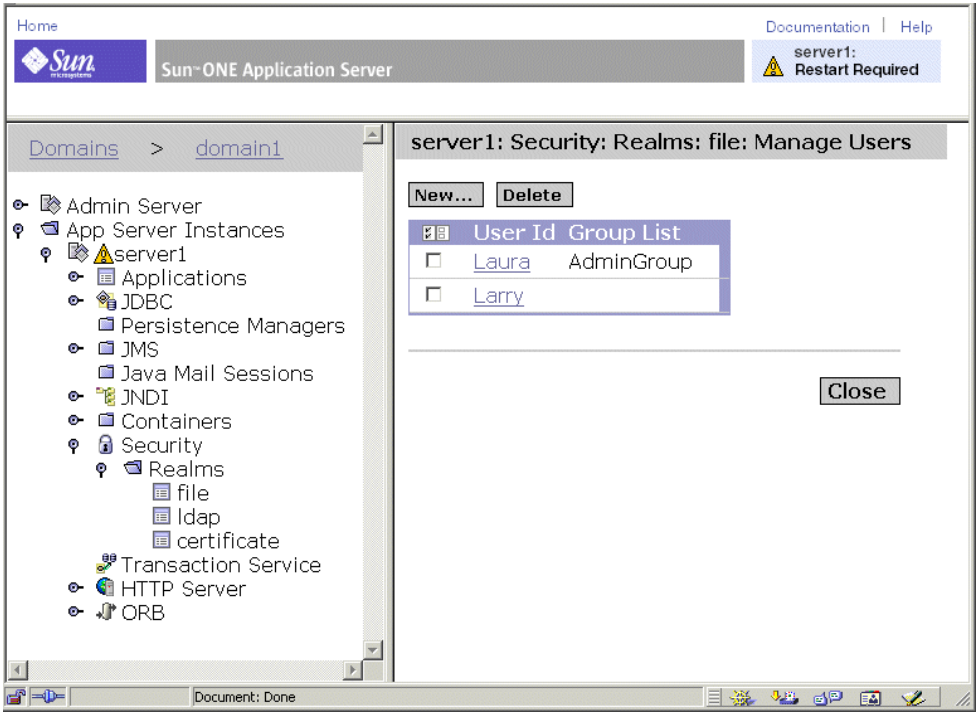

図 A-8 アプリケーションサーバー管理ツールとユーザーの管理

2. 「server1」->「セキュリティ」->「レルム」->「file」->「ユーザーを管理」までナビゲートします。

   管理ツールの「ユーザーを管理」機能を使うと、パスワードを伴う新規ユーザーの追加、ユーザーとグループとの関連付け、およびユーザーの削除が実行できます。

   以降の説明では、アプリケーションサーバー内でテストユーザーが定義済みであり、そのユーザーがテストする Web サービスに関連付けられたロールにマップされていることを前提としています。ロールとユーザーのマッピングについては 174 ページの「ユーザーとユーザーグループへのロールのマッピング」を参照してください。

   Web サービスのセキュリティ保護をテストする手順は次のとおりです。

1. アプリケーションを実行します。

   アプリケーションをテストクライアントと共に実行する手順については 35 ページの「Web サービスのテスト」を参照してください。操作手順は、Web サービスが多層アーキテクチャと Web 主体アーキテクチャのどちらを採用しているかによって異なります。

   テストクライアントによってアプリケーションの開始ページがブラウザに表示されます。ページの最上部にある認証リンクは「HTTP 基本認証用にユーザー名とパスワードを入力してください。」と表示されています。ページの下の部分には Web サービスのビジネスメソッドで使用されるボタンや入力フィールドが含まれています。

2. 認証リンクをクリックします。

   ブラウザによってログインページが表示されます (図 A-1 を参照)。

3. 無効なユーザー名またはパスワードを入力して、「変更を適用」をクリックします。

   ブラウザによって開始ページが再び表示されます。

4. 必要な任意のデータを入力し、アプリケーションのビジネスメソッドのボタンをクリックします。

   ブラウザによってエラーメッセージが表示されます。

5. 認証リンクをクリックします。

   ブラウザによってログインページが表示されます。

6. 有効なユーザー名およびパスワードを入力して、「変更を適用」をクリックします。

   ブラウザによって開始ページが再び表示されます。

7. 必要な任意のデータを入力し、アプリケーションのビジネスメソッドのボタンをクリックします。

   アプリケーションは認証されたユーザーに対する動作を実行します。

基本認証と WSDL

WSDL 標準では認証は提供されません。したがって、UDDI レジストリまたは WSDL からクライアントを生成した場合は、クライアントの「HTTP 基本認証」プロパティは、デフォルト値である `False` に設定されます。Web サービスが HTTP 基本認証をサポートしていることがわかっている場合は、クライアントのプロパティを手動で `True` に設定しておくことができます。

基本認証と UDDI 公開

デフォルトでは、Web サービスを UDDI レジストリに公開すると、Web サービスを介して WSDL ファイルにアクセスする WSDL URL が IDE によって指定されます。Web サービスが HTTP 基本認証によって保護されている場合は、クライアント開発者がクライアントを開発するための WSDL ファイルをダウンロードできないことがあります。IDE はこの問題を次のように解決します。

「認証」プロパティに `HTTP` **基本認証**が指定されている Web サービスを公開した場合、IDE は WAR ファイルの最上位ディレクトリに WSDL のコピーをパッケージ化します。Web サービスを公開する過程では Web サービスを呼び出すことなしに WSDL ファイルのコピーを参照させるような静的 WSDL URL を手動で提供しておく必要があり、この処置を取っておくことはWeb サービスの公開者の責任です。

UDDI レジストリに対する Web サービスの公開手順の詳細については、53 ページの「公開手順」を参照してください。WSDL URL を手動で編集するためのウィザードのページについては、図 2-31 を参照してください。ウィザードページのタイトルは、「サービスと WSDL を公開しますか、それともサービスだけを公開しますか?」となっています。

- **ウィザードで WSDL ファイルの URL を次のように変更します。**

  変更前

  `http://`*<hostname>*`:`*<portnumber>*`/`*<wsname>*`/`*<wsname>*`?WSDL`

  変更後

  `http://`*<hostname>*`:`*<portnumber>*`/`*<wsname>*`/`*<wsname>*`.wsdl`

HTTP/SSL 認証および暗号化の使用

製品環境での SSL セキュリティと証明書を管理するには、アプリケーションサーバー管理者とセキュリティ保護されたプロシージャが必要になります。開発環境では、開発者がアプリケーションサーバー管理者となるか、誰かに管理を依頼する必要があります。開発チーム全体の証明書管理を担当できる人物を、チームの一員に加えておくと便利でしょう。

この節での説明は、開発環境内で Web サービスを IDE にバンドルされている Sun ONE Application Server に配備することを前提にしています。

この節でいう「トラストストア」は、証明書が保持されているデータベース (または単純なファイル) を指しています。トラストストアの場所と形式、およびトラストストアを管理するためのツールは、使用している環境とアプリケーションサーバーによって異なります。

注 – このマニュアルでは「トラストストア」という用語の代わりに「キーストア」という用語が使われることもあります。通常のセキュリティに関する用語では、「トラストストア」は、あるエンティティによって使用される信頼のおける証明書を別のエンティティが保持しているデータベースを指しています。「キーストア」とは、あるエンティティが自身のキーまたは証明書を保持しているデータベースを指しています。しかし、これらの 2 つのデーターベースは同じこともあり、IDE は特に区別をしません。図 A-10 に、この両方の目的を果たす IDE オプションである SSL トラストストアを示しています。

Web サービス、テストクライアント、および SSL 認証と暗号化用のアプリケーションサーバーを構成するには、次の操作を実行する必要があります。

- Web サービスとテストクライアントに対する「HTTPS を使用」プロパティを `True` に設定します。
- Web サービスの証明書のコピーを保存するクライアントのトラストストアを作成します。
- クライアントトラストストアの場所をクライアントに知らせます。
- サーバーの証明書をエクスポートし、それをクライアントトラストストアにインポートします。
- アプリケーションサーバーで各ポートに対する SSL リスナーを作成します。これらのポートは、SSL セキュリティで保護された Web サービスのどれかが使用するものです。

  リスナーを作成することによって、クライアントがサーバーを認証することができます。認証を成功させるには、クライアントトラストストアにサーバーが所有する証明書のコピーが保存されている必要があります。
- (省略可能) クライアントの認証を有効にする手順は次のとおりです。
  - クライアントの証明書を取得します。
  - クライアントの証明書をクライアントトラストストアにインポートします。
  - クライアントの証明書をサーバートラストストアにインポートします。
  - アプリケーションサーバーで、Web サービスが使用するポートに対するクライアントの認証を有効にします。

  このことによって、サーバーがクライアントを認証することができます。認証を成功させるには、サーバートラストストアにクライアントが所有する証明書のコピーが保存されている必要があります。

Web サービスとテストクライアントへのプロパティの設定

HTTPS/SSL 認証用に Web サービスを構成する手順は次のとおりです。

1. Web サービスノードを右クリックし、「プロパティ」を選択します。
2. 「HTTPS を使用」プロパティを `True` に設定します。

テストクライアントを作成すると、IDE によって、「HTTPS を使用」プロパティに対応するWeb サービスのプロパティが同じように設定されます。既存のテストクライアントに対して次の操作を実行します。

1. クライアントノードを右クリックし、「プロパティ」を選択します。
2. 「HTTPS を使用」プロパティが、Web サービスの「HTTPS を使用」プロパティに一致するように設定します。

注 – WSDL 標準では認証は提供されません。したがって、UDDI レジストリまたは WSDL からクライアントを生成した場合は、クライアントの「HTTPS を使用」プロパティは、デフォルト値である `False` に設定されます。Web サービスが HTTP/SSL 基本認証をサポートしていることがわかっている場合は、クライアントのプロパティを手動で `True` に設定しておくことができます。

テストクライアントトラストストアの設定

最初にトラストストアを作成する必要があります。この節では、`keytool` ユーティリティを使ってトラストストアを作成する方法を説明します。このユーティリティは、Java™ 2 Software Development Kit, Standard Edition, 1.4 または 1.4.1 (J2SE™ SDK) で提供されているものです。

トラストストアの作成

注 – 製品環境では、トラストストアは一般的に、セキュリティ管理者によってセキュリティ保護された場所で設定されます。

UNIX システムでトラストストアを作成する手順は次のとおりです。

- トラストストアを作成するディレクトリに移動し、次のコマンドを指定します。

```
$JAVA_HOME/bin/keytool -genkey -alias localhost -dname "CN=<client
name>, OU=<organizational unit>, O=<organization>, L=<locality>, S=<state>,
C=<country code>", -keyalg RSA - keypass changeit -storepass changeit
-keystore client.keystore
```

Windows システムでトラストストアを作成する手順は次のとおりです。

- トラストストアを作成するディレクトリに移動し、次のコマンドを指定します。

```
%JAVA_HOME%\bin\keytool -genkey -alias localhost -keyalg RSA -keypass
changeit -storepass changeit -keystore client.keystore
```

keytool ユーティリティによって、クライアントのサーバーの名前、部署名、組織名、地域名、行政地域名 (県名など)、および国別コードの入力が促されます。サーバー名は、名字と名前の入力を促す最初のプロンプトで入力する必要があります。

図 A-9 に、Windows 環境でのトラストストアの作成を示します。

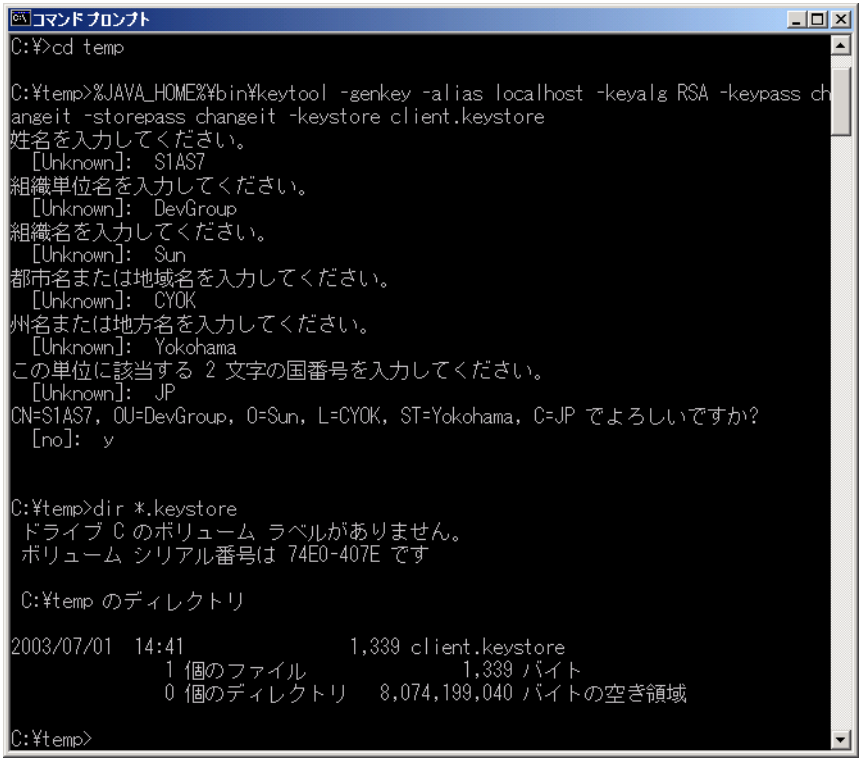

図 A-9 クライアントトラストストアの作成

IDE の広域トラストストアの設定

IDE にクライアントトラストストアの場所を知らせる手順は次のとおりです。

1. IDE のメインウィンドウから「ツール」->「オプション」->「分散アプリケーションサポート」->「Web サービス設定」を選択します。

   図 A-10 に示す「オプション」ダイアログが表示されます。

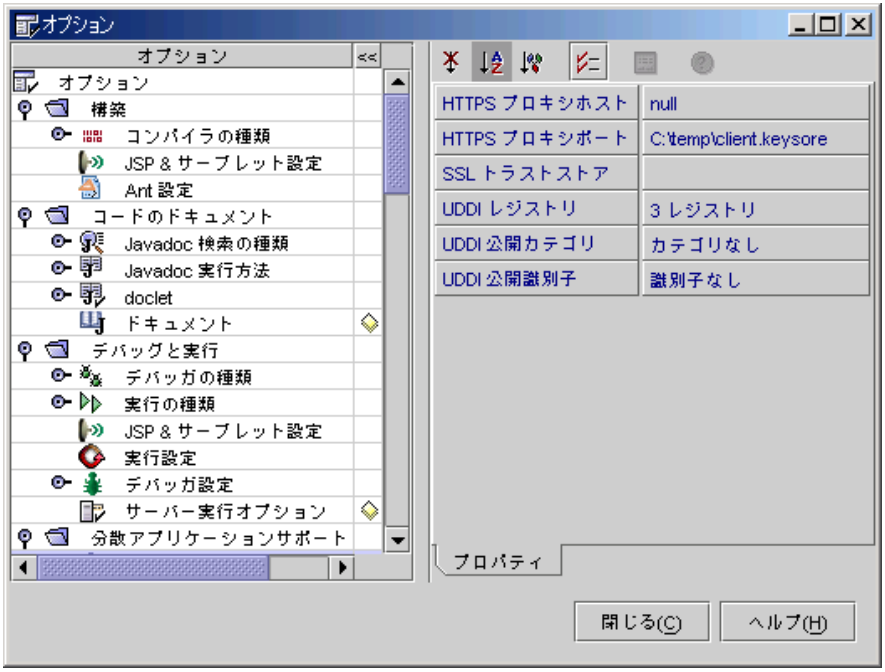

図 A-10　「SSL トラストストア」オプションの設定

2. 「SSL トラストストア」オプションをクリックしてから、省略符号ボタン (...) をクリックします。

   IDE がファイル選択ダイアログを開きます。

3. トラストストアファイルを選択して、「了解」をクリックします。

テストクライアントトラストストアへのサーバーの証明書のインポート

サーバー管理者は、サーバー管理ツールを使って証明書をファイルにエクスポートする必要があります。開発環境で、開発者が自分でこの操作を実行する場合は、手順をアプリケーションサーバーのマニュアルで確認してください。次に、Sun ONE Application Server 7 での手順を参考例として示します。

アプリケーションサーバー管理ツールを開始する手順は次のとおりです。

1. IDE の「実行時」タブでアプリケーションサーバーインスタンスを右クリックし、「Sun ONE 管理 GUI を起動」を選択します。

   IDE によってデフォルトの Web ブラウザが開始され、ブラウザに管理ツールが表示されます。

2. 「server1」の「セキュリティ」までナビゲートし、「証明書管理」タブをクリックしてメニューから「管理」を選択します。

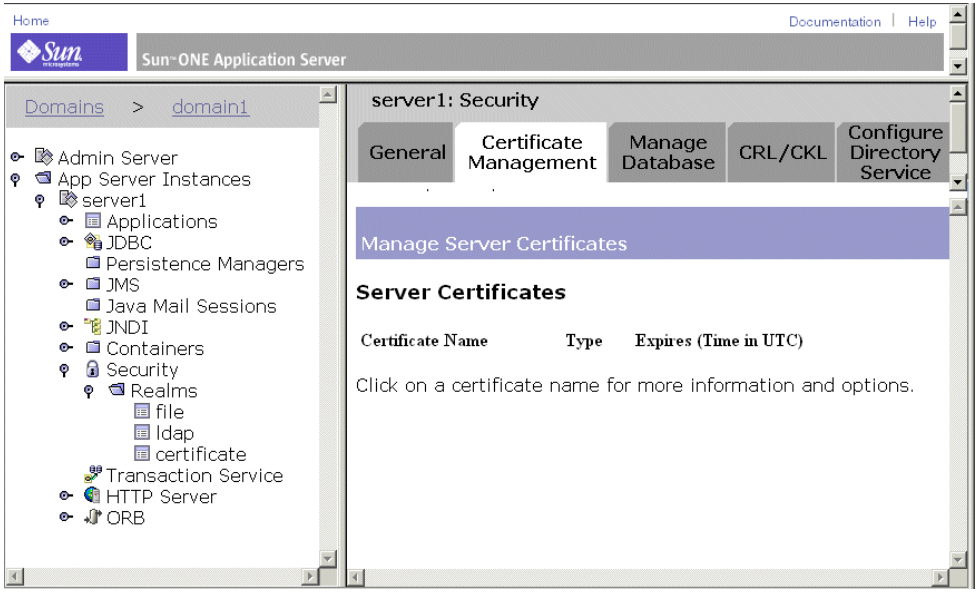

図 A-11　証明書の管理を表示したアプリケーションサーバー管理ツール

3. サーバーの証明書をファイルにエクスポートします。

   次に、keytool ユーティリティを使用して、サーバーの証明書を自身のクライアントトラストストアにインポートします。keytool コマンドは命名規則を除いて UNIX と Windows の両方で同じです。次のコマンドの例では、サーバーの証明書が *server.cer* という名前のファイルにエクスポート済みであることを前提としています。

   サーバーの証明書をクライアントトラストストアにインポートする手順は次のとおりです。

- 次のコマンドを入力します。

  ```
  $JAVA_HOME/bin/keytool -import -v trustcacerts -alias localhost -file
  server.cer -keystore client.keystore -keypass changeit -storepass changeit
  ```

  クライアントトラストストアには複数の信頼証明書を保存できます。

  クライアントトラストストアの内容を表示する手順は次のとおりです。

- **次のコマンドを入力します。**

  `$JAVA_HOME/bin/keytool -list -keystore` *client.keystore*

  `keytool` ユーティリティによって、トラストストアのパスワードの入力が促されます。

  図 A-12 に、Windows 環境でのトラストストアのリストを示します。

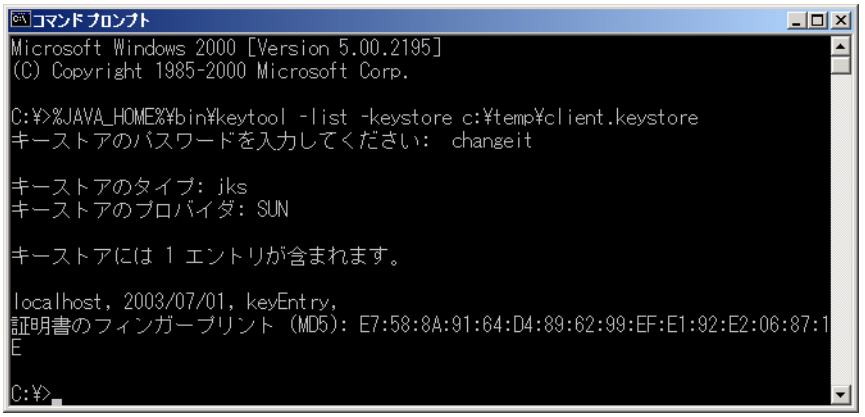

図 A-12 クライアントトラストストアのリスト

注 – 複数の開発者がいるチームでは、一人がチーム全体の証明書の管理を担当すると便利です。この開発者は、チームに対して単一のクライアントトラストストアを設定し、そこにすべての必要な証明書をインポートできます。

サーバーの設定

サーバー管理者は、Web サービスが使用するポートに対して SSL リスナーを作成する必要があります。開発環境で、開発者が自身でこの操作を実行する場合は、手順をアプリケーションサーバーのマニュアルで確認してください。次に、Sun ONE Application Server 7 での手順を参考例として示します。

アプリケーションサーバー管理ツールを開始する手順は次のとおりです。

1. **IDE の「実行時」タブでアプリケーションサーバーインスタンスを右クリックし、「Sun ONE 管理 GUI を起動」を選択します。**

   IDE によってデフォルトの Web ブラウザが開始され、ブラウザに管理ツールが表示されます。

2. 「server1」-「HTTP サーバー」-「HTTP リスナー」までナビゲートします。

   ここで表示されるページには既存の HTTP リスナーが表示されるので、その中からリスナーを削除したり、新規リスナーを追加したりできます (図 A-13 を参照)。

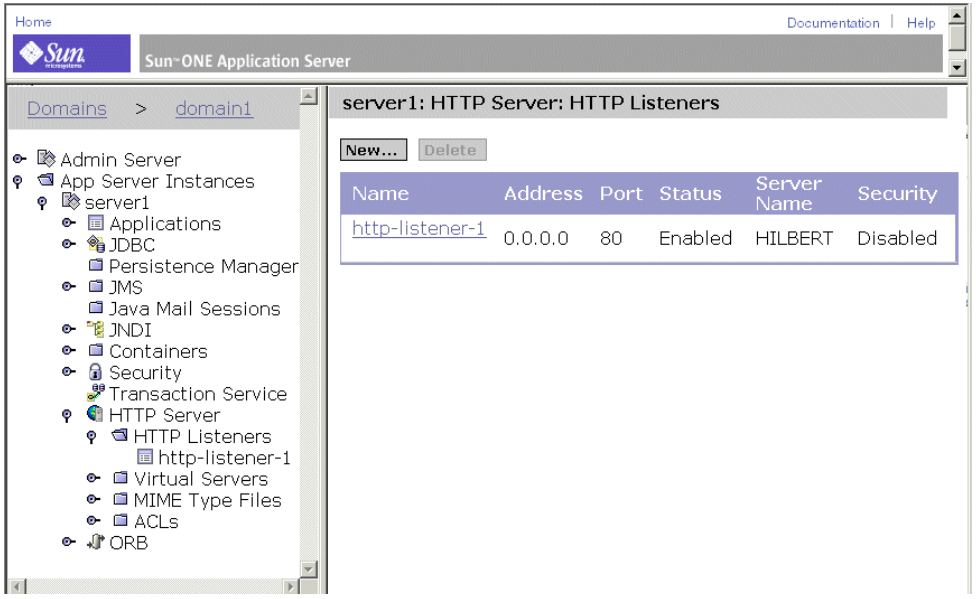

図 A-13　HTTP リスナーを表示したアプリケーションサーバー管理ツール

3. 既存の HTTP リスナーまでナビゲートして、プロパティを表示または変更できます (図 A-14 を参照)。

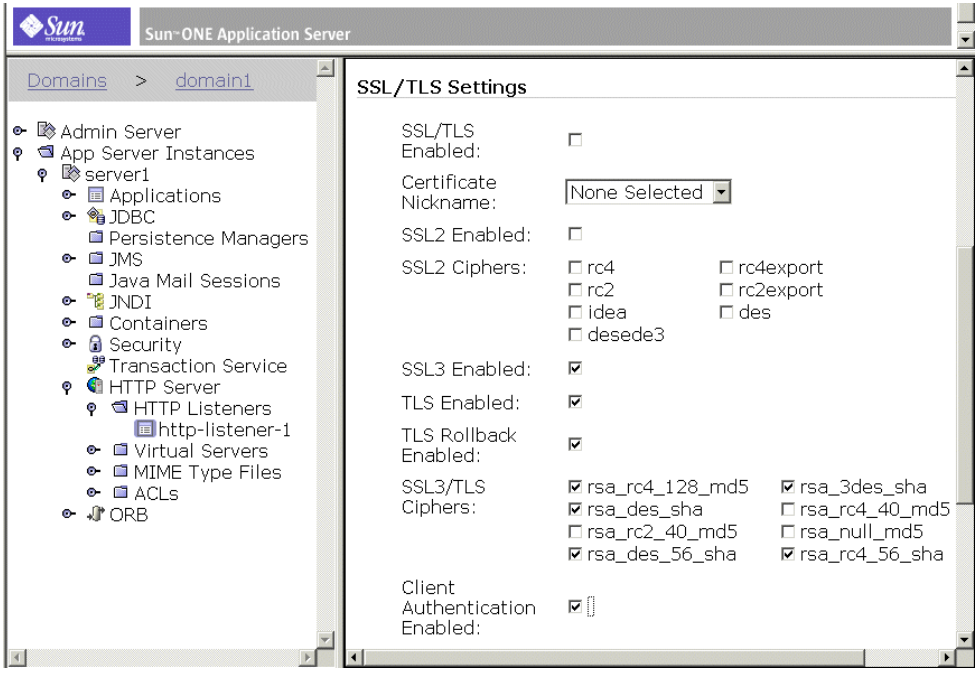

図 A-14　「HTTP リスナー」プロパティを表示したアプリケーションサーバー管理ツール

注 – サーバーに相互 (双方向) 認証を設定する場合は、Web サービスが使用するポートに関連付けられているリスナー用に「クライアント認証を有効」チェックボックスにチェックを入れます。

セキュリティ (HTTP/SSL 認証および暗号化) のテスト

SSL サーバー認証を使った Web サービスのセキュリティ保護をテストする手順は次のとおりです。

1. **アプリケーションを実行します。**

   アプリケーションをテストクライアントと共に実行する手順については 35 ページの「Web サービスのテスト」を参照してください。操作手順は、Web サービスが多層アーキテクチャと Web 主体アーキテクチャのどちらを採用しているかによって異なります。

   テストクライアントによってアプリケーションの開始ページがブラウザに表示されます。ページの最上部にあるリンクは「トラストストアとパスワードを入力」と表示されています (図 A-15 を参照)。ページの下の部分には Web サービスのビジネスメソッドで使用されるボタンや入力フィールドが含まれています。

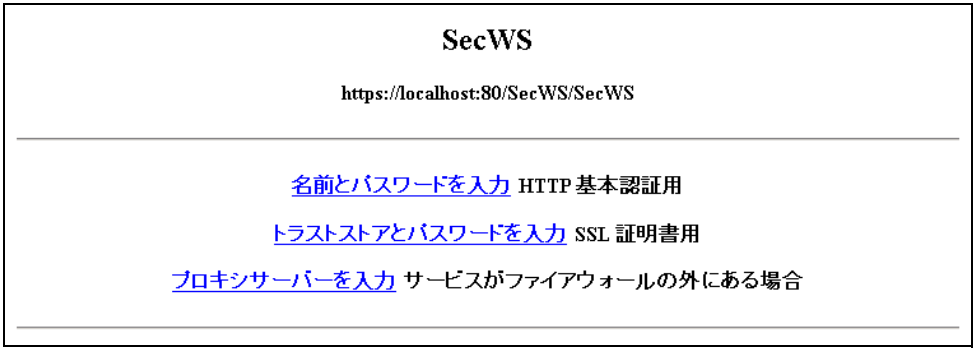

図 A-15　セキュリティリンクが表示されたクライアント開始ページ

2. **SSL 証明書リンクをクリックします。**

   クライアントのトラストストアのある場所を入力するためのページがブラウザに表示されます (図 A-16 を参照)。

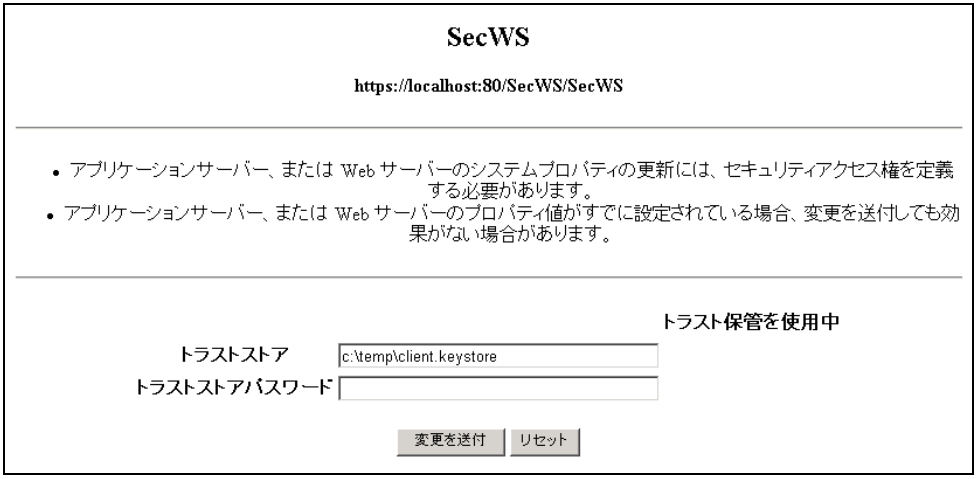

図 A-16 トラストストアとトラストストアのパスワードを設定するクライアントページ

このページは、IDE が生成したクライアントによって表示される開発環境をテストするためのページです。このページで設定したトラストストアの位置は、クライアントがその内部で実行されるサーバーによって使用されます。製品環境では、トラストストアの位置は一般的に、クライアントではなくセキュリティ管理者によって設定されます。

注 – トラストストアの場所をサーバーセッション内で 1 度だけ設定できます。トラストストアの場所を変更するには、サーバーを 1 度停止してから再起動する必要があります。これは、製品環境内でのトラストストアの場所は、クライアントではなく、セキュリティ管理者によって設定されているという前提に基づいてサーバーが設計されているためです。

---

3. **クライアントのトラストストアの場所とパスワードを入力し、「変更を送付」をクリックします。**

   ブラウザによって開始ページが再び表示されます。

4. **必要な任意のデータを入力し、アプリケーションのビジネスメソッドのボタンをクリックします。**

   アプリケーションは認証されたユーザーに対する動作を実行します。

注 – サーバーもクライアント認証をするように構成されている場合は、クライアントに証明書が必要です。また、サーバーの管理ツールを使ってクライアントの証明書をサーバーのトラストストアにインポートしておく必要もあります。

付録B

配備記述子

Web サービス用の実行時クラスを生成すると、Web モジュールおよび EJB モジュール配備記述子も生成されます。Web サービス J2EE アプリケーションのアセンブル時には、これらの配備記述子がアプリケーションに含まれます。配備記述子は、アプリケーションの実行時プロパティを設定するために使用される XML ファイルです。J2EE アプリケーションはこれらの記述子のフォーマットを定義します。

Web サービスの配備記述子は、開発中はいつでもソースエディタ内で参照できます。配備記述子を編集することもできます。この付録では、配備記述子の参照と編集方法について説明します。

配備記述子に伝達されるフィールド

ここでは、標準で Web サービスの配備記述子に伝達される IDE 内のフィールドを示します。これらの配備記述子の 1 つを編集した後に、標準で配備記述子に伝達されるフィールドの値を変更した場合は、配備記述子を手動で編集して新しい値を入力する必要があります。

EJB モジュール配備記述子に伝達されるフィールド

「オブジェクトの解決」ダイアログには、EJB モジュール配備記述子に伝達されるいくつかのフィールドが含まれます。配備記述子内の対応する要素は、すべて `ejb-ref` 要素のサブ要素です。次の表は、フィールドの一覧です。

「オブジェクトの解決」ダイアログ内のフィールド	EJB モジュール配備記述子内の要素
EJB 参照名	`description`
EJB 参照名	`ejb-ref-name`
EJB の型	`ejb-ref-type`
EJB ホームインタフェース	`home`
EJB リモートインタフェース	`remote`
EJB 名	`ejb-link`

注 – EJB メソッド呼び出しを XML コンポーネントの 1 つに追加すると、新しいターゲットオブジェクト定義が作成されます。メソッド呼び出しを特定の EJB コンポーネントに初めて追加するときに、「オブジェクトの解決」ダイアログに新しいターゲットオブジェクト定義が作成されます。一般的に、この定義は EJB モジュール配備記述子に新しい `ejb-ref` 要素として伝達されます。

Web サービスのプロパティシートには、「環境エントリ」という名前のプロパティがあります。このプロパティ内のフィールドは、EJB モジュール配備記述子に伝達されます。配備記述子内の対応する要素は、すべて `env-entry` 要素のサブ要素です。次の表は、フィールドの一覧です。

「環境エントリ」プロパティ内のフィールド	EJB モジュール配備記述子内の要素
名前	`env-entry-name`
説明	`description`
型	`env-entry-type`
値	`env-entry-value`

配備記述子の参照

Web モジュールまたは EJB モジュール配備記述子を表示する手順は、次のとおりです。

- **Web サービスノードを右クリックし、これらのメニュー項目の 1 つを選択します。**
  - 「配備記述子」->「Web モジュール」->「表示」
  - 「配備記述子」->「EJB モジュール」->「表示」

配備記述子は、ソースエディタ内で読み取り専用モードで表示されます。

配備記述子の編集

Web モジュールまたは EJB モジュール配備記述子を編集する手順は、次のとおりです。

1. **Web サービスノードを右クリックし、これらのメニュー項目の 1 つを選択します。**
   - 「配備記述子」->「Web モジュール」->「最終的な編集」
   - 「配備記述子」->「EJB モジュール」->「最終的な編集」

   図 B-1 に示す「最終的な編集」ダイアログが表示されます。このダイアログは配備記述子を編集した後に、Web サービスの実行クラスを生成し直したときに IDE によって記述子が再生成されないことを示します。

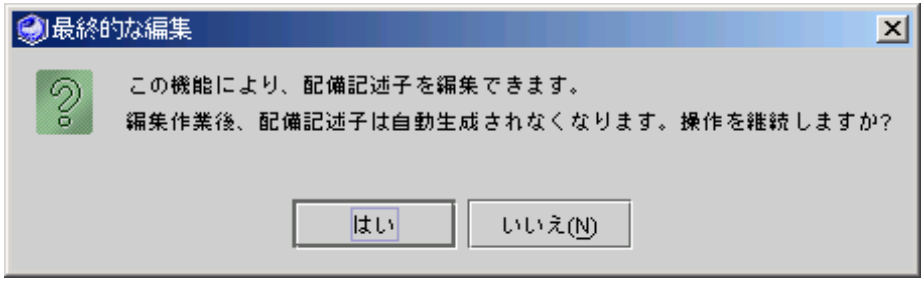

図 B-1　配備記述子の「最終的な編集」ダイアログ

2. **配備記述子を生成し直す必要がない場合は、「はい」をクリックします。それ以外の場合は、「いいえ」をクリックします。**

   配備記述子は、ソースエディタ内で編集モードで表示されます。Web サービス用の実行時クラスを生成し直す場合は、配備記述子は再生成されません。

索引

U

W

X

あ

え

お

か

く

け

こ

さ

し

す

せ

そ

た

て