Sun™ ONE Studio 5
J2EE アプリケーション
チュートリアル

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

Part No. 817-3299-10
2003 年 7 月, Revision A

原典： *Sun ONE Studio 5 J2EE Application Tutorial*
Part No: 817-2322-10
Revision A

目次

図目次

表目次

はじめに

Sun™ ONE Studio 5 J2EE アプリケーションチュートリアルにようこそ。このチュートリアルでは、Sun ONE Studio 5 統合開発環境 (IDE) に導入されている以下の機能の使用方法を学びます。

- EJB™ 2.0 ビルダー -『Enterprise JavaBeans Specification, Version 2.0』に基づく、Enterprise JavaBeans™ コンポーネントを開発、作成する
- EJB モジュールアセンブリ - EJB™ コンポーネントを EJB モジュールにアセンブルし、EJB Java Archive (JAR) ファイルにエクスポートする
- テストアプリケーション機能 - Sun ONE Application Server 7, Standard Edition ソフトウェアをアプリケーションサーバーとして使用し、クライアントを手動で作成せずに エンタープライズ Bean をテストする
- Web サービス機能 - 既存の EJB コンポーネントから SOAP Web サービスを構築し、Web ブラウザで表示可能な JSP™ ページを生成する
- Sun ONE Application Server への配備 - チュートリアル用アプリケーションをテストする

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

このチュートリアルでは、Java 2 Platform, Enterprise Edition (J2EE™) Blueprints リソースに記載されているアーキテクチャに準拠したアプリケーションを作成します。J2EE 準拠のアプリケーションを作成、開発、配備するために Sun ONE Studio 5, Standard Edition の機能を使用するには、このチュートリアルが役に立ちます。

チュートリアルを開始する前に、次の内容をよく理解しておく必要があります。

- Java™ プログラミング言語
- Enterprise JavaBeans の概念
- Java サーブレット構文
- JDBC™ 対応のドライバ構文
- JavaServer Pages™ 構文
- HTML 構文
- リレーショナルデータベースの概念 (表やキーなど)
- データベースの使用方法
- 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`

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

このマニュアルは、初めから順を追って読むことを前提に作成されています。チュートリアルの各章は、前の章で作成したコードに基づいて構成されています。

第 1 章では、この「DiningGuide」チュートリアルを学ぶにあたって必要なシステム要件を紹介します。また、Sun ONE Studio 5 IDE とSun ONE Application Server について、起動方法や IDE とサーバーを相互に認識させる方法、双方を Oracle データベースと交信させる方法、チュートリアルのデータベース表の作成方法、および、作成したデータベース表をもとに IDE でデータベーススキーマを作成する方法について説明します。

第 2 章 では、チュートリアル用アプリケーションの機能と構造について説明します。

第 3 章では、チュートリアル用アプリケーションの EJB 層を作成する手順と、それぞれの Bean をテストする、IDE のテストアプリケーション機能の使用方法を説明します。

第 4 章では、EJB 層からチュートリアルの Web サービスを作成するための IDE の使用方法と、Web サービスのテスト方法を説明します。

第 5 章では、提供されている Swing クライアントが、第 4 章で Web サービスモジュールから生成された出力にどのようにアクセスするか、またチュートリアル用アプリケーションがどのように実行されるかを説明します。

付録 A は、このチュートリアル用アプリケーションのソースファイルの全内容のまとめです。

付録 B は、このチュートリアル用アプリケーションのデータベーススクリプトの全内容のまとめです。

付録 Cでは、Oracle データベースを使用してアプリケーションを作成、実行できるようにチュートリアルを改造する方法を説明します。

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

チュートリアルを開始する前に

この章では、この J2EE アプリケーションのチュートリアルを開始する前に必要な作業について説明します。この章の内容は次のとおりです。

- 必要なソフトウェアの入手とインストール
- 3 ページの「ソフトウェアの起動」
- 9 ページの「データベース接続の設定」
- 12 ページの「チュートリアル用のデータベース表の説明」

注 – このマニュアルには、「DiningGuide アプリケーションファイル」の名前を参照している個所が出てきます。DiningGuide アプリケーションファイルには、チュートリアルアプリケーションの完成版と、そのアプリケーションの実行方法を説明した readme ファイル、および必要なデータベース表を作成するための SQL スクリプトファイルが含まれます。これらのファイルを 1 つの zip 形式のファイルにまとめたものが Sun ONE Studio 5 Developer Resources ポータル (`http://forte.sun.com/ffj/documentation/tutorialsandexamples.html`) にあり、ダウンロードすることができます。

必要なソフトウェアの入手とインストール

チュートリアルアプリケーションの作成および実行には、以下のソフトウェアが必要です。

- Sun ONE Studio 5, Standard Edition ソフトウェア
  - Sun ONE Studio 5, Standard Edition 統合開発環境 (IDE)
  - Sun ONE Application Server 7 ソフトウェア

Sun ONE Studio 5, Standard Edition のインストーラでは、両方の製品がインストールされます。ただし、Sun ONE Application Server 7 のサポートされているバージョンが見つかった場合は、インストールされません。たとえば、Solaris™ 9 Update 2 オペレーティング環境には、Sun ONE Application Server 7 が含まれます。

Sun ONE Studio 5, Standard Edition ソフトウェアは、次の場所から入手できます。

- Sun ONE Studio 5, Standard Edition の CD
- Sun ONE ポータル (`http://wwws.sun.com/software/sundev/jde/`)
- Sun ONE Developer Resources ポータル (`http://forte.sun.com/ffj/`)

- Java™ 2 Software Development Kit (J2SE™ SDK)、バージョン 1.4.1_02 以降

  システムに J2SE SDK がなかった場合、Sun ONE Studio 5, Standard Edition のインストーラは実行されません。システムに J2SE SDK がある場合は、インストーラが起動し、使用している J2SE SDK のバージョンが、プラットフォーム用の IDE および Sun ONE Application Server 7 に必要なバージョンかどうかが確認されます。必要なバージョンではなかった場合は、正しいバージョンをインストールする必要があることを示すメッセージが表示され、インストーラが終了します。J2SE SDK は、IDE と同じ場所から入手できます。

- PointBase ネットワークサーバーデータベースソフトウェア

  このチュートリアルでは、PointBase データベースを使用します。PointBase は、Sun ONE Studio 5, Standard Edition ソフトウェアと同時に、Sun ONE Application Server 7 ソフトウェアを含むサブディレクトリにインストールされます。IDE とアプリケーションサーバーを別々にインストールした場合は、アプリケーションサーバーに PointBase ソフトウェアが含まれない場合があります。PointBase ソフトウェアが含まれない場合は、PointBase ソフトウェアをダウンロードし、手動でインストールする必要があります。この方法については、『Sun ONE Application Server 7 入門ガイド』を参照してください。また、付録 C では、Oracle データベースでアプリケーションを作成する方法を説明しています。

- チュートリアル用データベース表

  チュートリアル用データベース表は、Sun ONE Studio 5 インストーラによって PointBase データベースのユーザーディレクトリに自動的に作成されます。これらの表については、12 ページの「チュートリアル用のデータベース表の説明」で説明しています。Oracle データベースにチュートリアル用データベース表をインストールする方法については、付録 Cを参照してください。

- Web ブラウザ

  チュートリアルアプリケーションのページを表示するには Web ブラウザが必要です。Web ブラウザには、Netscape Communicator™ または Microsoft Internet Explorer を使用できます。

全般的なシステム要件については、リリースノートまたは Sun ONE Studio 5 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/`) をご覧ください。

ソフトウェアの起動

この節では、ソフトウェアのインストール後に Sun ONE Studio 5 IDE と Sun ONE Application Server 7 を起動する方法を説明します。

IDE の起動

Sun ONE Studio 5 IDE は、複数の方法で起動できます。ここでは、1 つの方法だけを示します。ほかの方法については、『Sun ONE Studio 5, Standard Edition インストールガイド』を参照してください。

IDE を起動するには、次のようにします。

- **Sun ONE Studio 5 IDE を起動するには、プログラムの実行可能ファイルを実行します。**
  - Microsoft Windows では、「スタート」->「プログラム」->「Sun Microsystems」->「Sun ONE Studio 5 SE」->「Sun ONE Studio 5 SE」を選択します。
  - Solaris、UNIX、および Linux の各環境では、次のように端末ウィンドウで `runide.sh` スクリプトを実行します。

```
$ sh s1studio-install-directory/bin/runide.sh
```

ここで *s1studio-install-directory* 変数は、IDE のホームディレクトリです。デフォルトでは、`/opt/studio5_se` (UNIX のスーパーユーザー) です。

アプリケーションサーバーの起動

ここに示す手順を実行する前に、アプリケーションサーバーのドメインへの書き込み権が必要です。デフォルトのドメインはインストール時に作成され、アクセスにはスーパーユーザーの権限が必要です。スーパーユーザーの権限とは、Microsoft Windows システムでは administrator の権限、Solaris または Linux の環境では root 権限です。使用しているユーザー ID にスーパーユーザーの権限がある場合は、以下に示す手順で、デフォルト設定を使用してアプリケーションサーバーを起動できます。スーパーユーザーの権限を持たない標準ユーザーは、5 ページの「管理サーバーの起動 (標準ユーザー)」に示す手順に従う必要があります。

管理サーバーの起動 (スーパーユーザーの場合)

以前に管理サーバーを起動した場合は、実行中かどうかを確認します。詳細は、9 ページの「Sun ONE Application Server 7 がデフォルトサーバーであることの確認」を参照してください。管理サーバーを初めて起動する場合は、ここから始めます。

管理サーバーを起動するには、次のようにします。

1. **IDE で、エクスプローラの「実行時」タブを選択します。**

   エクスプローラの「実行時」タブに「サーバーレジストリ」ノードが表示されます。このノードの下には、インストールされているすべての Web サーバーとアプリケーションサーバーのサブノードがあります。また、デフォルトのサーバーを示すノードがあります。

2. **「サーバーレジストリ」ノードを選択します。**

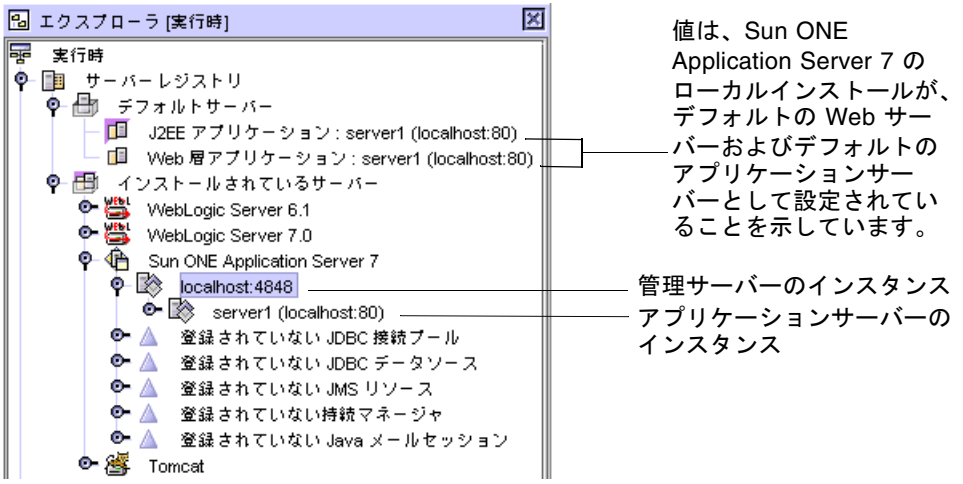
   管理サーバーを起動するかどうかを確認するウィンドウが開きます。この管理サーバーは、デフォルトドメインの管理サーバーを指し、権限を持つユーザーだけが実行できます。

3. **「了解」をクリックしてデフォルトの管理サーバーを起動します。**

   IDE によってデフォルトの管理サーバーが起動し、Sun ONE Application Server 7 が IDE のデフォルトのアプリケーションサーバーに設定されます。また、サーバーインスタンスの server1 が作成されます。

4. **「サーバーレジストリ」ノード、「インストールされているサーバー」ノード、「Sun ONE Application Server 7」ノードの順に展開します。**

   エクスプローラの「サーバーレジストリ」ノードは次のようになっています。



8 ページの「アプリケーションサーバーのインスタンスの起動」に示す手順に従って、サーバーインスタンスを起動します。

管理サーバーの起動 (標準ユーザー)

ユーザー ID にスーパーユーザーの権限がない場合は、ここに示す手順を実行する前に、スーパーユーザーにドメインを作成してもらう必要があります。作成方法は、『Sun ONE Studio 5, Standard Edition インストールガイド』で説明しています。

以前に IDE を起動し、管理サーバーを作成して起動した場合は、9 ページの「Sun ONE Application Server 7 がデフォルトサーバーであることの確認」の手順に従って、管理サーバーが実行中かどうかを確認します。管理サーバーを初めて起動する場合は、ここから始めます。

この手順を実行する前に、ドメインに関するプロパティの値をいくつか確認する必要があります。これらの値は、管理者に確認します。プロパティの値を確認したら、次の表に記入してください。

表 1-1　管理サーバーのプロパティの値

管理サーバーのプロパティ	値
管理サーバーホスト	
管理サーバーポート	
ユーザー名	
ユーザーパスワード	
ドメイン	

管理サーバーを起動するには、次のようにします。

1. **IDE で、エクスプローラの「実行時」タブを選択します。**

   エクスプローラの「実行時」タブに「サーバーレジストリ」ノードが表示されます。このノードの下には、インストールされているすべての Web サーバーとアプリケーションサーバーのサブノードがあります。また、デフォルトのサーバーを示すノードがあります。

2. **「サーバーレジストリ」ノードを選択します。**

   管理サーバーを起動するかどうかを確認するウィンドウが開きます。この管理サーバーは、デフォルトドメインの管理サーバーを指し、権限を持つユーザーだけが実行できます。

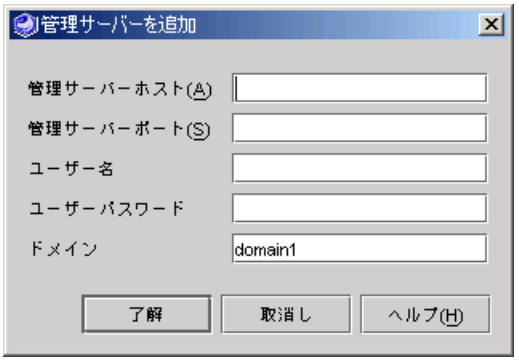
   「了解」をクリックすると、使用できない管理サーバーが作成され、起動されてしまいます。「取消し」をクリックします。

3. **管理サーバーを IDE に追加します。**

   a. **「サーバーレジストリ」ノード、「インストールされているサーバー」ノードを展開します。**

b. 「Sun ONE Application Server 7」ノードを右クリックして、「管理サーバーを追加」を選択します。

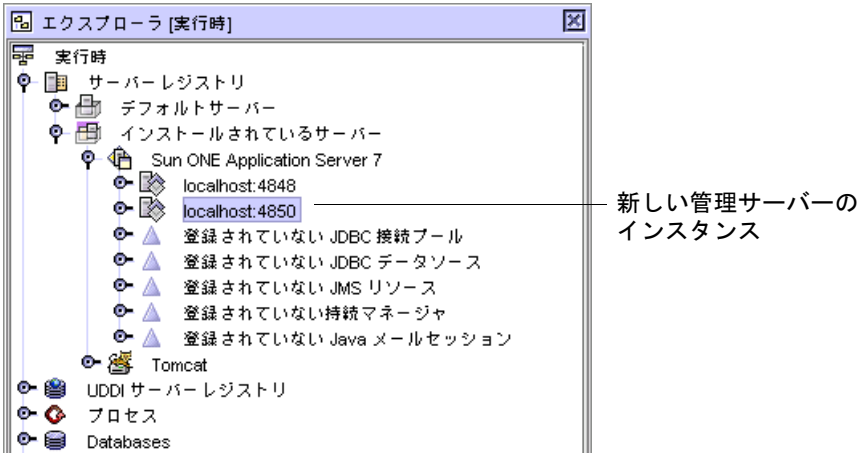
「管理サーバーを追加」ダイアログが表示されます。



c. 表 1-1 の値を入力し、「了解」をクリックします。

ここで、「管理サーバーに接続できません。管理サーバーがローカルの場合、サーバーを起動します。」というメッセージが表示された場合、「了解」をクリックして、エラーウィンドウを閉じます。管理サーバープロセスの起動状況を示す進捗ウィンドウが表示されます。

エクスプローラに、新しい管理サーバーノードが作成されます。次の画面イメージでは、新しい管理サーバーのホストは localhost で、ポート番号は 4850 です。



4. アプリケーションサーバーのインスタンスを作成します。

a. 新しい管理サーバーのノードを右クリックして、「サーバーインスタンスを作成」を選択します。

「サーバーインスタンスの値を入力」ダイアログが表示されます。

b. 名前と使用可能なポート番号を入力します。

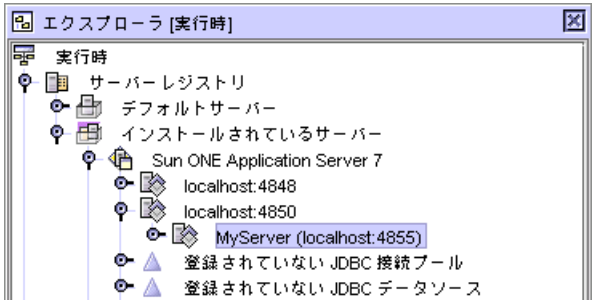
たとえば、「MyServer」および「4855」と入力します。

注 – Solaris システムと Linux システムでは、1024 未満のポート番号は予約されています。1024 以上のポート番号を使用してください。すべてのシステムで、デフォルトのアプリケーションサーバーで使用されるポート番号 80 は使用しないでください。

---

c. 「了解」をクリックします。

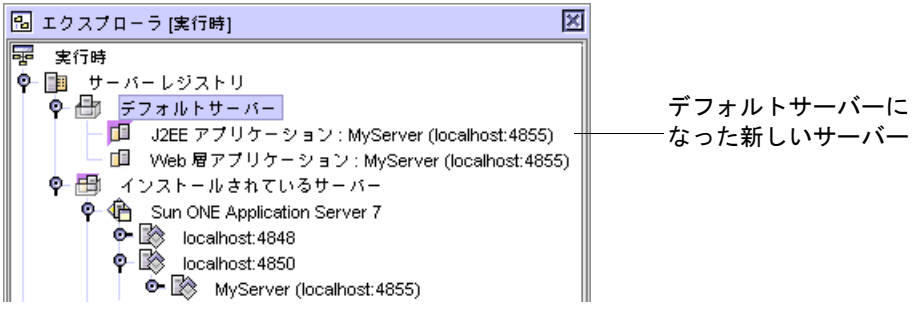
管理サーバーが起動し、出力ウィンドウとステータスバーにメッセージが表示されます。IDE に新しいサーバーインスタンスが作成されます。



5. 新しいサーバーインスタンスを右クリックし、「デフォルトとして設定」を選択して、デフォルトのアプリケーションサーバーと Web サーバーを設定します。

6. 「デフォルトサーバー」ノードを展開して、この処理が正常に実行されたことを確認します。

J2EE アプリケーションと Web 層アプリケーションのデフォルトサーバーに、新しいサーバーがデフォルトとして表示されます。



次の項に示す手順に従って、サーバーのインスタンスを起動します。

アプリケーションサーバーのインスタンスの起動

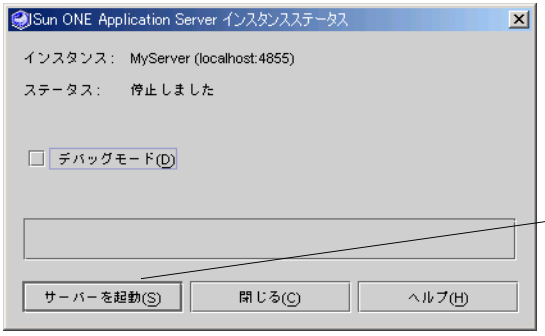
開発中にアプリケーションのテスト配備および配備を行ったとき、管理サーバーが実行中の場合は、IDE によってアプリケーションサーバーのインスタンスが自動的に起動されます。ここでは、この後に示す操作を行うために、アプリケーションサーバーのインスタンスを手動で起動します。

サーバーのインスタンスを起動するには、次のようにします (すべてのユーザー)。

1. **アプリケーションサーバーのノードを右クリックして、「ステータス」を選択します。**

注 – このノードが表示されない場合は、管理サーバーのインスタンスのノードを選択し、「再表示」を選択します。

次に示す「Sun ONE Application Server インスタンスステータス」ダイアログが表示されます (インスタンス名は異なる場合があります)。



2. **「サーバーを起動」ボタンをクリックします。**

   ダイアログに「サーバーを停止」ボタンがある場合は、サーバーはすでに実行中です。

   Microsoft Windows システムでは、進捗メッセージを示すコマンドウィンドウが表示されます。

   サーバーが起動すると、「Sun ONE Application Server インスタンスステータス」ダイアログに「ステータス: 実行中」と表示されます。

3. **「閉じる」をクリックします。**

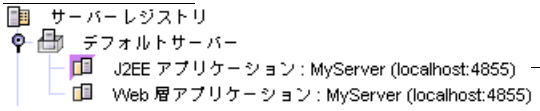
   9 ページの「データベース接続の設定」に進みます。

Sun ONE Application Server 7 がデフォルトサーバーであることの確認

Sun ONE Application Server 7 を以前に起動した場合は、次の手順で、このサーバーがまだデフォルトのサーバーであることを確認します。

1. IDE で、エクスプローラの「実行時」タブを選択します。
2. 「サーバーレジストリ」ノード、「デフォルトサーバー」サブノードを展開します。

   「J2EE アプリケーション」ノードのラベルが *server-instance* (*server-hostname*:*server-port-number*) の場合は、Sun ONE Application Server 7 がデフォルトのアプリケーションサーバーになっています。次の節に進みます。それ以外の場合は、以下の手順に進みます。



値は、Sun ONE Application Server 7 のローカルインストールが、デフォルトの Web サーバーとして設定されているところを示しています。

3. 「インストールされているサーバー」ノードの下で Web サーバーのインスタンスを右クリックし、「デフォルトとして設定」を選択します。

   サーバーが、J2EE と Web 層アプリケーションのデフォルトのサーバーとして設定されます。

データベース接続の設定

エンタープライズアプリケーションは、データベースとの対話に JDBC™ (Java Database Connectivity) API を使用します。エンタープライズアプリケーションを配備して Sun ONE Application Server 7 で実行するには、アプリケーションサーバー環境で JDBC 関連の作業を行う必要があります。この作業は、次のとおりです。

- データベースの JDBC ドライバを有効にする
- 接続プールの作成

  エンタープライズアプリケーションでは、データベース接続のプールを作成することによってシステム内のビジネスオブジェクトがデータベースアクセスを共有できるようにする必要があります。
- JDBC データソースの作成

  JDBC データソース (JDBC リソースともいう) では、 `getConnection()` メソッドを使用してデータベースに接続することができます。

- JDBC 持続マネージャの作成

  持続マネージャは、コンテナにインストールされているエンティティ Bean の持続性を担当するコンポーネントです。

ほとんどの環境では、Sun ONE Studio 5 IDE によって、チュートリアルが必要とする値が提供され、こうした作業のすべてが自動的に行われます。これらの作業が行われる方法は、ユーザー ID の特権内容と、使用しているプラットフォームによって異なります。詳細は、10 ページの「JDBC リソースの設定 (Microsoft Windows のスーパーユーザー)」 および 11 ページの「JDBC リソースの設定 (administrator 以外のユーザー)」を参照してください。

ただし、次に説明するように、JDBC ドライバの自動的な有効に対する例外事項は、ユーザー ID に依存しません。

JDBC ドライバを有効にする

PointBase 用の JDBC ドライバは、Sun ONE Studio 5 IDE と Sun ONE Application Server 7 を一緒にインストールする標準のインストールで自動的にインストールされます。

注 – Sun ONE Studio 5 IDE と Sun ONE Application Server 7 を別々にインストールした場合は、手動で行わなければならない作業があります。これらの作業については、『Sun ONE Studio 5, Standard Edition インストールガイド』で説明しています。

JDBC リソースの設定 (Microsoft Windows のスーパーユーザー)

Sun ONE Studio 5のインストーラは、administrator 特権を持つ Microsoft Windows ユーザー用に自動的に 3 つの接続リソースを作成します。

注 – Sun ONE Studio 5 IDE とSun ONE Application Server 7 を別々にインストールした場合は、11 ページの「JDBC リソースの設定 (administrator 以外のユーザー)」で説明する手順を使用して JDBC リソースの設定をする必要があります。

IDE でリソースを表示する

1. **「エクスプローラ」の「実行時」区画で「サーバーレジストリ」、「インストールされているサーバー」の順にノードを展開します。**

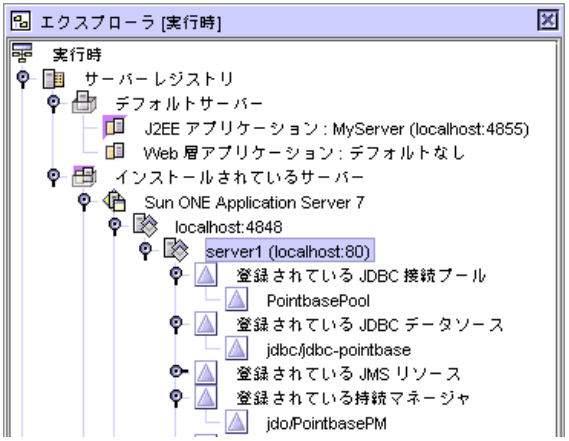
2. デフォルト管理サーバー (localhost:4848)、デフォルトアプリケーションサーバー (server1(localhost:80)) の順にノードを展開します。

3. JDBC 接続プール、JDBC データソース、および持続マネージャの登録リソースを展開します。

   以下に次のノードが表示されます。

   - JDBC 接続プール: PointbasePool
   - JDBC データソース: jdbc/jdbc-pointbase
   - 持続マネージャ jdo/PointbasePM

   エクスプローラの表示は以下のようになります。



JDBC リソースの設定 (administrator 以外のユーザー)

注 – この作業に進む前に、管理サーバーとアプリケーションサーバーの両方が稼働していることを確認してください (3 ページの「ソフトウェアの起動」を参照)。

このチュートリアル用の JDBC 接続リソースを作成する

1. 「エクスプローラ」の「実行時」区画で「サーバーレジストリ」、「インストールされているサーバー」の順にノードを展開します。

2. アプリケーションサーバーインスタンスを見つけます。

   MyServer (localhost:4855) というような *app-server-name* (*app-server-host*:*app-server-port*) の形式のラベルが付いています。

3. アプリケーションサーバーインスタンスのノードを右リックして、「PointBase JDBC リソースを事前構成」を選択します。

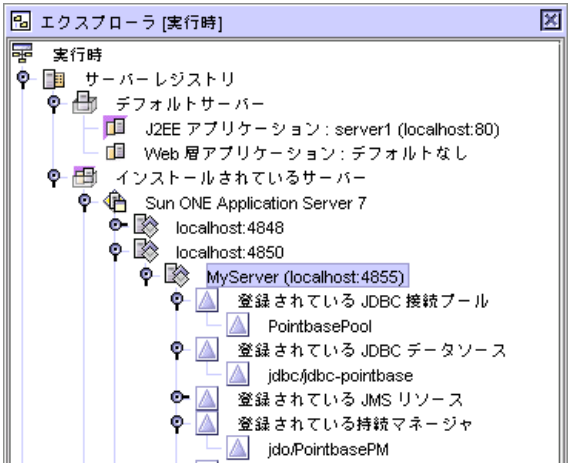
   砂時計などのタイマーアイコンが現れます。処理が完了すると、カーソルが通常のアイコンに戻ります。

4. JDBC 接続プール、JDBC データソース、および持続マネージャの登録リソースを展開します。

   以下に次のノードが表示されます。

   - JDBC 接続プール: PointbasePool
   - JDBC データソース: jdbc/jdbc-pointbase
   - 持続マネージャ: jdo/PointbasePM

   エクスプローラの表示は以下のようになります。



チュートリアル用のデータベース表に進みます。

チュートリアル用のデータベース表の説明

DiningGuide チュートリアルでは、Restaurant と CustomerReview という 2 つのデータベース表を使用します。これらの表は、Sun ONE Studio 5 ソフトウェアと Sun ONE Application Server 7 を一緒にインストールする標準のインストールで PointBase のデフォルトデータベースに自動的に作成されます。

これに対し、Sun ONE Studio 5 ソフトウェアと Sun ONE Application Server 7 を一緒にインストールしなかった場合は、10 ページの「JDBC ドライバを有効にする」で説明している手順に従う必要があります。この作業では、PointBase 用の JDBC ドライバを有効にしたうえで、デフォルトのサンプルデータベースをユーザーディレクトリにコピーして、ユーザーが表を使用できるようにします。

このチュートリアルでは、表 1-2に示すデータベーススキーマを使用します。これらのスキーマは、Sun ONE Studio 5 インストーラによって PointBase データベースに作成されます。

表 1-2 DiningGuide のデータベース表

表名	列	主キー	その他
Restaurant	restaurantName	○	
	cuisine		
	neighborhood		
	address		
	phone		
	description		
	rating		
CustomerReview	restaurantName	○	CustomerName との複合主キー (Restaurant 表の restaurantName を参照している)
	customerName	○	
	review		

Restaurant 表には、表 1-3 に示すレコードが含まれます。

表 1-3 Restaurant 表のレコード

restaurant -Name	cuisine	neighborhood	address	phone	description	rating
French Lemon	Mediterranean	Rockridge	1200 College Avenue	510 888 8888	Very nice spot.	5
Bay Fox	Mediterranean	Piedmont	1200 Piedmont Avenue	510 888 8888	Excellent.	5

CustomerReview 表には、表 1-4 に示すレコードが含まれます。

表 1-4　CustomerReview 表のレコード

restaurantName	customerName	comment
French Lemon	Fred	Nice flowers.
French Lemon	Ralph	Excellent Service

これで、チュートリアルアプリケーションを開始する準備ができました。構築するアプリケーションの概要は、第 2 章を参照してください。すぐに構築を開始するには、第 3 章を参照してください。

第2章

チュートリアルアプリケーションの概要

ここでは、チュートリアルのサンプルアプリケーションを通して、Sun ONE Studio 5, Standard Edition の機能を使用した簡単な J2EE アプリケーションを作成する方法を学習します。

この章では作成するアプリケーションを紹介し、アプリケーションの要件、およびその要件を満たすアーキテクチャについて説明します。また、最後の節では Sun ONE Studio 5, Standard Edition の機能 (EJB ビルダー、テストアプリケーション機能、新規 Web サービスウィザードなど) の使用方法について説明します。

この章の構成は次のとおりです。

チュートリアルアプリケーションの機能

チュートリアルアプリケーションである「DiningGuide」は、利用できるレストランとその特徴をまとめたリストを表示する、簡単なレストランガイドアプリケーションです。このアプリケーションの利用者は、特定のレストランに関する利用者の批評を表示したり、レストランのレコードに自分のコメントを追加したりできます。表示されるレストランの特徴としては、店名、料理の種類、近隣情報、所在地、電話番号、簡単な説明、評価 (1 ～ 5) などがあります。

ユーザーは、このアプリケーションのインタフェースと以下のように対話します。

- レストランの一覧を表示する
- 特定のレストランに関する利用者のコメントのリスト表示を要求する
- そのレストランに対するコメントを入力してリストに追加する

アプリケーションとユーザーの対話シナリオ

DiningGuide アプリケーションとそのユーザーの対話は、ユーザーがデータベース内のすべてのレストランレコードを一覧表示するクライアントを実行することで始まり、 ユーザーがアプリケーションのクライアントを終了したときに終了します。このチュートリアルには、アプリケーションの機能との対話方法を具体的に示すための単純な Swing クライアントが用意されています。しかし、Web クライアントや他のアプリケーションなど、これ以外の種類のクライアントも DiningGuide アプリケーションのビジネスメソッドにアクセスすることができます。

次のシナリオは、アプリケーションでどのような対話が行われ、アプリケーションにどのような条件が求められるのかを説明します。

1. ユーザーがアプリケーションの `RestaurantTable` クラスを実行します。

   アプリケーションは「DiningGuide Restaurant Listing」ウィンドウを表示します。このウィンドウには、すべてのレストランの店名と料理の種類、所在地、電話番号、短いコメント、および 1 から 5 の評価が表示されます。またこのページには、「View Customer Comments」というボタンがあります。

2. ユーザーがリストからレストランのレコードを選択し、「View Customer Comments」ボタンをクリックします。

   アプリケーションは、選択されたレストランに関する利用者のコメントがすべて入った「All Customer Reviews By Restaurant Name」ウィンドウを表示します。

3. 「利用者コメント」ウィンドウで、ユーザーが「Customer Name」および「Review」フィールドにテキストを入力して、「Submit Customer Review」ボタンをクリックします。

   アプリケーションは入力されたユーザー名とコメントを CustomerReview データベース表に追加し、新規追加されたレコードの入った「All Customer Reviews By Restaurant Name」ウィンドウを再表示します。

4. ユーザーが「Restaurant Listing」ウィンドウに戻り、別のレストランを選択して、「View Customer Comments」ボタンをクリックします。

   アプリケーションは、選択されたレストランに対するコメントがすべて入った新しい「All Customer Reviews By Restaurant Name」ウィンドウを表示します。

アプリケーションの機能仕様

上記のような対話シナリオをサポートする DiningGuide アプリケーションのユーザーインタフェースの主な機能としては、以下が挙げられます。

- すべてのレストランデータのマスタービュー (リスト表示)
- 特定のレストランに関するすべての利用者のコメントを読み込むためのボタン (マスターレストランリストのウィンドウに表示)

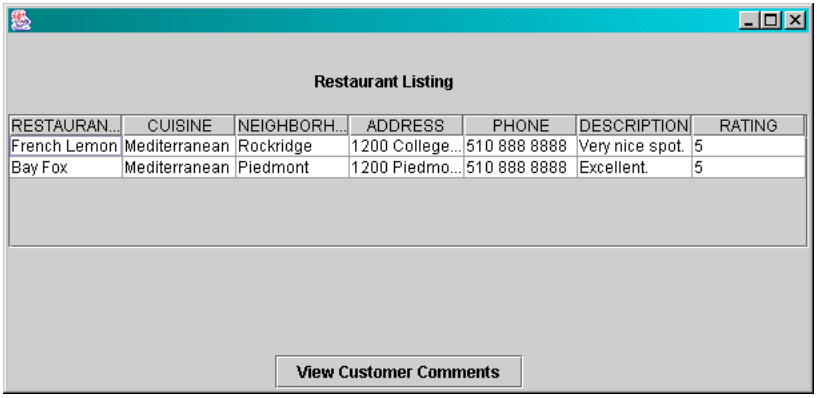
- 特定のレストランに対するすべての利用者のコメントデータのマスタービュー
- 新しいコメントを追加するためのボタン (「利用者コメントリスト」ウィンドウに表示)
- 現在のレストランの新規ユーザー名とコメントを入力するためのテキスト入力フィールド (ユーザー批評リストウィンドウに表示)
- 入力されたコメントデータをデータベースに送信するためのボタン (ユーザー批評リストウィンドウに表示)

ユーザーから見たチュートリアルアプリケーション

15 ページの「チュートリアルアプリケーションの機能」で説明したシナリオと機能仕様が、ユーザーから見た場合にどのように実現されるかを示します。

1. **Sun ONE Studio 5 エクスプローラで「RestaurantTable」ノードを右クリックして「実行」を選択します。**

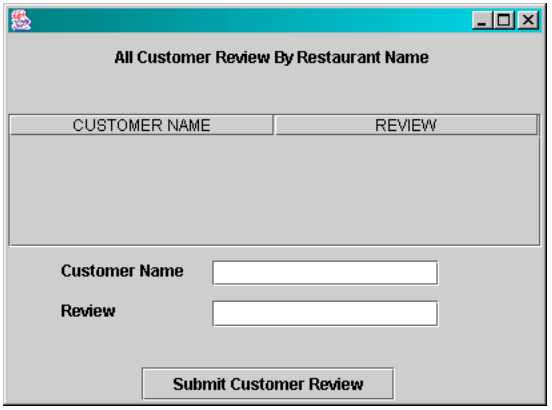
   IDE が実行モードに切り替わります。実行ウィンドウに Restaurant ノードが現れ、以下のような「RestaurantTable」ウィンドウが表示されます。



   このウィンドウに表示されているのは、12 ページの「チュートリアル用のデータベース表の説明」で説明している Restaurant 表のデータです。

2. **特定のレストランに関する利用者のコメントを表示するには、店名を選択して、「View Customer Comments」ボタンをクリックします。**

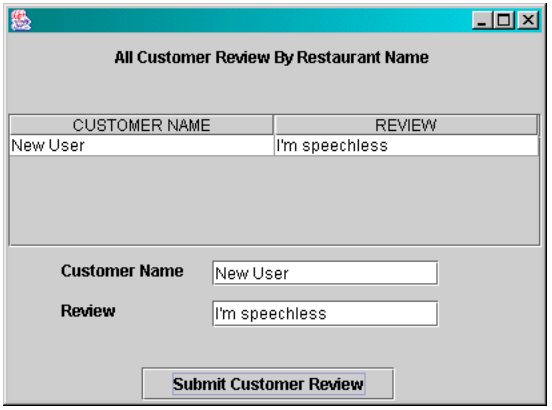
   たとえば Bay Fox レストランを選択してください。「CustomerReviewTable」ウィンドウが表示されます。



この場合は、データベースに何もコメントがないため、レコードは表示されません。表 1-4 を参照してください。

3. **コメントを追加するには、利用者名とコメントを入力して、「Submit Customer Review」ボタンをクリックします。**

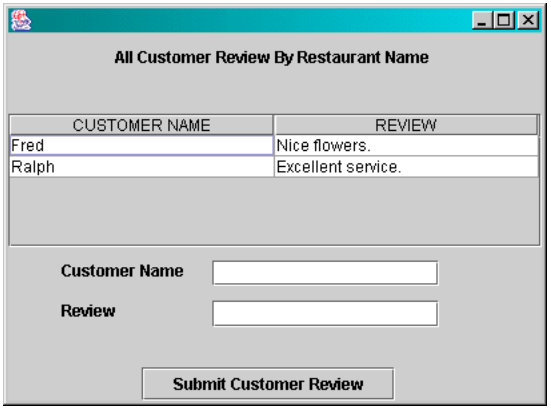
   たとえば利用者名として `New User`、コメントとして `I'm speechless!` と入力してください。「利用者コメント」ウィンドウが再表示されます。



他のレストランのコメントを表示してみます。

4. **「Restaurant List」ウィンドウで「French Lemon」を選択して、「View Customer Comments」ボタンをクリックします。**

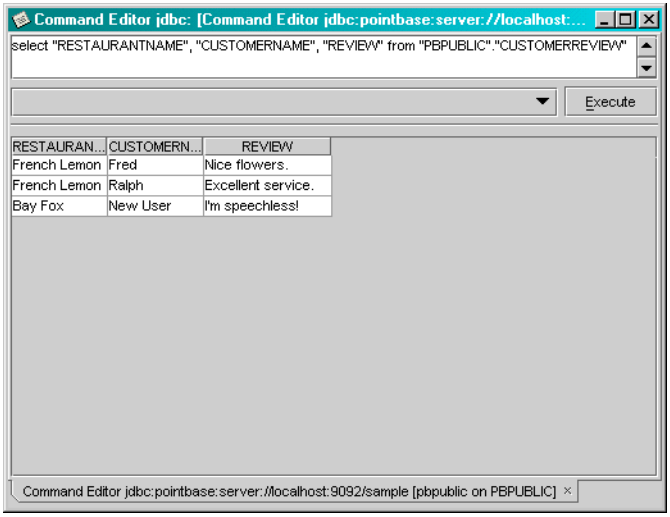
   新しい「利用者コメントリスト」ウィンドウに French Lemon レストランに対するコメントが表示されます。



利用者コメントレコードが 2 つ表示されています。表 1-4 で確認してください。

5. 続けて利用者コメントレコードを追加して、表示してみます。
6. 練習を終えたら、アプリケーションのウィンドウのいずれかを閉じることによってアプリケーションを終了します。
7. 新しい利用者のコメントレコードがデータベースに書き込まれたことを確認するには、IDE でエクスプローラの「実行時」タブを選択します。
8. 「データベース」、「PointBase 接続」、「表」の順にノードを展開します。
9. 「CUSTOMERREVIEW」表を右クリックして、「データを表示」を選択します。

   コマンドエディタウィンドウに、たとえば手順 3 で入力したような新しい CustomerReview レコードがすべて表示されます。



チュートリアルアプリケーションの構造

チュートリアルアプリケーションの中枢は、エンティティタイプの 2 つのエンタープライズ Bean と 2 つの詳細クラス、および 1 つのセッション Bean からなる EJB 層です。2 つのエンティティ Bean は 2 つの DiningGuide データベース表 (Restaurant 表および CustomerReview 表) を示します。2 つの詳細クラスはエンティティ Bean フィールドを反映するクラスで、それぞれのフィールドの取得メソッドおよび設定メソッドが含まれます。詳細クラスを使用する目的は、データベースデータを読み込む際のエンティティ Bean に対するメソッド呼び出し回数を減らすことにあります。セッション Bean はクライアント (Web サービス経由) とエンティティ Bean 間の対話を管理します。

図 2-1 は DiningGuide アプリケーションのアーキテクチャ図です。

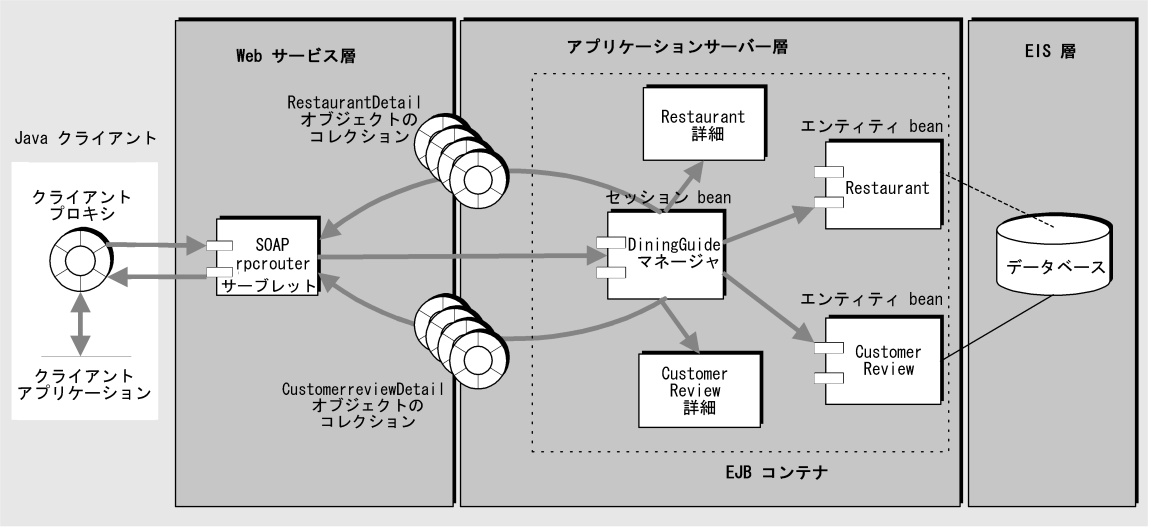


図 2-1 DiningGuide アプリケーションのアーキテクチャ

図 2-1 のクライアントには クライアントプロキシが含まれています。このプロキシは SOAP 実行環境システムを使用して、Web サーバー上の SOAP 実行システムと通信します。要求は SOAP メッセージとして渡されます。Web サービスは、この SOAP メッセージを EJB 層のセッション Bean のメソッドの呼び出しに変換します。その応答はセッション Bean が Web サービスに返し、SOAP メッセージに戻されてクライアントプロキシに渡され、最終的には表示データまたはアクションに変換されます。

SOAP 要求は XML ラッパーであり、Web サービスのメソッド呼び出しとシリアル化された形式の入力データを含みます。

アプリケーションの構成要素

ここでは、図 2-1 に示すアプリケーションの構成要素をまとめています。

- アプリケーションサービス層 (EJB 層)

  このチュートリアルでは、最初に EJB 層を作成してテストします。EJB 層は、次の要素で構成されています。

  - 2 つのエンティティエンタープライズ Bean - コンテナ管理の持続性 (Container-Managed Persistence: CMP) を使用し、アプリケーションの 2 つのデータベース表を表します。
  - 2 つの詳細クラス - 返されたデータベースレコードを保持します。
  - ステートレスセッションエンタープライズ Bean - クライアントの要求を管理し、クライアントに返されるオブジェクトをフォーマットします。

- Web サービス層
  - Web モジュール - セッション Bean のメソッド実行用のサーブレットと JSP ページが含まれます。

    このモジュールは、セッション Bean に対するテストアプリケーションを作成したときに自動的に作成されます。
  - Web サービス論理ノード - Web サービス全体を表し、Web サービスの変更や構成を可能にします。
  - クライアントプロキシ - Web サービスを配備したときに生成されます。
  - WSDL (Web Services Descriptive Language) ファイル - クライアント用の Web サービスの記述です。
- クライアント

  クライアントコンポーネントは、アプリケーションのページを表示する Swing クライアントです。第 5 章 では、第 4 章で Web サービスに作成するクライアントプロキシをインスタンス化するコードを、提供されるクライアントページからコピーします。

EJB 層の詳細

DiningGuide アプリケーションの EJB 層には、エンティティタイプの 2 つの エンタープライズ Bean、2 つの詳細クラス、およびその他のコンポーネント (セッション Bean) が含まれており、このうちのセッション Bean を使用して、クライアントとエンタープライズ Bean のやりとりを管理します。

- Restaurant CMP EJB コンポーネント

  Restaurant Bean は、コンテナ管理の持続性 (CMP) を使用して Restaurant データベース表のデータを表すエンティティ Bean です。
- Customerreview CMP EJB コンポーネント

  CMP タイプのエンティティ Bean である Customerreview エンティティ Bean も、CustomerReview データベース表のデータを表します。
- RestaurantDetail クラス

  このコンポーネントには、Restaurant エンティティ Bean と同じフィールド、および各フィールドに対する getter/setter メソッド (エンティティ Bean のリモート参照からこのデータを取得するためのもの) が定義されています。このコンストラクタは、レストランデータを表すオブジェクトをインスタンス化します。また、このオブジェクトはクライアントで JSP ページ、HTML ページ、または Swing コンポーネントにフォーマットされ、表示できるようになります。
- CustomerreviewDetail クラス

  このコンポーネントは、RestaurantDetail クラスが Restaurant エンティティ Bean に対して機能するのと同様に、Customerreview エンティティ Bean に対して機能します。

- DiningGuideManager セッション EJB コンポーネント

  このコンポーネントはステートレスセッション Bean で、クライアントとエンティティ Bean のやりとりを管理します。

チュートリアルアプリケーションの作成に必要な作業

チュートリアルアプリケーションを作成する作業は、3 つの章に分けて説明します。まず、第 3 章で EJB 層を作成し、作業を進めながら IDE のテストアプリケーション機能を使用して各エンタープライズ Bean をテストします。次に、トラフィックを管理するセッション Bean を作成します。この作業は、Web サービスとクライアントを手動で作成する際の一般的なモデルです。

第 4 章では、Web サービスを作成し、どの EJB 層のビジネスメソッドを参照するか指定します。また、Web サービスを配備して、そのときに生成されるクライアントプロキシをテストします。

第 5 章では、提供されている 2 つの Swing クラスをアプリケーションにインストールおよび実行し、アプリケーションをテストします。

EJB コンポーネントの作成

第 3 章では、Sun ONE Studio 5 の機能を使用して以下のことを行う方法について学習します。

- EJB ビルダーを使用してエンティティ Bean とセッション Bean を簡単に作成する
- データベーススキーマから、取得メソッドや設定メソッドを使用してクラスを生成する
- テストアプリケーション機能を使用し、エンタープライズ Bean から J2EE テストアプリケーションをアセンブルする
- J2EE アプリケーションに EJB 参照を追加する
- J2EE リファレンス実装アプリケーションサーバーにテストアプリケーションを配備する
- テストアプリケーション機能によって作成されたテストクライアントページから、エンタープライズ Bean メソッドを実行する

EJB ビルダーの使用方法

EJB ビルダーウィザードは、エンタープライズ Bean を構成するさまざまなコンポーネントを自動的に作成します。ここで自動生成されるセッション Bean は、ステートレス Bean でもステートフル Bean でもよく、また、エンティティ Bean はコンテナ管理の持続性 (CMP) をもつ Bean でも Bean 管理の持続性 (BMP) の特徴を持つ Bean でもかまいません。第 3 章 では、既存のデータベース表に基づいて 2 つの CMP エンティティ Bean を作成します。

エンティティ Bean を作成しながら、作成プロセス中にデータベースにどのように接続するか、また、フィールドが表の列を表すエンティティ Bean をどのように生成するかについて学習します。Bean の基本的な部分は、Java コードを使用して Sun ONE Studio 5 のエクスプローラに生成されます。この Java コードは、ホームインタフェース、リモートインタフェース、Bean クラス、(適切な場合は) 主キークラスに対し、すでに生成されているものです。また、Bean を全体的に表す論理的な Bean ノードを使用して Bean プロパティを編集、変更する方法を学習します。EJB ビルダーの GUI 機能を使用し、作成メソッド、検索メソッド、ビジネスメソッドを追加する方法についても学習します。

詳細クラスの作成

詳細クラスには、エンティティ Bean と同じフィールドが存在する必要があります。2 つのクラスを作成し、それらクラスに適切な Bean プロパティを追加します。各プロパティの追加では、そのプロパティに対する補助メソッドを自動的に生成するオプションを有効にします。これは、アプリケーションが要求する取得メソッドと設定メソッドを取得するためです。続いて、プロパティをインスタンス化する各クラスのコンストラクタをコーディングします。最後に、2 つのエンティティ Bean のそれぞれに、対応する詳細クラスのインスタンスを返すためのコードを追加します。

テストアプリケーション機能

Sun ONE Studio 5 には、テスト用のクライアントを作成せずに、エンタープライズ JavaBean コンポーネントをテストする機能が用意されています。この機能では、Sun ONE Application Server 7 をアプリケーションサーバーとして使用し、エンタープライズ Bean を、Web モジュールとクライアント JSP ページを含んだ J2EE アプリケーションの一部として配備します。これらの JSP ページは 1 枚の HTML ページにまとめられているため、Web ブラウザから Bean の 1 つのインスタンスを作成し、そのビジネスメソッドを実行することができます。

3 つの エンタープライズ Bean に対しては、それぞれ個別のテストアプリケーションを作成します。エンティティ Bean に対しては、テストアプリケーションは J2EE アプリケーションを生成します。この J2EE アプリケーションには Web モジュールが含まれており、クライアントが Web ブラウザから使用するために自動的に生成された JSP ページと、エンティティ Bean に対する EJB モジュールで構成されています。セッション Bean はそれらエンティティ Bean のメソッドを呼び出すため、その EJB

モジュールにもまた エンティティ Bean の EJB モジュールが含まれる必要があります。IDE でコマンドを使用し、セッション Bean の EJB モジュールに対してエンティティ Bean の参照を追加します。テストアプリケーションの作成中に作成された EJB モジュールは、後で Web サービスによって参照されます。

Web ブラウザでセッション Bean をテストすると、アプリケーションのすべてのビジネスメソッドを実行することができます。第 3 章 の最後には、独自の Web サービスとクライアントを手動で作成する場合に、テストクライアントアプリケーションを使用するためのガイドラインが記載されています。

チュートリアルアプリケーションの Web サービスの作成

第 4 章では、Sun ONE Studio 5 の機能を使用して以下のことを行う方法を学びます。

- 論理 Web サービスを作成する
- Web サービスで参照するセッションビジネスメソッドを指定する
- Web サービスを入れる J2EE アプリケーションを作成する
- Web サービスの実行時クラスとクライアントページを生成する
- Web サービスのクライアントプロキシを生成する

Web サービスの作成

Web サービスは、そのサービスに含まれるオブジェクトセット全体を表す論理的なエンティティで、Web サービスの変更と構成を容易に行えるようにします。Web サービスは、新規ウィザードを使用して名前とパッケージの場所を定義することによってエクスプローラに作成します。このウィザードの実行中に、Web サービスに参照させるビジネスメソッドの指定が求められます。

Web サービスのプロパティの 1 つとして JAX-RPC 実行環境の場所を示す URL を指定し、Web サービスの実行時クラス (Web サービスを実装する EJB コンポーネント) を生成します。

テストクライアントの作成

フロントエンドクライアントとバックエンド J2EE アプリケーションからなるテストクライアントを作成して、 セッション Bean の EJB モジュールと Web サービス への参照を追加します。この参照の追加によって、Web サービスの WAR および EJB JAR ファイルが利用可能になり、それらのプロパティをカスタマイズすることができます。カスタマイズするプロパティとしては、Web コンテキストプロパティがあります。これで DiningGuide の J2EE アプリケーションが完成し、配備準備完了です。

Web サービスの配備とテストクライアントの作成

Web サービスを含む J2EE アプリケーションを配備すると、IDE によってクライアントプロキシとサポートファイルが自動的に生成されます。サポートファイルとしては、参照される各メソッドの JSP ページ、 JSP エラーページ、開始ページなどです。

Web サービスのテスト

IDE のコマンドを使用して、DiningGuide アプリケーションを配備します。アプリケーションサーバーが起動し、1 つのページにすべての操作をまとめた、テストクライアントの開始ページを表示します。生成される JSP ページには、入力パラメータが必要なときの入力フィールドと、操作を実行するための「Invoke」ボタンが含まれます。これらの手段を利用して、Web サービスがどのようにしてセッション Bean の各メソッドを呼び出すのかを確認します。

他の開発者が Web サービスを利用できるようにする

このチュートリアルでは、UDDI レジストリに Web サービスを公開する方法は説明しませんが、テスト目的で他の開発者が Web サービスを利用できるようにする非公式な方法について説明します。この方法を使うと、WSDL ファイルを生成してサーバーに置くか、電子メールなどの他の方法で配布することによって、他の開発者が利用できるようにします。他の開発者はこのファイルからクライアントプロキシを生成し、Web サービスで利用できるメソッドを特定し、それに従ってクライアントを作成できます。また、配備した Web サービスの URL を教えることによって、他の開発者が Web サービスに対してクライアントをテストすることもできます。

Sun ONE Studio 5 IDE にはまた、テストに利用できるシングルユーザー用内部 UDDI レジストリも用意されています。StockApp のサンプルはこのデバイスを使用して Web サービスを公開する具体例で、Sun ONE Studio 5 Developer ポータルサイトの Examples and Tutorials ページから入手できます。Examples and Tutorials ページの URL は以下のとおりです。

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

UDDI レジストリへの Web サービスの公開についての詳細は、Sun ONE Studio 5 プログラミングシリーズの『Web サービスのプログラミング』をご覧ください。

提供クライアントのインストールと利用

このチュートリアルの付録 A には、DiningGuide アプリケーションの働きを具体的に理解するための、単純な Swing クライアントのコードが記載されています。このクライアントは、データベース表ごとに 1 つの Swing クラスで構成されます。2 つのクラスを作成して、デフォルトのコードを提供コードに置き換え、 主クラスを実行するだけです。

提供コードを調べることによって、クライアントがアプリケーションのメソッドにアクセスする方法が分かります。最初に、クライアントはクライアントプロキシをインスタンス化する必要があります。このインスタンス化によって、クライアントがクライアントプロキシのメソッドを利用できるようになります。それらのメソッド (図 2-1 を参照) は、アプリケーションの EJB 層のメソッドにアクセスする際に SOAP 実行環境が使用します。

最後に

このチュートリアルアプリケーションは、Sun ONE Studio 5, Standard Edition の主要機能を具体的に紹介するための実行アプリケーションとして、簡潔で比較的短期間 (1 日程度) で作成できるように設計されています。このため、次のような制約があります。

- エラー処理がない
- デバッグプロシージャがない
- Web サービスの公開の説明がない

すぐに完成できるように単純なアプリケーションとして設計されているとはいえ、このチュートリアルアプリケーション全体のインポートやそのソースファイルの表示、作成するメソッドへのメソッドコードのコピーが行えると便利です。DiningGuide アプリケーションは、以下の URL の Sun ONE Studio 5 Developer ポータルサイトの Examples and Tutorials ページから入手できます。

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

第3章

DiningGuide アプリケーションの EJB 層の作成

この章では、DiningGuide チュートリアルアプリケーションの EJB 層の作成方法を手順に従って説明します。その過程で EJB ビルダーを使用してエンティティ Bean とセッション Bean の両方を作成し、IDE のテスト機能を使用してそれら Bean をテストします。この章の内容は次のとおりです。

この章の作業を完了すると、1 つのテストアプリケーションとして配備した DiningGuide アプリケーションの EJB 層全体を実行できるようになります。

EJB 層を完成すると、独自の Web サービスやクライアントページを自由に作成できるようになります。また、そのまま第 4 章に進み、Sun ONE Studio 5 の Web サービス機能を使用してアプリケーションの Web サービスを作成する方法を学ぶこともできます。

EJB 層の概要

この章では、チュートリアルアプリケーションの中枢モジュールである EJB 層を作成します。各コンポーネントを作成するたびに、IDE のテストアプリケーション機能を使用してテストを行います。このテストアプリケーションでは、テスト用の Web サービスとクライアントを自動的に作成します。

作成する EJB 層には、次の Bean が定義されます。

- Restaurant エンティティ Bean
- Customerreview エンティティ Bean
- DiningGuideManager セッション Bean
- RestaurantDetail bean
- CustomerreviewDetail bean

J2EE アーキテクチャにおける EJB 層の役割についての詳細は、Sun ONE Studio 5 プログラミングシリーズの『Enterprise JavaBeans コンポーネントのプログラミング』を参照してください。このマニュアルでは、あらゆる Bean 要素を完全解説しているとともに、エンタープライズ Bean においてトランザクション、持続性、セキュリティがどのようにサポートされているかについても説明しています。

このチュートリアルの例と同様に、EJB 層とそこから生成される Web サービスを使用するアプリケーションがどのようなものであるかを学びたい場合は、Sun ONE Studio 5 Example ページ (`http://forte.sun.com/ffj/documentation/tutorialsandexamples.html`) にある PartSupplier というサンプルアプリケーションを参照してください。

エンティティ Bean

エンティティ Bean には、概念を定義する共有データセットに対して一貫したインタフェースが用意されています。このチュートリアルでは、2 つの概念 (**レストラン**、**利用者のコメント**) を使用します。ここで作成する Restaurant および Customerreview のエンティティ Bean は、第 1 章で作成したデータベース表を表します。

これらのエンティティ Bean には、コンテナ管理の持続性 (CMP) または Bean 管理の持続性 (BMP) のいずれかを定義することができます。BMP エンティティ Bean では、開発者が、Bean フィールドをデータベース表の列にマップするためのコードを記述する必要があります。CMP エンティティ Bean では、EJB の実行環境で持続性の操作を管理します。このチュートリアルでは、CMP エンティティ Bean を使用します。IDE の EJB ビルダーウィザードを使用し、データベースに接続してマップする列を指定します。ウィザードでは、データベースにマップされるエンティティ Bean を作成します。

EJB ビルダーでは、CMP エンティティ Bean のフレームワークを作成します。具体的には、エンティティ Bean を体系付けてカスタマイズを容易にするための論理ノード、必要なホームインタフェース、リモートインタフェース、Bean クラスを作成します。

このエンティティ Bean の作成、検索、ビジネスメソッドは手動で定義します。メソッドを定義すると、IDE によって適切な Bean コンポーネントにそのメソッドが自動的に伝達されます。たとえば 生成メソッドは Bean のホームインタフェースに、また、対応する ejbCreate メソッドは Bean のクラスに伝達されます。メソッドを編集すると、その変更もまた伝達されます。

検索メソッドでは、目的のオブジェクトを探すための適切なデータベース文を定義する必要があります。EJB 2.0 アーキテクチャには、EJB QL という、データベースに依存しないバージョンの SQL が定義されており、データベースの定義文には、この SQL を使用します。配備するとき、Sun ONE Application Server プラグインは EJB QL を実際のデータベースに合った SQL に変換して配備記述子に書き込みます。

セッション Bean

エンティティ Bean が共有データを表すのに対し、セッション Bean は、共有されずに複数の概念にまたがっているデータにアクセスします。この種の Bean は、特定の処理を行うために必要なステップを管理することもできます。セッション Bean には、ステートフルとステートレスの 2 つのタイプがあります。「ステートフル」セッション Bean は、クライアントとの会話状態を維持しながら、そのクライアントのためだけに仕事をします。これに対して「ステートレス」セッション Bean は、会話状態を維持することはなく 1 つのクライアント専用にはなりません。スレートレス Bean は、あるクライアントのためのメソッド呼び出しを終了すると、別のクライアントからの要求にサービスを提供することができます。

DiningGuide アプリケーションにおけるクライアント要求としては、データベース内のすべてのレストランのデータの取得、特定のレストランに関するすべての利用者コメントの検出などが考えられます。また、特定のレストランに関するコメントを送信するのもクライアント要求です。これらの要求の間に関連性はなく、会話状態を維持する必要はありません。このため、DiningGuide アプリケーションはステートレスセッション Bean を使用して、それぞれの要求に必要なさまざまなステップを管理します。

セッション Bean は、クライアントの要求を満たすためにレストランと利用者コメントレコードのコレクションを繰り返し作成します。この処理は、フィールドごとの getter および setter メソッドをエンティティ Bean に追加することによっても実現できますが、この方法では、セッション Bean が表の行を取り出す必要があるたびに個々のフィールドについてメソッド呼び出しが必要になります。メソッドの呼び出し回数を減らすため、チュートリアルアプリケーションでは、詳細クラスと呼ばれる特殊なヘルパークラスを使用して、行のデータを保持します。

詳細クラス

詳細クラスは、エンティティ Bean のコンテナ管理フィールドに対応するフィールドと、各フィールド用の getter および setter メソッドを持つ Java クラスです。エンティティ Bean をルックアップする際、セッション Bean は対応する詳細クラスを使用することによって、エンティティ Bean が返す各リモート参照のインスタンスを作成します。セッション Bean は、表示する行データをインスタンス化する際、単に詳細クラスのコンストラクタを呼び出すだけです。こうしてセッション Bean が行インスタンスのコレクションを作成すると、次の段階としてクライアントが表示する

HTML ページの形式に変換することが可能なります。エンティティ Bean インスタンス用のリモート参照を返すより、クライアントに詳細クラスインスタンスを返す方が、ネットワーク帯域幅の消費量が少なくてすみます。

図 3-1 は、詳細クラスの仕組みの概念図です。

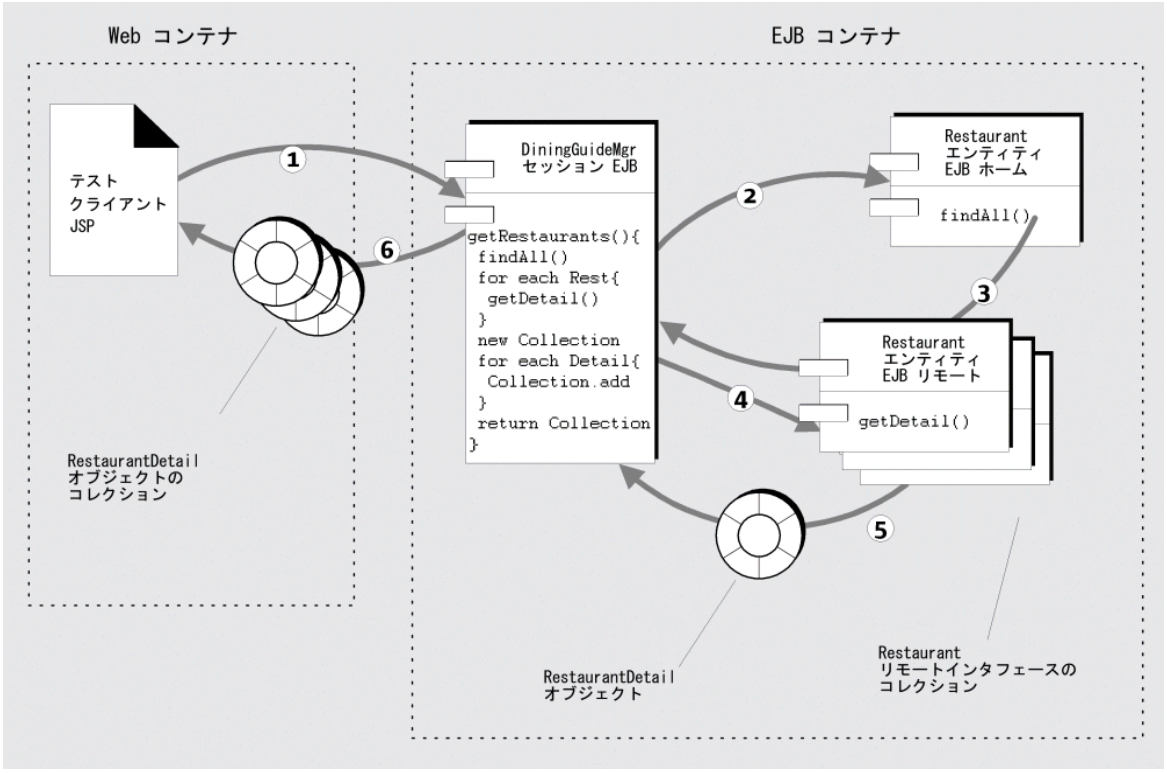


図 3-1 詳細クラスの働き

図 3-1 中の番号が振られた矢印はそれぞれ、以下のアクションを意味します。

1. Web コンテナが、クライアントからの全レストランデータの要求を DiningGuideManager セッション Bean に渡します。
2. セッション Bean が、Restaurant エンティティ Bean の findAll メソッドを呼び出して、Restaurant エンティティ Bean をルックアップします。
3. findAll メソッドが、エンティティ Bean に対する使用可能なすべてのリモート参照を取得します。
4. 返されたリモート参照ごとに、セッション Bean が Restaurant Bean の getRestaurantDetail メソッドを呼び出して、RestaurantDetail クラスをフェッチします。

5. getRestaurantDetail が RestaurantDetail オブジェクトを返し、それがコレクションに追加されます。

6. セッション Bean がすべての RestaurantDetail オブジェクトのコレクションを Web コンテナに返し、これがクライアント表示用の適切な形式にデータを変換します。

手順の概要

EJB 層を作成するには、次の 6 つの作業を行う必要があります。

1. エンティティ Bean の作成

   最初に IDE で、チュートリアルのデータベース表を基にデータベーススキーマを作成します。続いて EJB ビルダーのウィザードを使用して、表を基にした CMP エンティティ Bean を作成します。生成、検索、およびビジネスメソッドを追加、コーディングすることによってエンティティ Bean をプログラミングします。

2. 表示用にクライアントに渡す詳細クラスの作成

   対応するエンティティ Bean と同じフィールドを持つ 2 つの JavaBeans クラスを作成して、同じ補助メソッドを含めます。これらのクラスは、パラメータやフィールド、戻り値の型としてエンティティ Bean によって使用されます。エンティティ Bean インスタンス用のリモート参照を返すより、クライアントにクラスを返す方が、ネットワーク帯域幅の面で効率的です。

3. エンティティ Bean に対する詳細クラスをフェッチするビジネスメソッドの作成

4. IDE のテストアプリケーション機能によるエンティティ Bean のメソッドのテスト

   各エンティティ Bean から、自動的にテストクライアントを 1 つ作成します。このテストクライアントを Web ブラウザで表示して、対応する Bean のインスタンスを作成し、そのインスタンスに対してビジネスメソッドを実行することができます。

5. セッション Bean の作成

   EJB ビルダーを使用して、ステートレスセッション Bean を作成します。エンティティ Bean に対するルックアップを行う Bean の生成メソッドと、各エンティティ Bean の詳細クラスから詳細オブジェクトを取得してコレクションを作成する取得メソッドをプログラミングします。データベースに利用者コメントレコードを作成するメソッドを作成します。また、SOAP 実行環境が必要とするダミーのビジネスメソッドも 2 つ作成します。

6. セッション Bean のメソッドのテスト

   セッション Bean をテストする前に、EJB モジュールのプロパティシートに、CMP エンティティ Bean への参照を追加します。そしてそのセッション Bean から、EJB モジュールを含むテストアプリケーションを作成します。このモジュールに

は、エンティティ Bean のテストアプリケーションからの EJB モジュールを追加し、テストクライアントを使用することによって、セッション Bean のインスタンスを作成し、そのメソッドを実行します。

注 – チュートリアルアプリケーションの作成を開始するには、第 1 章で説明した設定手順をすべて完了しておく必要があります。

EJB ビルダーによるエンティティ Bean の作成

ここでは、第 1 章で作成した 2 つのデータベース表を表す 2 つのエンティティ Bean、Restaurant と Customerreview を作成します。

EJB アーキテクチャのバージョン 2.0 では、エンティティ Bean はローカルインタフェースかリモートインタフェース、あるいはその両方を持つことができます。どれを使用するかの基準は、Bean のメソッドを呼び出すクライアントがエンティティ Bean に対して、リモートにあるかローカルにあるかに依存します。このチュートリアルでは、Web サービスから エンティティ Bean のメソッドへのアクセス方法に柔軟性を持たせるため、リモートとローカル両方のインタフェースを持つ Bean を作成します。セッション Bean がエンティティ Bean のメソッドにアクセスするケース (ローカルインタフェースを使用) と、 Web サービスがそれらメソッドに直接アクセスするケース (リモートインタフェースを使用) の 2 つケースが考えられます。

参考 – EJB ビルダーの操作についての詳細は、Sun ONE Studio 5 ヘルプのEJB コンポーネントに関する項目を参照してください。

注 – 付録 A に、完成したエンティティ Bean のソースコードがあります。

Restaurant および Customerreview エンティティ Bean の作成

最初に、IDE でアプリケーションを保存するディレクトリを作成し、そのディレクトリをマウントします。ディレクトリをマウントすると、IDE のクラスパスに追加されます。次に、マウントしたディレクトリで、2 つのデータベース表を基にデータベーススキーマを作成します。最後に EJB 層用のパッケージを作成し、そのパッケージ内に、データベーススキーマを基にした 2 つのエンティティ Bean を作成します。

チュートリアルのディレクトリの作成

以下のようにしてIDE のファイルシステムでチュートリアルのファイルを保存するディレクトリを作成し、マウントします。

1. **ファイルシステムの適当な場所に、「`DiningGuide`」という名前でディレクトリを作成します。**

注 – 別のディレクトリのサブディレクトリとしてこのディレクトリを作成すると、このチュートリアルで作成するメソッドの指定が長くなる可能性があります。すると、Microsoft Windows 2000 の Service Pack 3 以前のものをインストールしているプラットフォームでは、実行時に問題が発生する可能性があります。この問題を回避するには、ディスクまたはボリュームの最上位 (たとえば `c:\DiningGuide`) にディレクトリを作成します。

---

2. **`DiningGuide` ディレクトリをマウントします。**

   エクスプローラでオブジェクトをマウントすると、そのオブジェクトが IDE のクラスパスに追加されます。

   a. **Sun ONE Studio 5 から「ファイル」→「ファイルシステムをマウント」を選択します。**

      新規ウィザードが表示されます。

   b. **「ローカルディレクトリ」を選択して「次へ」をクリックします。**

      新規ウィザードに「ディレクトリを選択」ページが表示されます。

   c. **ファイル検索機能を使用して `DiningGuide` ディレクトリを見つけ、そのディレクトリを選択して、「完了」をクリックします。**

      ディレクトリ （たとえば `c:\DiningGuide`) がマウントされて、エクスプローラに表示されます。

チュートリアルの表のデータベーススキーマの作成

Restaurant および Customerreview データベース表を基にしたデータベーススキーマを作成します。IDE はデータベースから表の定義を読み取り、スキーマを作成します。

1. 「ツール」→「PointBase ネットワークサーバー」→「サーバーを起動」を選択し、PointBase サーバーを起動します。

   「サーバーを起動」コマンドが選択不可の場合、サーバーはすでに稼働しています。

2. PointBase サンプルデータベースへの接続を開きます。

   a. エクスプローラの「実行時」タブを選択します。

   b. 「データベース」ノードを展開します。

   - jdbc:pointbase:server://localhost:9092/sample [PBPUBLIC on PBPUBLIC] というラベルの四角いアイコンがあり、そのアイコンが完全な四角形の場合は、手順 3に進みます。
   - 四角いアイコンが不完全な四角形の場合は、アイコンを右クリックして、「接続」を選択します。

     接続が開くと、不完全な四角形のアイコンが完全な四角形として再表示されます。

注 – IDE とアプリケーションサーバーを別々にインストールしていて、四角いアイコンがなく、IDE で PointBase 用 JDBC ドライバを有効にしていない場合は、『Sun ONE Studio 5, Standard Edition インストールガイド』の IDEへの外部 PointBase データベースの接続に関する説明を参照してください。

---

3. スキーマの作成を開始します。

   a. エクスプローラで「DigingGuide」ノードを右クリックして、「新規」→「すべてのテンプレート」を選択します。

   新規ウィザードの「テンプレートを選択」ページが表示されます。

   b. 「データベース」ノードを展開して「データベーススキーマ」を選択し、「次へ」をクリックします。

   新規ウィザードの「新規オブジェクト名」ページが表示されます。

4. 「名前」フィールドに **`dgSchema`** と入力して、「次へ」をクリックします。

   ウィザードの「データベース接続」ページが表示されます。

5. スキーマのソースデータベースを作成します。

   a. 「既存の接続」オプションを有効にします。

b. 一覧から jdbc:pointbase:server://localhost:9092/sample [PBPUBLIC on PBPUBLIC] を選択します。

c. 「次へ」をクリックします。

「表とビュー」ページが表示されます。

6. スキーマで基にするテーブルを選択します。

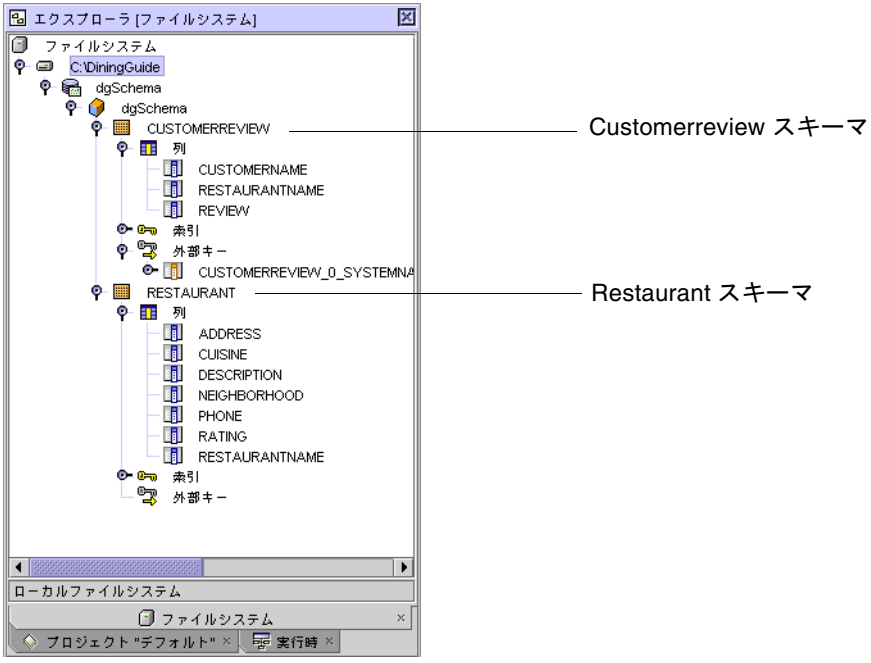
a. 使用可能な表のリストで CUSTOMERREVIEW を選択して、「追加」ボタンをクリックします。

b. 同じリストで RESTAURANT を選択して、「追加」ボタンをクリックします。

選択した表とビューの一覧に CUSTOMERREVIEW および RESTAURANT の表が表示されます。

c. 「完了」をクリックします。

エクスプローラ内の DiningGuide ディレクトリの下に新しいデータベーススキーマが表示されます。サブノードをすべて展開すると、次のようになります。



EJB 層用 Java パッケージの作成

マウントした DigingGuide ノード内に、EJB 層 （アプリケーションの全データ) を保持する Java パッケージを作成します。

1. 「DigingGuide」ノードを右クリックして、「新規」→「Java パッケージ」を選択します。
2. 新規パッケージに `Data` という名前を付けて、「完了」をクリックします。
   DigingGuide ディレクトリ下に新しい「Data」パッケージが表示されます。

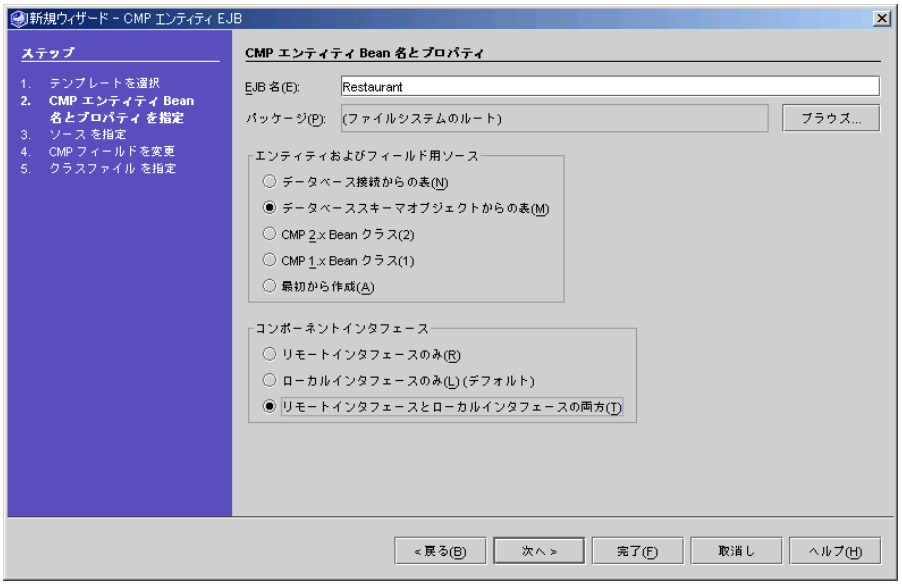
Restaurant エンティティ Bean の作成

以下のようにして Restaurant 表を基にしたエンティティ Bean を作成します。

1. Restaurant エンティティ Bean の作成を開始します。
   a. 新しい「Data」パッケージを右クリックして、「新規」→「すべてのテンプレート」を選択します。
   b. 「テンプレート」ウィザードで「J2EE」ノードを展開し「CMP エンティティ EJB」を選択して、「次へ」をクリックします。
      新規ウィザードの「CMP エンティティ Bean 名とプロパティ」ページが表示されます (この設定は EJB ビルダーモジュールによって使用されます)。ウィザードの任意のページで「ヘルプ」ボタンをクリックすると、CMP エンティティ Bean の作成に関する、状況に応じたヘルプを見ることができます。
   c. CMP エンティティ Bean に `Restaurant` という名前を付けて、次のオプションを選択します。

オプションの分類	選択するオプション
エンティティおよびフィールド用ソース	データベーススキーマオブジェクトからの表
コンポーネントインタフェース	リモートインタフェースとローカルインタフェースの両方

新規ウィザードが次のような表示になります。



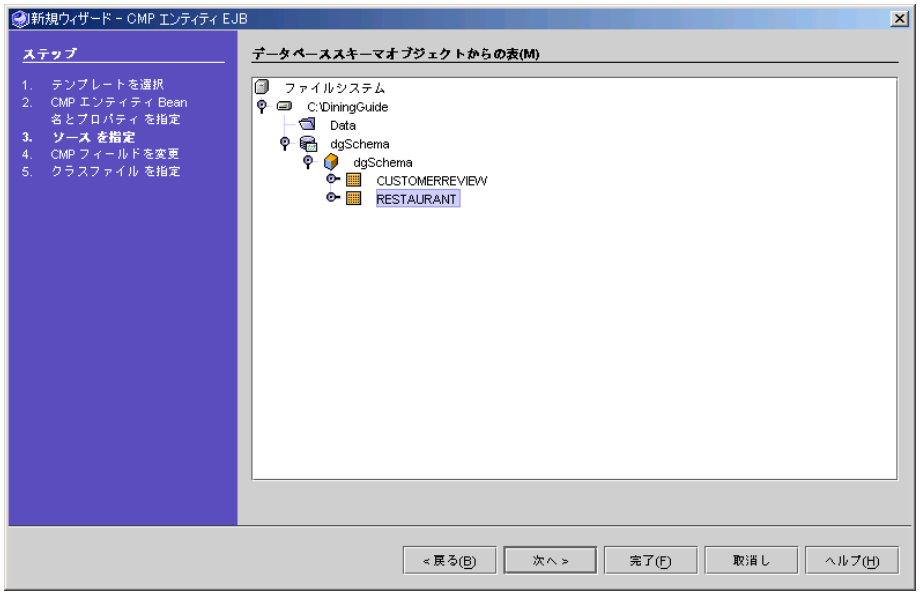
d. 「次へ」をクリックします。

「データベーススキーマオブジェクトからの表」ページが表示されます。

2. RESTAURANT スキーマを指定します。

a. 「DigingGuide」ノードを展開して、「dgSchema」ノードの下のノードを展開し、「Restaurant」表を選択します。

ページは次のようになります。



b. 「次へ」をクリックして「CMP フィールド」ページに進みます。

3. 必要に応じてフィールドをカスタマイズして CMP Bean を定義します。

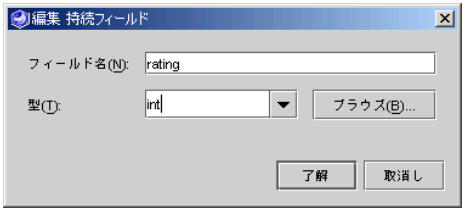
Restaurant データベース列の横に、ウィザードによって Restaurant エンティティ Bean が作成されるときに対応づけられる Java フィールドが表示されます。このチュートリアルでは、「rating」フィールドの型を変更すると仮定します。

a. 「rating」フィールドを選択し、「編集」ボタンをクリックします。

「持続フィールド」ダイアログが表示されます。

b. 「型」フィールドのテキストを削除して、`int` と入力します。

ダイアログは以下のような表示になります。



c. 「了解」をクリックしてダイアログを閉じます。

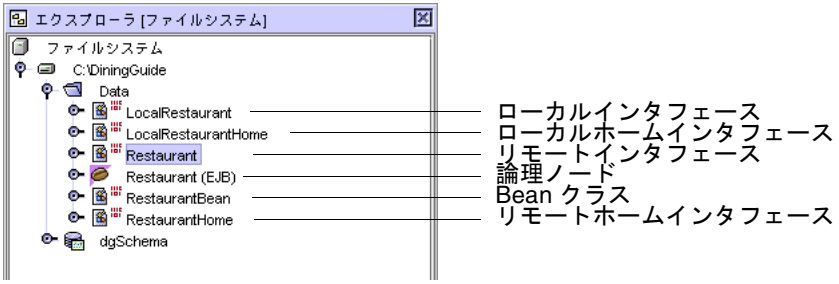
「rating」Java フィールドの型が `int` に変更されています。

d. 「次へ」をクリックします。

「CMP エンティティ Bean クラスファイル」ページが表示され、Restaurant Bean 用に作成される部品が表示されます。新しいエンティティ Bean には、EJB ビルダーウィザードによって、データベース表と同じ名前が自動的に付けられます。

4. 「完了」をクリックして、Bean の作成を終了します。

Restaurant エンティティ Bean とその全部品が作成され、「エクスプローラ」ウィンドウに表示されます。



部品のうちの 5 つはインタフェース、1 つは Bean クラスです。6 番目の部品の**論理ノード**は、エンタープライズ Bean のすべての要素を 1 つのグループにまとめて、簡単に扱えるようにします。

Customerreview エンティティ Bean の作成

ここでは、Restaurant Bean を作成したのと同様の手順で Customerreview エンティティ Bean を作成します。

1. Restaurant エンティティ Bean の作成を開始します。

a. 「Data」パッケージを右クリックして、「新規」→「CMP エンティティ EJB」選択します。

この項目がコンテキストメニューに表示されるようになっています。このショートカットは「新規」メニューから項目を選択すると作成される便利な機能です。

b. CMP Bean に Customerreview という名前を付けて、次のオプションを選択します。

オプションの分類	選択するオプション
エンティティおよびフィールド用ソース	データベーススキーマオブジェクトからの表
コンポーネントインタフェース	リモートインタフェースとローカルインタフェースの両方

c. 「次へ」をクリックします。

「データベーススキーマオブジェクトからの表」ページが表示されます。

2. CUSTOMERREVIEW スキーマを指定します。

a. 「DigingGuide」ノードと「dgSchema」ノードの下のすべてノードを展開します。

b. 「CUSTOMERREVIEW」表を選択します。

c. 「次へ」をクリックします。

3. Bean の作成を終了します。

a. 「CMP フィールド」ページで「次へ」をクリックします。

b. 最終ページ (「CMP エンティティ Bean クラスファイル」ページ) で「完了」をクリックします。

エクスプローラの「Data」パッケージに `Customerreview` エンティティ Bean が表示されます。CustomerreviewKey という主キークラスもあることに注目してください。これは、エンティティ Bean に複合主キーがある場合に自動的に作成されるクラスです。第 1 章の 表 1-2 を参照して、この表の複合主キーを確認してください。)

CMP エンティティ Bean の生成メソッドの作成

ここでは、両方の エンティティ Bean の生成メソッドを作成して、パラメータと、それら Bean のインスタンスのフィールドを初期化するコードを追加します。

Restaurant Bean の生成メソッドの作成

以下の手順で、Restaurant エンティティ Bean の生成メソッドを作成します。

1. エクスプローラで Restaurant (EJB) 論理ノード (Bean のアイコン) を右クリックします。
2. コンテキストメニューから「生成メソッドを追加」を選択します。

   「新規生成メソッドを追加」ダイアログが表示されます。
3. Restaurant データベースのすべての列にメソッドのパラメータを追加します。

   a. 「パラメータ」セクションにある「追加」ボタンをクリックします。

      「メソッドのパラメータを入力」ダイアログが表示されます。

   b. フィールド名として **`restaurantname`** と入力します。

c. 型として `java.lang.String` を選択します。

d. 「了解」をクリックするか Enter を押してメソッドを作成し、ダイアログを閉じます。

e. 同様にその他のパラメータを作成します。

フィールド名	型
cuisine	java.lang.String
neighborhood	java.lang.String
address	java.lang.String
phone	java.lang.String
description	java.lang.String
rating	int

注 – これらのパラメータを作成する順序は、テストアプリケーション機能を使用して Bean をテストするときに重要な意味を持ちます。このため、上記に列挙している順序でパラメータを作成してください。

デフォルトで作成される 2 つの例外はそのまま残します。ホームとローカルホームの両方のインタフェースにメソッドが追加されることを確認します。

4. 「了解」をクリックします。

   RestaurantHome と LocalRestaurantHome インタフェースのそれぞれに生成メソッドが 1 つずつ伝達され、Restaurant Bean クラス (ReastaurantBean) に jbCreate メソッドが伝達されます。この Bean クラスには、関係する ejbPostCreate メソッドも追加されます。

5. Restaurant(EJB) 論理ノードを展開して、「生成メソッド」フォルダを開き、生成メソッドをダブルクリックします。

   Bean の ejbCreate メソッドにカーソルが置かれた状態で、ソースエディタが表示されます。

注 – 「生成メソッド」ノードを右クリックして、「ヘルプ」を選択すると、生成メソッドに関するオンラインヘルプを見ることができます。

---

6. Bean インスタンスのフィールドを初期化するコード (太字の部分) を ejbCreate method メソッドの本体に追加します。

```
public String ejbCreate(java.lang.String restaurantname,
java.lang.String cuisine, java.lang.String neighborhood,
java.lang.String address, java.lang.String phone,
java.lang.String description, int rating) throws
javax.ejb.CreateException {
    if (restaurantname == null) {
//ソースエディタで次の 2 行を 1 行にする
        throw new javax.ejb.CreateException("The restaurant name
is required.");
    }
    setRestaurantname(restaurantname);
    setCuisine(cuisine);
    setNeighborhood(neighborhood);
    setAddress(address);
    setPhone(phone);
    setDescription(description);
    setRating(rating);

    return null;
}
```

参考 – 入力またはコピー操作によってソースエディタにコードを入力した後、そのコードブロックを選択して Ctrl+Shift F を押すと、書式を整えることができます。PDF ファイルからコピーする場合は、コードのコメントで指示しているようにワードラップで途切れた行を連結してください。

Restaurant エンティティ Bean の生成メソッドが呼び出されると、この Bean のコンテナ管理フィールドに基づいて、データベースに新しいレコードが作成されます。

7. 「Restaurant(EJB)」論理ノードを選択し、F9 キーを押して Bean をコンパイルします。

   Restaurant エンティティ Bean がコンパイルされます。

Customerreview Bean の生成メソッドの作成

以下の手順で、Customerreview エンティティ Bean の生成メソッドを作成します。

1. Customerreview(EJB) 論理ノード (Bean のアイコン) を右クリックして、「生成メソッドを追加」を選択します。

2. 「追加」ボタンを使用して、CustomerReview 表の各列に 1 つ、合計で 3 つのパラメータを作成します。

フィールド名	型
restaurantname	java.lang.String
customername	java.lang.String
review	java.lang.String

注 – 前の手順 3 のとき同様、これらのパラメータは、列挙されている順序で作成してください。

デフォルトで作成される 2 つの例外はそのまま残します。ホームとローカルホームの両方のインタフェースにメソッドが追加されることを確認します。

3. 「了解」をクリックします。

4. Customerreview(EJB) 論理ノードを開いて、「生成メソッド」フォルダを開き、生成メソッドをダブルクリックします。

   Bean の ejbCreate メソッドにカーソルが置かれた状態で、ソースエディタが表示されます。

5. Bean インスタンスのフィールドを初期化するコード (太字の部分) を ejbCreate メソッドの本体に追加します。

```
public CustomerreviewKey ejbCreate(java.lang.String
restaurantname, java.lang.String customername, java.lang.String
review) throws javax.ejb.CreateException {
    if ((restaurantname == null) || (customername == null)) {
//ソースエディタで次の 2 行を 1 行にする
        throw new javax.ejb.CreateException("Both the restaurant
name and customer name are required.");
    }
    setRestaurantname(restaurantname);
    setCustomername(customername);
    setReview(review);

    return null;
}
```

参考 – 入力またはコピー操作によってソースエディタにコードを入力した後、そのコードブロックを選択して Ctrl+Shift F を押すと、書式を整えることができます。PDF ファイルからコピーする場合は、コードのコメントで指示しているようにワードラップで途切れた行を連結してください。

ejbCreate メソッドが呼び出されると、この Bean のコンテナ管理フィールドに基づいて、データベースに新しいレコードが作成されます。

6. **「Customerreview(EJB)」論理ノードを選択し、F9 キーを押して Bean をコンパイルします。**

   Customerreview エンティティ Bean がコンパイルされます。

次に、コンテキストから、すべてのインスタンスまたは選択されたインタンスを特定する検索メソッドを両方のエンティティ Bean に作成します。

エンティティ Bean に対する検索メソッドの作成

ここでは、すべてのレストランデータを検出する findAll メソッドを Restaurant Bean に作成します。また、特定のレストランのコメントデータを検出する findByRestaurantName メソッドを Customerreview Bean に作成します。

findByPrimaryKey を以外の各検索メソッドは、配備記述子の照会要素に関連づける必要があります。2 つのエンティティ Bean の検索メソッドの作成では、 SQL 文には、データベースに依存しない EJB 2.0 仕様の言語、すなわち EJB QL を指定します。配備の際に、アプリケーションサーバープラグインは、EJB QL をターゲットのデータベースの SQL に変換します。

Restaurant Bean の findAll メソッドの作成

Restaurant Bean の findAll メソッドを作成するには、次の操作を行います。

1. **「Restaurant (EJB)」論理ノードを右クリックして、「検索メソッドを追加」を選択します。**

   「新規検索メソッドを追加」ダイアログが表示されます。

2. **「名前」フィールドの「find」という文字列の後に `All` を入力します。**
3. **戻り値の型として「`java.util.Collection`」を選択します。**
4. **デフォルトの 2 つの例外をそのまま受け入れます。**

5. 以下のように EJB QL 文を定義します。

EJB QL 文	テキスト
SELECT	`Object(o)`
FROM	`Restaurant o`

6. ホームとローカルホームの両方のインタフェースにメソッドが追加されることを確認します。
7. 「了解」をクリックします。

   Restaurant Bean のローカルおよびローカルホームインタフェースに findAll メソッドが作成されます。

注 – 「検索メソッド」ノードを右クリックして、「ヘルプ」を選択すると、検索メソッドに関するオンラインヘルプを見ることができます。

---

8. 「Restaurant(EJB)」論理ノードを選択し、F9 キーを押して Bean をコンパイルします。

   Restaurant エンティティ Bean がコンパイルされます。

Customerreview Bean の findByRestaurantName メソッドの作成

Customerreview Bean の findByRestaurantName メソッドを作成するには、次の操作を行います。

1. 「Customerreview (EJB)」 論理ノードを右クリックして、「検索メソッドを追加」を選択します。

   「新規検索メソッドを追加」ダイアログが表示されます。
2. 「名前」フィールドの「find」という文字列の後に **`ByRestaurantName`** を入力します。
3. 戻り値の型として「`java.util.Collection`」を選択します。
4. パラメータの「追加」ボタンをクリックします。

   「メソッドのパラメータを入力」ダイアログが表示されます。
5. フィールド名として **`restaurantname`** と入力します。
6. 型として「`java.lang.String`」を選択します。
7. 「了解」をクリックします。

8. デフォルトの 2 つの例外をそのまま受け入れます。
9. 以下のように EJB QL 文を定義します。

EJB QL 文	テキスト
SELECT	`Object(o)`
FROM	`Customerreview o`
WHERE	`o.restaurantname = ?1`

(Where 句に使用する数値は、検索メソッド内のパラメータ位置によって異なります。この場合、パラメータは 1 つだけのため、数値は 1 になります。)

10. ホームとローカルホームの両方のインタフェースにメソッドが追加されることを確認します。
11. 「了解」をクリックします。

    Restaurant Bean のローカルおよびローカルホームインタフェースに findByRestaurantName メソッドが作成されます。
12. 「Customerreview(EJB)」論理ノードを選択し、F9 キーを押して Bean をコンパイルします。

    Customerreview エンティティ Bean がコンパイルされます。

テスト用のビジネスメソッドの作成

ここでは、エンティティ Bean ごとに、そのパラメータの値を返すビジネスメソッドを作成します。ビジネスメソッドを作成することによって、後で Bean をテストすることができます。Restaurant に対しては getRating メソッド、Customerreview に対しては getReview メソッドを作成します。

Restaurant Bean の getRating メソッドの作成

Restaurant Bean に対する getRating ビジネスメソッドを作成するには、以下の操作を行います。

1. Restaurant エンティティ Bean のビジネスメソッドを確認します。
   a. 「Restaurant(EJB)」論理ノードを展開してから、その「ビジネスメソッド」ノードを展開します。

      このエンティティ Bean には、まだビジネスメソッドはありません。

b. Restaurant Bean のクラス (RestaurantBean) を展開して、その「メソッド」ノードを展開します。

Bean の各フィールドに getRating メソッドなどの補助メソッドが用意されています。

これらのメソッドは、データソースとの同期の際にコンテナによって使用されます。開発でこれらのメソッドを使用するには、メソッドをビジネスメソッドとして作成する必要があります。

2. getRating ビジネスメソッドを作成します。

a. 「Restaurant (EJB)」 論理ノードを右クリックして、「ビジネスメソッドを追加」を選択します。

「新規ビジネスメソッドを追加」ダイアログが表示されます。

b. 「名前」フィールドに**`getRating`** と入力します。

c. 「戻り値の型」フィールドの一覧から `int` を選択します。

d. デフォルトの例外 (RemoteException) とメソッドの追加先をそのまま受け入れます。この追加先のリモートインターフェースおよびローカルインターフェースの両方にメソッドが作成されます。

e. 「了解」をクリックします。

3. Restaurant エンティティ Bean のビジネスメソッドを再度確認します。

論理ノードの「ビジネスメソッド」フォルダの下に getRating メソッドがあり、ビジネスメソッドとして利用します。getRating メソッドを使用すると、選択されたレストランレコードの rating 列の値が返されます。

4. 「Restaurant(EJB)」論理ノードを右クリックし、コンテキストメニューから「EJB の検査」を選択します。

Restaurant エンティティ Bean がコンパイルされます。続いて、Customerreview Bean に対する同様のメソッドを作成します。

Customerreview Bean の getReview メソッドの作成

Customerreview Bean に getReview ビジネスメソッドを作成するには、次の操作を行います。

1. 「Customerreview (EJB)」 論理ノードを右クリックして、「ビジネスメソッドを追加」を選択します。

「新規ビジネスメソッドを追加」ダイアログが表示されます。

2. 「名前」フィールドに**`getReview`** と入力します。

3. 「戻り値の型」フィールドに `java.lang.String` と入力します。

   デフォルトの例外 (RemoteException) とメソッドの追加先をそのまま受け入れます。この追加先のリモートインタフェースおよびローカルインタフェースの両方にメソッドが作成されます。

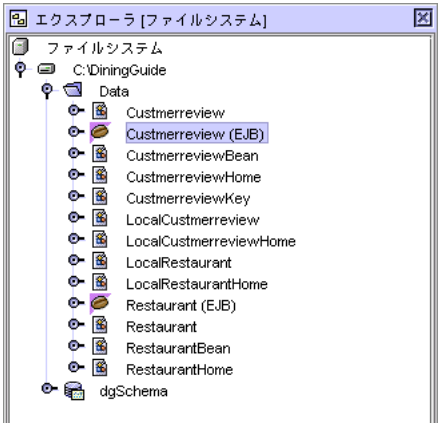
4. 「了解」をクリックします。

   これで getReview メソッドをビジネスメソッドとして使用できます。getReview メソッドを使用すると、選択されたレストランレコードの review 列の値が返されます。

5. 「Customerreview(EJB)」論理ノードを右クリックし、コンテキストメニューから「EJB の検査」を選択します。

   Customerreview エンティティ Bean がコンパイルされます。

6. エクスプローラから Bean が未コンパイルであることを示すアイコンがなくなったことを確認します。



エンティティ Bean データを表示する詳細クラスの作成

31 ページの「詳細クラス」で説明したように、このチュートリアルアプリケーションでは、詳細クラスを使用して表の行データを保持することによって、エンティティ Bean に対するメソッド呼び出しを表示したり、その回数を減らしたりします。それらの詳細クラスのフィールド、フィールドごとの補助メソッド、各フィールドを設定するコンストラクタは、対応する エンティティ Bean と同じである必要があります。

注 – 付録 A に、完成した詳細クラスのソースコードがあります。

詳細クラスの作成

最初に RestaurantDetail クラスと CustomerreviewDetail クラスを作成します。

1. **エクスプローラで「Data」パッケージを右クリックして、「新規」→「すべてのテンプレート」を選択します。**
   新規ウィザードが表示されます。
2. **ウィザードの「テンプレートを選択」ページで「Java Bean」ノードを展開し、「Bean」を選択します。**
3. **「次へ」をクリックします。**
   ウィザードの「新規オブジェクト名」ページが表示されます。
4. **「名前」フィールドに `RestaurantDetail` と入力し、「完了」をクリックします。**
   エクスプローラに新しい Bean が表示されます。
5. **「Data」パッケージノードを右クリックして、「新規」→「Bean」を選択します。**
   IDE によって Java Bean テンプレートへのショートカットが作成されています。
6. **「名前」フィールドに `CustomerreviewDetail` と入力し、「完了」をクリックします。**

詳細クラスのプロパティとその補助メソッドの作成

続いて、詳細クラスにエンティティ Bean の CMP フィールドの複製を作成します。このためには、これらのフィールドを Bean プロパティとして追加します。(エンティティ Bean の Bean クラスの「Bean パターン」の内容を見ると、CMP フィールドがプロパティとして保存されていることが分かります。) フィールドを追加しながら、各フィールド用の補助メソッドを自動的に作成することができます。

詳細クラスのプロパティとメソッドを作成するには、次の操作を行います。

1. **「RestaurantDetail」、「クラス RestaurantDetail」の順にノードを展開します。**
2. **「Bean パターン」ノードを右クリックして、「追加」→「プロパティ」を選択します。**
   「新規プロパティパターン」ダイアログが表示されます。

3. 「名前」フィールドに **restaurantname** と入力します。
4. 型として「String」を選択します。
5. 「フィールドを生成」オプションを選択します。
6. 「Return 文を生成」オプションを選択します。
7. 「Set 文を生成」オプションを選択します。
8. 「了解」をクリックします。
9. 手順 2 ～ 手順 8 を繰り返すことによって、以下のプロパティを作成します。

```
cuisine (String)
neighborhood (String)
address (String)
phone (String)
description (String)
rating (int)
```

10. RestaurantDetail bean の「メソッド」ノードを展開します。

    各フィールドに対する補助メソッドが生成されています。
11. 「CustomerreviewDetail」、「class CustomerreviewDetail」の順にノードを展開します。
12. 手順 2 から手順 8 を繰り返すことによって、「Bean パターン」ノードに次のプロパティを作成します。

```
restaurantname (String)
customername (String)
review (String)
```

詳細クラスのコンストラクタの作成

クラスのフィールドをインスタンス化する詳細クラスのコンストラクタを作成するには、次の操作を行います。

1. RestaurantDetail Bean を展開して 「クラス RestaurantDetail」 ノードを右クリックし、「追加」→「コンストラクタ」を選択します。

   「新規コンストラクタを編集」ダイアログが表示されます。
2. 次のメソッドパラメータを追加して、「了解」をクリックします。

```
java.lang.String restaurantname
java.lang.String cuisine
java.lang.String neighborhood
java.lang.String address
```

```
java.lang.String phone
java.lang.String description
int rating
```

3. フィールドを初期化するコード (次のコードの太字部分) を RestaurantDetail コンストラクタに追加します。

```
public RestaurantDetail(java.lang.String restaurantname,
java.lang.String cuisine, java.lang.String neighborhood,
java.lang.String address, java.lang.String phone,
java.lang.String description, java.lang.Integer rating){
    System.out.println("Creating new RestaurantDetail");
    setRestaurantname(restaurantname);
    setCuisine(cuisine);
    setNeighborhood(neighborhood);
    setAddress(address);
    setPhone(phone);
    setDescription(description);
    setRating(rating);
}
```

参考 – ソースエディタにコピーしたコードブロックを選択して、Ctrl+Shift F を押すと、その部分の書式を整えることができます。

---

4. 同様に次のパラメータを使用して、CustomerreviewDetail クラスにコンストラクタを追加します。

```
java.lang.String restaurantname
java.lang.String customername
java.lang.String review
```

5. フィールドを初期化するコード (次のコードの太字部分) を CustomerreviewDetail コンストラクタに追加します。

```
public CustomerreviewDetail(java.lang.String restaurantname,
java.lang.String customername, java.lang.String review) {
    System.out.println("Creating new CustomerreviewDetail");
    setRestaurantname(restaurantname);
    setCustomername(customername);
    setReview(review);
}
```

6. 「Data」パッケージを右クリックして、「すべてをコンパイル」を選択します。

   パッケージがコンパイルされます。

続いて、詳細クラスのインスタンスを取り出す取得メソッドをエンティティ Bean に作成します。

エンティティ Bean に対する、詳細クラスをフェッチするビジネスメソッドの作成

ここでは、対応する詳細クラスのインスタンスを返すメソッドを各エンティティ Bean に作成します。

これらのメソッドを作成するには、次の操作を行います。

1. **エクスプローラ で「Restaurant (EJB)」論理ノードを右クリックして、「ビジネスメソッドを追加」を選択します。**
   「新規ビジネスメソッドを追加」ダイアログが表示されます。
2. **「名前」フィールドに `getRestaurantDetail` と入力します。**
3. **「戻り値の型」では、「ブラウズ」ボタンを使用して、「`RestaurantDetail`」クラスを選択します。**

参考 – 必ず Bean のノードではなく、クラス () を選択します。

「戻り値の型」フィールドに Data.RestaurantDetail と表示されます。

4. **「リモートインタフェースとローカルインタフェースの両方」オプションが有効であることを確認します。**
5. **他のすべての値はデフォルトのままで「了解」をクリックしてメソッドを作成します。**
6. **論理 Bean の「ビジネスメソッド」ノード下にある新しいメソッドを見つけてダブルクリックし、ソースエディタで表示します。**
7. **メソッドの本体に次の太字のコードを追加します。**

```
public Data.RestaurantDetail getRestaurantDetail() {
    return (new RestaurantDetail(getRestaurantname(),
getCuisine(),getNeighborhood(), getAddress(), getPhone(),
getDescription(), getRating()));
}
```

8. **「Restaurant(EJB)」論理ノードを選択し、F9 キーを押してコードをコンパイルします。**

9. エクスプローラ で「Customerreview (EJB)」論理ノードを右クリックして、「ビジネスメソッドを追加」を選択します。
10. 「名前」フィールドに **getCustomerreviewDetail** と入力します。
11. 戻り値の型では、「ブラウズ」ボタンを使用して、「クラス CustomerreviewDetail」を選択します。
12. 「リモートインタフェースとローカルインタフェースの両方」オプションが有効であることを確認します。
13. 他のすべての値はデフォルトのままで、「了解」をクリックしてメソッドを作成します。
14. ソースエディタでメソッドを開き、そのソースコードに太字部分を追加します。

```
public Data.CustomerreviewDetail getCustomerreviewDetail() {
    return (new CustomerreviewDetail(getRestaurantname(),
getCustomername(), getReview()));
}
```

15. 「Data」パッケージを右クリックして、「すべてをコンパイル」を選択します。
    パッケージ全体がコンパイルされます。

    これで、このチュートリアルアプリケーションのエンティティ Bean およびその詳細クラスヘルパーの作成は完了しました。次に行うのは Bean のテストです。

エンティティ Bean のテスト

Sun ONE Studio 5 IDE では、テストクライアントを作成して、エンティティ Bean でメソッドを実行することができます。テストクライアントには、Bean インスタンスを作成し、Web ブラウザでその Bean の生成、検索、ビジネスメソッドを実行することを可能にする JavaServer Pages 技術が採用されています。

このテストクライアントを使用して、Restaurant Bean の生成メソッドおよび getRating メソッドを実行してみてください。

Restaurant Bean のテストクライアントの作成

テストクライアントを作成すると、EJB モジュール 1 つと J2EE アプリケーションモジュール 1 つ、それに多数のサポート要素が自動的に生成されます。

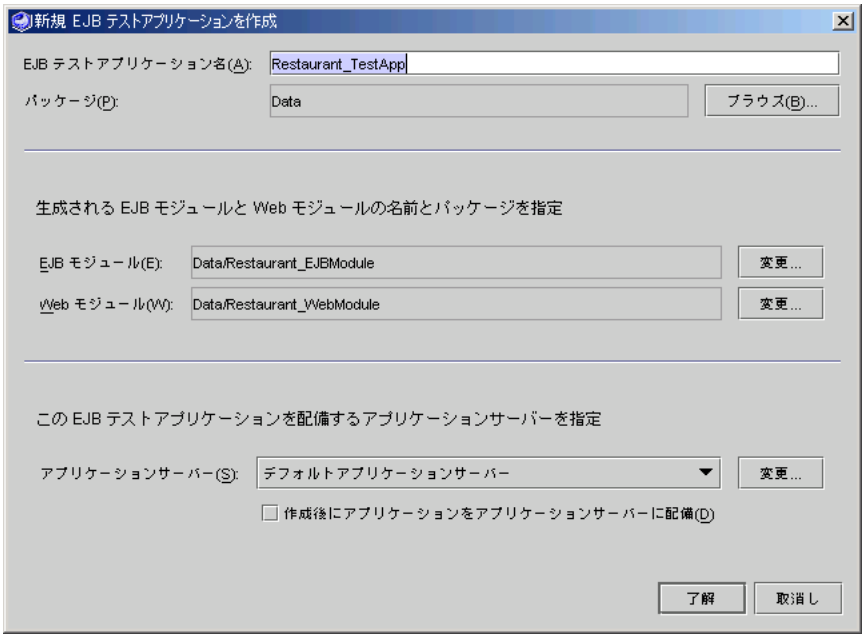
Restaurant エンティティ Bean のテストクライアントを作成するには、以下の操作を行います。

1. 「Restaurant (EJB)」論理ノードを右クリックして、「新規 EJB テストアプリケーションを作成」を選択します。

   EJB テストアプリケーションウィザードが表示されます。

2. すべてのデフォルト値をそのまま受け入れます。

   ウィザードが次のような表示になります。

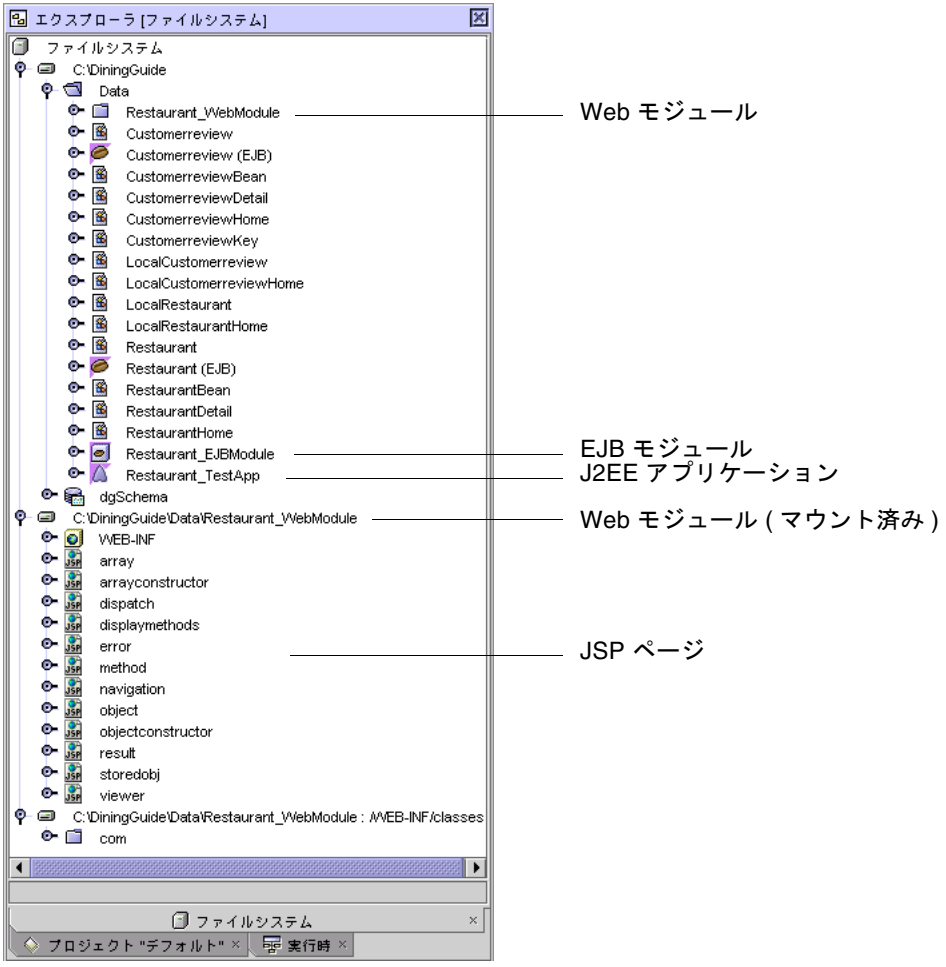


3. 「了解」をクリックします。

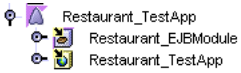
   進捗モニターが少しの間表示され、処理が完了すると消えます。別のウィンドウが表示され、作成された Web モジュールが「プロジェクト」タブでも見えることを示して自動的に消えます。消えない場合は、「了解」をクリックしてウィンドウを閉じてください。

4. 生成されたテストオブジェクトをエクスプローラで確認します。

   Restaurant_EJBModule という EJB モジュールと Restaurant_WebModule という Web モジュール (別にマウントされている)、Restaurant_TestApp という J2EE アプリケーションが生成されています。Web モジュールには、テストクライアントをサポートする JSP ページがいくつか含まれています。J2EE アプリケーションには、EJB モジュールと Web モジュールに対する参照が含まれています。



また、作成された J2EE アプリケーションには、Web モジュールと EJB モジュールに対する参照が含まれています。これらのオブジェクトは、Restaurant_TestApp を展開することによって見ることができます。



Sun ONE Application Server 7 プラグインへのデータベース情報の指定

テストクライアントがデータベースを検出してログオンするためには、EJB モジュールのアプリケーションサーバーのプロパティに、データベースに関する情報を追加する必要があります。

必要な情報を追加するには、次の操作を行います。

1. エクスプローラで EJB モジュール (Restaurant_EJBModule) を展開して、その下の「Restaurant」ノード (Restaurant Bean への参照) を選択し、そのプロパティを表示します。

   「プロパティ」ウィンドウが表示されていない場合は、「表示」→「プロパティ」を選択します。

2. 「プロパティ」ウィンドウの「Sun ONE AS」タブを選択します。

3. プロパティが以下の設定になっていることを確認します。

プロパティ	値
マップされたフィールド	7 コンテナ管理のマップされたフィールド
マップされた主表	RESTAURANT
マップされたスキーマ	dgSchema

   上記の値が表示されている場合は、手順 5 に進みます。

4. 上記の値が表示されない場合は、次の手順で Restaurant Bean を再びマップします。

   a. 「マップされたフィールド」プロパティの値フィールドを選択して、省略符号ボタン (...) をクリックします。

      データベースへのマップウィザードが表示されます。

   b. 「次へ」をクリックして、「表を選択」ページを表示します。

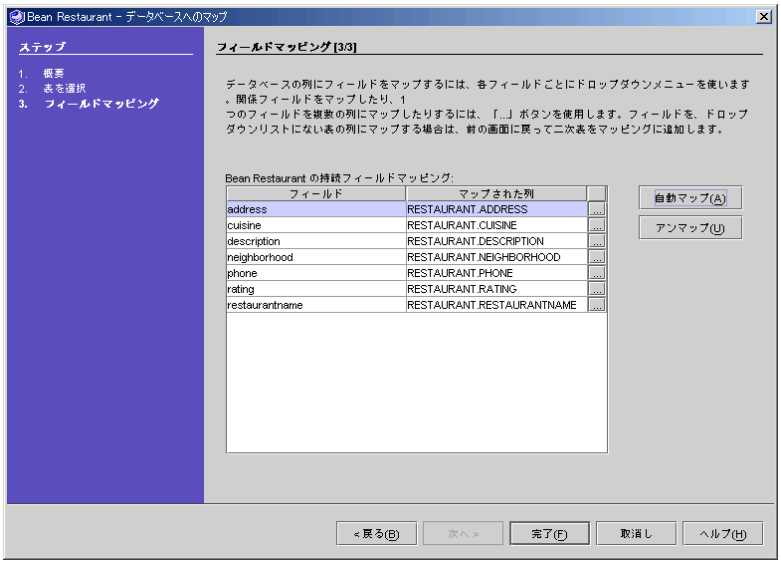
   c. 「主表」フィールドのドロップリストから「RESTAURANT」を選択します。

      一覧の中に「RESTAURANT」がない場合は、「ブラウズ」ボタンを使用して、restSchema スキーマ内にある表を検索します。

   d. 「次へ」をクリックして、「フィールドマッピング」ページを表示します。

   e. フィールドがマップされていない場合は、「自動マップ」ボタンをクリックします。

      各フィールドにマップされる値が表示されます。



f. 「完了」をクリックします。

手順 3 に示す値が表示されます。

5. EJB モジュール (Restaurant_EJBModule) を選択して、そのプロパティを表示します。

6. 「プロパティ」ウィンドウの「Sun ONE AS」タブを選択します。

7. 「CMP リソース」プロパティの値フィールドをクリックして、省略符号ボタン (...) を表示します。

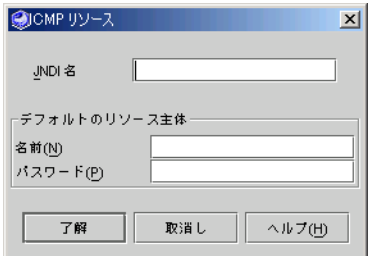
8. 省略符号ボタン (...) をクリックして、「CMP リソース」プロパティのプロパティエディタを表示します。

9. 「JNDI 名」フィールドに **`jdo/PointbasePM`** と入力します。

これは、10 ページの「JDBC リソースの設定 (Microsoft Windows のスーパーユーザー)」 または 11 ページの「JDBC リソースの設定 (administrator 以外のユーザー)」 で定義した、JDBC 持続マネージャの JNDI 名です。

10. 「名前」フィールドと「パスワード」フィールドに、それぞれデータベースのユーザー名とパスワードを入力します。

PointBase サンプルデータベースの場合は、これはともに PBPUBLIC です。エディタの表示は次のようになります。



11. 「了解」をクリックして値を適用し、プロパティエディタを閉じます。

テストアプリケーションで使用するデータベースを設定する作業が終了しました。これで、テストアプリケーションを配備できます。

Restaurant Bean のテストアプリケーションの配備と実行

注 – テストアプリケーションなど、データベースにアクセスする J2EE アプリケーションを配備する前に、PointBase サーバーが実行されていることを確認してください。また、 Sun ONE Application Server 7 サーバーが稼働していて、IDE のデフォルトのアプリケーションサーバーになっていることを確認してください。9 ページの「Sun ONE Application Server 7 がデフォルトサーバーであることの確認」を参照してください。

Restaurant テストアプリケーションを配備するには、以下の操作を行います。

- Restaurant_TestApp」J2EE アプリケーションノードを右クリックして、コンテキストメニューから「実行」を選択します。

「進捗モニター」ウィンドウに、配備プロセスの進捗状況が示されます。出力ウィンドウ上のサーバーインスタンスのログファイルのタブに、進捗メッセージが表示されます。アプリケーションが正常に配備されると、完了のメッセージが表示されます。

IDE がデフォルトの Web ブラウザを起動し、テストアプリケーションのホーム URL を表示します。

`http://`*server-host*`:`*server-port-number*`/Restaurant_TestApp/dispatch.jsp`

ブラウザには、テストクライアントが次のように表示されます。

注 – ブラウザが見えない場合は、IDE のウィンドウの背後に隠れている可能性があります。出力ウィンドウにあるサーバーインスタンスのログファイルのタブを確認すると、アプリケーションが配備されたかどうかがわかります。アプリケーションが正常に配備されると、完了のメッセージが表示されます。

テストクライアントを使用した Restaurant Bean のテスト

テストクライアントの Web ページが表示されたら、Restaurant Bean のホームインタフェースの生成メソッドを使用して、Bean のインスタンスを作成します。そして、そのインスタンスでビジネスメソッド (この場合は getRating) をテストします。

`Restaurant` Bean をテストするには、次の操作を行います。

1. **生成メソッドを呼び出して、Restaurant Bean のインスタンスを作成します。**

   生成メソッドは、「Data.RestaurantHome でメソッドを呼び出す」という見出しの下にあり、その下に 7 つのフィールドがあります。これらのフィールドには名前がありませんが、順序が43 ページの「CMP エンティティ Bean の生成メソッドの作成」の手順 3 でフィールドを作成したのと同じ順序なので、どのようなフィールドであるかを推測することができます。

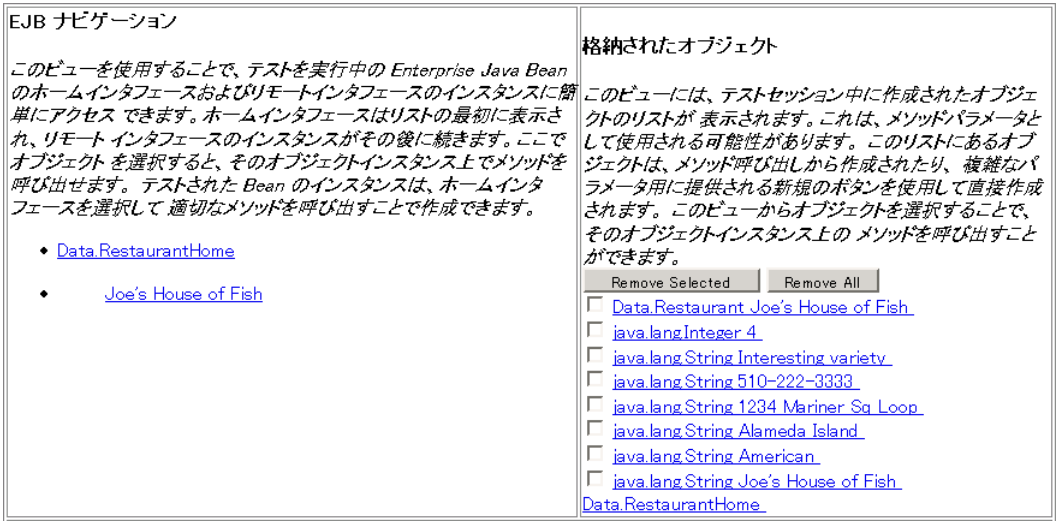
注 – エクスプローラで Restaurant.create メソッドをダブルクリックすると、ソースエディタにそのメソッドが表示されます。フィールドの順序は、メソッドの定義にあるとおりです。

参考 – パラメータを別の順序で表示する場合は、エクスプローラで「Restaurant.create」メソッドノードを右クリックし、「カスタマイズ」を選択します。「カスタマイザ」ダイアログで「上へ」ボタンと「下へ」ボタンをクリックして選択し、パラメータの順序を調整します。その後で、エクスプローラでノードを右クリックして「配備」を選択し、テストアプリケーションを再配備します。

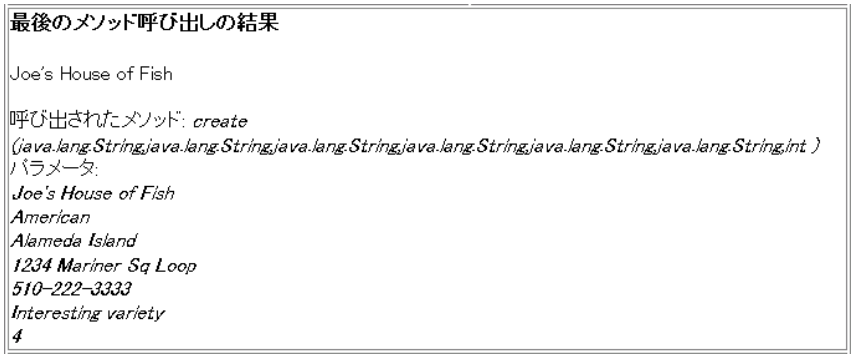
各フィールドに任意のデータを入力します。次に例を示します (フィールドの順序は異なる場合があります)。

2. メソッドの横の「Invoke」ボタンをクリックします。

配備されたテストアプリケーションによって、作成したレコードがテストデータベースに追加されます。以下に示すように、追加された Restaurant インスタンスの restaurantname 値が左上に、新しいデータオブジェクトが右上に示されます。



結果は、結果領域に表示されます。



3. 「Invoke」ボタンをクリックして、 Restaurant Bean の findAll メソッドをテストします。

   結果領域には、次のように表示されます。

ここでは、3 つの内容が返されます。これは、手順 2 で作成した新しいデータベースレコードが、第 1 章で作成した 2 つのデータベース表に追加されたことを表しています。

4. **Bay Fox** と入力してメソッドの「Invoke」ボタンをクリックし、findByPrimaryKey メソッドをテストします。

   結果領域を見ると、Bay Fox レコードが返されたことが分かります。

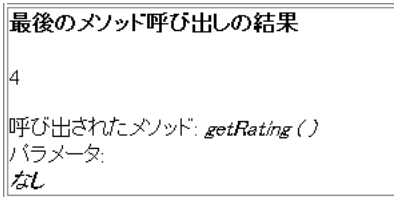
   次に、エンティティ Bean のビジネスメソッドをテストします。

5. 左上のインスタンス一覧で Data.RestaurantHome の下に表示されている「Joe's House of Fish」をクリックします。

   これで「Joe's House of Fish でメソッドを呼び出す」領域の下に getRating メソッドが示されます。

6. getRating メソッド横の「Invoke」ボタンをクリックします。

   この操作の結果は、結果領域に次のように表示されます。



これは、データベースに新しいレコードが作成され、getRating メソッドを使用してフィールドのいずれかの値が取得されたことを表しています。

作成されたオブジェクトを選択し、そのメソッドを呼び出してテストを続行します。たとえば、Data.RestaurantDetail のオブジェクトの 1 つを選択すると、取得メソッドを呼び出してそのデータを表示したり、あるいは設定メソッドを呼び出して、新しいデータをデータベースに書き込んだりすることができます。

7. テストが終了したら、Web ブラウザで別の URL を指定するかブラウザを終了し、テストクライアントを終了します。

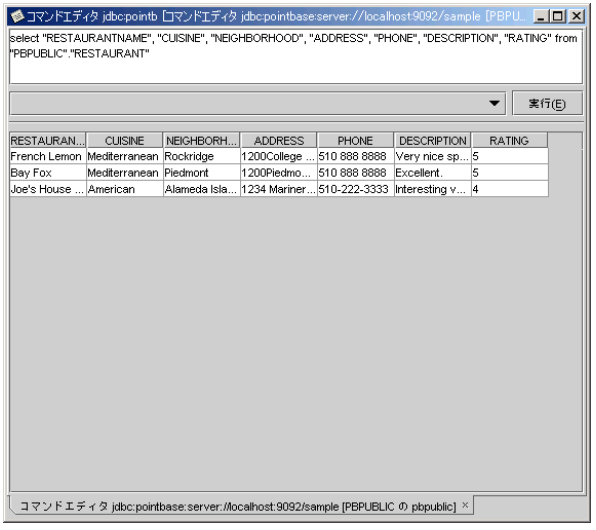
データベースへの追加内容の確認

Restaurant_TestApp アプリケーションによってデータベースにデータが追加されたことを確認するには、以下の操作を行います。

1. IDE のエクスプローラの「実行時」タブを選択します。
2. 「データベース」、「PointBase 接続」、「表」の順にノードを展開します。

   サンプルデータベース内の各テーブルのノードが表示されます。
3. 「RESTAURANT」表を右クリックして、「データを表示」を選択します。

   コマンドエディタウィンドウに、62 ページの「テストクライアントを使用した Restaurant Bean のテスト」の手順 1 で入力した新しい Restaurant レコードが表示されます。



確認を終えたら、セッション Bean の作成に進みます。

注 – アプリケーションサーバーのプロセス (「実行」ウィンドウに示されるプロセス) を停止する必要はありません。再配備のたびに、サーバーはアプリケーションの配備を元に戻し、再配備するからです。IDE が終了すると、アプリケーションサーバーインスタンスのプロセスの終了を告げるダイアログが表示されます。必要ならば、「実行」ウィンドウで「`server1` (*hostname*:*host-port*)」ノードを右クリックし、「プロセスを終了」を選択することによって、いつでも手動でプロセスを終了することもできます。

Customerreview Bean のテストクライアントの作成

Customerreview エンティティ Bean のテストクライアントを作成するには、56 ページの「Restaurant Bean のテストクライアントの作成」と同じ手順を使用します。ただし、使用する値は、Customerreview Bean に適切な値に置き換えます。

テストクライアントアプリケーションを作成する手順の概要は、次のとおりです。

1. 「Customerreview(EJB)」論理 Bean を選択して、「新規 EJB テストアプリケーションを作成」を選択します。
2. すべてのデフォルト値をそのまま受け入れて、「了解」をクリックします。

59 ページの「Sun ONE Application Server 7 プラグインへのデータベース情報の指定」の手順に従って、プラグインにデータベース情報を追加します。ただし、使用する値は、Customerreview Bean に適切な値に置き換えます。

プラグインにデータベース情報を追加する手順の概要は、次のとおりです。

1. EJB モジュール (Customerreview_EJBModule) を展開し、その下の「Customerreview」ノードを選択して、そのプロパティを表示します。
2. 「プロパティ」ウィンドウの「Sun ONE AS」タブを選択します。
3. 次のプロパティの値を確認します。

プロパティ	値
マップされたフィールド	3 コンテナ管理のマップされたフィールド
マップされた主表	CUSTOMERREVIEW
マップされたスキーマ	dgSchema

上記の値が表示されている場合は、手順 5 に進みます。

4. 上記の値が表示されない場合は、次の手順で Customerreview Bean を再びマップします。
   a. 「マップされたスキーマ」を `Data/dgSchema` に設定します。
   b. 「マップされた主表」を `CUSTOMERREVIEW` に設定します。
   c. 「マップされたフィールド」プロパティの値フィールドの省略符号ボタン (...) をクリックします。
   d. ウィザードで、「次へ」をクリックして、「表を選択」ページを表示します。
   e. 「主表」フィールドのドロップリストから「`CUSTOMERREVIEW`」を選択します。

f. 「次へ」をクリックして、「フィールドマッピング」ページを表示します。

g. フィールドがマップされていない場合は、「自動マップ」ボタンをクリックします。

h. 「完了」をクリックします。

5. EJB モジュール (Customerreview_EJBModule) のプロパティを表示します。
6. 「プロパティ」ウィンドウの「Sun ONE AS」タブを選択します。
7. 「CMP リソース」プロパティの値フィールドをクリックして、省略符号ボタン (...) をクリックします。
8. 「CMP リソース」プロパティのプロパティエディタで、「JNDI 名」フィールドに **`jdo/PointBasePM`** と入力します。
9. 「名前」フィールドと「パスワード」フィールドに、それぞれデータベースのユーザー名とパスワードを入力します。

   PointBase サンプルデータベースの場合は、これはともに PBPUBLIC です。
10. 「了解」をクリックして値を適用し、プロパティエディタを閉じます。

Customerreview Bean のテストアプリケーションの配備と実行

注 – テストアプリケーションなど、データベースにアクセスする J2EE アプリケーションを配備する前に、PointBase サーバーが実行されていることを確認してください。また、 Sun ONE Application Server 7 サーバーが稼働していて、IDE のデフォルトのアプリケーションサーバーになっていることを確認してください。9 ページの「Sun ONE Application Server 7 がデフォルトサーバーであることの確認」を参照してください。

Customerreview テストアプリケーションを配備するには、次の操作を行います。

- **「Customerreview_TestApp」J2EE アプリケーションノードを右クリックして、コンテキストメニューから「実行」を選択します。**

  「進捗モニター」ウィンドウに、配備プロセスの進捗状況が示されます。出力ウィンドウ上のサーバーインスタンスのログファイルのタブに、進捗メッセージが表示されます。アプリケーションが正常に配備されると、完了のメッセージが表示されます。

  IDE がデフォルトの Web ブラウザを起動し、テストアプリケーションのホーム URL を表示します。

  `http://`*server-host*`:`*server-port-number*`/Customerreview_TestApp/dispatch.jsp`

注 – ブラウザが見えない場合は、IDE のウィンドウの背後に隠れている可能性があります。出力ウィンドウにあるサーバーインスタンスのログファイルのタブを確認すると、アプリケーションが配備されたかどうかがわかります。アプリケーションが正常に配備されると、完了のメッセージが表示されます。

Customerreview エンティティ Bean のテスト

テストクライアントの Web ページが表示されたら、Customerreview Bean のホームインタフェースの生成メソッドを使用して、Bean のインスタンスを作成します。次に、そのインスタンスに対してビジネスメソッド (この場合は getCustomerreview) をテストします。

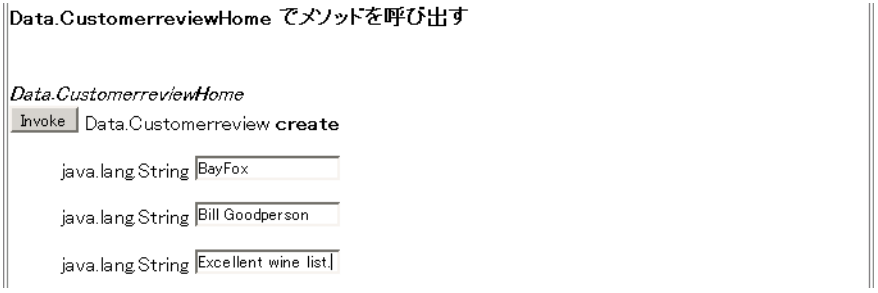
Customerreview Bean をテストするには、次の操作を行います。

1. **生成メソッドを呼び出して、Customerreview Bean のインスタンスを作成します。**

   生成メソッドは、「Data.CustomerreviewHome でメソッドを呼び出す」という見出しの下にあり、その下に 3 つのフィールドがあります。

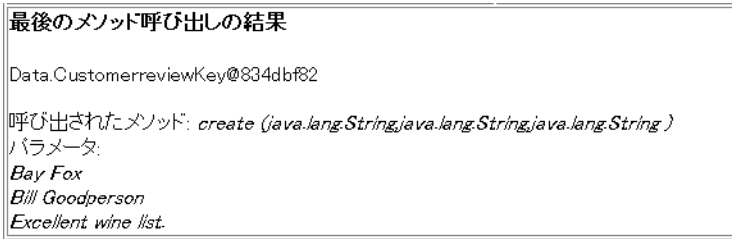
2. **3 つのフィールドに値を入力します。**

   各フィールドに任意のデータを入力します。次に例を示します (フィールドの順序は異なる場合があります)。



3. **生成メソッドの横の「Invoke」ボタンをクリックします。**

   配備されたテストアプリケーションによって、作成したレコードがテストデータベースに追加されます。以下に示すように、追加された Restaurant インスタンスの restaurantname 値が左上に、新しいデータオブジェクトが右上に示されます。



4. findByRestaurantName メソッドのフィールドに **`French Lemon`** と入力して、「Invoke」ボタンをクリックします。

   結果は次のようになります。French Lemon レコードが返されています。

5. 左上の「EJB ナビゲーション」区画で、CustomerreviewKey インスタンスを選択します。

6. インスタンスの getReview メソッドを検索して、その「Invoke」ボタンをクリックします。

   結果には、たとえば次のように、手順 2 と手順 3 で作成したインスタンスの利用者コメントが表示されます。

   作成されたオブジェクトを選択し、そのメソッドを呼び出すことで、テストを継続します。

7. テストが終了したら、ブラウザを終了したり別の URL を指定したりできます。また、何もしなくてもかまいません。

データベースへの追加内容の確認

66 ページの「データベースへの追加内容の確認」で Restaurant_TestApp をテストしたように、Customerreview_TestApp アプリケーションによってデータベースにデータが挿入されたことを確認します。

1. IDE のエクスプローラの「実行時」タブを選択します。
2. 「Databases」、PointBase 接続ノード、「Tables」の順にノードを展開します。
3. 「CUSTOMERREVIEW」表を右クリックして、「データを表示」を選択します。

   コマンドエディタに新しいコメントレコードが表示されます。

EJB ビルダーによるセッション Bean の作成

ここでは、クライアント (第 4 章で作成する Web サービスのこと) とエンティティ Bean 間の会話を管理するステートレスセッション Bean を作成します。

注 – 付録 A に、完成したセッション Bean のソースコードがあります。

EJB アーキテクチャのバージョン 2.0 では、セッション Bean はローカルインタフェース、またはリモートインタフェース、あるいはその両方を持つことができます。このチュートリアルでは、セッション Bean のメソッドは、テストアプリケーション (セッション Bean にローカル)、Web サービス (ローカル)、クライアント (リモート) によって呼び出されます。このため、ローカルとリモートの両方のインタフェースを持つセッション Bean を作成します。

1. Sun ONE Studio 5 エクスプローラで「Data」パッケージを右クリックして、「新規」→「すべてのテンプレート」を選択します。

   新規ウィザードの「テンプレートを選択」ページが表示されます。
2. 「J2EE」ノードを展開して「セッション EJB」を選択します。
3. 「次へ」をクリックします。

   「セッション Bean 名とプロパティ」ページが表示されます。

i. 「名前」フィールドに **DiningGuideManager** と入力して、次のオプションを選択します。

オプションの分類	選択するオプション
状態	ステートレス
トランザクション型	コンテナ管理
コンポーネントインタフェース	リモートインタフェースとローカルインタフェースの両方

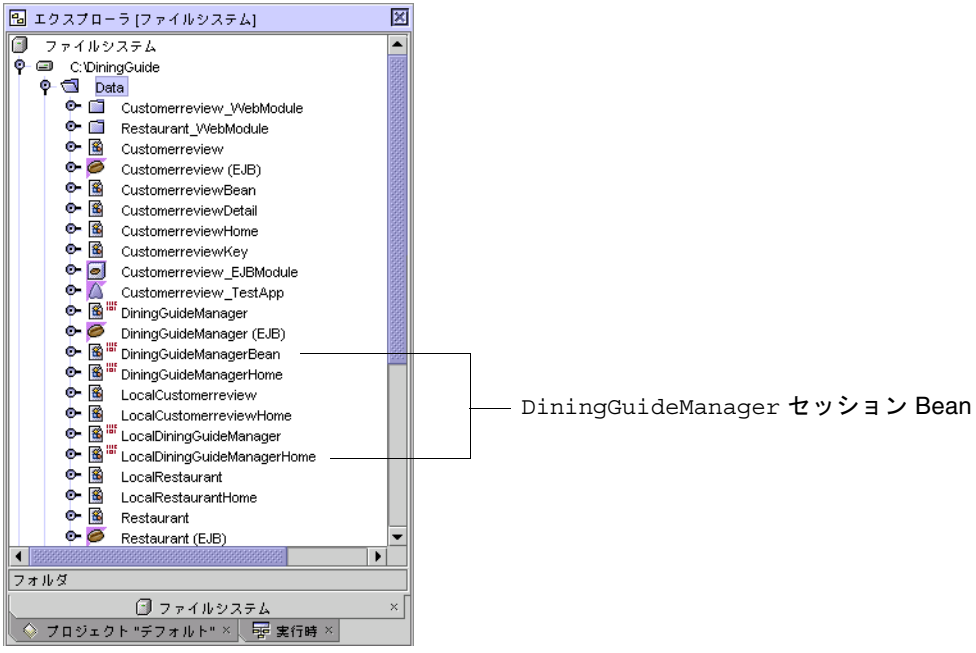
4. 「次へ」をクリックします。

   このセッション Bean について作成されるすべてのコンポーネントの一覧が入った「セッション Bean クラスファイル」ページが表示されます。

   すべてのコンポーネントに DiningGuideManager に基づく名前が付けられていることに注意してください。

5. 「完了」をクリックします。

   エクスプローラに DiningGuideManager セッション Bean が表示されます。



続いて、このセッション Bean のメソッドを作成します。

セッション Bean の生成メソッドのコーディング

生成メソッドは、DiningGuideManager セッション Bean を作成したときにすでに作成されています。ここで変更します。

ステートレスセッション Bean の生成メソッドには、初期化が必要な状態を持たないため引数はありません。DiningGuideManager セッション Bean の生成メソッドは最初に初期コンテキストを作成し、そのコンテキストを使用して必要なリモート参照を取得します。

1. 論理 Bean の「生成メソッド」フォルダ内にある DiningGuideManager の生成メソッドを見つけてダブルクリックし、ソースエディタで表示します。
2. RestaurantHome インタフェースへのリモート参照用の JNDI ルックアップを使用してメソッドのコーディングを開始します。

```
public void ejbCreate() {
//ソースエディタで次の 2 行を 1 行にする
    System.out.println("Entering
DiningGuideManagerEJB.ejbCreate()");
    Context c = null;
    Object result = null;

    if (this.myRestaurantHome == null) {
        try {
            c = new InitialContext();
            result = c.lookup("java:comp/env/ejb/Restaurant");
            myRestaurantHome =
(RestaurantHome)javax.rmi.PortableRemoteObject.narrow(result,
RestaurantHome.class);
        }
        catch (Exception e) {System.out.println("Error: "+ e); }
    }
```

参考 – ソースエディタに入力したコードは、その部分を選択して Ctrl+Shift F を押すと、書式を整えることができます。また、コード内にコメントがある個所では、コメントに従って改行を削除する必要があります。

---

3. 前のコードの下に CustomerreviewHome 用の同様の JNDI ルックアップを追加します。

```
    Context crc = null;
    Object crresult = null;

    if (this.myCustomerreviewHome == null) {
        try {
            crc = new InitialContext();
            result =
crc.lookup("java:comp/env/ejb/Customerreview");
            myCustomerreviewHome =
(CustomerreviewHome)javax.rmi.PortableRemoteObject.narrow(result
, CustomerreviewHome.class);
        }
        catch (Exception e) {System.out.println("Error: "+ e); }
    }
```

4. javax.naming パッケージのインポート文を追加します。

   ファイルの先頭にインポート文を追加してください。javax.naming をインポートするのは、すぐ前に使用したルックアップメソッドが含まれているためです。

```
import javax.naming.*;
```

5. myRestaurantHome フィールドと myCustomerreviewHome フィールドを宣言します。

   DiningGuideManagerEJB セッション Bean の定義の import 文の後に、これらの宣言を追加します。

```
public class DiningGuideManagerBean implements
javax.ejb.SessionBean {
    private javax.ejb.SessionContext Context;
    private RestaurantHome myRestaurantHome;
    private CustomerreviewHome myCustomerreviewHome;
```

6. 「DiningGuideManager(EJB)」論理ノードを選択し、F9 を押して Bean をコンパイルします。

   続いて、DiningGuideManager のビジネスメソッドを作成します。

詳細データを取得するビジネスメソッドの作成

DiningGuideManager Bean には、クライアントからレストラン一覧の要求を受け取ったときにすべてのレストランデータを取り出すメソッドが必要です。同様に、クライアントから利用者コメント一覧の要求があったときに特定のレストランに関するコメントデータを取り出すためのメソッドも必要です。ここでは、これらの要求を満たす getAllRestaurants とgetCustomerreviewsByRestaurant というメソッドを作成します。

getAllRestaurants メソッドの作成

getAllRestaurants ビジネスメソッドを作成するには、以下の操作を行います。

1. 「DiningGuideManager」論理ノードを右クリックして、「ビジネスメソッドを追加」を選択します。

   「新規ビジネスメソッドを追加」ダイアログが表示されます。

2. 「名前」フィールドに**`getAllRestaurants`** と入力します。

3. 「戻り値の型」フィールドに **`java.util.Vector`** と入力します。

4. 「リモートインタフェースとローカルインタフェースの両方」オプションが有効であることを確認します。

   このメソッドは、リモートとローカル両方のインタフェースに追加されます。

5. 他のすべての値はデフォルトのままで、「了解」をクリックします。

   DiningGuideManager セッション Bean のビジネスメソッドにメソッドが作成されます。

6. ソースエディタでメソッドを開き、そのソースコードに次の太字部分を追加します。

```
public java.util.Vector getAllRestaurants() {
//ソースエディタで次の 2 行を 1 行にする
    System.out.println("Entering
DiningGuideManagerEJB.getAllRestaurants()");
    java.util.Vector restaurantList = new java.util.Vector();
    try {
        java.util.Collection rl = myRestaurantHome.findAll();
        if (rl == null) { restaurantList = null; }
        else {
            RestaurantDetail rd;
            java.util.Iterator rli = rl.iterator();
            while ( rli.hasNext() ) {
                rd
=((Restaurant)rli.next()).getRestaurantDetail();
                System.out.println(rd.getRestaurantname());
                System.out.println(rd.getRating());
                restaurantList.addElement(rd);
            }
        }
    }
    catch (Exception e) {
//ソースエディタで次の 2 行を 1 行にする
        System.out.println("Error in
DiningGuideManagerEJB.getAllRestaurants(): " + e);
    }
//ソースエディタで次の 2 行を 1 行にする
    System.out.println("Leaving
DiningGuideManagerEJB.getAllRestaurants()");
    return restaurantList;
}
```

注 – コードのコメントで指示しているように、必ず行の区切りを削除してください。

このコードは、コンテキスト内の Restaurant Bean のリモート参照ごとに RestaurantDetail のインスタンスを 1 つ取得し、そのインスタンスを restaurantList というベクトルに追加して、そのベクトルを返します。

7. 「DiningGuideManager(EJB)」論理ノードを選択し、F9 を押して Bean をコンパイルします。

続いて、利用者コメントの一覧を取得する同様のメソッドを作成します。

getCustomerreviewsByRestaurant メソッドの作成

getCustomerreviewsByRestaurant メソッドを作成するには、以下の操作を行います。

1. 「DiningGuideManager」論理ノードを右クリックして、「ビジネスメソッドを追加」を選択します。

   「新規ビジネスメソッドを追加」ダイアログが表示されます。

2. 「名前」フィールドに **`getCustomerreviewsByRestaurant`** と入力します。
3. 「戻り値の型」フィールドに **`java.util.Vector`** と入力します。
4. 「追加」ボタンをクリックし、次の値を持つパラメータを追加します。

パラメータ名	型
restaurantname	java.lang.String

5. 「了解」をクリックしてダイアログを閉じ、メソッドパラメータを作成します。
6. 「リモートインタフェースとローカルインタフェースの両方」オプションが有効であることを確認します。
7. 他のすべての値はデフォルトのままで、再び「了解」をクリックしてビジネスメソッドを作成します。

   `DiningGuideManager` セッション Bean にメソッドが作成されます。

8. ソースエディタでメソッドを開き、そのソースコードに次の太字部分を追加します。

```
public java.util.Vector
getCustomerreviewsByRestaurant(java.lang.String
        restaurantname) {
//ソースエディタで次の 2 行を 1 行にする
    System.out.println("Entering
DiningGuideManagerEJB.getCustomerreviewsByRestaurant()");
    java.util.Vector reviewList = new java.util.Vector();
    try {
        java.util.Collection rl =
myCustomerreviewHome.findByRestaurantName(restaurantname);
        if (rl == null) { reviewList = null; }
        else {
            CustomerreviewDetail crd;
            java.util.Iterator rli = rl.iterator();
            while ( rli.hasNext() ) {
                crd =
((Customerreview)rli.next()).getCustomerreviewDetail();
                System.out.println(crd.getRestaurantname());
                System.out.println(crd.getCustomername());
                System.out.println(crd.getReview());
                reviewList.addElement(crd);
            }
        }
    }
    catch (Exception e) {
//ソースエディタで次の 2 行を 1 行にする
        System.out.println("Error in
DiningGuideManagerEJB.getCustomerreviewsByRestaurant(): " + e);
    }
//ソースエディタで次の 2 行を 1 行にする
    System.out.println("Leaving
DiningGuideManagerEJB.getCustomerreviewsByRestaurant()");
    return reviewList;
}
```

注 – コードのコメントで指示しているように、必ず行の区切りを削除してください。

getAllRestaurants メソッド同様、このメソッドは、コンテキスト内の Customerreview Bean のリモート参照ごとに CustomerreviewDetail のインスタンスを 1 つ取得し、そのインスタンスを reviewList というベクトルに追加して、そのベクトルを返します。

9. 「DiningGuideManager(EJB)」論理ノードを選択し、F9 を押して Bean をコンパイルします。

利用者コメントレコードを作成するビジネスメソッドの作成

ここでは、Customerreview エンティティ Bean の生成メソッドを呼び出して、データベースの新しいレコードを作成するビジネスメソッドを作成します。

createCustomerreview メソッドを作成するには、以下の操作を行います。

1. 「DiningGuideManager」論理ノードを右クリックして、「ビジネスメソッドを追加」を選択します。
   「新規ビジネスメソッドを追加」ダイアログが表示されます。
2. 「名前」フィールドに**`createCustomerreview`** と入力します。
3. 「戻り値の型」フィールドに **`void`** と入力します。
4. 「追加」ボタンをクリックし、次の値を持つ 3 つのパラメータを追加します。

パラメータ名	型
restaurantname	java.lang.String
customername	java.lang.String
review	java.lang.String

5. 「了解」をクリックして「パラメータを追加」ダイアログを閉じます。
6. 「リモートインタフェースとローカルインタフェースの両方」オプションが有効であることを確認します。
7. 再び「了解」をクリックしてビジネスメソッドを作成します。
   DiningGuideManager セッション Bean にメソッドが作成されます。

8. ソースエディタでメソッドを開き、そのソースコードに次の太字部分を追加します。

```
public void createCustomerreview(java.lang.String restaurantname,
java.lang.String customername, java.lang.String review) {
//ソースエディタで次の 2 行を 1 行にする
    System.out.println("Entering
DiningGuideManagerEJB.createCustomerreview()");
    try {
        Customerreview customerrev =
myCustomerreviewHome.create(restaurantname, customername,
review);
    } catch (Exception e) {
//ソースエディタで次の 2 行を 1 行にする
        System.out.println("Error in
DiningGuideManagerEJB.createCustomerreview(): " + e);
    }
//ソースエディタで次の 2 行を 1 行にする
    System.out.println("Leaving
DiningGuideManagerEJB.createCustomerreview()");
}
```

注 – コード内にコメントがある 3 個所では、コメントに従って改行を削除し、1 行にする必要があります。

---

9. 「DiningGuideManager(EJB)」論理ノードを選択し、F9 を押して Bean をコンパイルします。

詳細クラス型を返すビジネスメソッドの作成

第 4 章で作成する Web サービス は SOAP RPC Web サービスの JAX-PRC 実装です。この SOAP (Simple Object Access Protocol) というプロトコルは抽象メッセージ技術であり、Web サービス同士が HTTP と XML を使用して互いに通信することを可能にします。Java の型を XML に正しくマッピングするには、SOAP 実行環境は、Web サービスによって呼び出されるすべてのメソッドが使用する、Java のあらゆる型の情報を持っている必要があります。つまり、DigingGuide アプリケーションでも、 Web サービスはセッション Bean のメソッドを呼び出しますから、Web サービスはそれらのメソッドが使用するあらゆる型の情報を持っている必要があります。

SOAP 実行環境が情報を持つことができない型に、コレクションを構成するオブジェクトの型があります。この章で少し前に作成したメソッド (getAllRestaurants と getCustomerreviewsByRestaurant) はすべて詳細クラスのコレクションを返します。このため、詳細クラスごとにクラスを返すメソッドを作成することによって、SOAP

実行環境にそれらクラスに関する情報を提供する必要があります。ここでは、このためのメソッドとして getRestaurantDetail および getCustomerreviewDetail を作成します。

ここでは、エンティティ Bean に作成したメソッドと同じ名前を持つメソッドを作成しますが (55 ページの「エンティティ Bean に対する、詳細クラスをフェッチするビジネスメソッドの作成」)、ここで作成するメソッドは空で、その目的は、単に SOAP 実行時に必要な戻り値の型を提供することにあります。

Sun ONE Studio 5 Web サービスおよび SOAP 実行環境についての詳細は、Sun ONE Studio 5 プログラミングシリーズの『Web サービスのプログラミング』を参照してください。

getRestaurantDetail メソッドの作成

getRestaurantDetail メソッドを作成するには、以下の操作を行います。

1. 「DiningGuideManager」論理ノードを右クリックして、「ビジネスメソッドを追加」を選択します。

   「新規ビジネスメソッドを追加」ダイアログが表示されます。

2. 「名前」フィールドに **`getRestaurantDetail`** と入力します。

3. 戻り値の型では、「ブラウズ」ボタンを使用して、RestaurantDetail クラスを選択します。

   必ず Bean のノードではなく、クラス () を選択します。「戻り値の型」フィールドに Data.RestaurantDetail と表示されます。

4. 「リモートインタフェースとローカルインタフェースの両方」オプションが有効であることを確認します。

5. 他のすべての値はデフォルトのままで、「了解」をクリックしてビジネスメソッドを作成し、ダイアログを閉じます。

   DiningGuideManager セッション Bean にメソッドが作成されます。

6. ソースエディタでメソッドを開き、そのソースコードに次の太字部分を追加します。

```
public Data.RestaurantDetail getRestaurantDetail() {
    return null;
}
```

getCustomerreviewDetail メソッドの作成

getCustomerreviewDetail メソッドを作成するには、以下の操作を行います。

1. 「DiningGuideManager」論理ノードを右クリックして、「ビジネスメソッドを追加」を選択します。

   「新規ビジネスメソッドを追加」ダイアログが表示されます。

2. 「名前」フィールドに **`getCustomerreviewDetail`** と入力します。

3. 「戻り値の型」では、「ブラウズ」ボタンを使用して、「`CustomerreviewDetail`」クラスを選択します。

   「戻り値の型」フィールドに Data.CustomerreviewDetail と表示されます。

4. 「リモートインタフェースとローカルインタフェースの両方」オプションが有効であることを確認します。

5. 「了解」をクリックしてビジネスメソッドを作成し、ダイアログを閉じます。

   DiningGuideManager セッション Bean にメソッドが作成されます。

6. ソースエディタでメソッドを開き、そのソースコードに次の太字部分を追加します。

```
public Data.CustomerreviewDetail getCustomerreviewDetail() {
    return null;
}
```

7. 「DiningGuideManager(EJB)」 を右クリックして、「EJB の検査」を選択します。

   DiningGuideManager セッション Bean が検査されます。

EJB 参照の追加

セッション Bean を配備するとき、その Bean のプロパティには、自身が呼び出すあらゆるエンティティ Bean のメソッドに対する参照が含まれている必要があります。ここでは、セッション Bean に対する参照の追加を行います。EJB モジュールにセッション Bean をアセンブルした後で追加することはできません。

1. エクスプローラで「DiningGuideManager(EJB)」論理ノードを選択します。

2. Bean のプロパティシートを表示します。

   「プロパティ」ウィンドウが表示されていない場合は、「表示」→「プロパティ」を選択します。

3. 「プロパティ」ウィンドウの「参照」タブを選択します。

4. 「EJB 参照」フィールドをクリックして、省略符号ボタン (...) をクリックします。

   「EJB 参照」プロパティエディタが表示されます。

5. 「追加」ボタンをクリックします。

   「EJB 参照を追加」プロパティエディタが表示されます。

6. 「参照名」フィールドに **`ejb/Restaurant`** と入力します。

7. 「参照される EJB 名」フィールドでは、「ブラウズ」ボタンをクリックします。

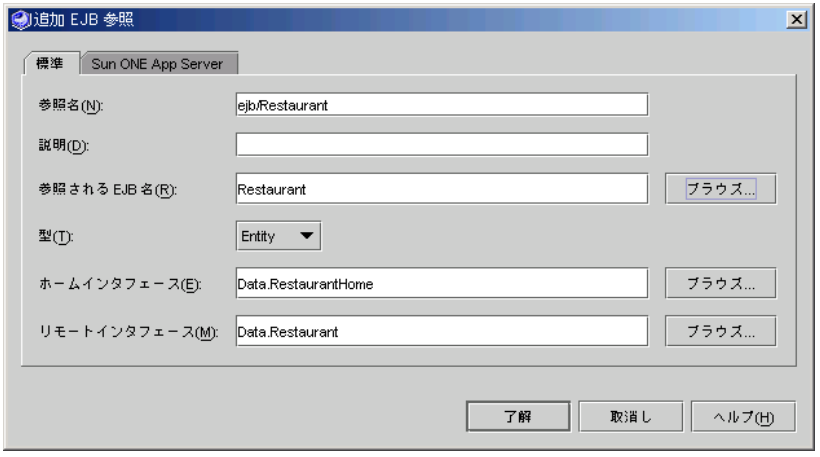
   「EJB を選択」ブラウザが表示されます。

8. 「DiningGuide/Data」ノードの下にある Restaurant (EJB) Bean を選択して、「了解」をクリックします。

   「ホームインタフェース」と「リモートインタフェース」フィールドの両方に自動的にデータが入力されることに注意してください。

9. 「型」フィールドを「`Entity`」に設定します。

   「EJB 参照を追加」プロパティエディタの表示は以下のようになります。



10. 「Sun ONE App Server」タブを選択します。

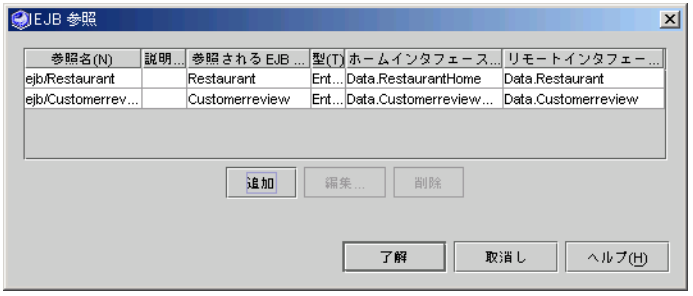
11. 「JNDI 名」フィールドに **`ejb/Restaurant`** と入力して、「了解」をクリックします。

    これは、Restaurant Bean の作成時に割り当てられたデフォルトの JNDI 名です。

12. 同様に Customerreview エンティティ Bean への参照を追加して、プロパティが次の値を持つようにします。

タブ	プロパティ	値
標準	参照名	ejb/Customerreview
	参照される EJB 名	Customerreview
	型	エンティティ
	ホームインタフェース	Data.CustomerreviewHome
	リモートインタフェース	Data.Customerreview
Sun ONE App Server	JNDI 名	ebj/Customerreview

「EJB 参照」ダイアログは次のような表示になります。



13. 「了解」を選択して「プロパティエディタ」ウィンドウを閉じます。

これでチュートリアルアプリケーションの EJB 層が完成し、テストできる状態になりました。エンティティ Bean のテストのとき同様、クライアントがブラウザで読み取り可能な Web 層と JSP ページは、IDE のテストアプリケーション機能によって自動的に作成されます。

セッション Bean のテスト

ここでは、IDE のテストアプリケーション機能を使用して、DiningGuideManager セッション Bean をテストします。このセッション Bean のメソッドは EJB 層のすべてのコンポーネントのメソッドへのアクセスを可能にするため、EJB 層全体がテストされることになります。

セッション Bean のテストクライアントの作成

ここでは、DiningGuideManager Bean からテストアプリケーションを作成し、EJB モジュールに 2 つのエンティティ Bean を追加します。

セッション Bean 用のテストクライアントを作成するには、次の操作を行います。

1. 「DiningGuideManager」論理ノードを右クリックして、「新規 EJB テストアプリケーションを作成」を選択します。

   EJB テストアプリケーションウィザードが表示されます。

2. すべてのデフォルト値をそのまま受け入れて、「了解」をクリックします。

   進捗モニターが少しの間表示され、処理が完了すると消えます。別のウィンドウが表示され、作成された Web モジュールが「プロジェクト」タブでも見えることを示して自動的に消えます。消えない場合は、「了解」をクリックしてウィンドウを閉じてください。

3. 生成されたテストオブジェクトをエクスプローラで確認します。

   IDE によって次のオブジェクトが作成されています。

   - EJB モジュール (DiningGuideManager_EJBModule)
   - Web モジュール (DiningGuideManager_WebModule)
   - J2EE アプリケーション (DiningGuideManager_TestApp))

   EJB モジュールと Web モジュールは、Data パッケージのサブノードとして、また J2EE アプリケーションに含まれるモジュールとして表示されます。また、Web モジュールは別にマウントされています。

EJB モジュールへのエンティティ Bean 参照の追加

EJB モジュールには DiningGuideManager Bean しか含まれていないため、2 つのエンティティ Bean を追加する必要があります。

1. 「DiningGuideManager_EJBModule」 を右クリックして、「EJB を追加」を選択します。

   「EJB を EJB モジュールに追加」ブラウザが表示されます。

2. DiningGuide ファイルシステムを展開して、「Data」パッケージを開きます。

3. Control を押しながら、Restaurant および Customerreview 論理 Bean を順にクリックし、両方の Bean を選択します。

4. 「了解」をクリックします。

   IDE は、EJB モジュールとテストアプリケーション内のその参照に 2 つのエンティティ Bean への参照を追加します。DiningGuideManager_EJBModule は以下のような表示になります。

5. 「ファイル」→「すべてを保存」を選択します。

Sun ONE Application Server 7 プラグインへのデータベース情報の指定

EJB モジュールの Sun ONE Application Server 7 のプロパティにデータベース情報を追加する必要があります。この作業は、59 ページの「Sun ONE Application Server 7 プラグインへのデータベース情報の指定」で、エンティティ Bean のテストクライアントについて行った作業と同じです。

必要な情報を追加するには、次の操作を行います。

1. エクスプローラで EJB モジュール (DiningGuideManager_EJBModule) を展開し、その下の「Restaurant」ノード (Restaurant Bean への参照) を選択して、そのプロパティを表示します。

   「プロパティ」ウィンドウが表示されていない場合は、「表示」→「プロパティ」を選択します。

2. 「プロパティ」ウィンドウの「Sun ONE AS」タブを選択します。

注 – 「プロパティ」ウィンドウに「Sun ONE AS」タブがない場合、サーバーレジストリに、Sun ONE Application Server 7 サーバーのインスタンスがないことが考えられます。この問題の解決方法については、9 ページの「Sun ONE Application Server 7 がデフォルトサーバーであることの確認」を参照してください。

---

3. 次のプロパティの値を確認します。

プロパティ	値
マップされたフィールド	7 コンテナ管理のマップされたフィールド
マップされた主表	RESTAURANT
マップされたスキーマ	dgSchema

上記の値が表示されている場合は、手順 5 に進みます。

4. 上記の値が表示されない場合は、次の手順で Restaurant Bean を再びマップします。

   a. 「マップされたスキーマ」を「`Data/dgSchema`」に設定します。

   b. 「マップされた主表」プロパティを「`RESTAURANT`」に設定します。

   c. 「マップされたフィールド」プロパティの値フィールドをクリックして、省略符号ボタン (...) をクリックします。

      データベースへのマップウィザードが表示されます。

   d. 「次へ」をクリックして、「表を選択」ページを表示します。

   e. 「主表」フィールドのドロップリストから「`RESTAURANT`」を選択します。

      一覧の中に「RESTAURANT」がない場合は、「ブラウズ」ボタンを使用して、dgSchema スキーマ内にある表を探します。

   f. 「次へ」をクリックして、「フィールドマッピング」ページを表示します。

   g. フィールドがマップされていない場合は、「自動マップ」ボタンをクリックします。

      各フィールドにマップされる値が表示されます。

   h. 「完了」をクリックします。

      手順 3 に示す値が表示されます。

5. 必要な場合は、Customerreview 参照について手順 1 から手順 4 を繰り返します。

6. 「EJB モジュール (DiningGuideManager_EJBModule)」 を選択して、プロパティを表示します。

7. 「プロパティ」ウィンドウの「Sun ONE AS」タブを選択します。

8. 「CMP リソース」プロパティの値フィールドをクリックして、省略符号ボタン (...) をクリックします。

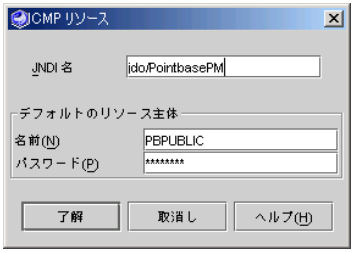
   「CMP リソース」プロパティのプロパティエディタが表示されます。

9. 「JNDI 名」フィールドに **jdo/PointbasePM** と入力します。

   これは、10 ページの「JDBC リソースの設定 (Microsoft Windows のスーパーユーザー)」または11 ページの「JDBC リソースの設定 (administrator 以外のユーザー)」で定義した、JDBC 持続マネージャの JNDI 名です。

10. 「名前」フィールドと「パスワード」フィールドに、それぞれデータベースのユーザー名とパスワードを入力します。

    PointBase サンプルデータベースの場合は、これはともに PBPUBLIC です。エディタの表示は次のようになります。



11. 「了解」をクリックして値を適用し、プロパティエディタを閉じます。

    テストアプリケーションで使用するデータベースを設定する作業が終了しました。これで、テストアプリケーションを配備できます。

12. 「ファイル」→「すべてを保存」を選択して、変更内容を保存します。

テストアプリケーションの配備と実行

エンティティ Bean の 2 つのテストアプリケーションが配備されている場合は、セッション Bean のテストアプリケーションを配備する前に、配備を取り消す必要があります。これは、エンティティ Bean のテストアプリケーションと DiningGuideManager_TestApp アプリケーションでは、使用される Restaurant Bean と Customerreview Bean への JNDI ルックアップが同じであるためです。配備を取り消さなかった場合、DiningGuideManager テストアプリケーションは配備はされますが、読み込まれません。

注 – この節の作業の進む前にアプリケーションサーバーインスタンスが稼働していて、デフォルトのアプリケーションサーバーとして設定されていることを確認しておいてください。

エンティティ Bean のテストアプリケーションの配備取り消し

以前に配備されたアプリケーションの配備を取り消すには、次の操作を行います。

1. エクスプローラの「実行時」タブをクリックして、「実行時」ページを表示します。
2. 「インストールされているサーバー」ノードの「Sun ONE Application Server 7」ノードの下にある「*servername* (*hostname* : *host-number*) 」インスタンスノードを開きます。

   たとえば、デフォルトのサーバーインスタンスは server1(localhost:80) です。実際には、MyServer(localhost:82) という他のラベルのこともあります。
3. アプリケーションサーバーが稼働していることを確認します。

   a. アプリケーションサーバーのノードを右クリックして「状態」を選択します。

      「アプリケーションサーバーインスタンスステータス」ダイアログが表示されます。ステータスが「実行中」の場合は、手順 4 に進みます。

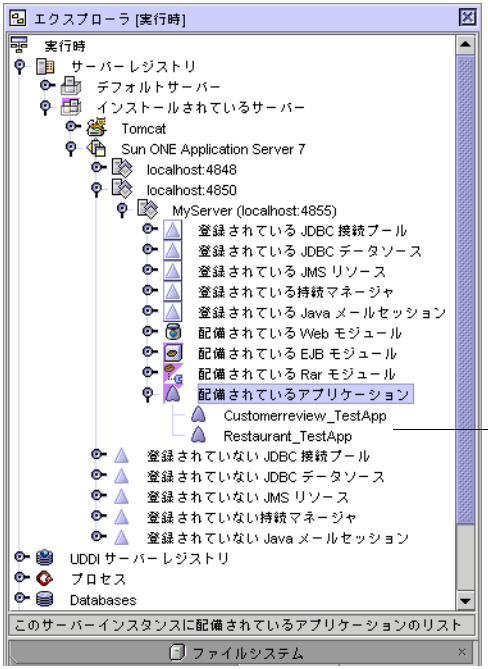
   b. 「サーバーを起動」ボタンをクリックします。

      コマンドウィンドウが開いて、進捗メッセージが表示されます。サーバーが起動すると、ウィンドウに「Application onReady complete」というメッセージが表示されます。

   c. インスタンスステータスのダイアログで「閉じる」をクリックします。

      必要に応じてアイコン化するのはかまいませんが、コマンドウィンドウは閉じないでください。
4. 「配備されているアプリケーション」ノードを展開します。

   エンティティ Bean の 2 つのテストアプリケーションが表示されます。



何も表示されない場合は、「配備されているアプリケーション」ノードを右クリックして「リストを再表示」を選択します。

5. **一方のアプリケーションを右クリックして、「配備の取消し」を選択します。**
   アプリケーションの配備が取り消されます。

6. **手順 5 を繰り返して、他方のアプリケーションの配備を取り消します。**

DiningGuideManager テストアプリケーションの配備

注 – テストアプリケーションなど、データベースにアクセスする J2EE アプリケーションを配備する前に、PointBase サーバーが実行されていることを確認してください。

DiningGuideManager テストアプリケーションを配備するには、次の操作を行います。

1. **エクスプローラの「ファイルシステム」タブをクリックして、「ファイルシステム」ページを表示します。**

2. 「DiningGuideManager_TestApp」J2EE アプリケーションノードを右クリックして、コンテキストメニューから「実行」を選択します。

   「進捗モニター」ウィンドウに、配備プロセスの進捗状況が示されます。出力ウィンドウ上のサーバーインスタンスのログファイルのタブに、進捗メッセージが表示されます。アプリケーションが正常に配備されると、完了のメッセージが表示されます。

   IDE によってデフォルトの Web ブラウザが起動され、テストアプリケーションのホーム URL が表示されます。URL は、アプリケーションサーバーがローカルにインストールされている場合は、
   「`http://localhost/DiningGuideManager_TestApp/`」のようになります。
   サーバーがリモートにインストールされている場合は、異なる URL になります。

テストクライアントによるセッション Bean のテスト

テストクライアントの Web ページで 生成メソッドを実行して DiningGuideManager セッション Bean のインスタンスを作成し、そのインスタンスでビジネスメソッド (getRating) をテストします。

DiningGuideManager Bean をテストするには、以下の操作を行います。

1. DiningGuideManagerHome の生成メソッドを呼び出して、DiningGuideManager セッション Bean のインスタンスを作成します。

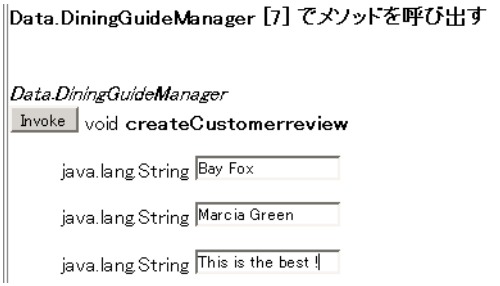
   Data.DiningGuideManager[*x*] インスタンスがインスタンスリストに表示されます。これで、Bean の取得メソッドをテストすることができます。

2. 新しい「Data.DiningGuideManager[*x*] 」をクリックします。

   getAllRestaurants および getCustomerreviewsByRestaurant メソッドが使用できるようになります。

3. createCustomerreview フィールドに任意のデータを入力します。

   次に例を示します。



4. createCustomerreview メソッドの「Invoke」ボタンをクリックします。

   配備されたテストアプリケーションによって、作成したレコードがテストデータベースに追加され、「Stored Objects」セクション (右上) にパラメータ値、結果領域に結果が表示されます。

5. getAllRestaurants メソッドの「Invoke」ボタンをクリックします。

   62 ページの「テストクライアントを使用した Restaurant Bean のテスト」でデータベースに Joe's House of Fish のレコードを作成した場合は、作成されたオブジェクトリスト (右上) にサイズ 3 のベクトルが表示され、メソッド呼び出しの結果が以下のように表示されます (実際の数字は異なるかもしれません。また、レコードを作成しなかった場合は、これとは異なる結果になります)。

6. getCustomerreviewDetail メソッドの「Invoke」ボタンをクリックします。

   結果領域に結果が表示されます。

7. getCustomerreviewsByRestaurant メソッドのフィールドに **`Joe's House of Fish`** と入力して、「Invoke」ボタンをクリックします。

   CustomerreviewDetail レコードは返されません。これは、データベースに利用者のコメントがないためです。今度は French Lemon のレコードで試してみます。

8. 同じフィールドに **`French Lemon`** と入力して、メソッドを呼び出します。

   2 つの CustomerreviewDetail レコードが返されます。

9. getRestaurantDetail メソッドの「Invoke」ボタンをクリックします。

   結果領域に結果が表示されます。

10. テストを終えたら、ブラウザで別の URL を開くか、ブラウザを終了することによって、テストクライアントを停止します。

注 – アプリケーションサーバーのプロセス (「実行」ウィンドウに示されるプロセス) を停止する必要はありません。再配備のたびに、サーバーはアプリケーションの配備を元に戻し、再配備するからです。IDE が終了すると、アプリケーションサーバーインスタンスのプロセスの終了を告げるダイアログが表示されます。必要ならば、「実行」ウィンドウで「`server1` (*hostname*:*host-port*)」ノードを右クリックし、「プロセスを終了」を選択することによって、いつでも手動でプロセスを終了することができます。

データベースへの追加内容の確認

DiningGuideManager_TestApp アプリケーションによって、データベースにデータが追加されたことを確認するには、66 ページの「データベースへの追加内容の確認」および71 ページの「データベースへの追加内容の確認」で説明されている手順を繰り返します。

これで、Web サービスの作成を開始する準備ができました。

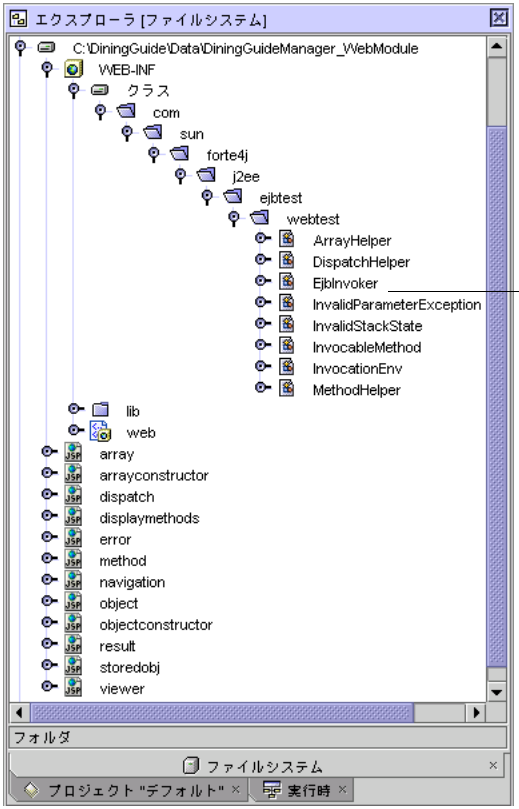
クライアントを作成する際の注意事項

これで、DiningGuide アプリケーションの EJB 層を正しく完成することができました。Sun ONE Studio 5 IDE の Web サービスモジュールを使用したアプリケーションの Web サービスの作成 (第 4 章) と、クライアント用に提供されている Swing クラスのインストール (第 5 章) をすることができます。

独自の Web サービスとクライアントを作成する場合は、Sun ONE Studio 5 テストアプリケーションにあるガイドラインを参照してください。

DiningGuideManager Bean のように、セッション Bean にアクセスする Web サービスには、ルックアップメソッドを持つサーブレットや JSP ページを定義しておく必要があります。これは、EJB 層で、エンティティ Bean のホームインタフェースとホームオブジェクトを取得するためです。テストアプリケーション機能によって作成された Web モジュールには、必要なコードのサンプルが用意されています。

ルックアップメソッドの例は、Web モジュールの下の EjbInvoker クラスにあります。`WEB--INF/Classes/com/sun/forte4j/j2ee/ejbtest/webtest` ディレクトリを参照してください。



たとえば、次のメソッドにルックアップコードの例があります。

- EjbInvoker.getHomeObject
- EjbInvoker.getHomeInterface
- EjbInvoker.resolveEjb

第4章

DigingGuide アプリケーションの Web サービスの作成

この章では、Sun ONE Studio 5 IDE を使用して DiningGuide アプリケーション用の Web サービスを作成する方法を学びます。この章の内容は以下のとおりです。

- この後の Web サービスの概要
- 99 ページの「チュートリアルアプリケーションの Web サービスの作成」
- 103 ページの「Web サービスのテスト」
- 117 ページの「他の開発者が Web サービスを利用できるようにする」

Sun ONE Studio 5 の Web サービス機能についての全容は、Sun ONE Studio 5 プログラミングシリーズの『Web サービスのプログラミング』を参照してください。このマニュアルは、Sun ONE Studio 5 のポータルのドキュメントサイト (`http://sun.co.jp/software/sundev/jde/documentation/`) から入手できます。また、個々の機能については、Sun ONE Studio 5 のオンラインヘルプを参照してください。

Web サービスの概要

この章では、DiningGuide アプリケーションの Web サービスを作成します。この一環として、いくつかのコンポーネントを自分で作成したり、生成したりします。

自分で作成するコンポーネントは以下のとおりです。

- 論理 Web サービス (DGWebService Web サービス)
- J2EE アプリケーション - セッション Bean の EJB モジュールと Web サービスの両方を参照します。

生成するクラスは以下のとおりです。

- 実行時クラス - Web サービスを実装するための EJB コンポーネントです。
- テストクライアント

- テストクライアントファイル

Web サービス

Web サービスとその作成・プログラミング方法についての全容は、『Web サービスのプログラミング』をご覧ください。また、個々の Web サービス項目と手順については、Sun ONE Studio 5 のオンラインヘルプを参照してください。

このチュートリアルでは、クライアントがアクセスできるようにするメソッドに対する参照を Web サービスに作成することによって、Web サービスの機能を実現します。

実行時クラス

Web サービスとそのメソッド参照を作成したら、その実行時クラスを生成します。実行時クラスを直接操作するのではなく、Web サービスを含むパッケージに生成されたものを確認します。

クライアントファイル

クライアントを作成してそのファイルを生成すると、クライアントページが作成されます。Web サービスのテストには、それらのクライアントページを使用します。また、それらのページは、フル機能の参照メソッドをプログラミングする際のスタート地点またはガイドとして利用することもできます。クライアントファイルとしては、各参照メソッドの JSP ページや Web サービス のエラーを表示するための JSP ページ、Web ブラウザでメソッドの JSP ページをまとめて表示するための開始ページなどがあります。

開始ページには、参照されたメソッド用に生成される各 JSP の HTML フォームが 1 つ含まれます。メソッドがパラメータを必要とする場合、HTML フォームにはそれに応じた入力フィールドが含まれます。メソッドのテストでは、必要に応じて各パラメータのデータを入力し、メソッドの「呼び出し」ボタンをクリックします。この結果、次のアクションが発生します。

1. JSP ページが SOAP クライアントファイルに要求を渡します。
2. SOAP クライアントファイルがアプリケーションサーバー上の JAX-RPC 実行環境システムに要求を渡します。

   SOAP 要求は XML ラッパーで、Web サービスに対するメソッド呼び出しとシリアル化された形式の入力データを含みます。

3. アプリケーションサーバー上の JAX-RPC 実行環境システムが SOAP 要求を、DiningGuide Web サービスの参照する適切なメソッドに対するメソッド呼び出しに変換します。
4. メソッド呼び出しが、EJB 層の適切なビジネスメソッドに渡されます。
5. 処理された応答がチェーンをのぼって SOAP クライアントファイルに返されます。
6. SOAP クライアントファイルが応答を JSP ページに渡し、そのページが Web ページに表示されます。

チュートリアルアプリケーションの Web サービスの作成

チュートリアルの Web サービスを作成するには、以下の 2 つの作業を行う必要があります。

1. アプリケーション用の論理 Web サービスの作成。

   IDE の Web サービスウィザードを使用して、論理 Web サービスを作成し、参照するメソッドを指定します。指定するメソッドは、DigingGuideManager セッション Bean 用に作成した 5 つのビジネスメソッドです。

2. Web サービス実行時クラスの生成

   この作業では、Web サービスのテストと実装に使用されるサポート用 EJB コンポーネントを生成します。

論理 Web サービスの作成

ここでは、新規 Web サービスウィザードを使用して、論理 Web サービスを作成します。このウィザードには、アーキテクチャの選択肢として多層と Web 主体の 2 つがあります。DigingGuide アプリケーションの Web サービス は EJB 層のコンポーネント上のメソッドを呼び出すため、アーキテクチャの選択では多層を選択します。

ウィザードではまた、Web サービスが呼び出すメソッドを選択するよう求められます。これに対しては、EJB 層のセッション bean の 5 つのビジネスメソッドを選択します。

チュートリアルの論理 Web サービスを作成するには、以下の操作を行います。

1. エクスプローラでマウントされている DiningGuide ファイルシステムを右クリックして、「新規」→「Java パッケージ」を選択します。

   新規パッケージダイアログが表示されます。

2. 名前として **`WebService`** と入力し、「完了」をクリックします。

   DigingGuide ディレクトリに WebService パッケージが表示されます。

3. 「WebService」パッケージを右クリックして、「新規」→「すべてのテンプレート」を選択します。

   新規ウィザードの「テンプレートを選択」ページが表示されます。

4. 「Web サービス」ノードを展開して「Web サービス」を選択し、「次へ」をクリックします。

   新規ウィザードに「Web サービス」ページが表示されます。

5. 「名前」フィールドに **`DGWebService`** と入力して、次のオプションを選択します。

オプションの分類	選択するオプション
フォーム作成	Java メソッド
アーキテクチャ	多層

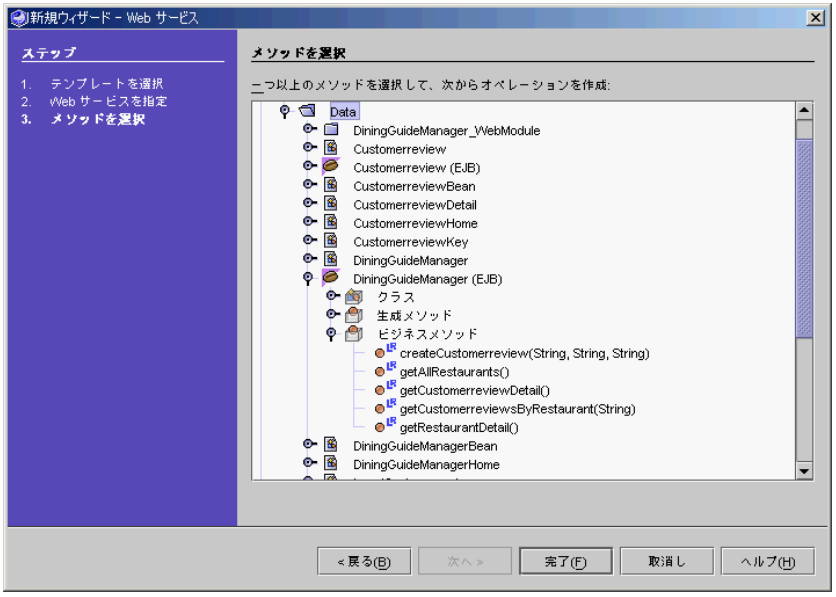
6. 「次へ」をクリックします。

   新規ウィザードに「メソッドを選択」ページが表示されます。

7. 「Data」、「DiningGuideManager(EJB)」、「ビジネスメソッド」の順にノードを展開します。

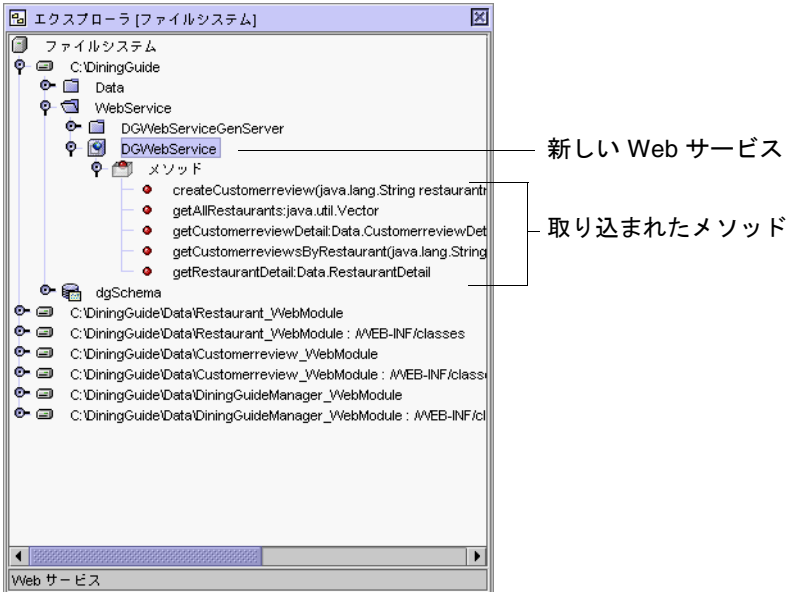
8. Shift キーを押しながら、クリックすることによって、DigingGuideManger のビジネスメソッドをすべて選択します。

   メソッドのページは次のようになります。



9. 「完了」をクリックします。

エクスプローラの WebService パッケージの下に、新しい「DGWebService」Web サービス (青色の球が入ったアイコン) が表示されます。このノードを開くと、エクスプローラの表示が次のようになります。



Web サービスの実行時クラスの生成

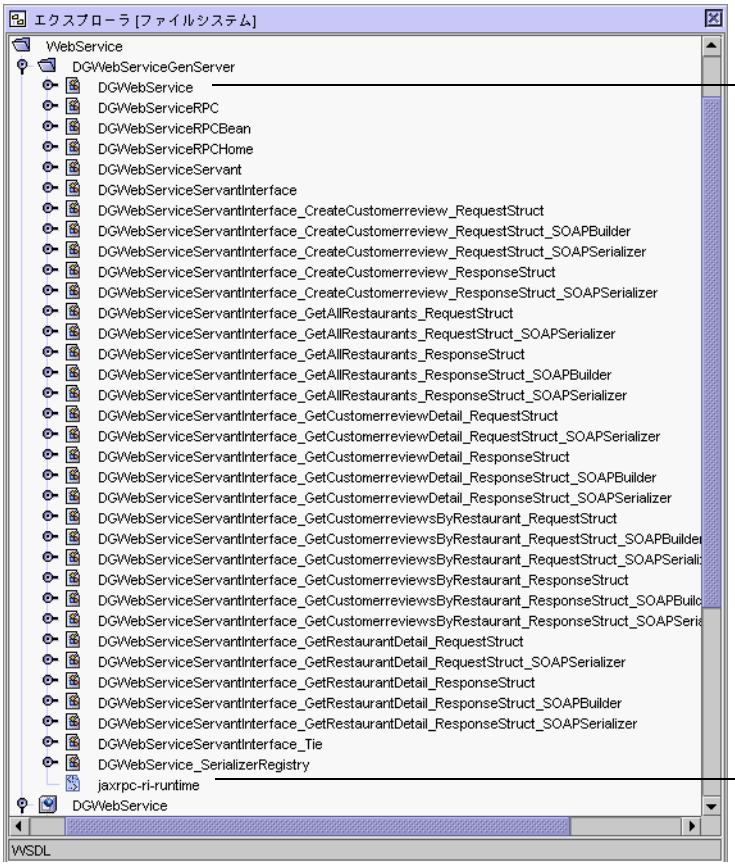
J2EE アプリケーションとして Web サービスをアセンブルしてテスト用に配備するには、Web サービスの実行時クラスを生成する必要があります。多層アーキテクチャの場合、IDE は多数のクラス (このうちの 3 つは生成された EJB コンポーネント用) を生成することによって、Web サービスを実装します。

注 – 管理サーバーが稼働していて、アプリケーションサーバーがデフォルトのサーバーとして設定されていることを確認してください。

Web サービスの実行時クラスを生成するには、次の操作を行います。

1. **「DGWebService」ノードを右クリックして、「Web サービスファイルを生成」を選択します。**

   処理が完了すると、IDE の出力ウィンドウに「完了」というメッセージが表示されます。SOAP RPC Web サービスを実装するための EJB コンポーネントとなる実行時クラスが、エクスプローラに表示されます。



2. DGWebService ノードのプロパティを表示します。

   エクスプローラの下の「プロパティ」ウィンドウで表示するか、ノードを右リックしてコンテキストメニューから「プロパティ」を選択します。

3. アプリケーションサーバーのホスト名とポート番号が SOAP RPC URL プロパティの値に反映されていることを確認します。

   たとえばアプリケーションサーバーのホスト名が tech5 で、ポート番号が 4855 の場合、URL は以下のようになります。

   ```
   http://tech5:4855/DGWebService/DGWebService
   ```

Web サービスのテスト

Web サービスのテストでは、次の作業を行う必要があります。

1. 以下のテストクライアントの作成
   - テストクライアント
   - EJB モジュールと Web サービスの両方を参照する J2EE アプリケーション
2. テストアプリケーションの配備
3. テストアプリケーションを使用した Web サービスのテスト

Web サービステストアプリケーションは、Web サービスでの XML 操作ごとに 1 つの JSP ページと、ブラウザでそれらのページをまとめて表示する開始ページを生成します。テストクライアントを実行するということは、開始ページから XML 操作を行うということです。

テストクライアントおよびテストアプリケーションの作成

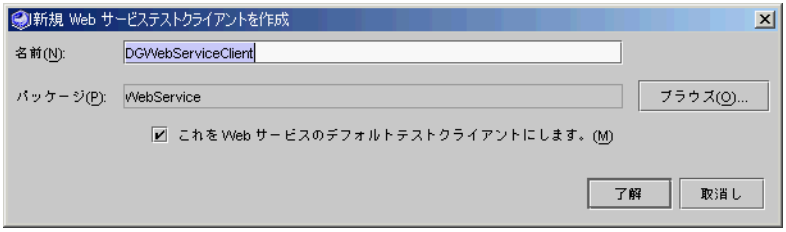
Web サービスをテストするには、テストクライアントと J2EE アプリケーションを作成します。このうちの J2EE アプリケーションに EJB モジュールと Web サービスモジュールを追加します。

参考 – テストクライアントの作成では、そのクライアントを Web サービス用のデフォルトのテストクライアントにします。J2EE アプリケーションを配備すると、テストクライアントも配備されます。

Web サービスのクライアントアプリケーションを作成して配備するには、次の操作を行います。

1. **エクスプローラで 「DGWebService」ノード () を右クリックして、「新規 Web サービステストクライアントを作成」を選択します。**

   「新規 Web サービステストクライアント」ダイアログが表示されます。



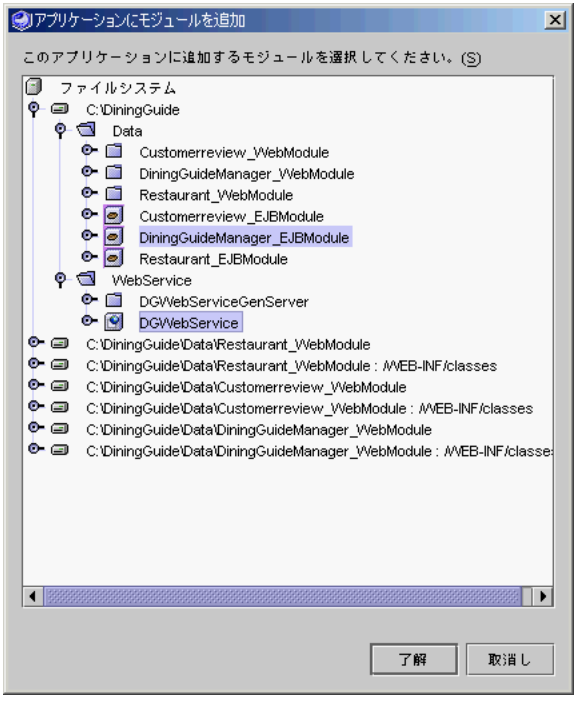
   デフォルトでは、このクライアントを Web サービスのデフォルトのテストクライアントにするオプションが選択されています。

2. すべてのデフォルト値をそのまま受け入れて、「了解」をクリックします。
   エクスプローラにクライアントノード () が表示されます。続いて、Web サービスの J2EE アプリケーションを作成します。
3. 「WebService」パッケージを右クリックして、「新規」→「すべてのテンプレート」を選択します。
4. 新規ウィザードで「J2EE」ノードを展開し、「アプリケーション」選択して、「次へ」をクリックします。
5. 「名前」フィールドに **`DGApp`** と入力し、「完了」をクリックします。
   WebService パッケージの下に J2EE アプリケーションノード () が表示されます。

J2EE アプリケーションへの Web サービスの追加

続いて、このアプリケーションに Web サービスを追加します。

1. 「DGApp」ノードを右クリックして、「モジュールを追加」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。
2. 「DigingGuide」ファイルシステムを展開し、「Data」および「WebService」パッケージの両方を展開します。
3. Control を押しながらクリックすることによって、「DiningGuideManager_EJBModule」と「DiningGuideWebService」の両方のノードを選択します。
   ダイアログは以下のような表示になります。



4. 「了解」をクリックして選択を確定し、ダイアログを閉じます。
5. エクスプローラで「DGApp」J2EE アプリケーションを展開します。

   DGWebService の WAR および EJB JAR の両方のファイルとともに DiningGuideManager_EJBModule がアプリケーションに追加されていることが分かります。

テストアプリケーションの配備

セッション Bean のテストアプリケーションを配備する前に、配備されているテストアプリケーションの配備をすべて取り消す必要があります。これは 88 ページの「テストアプリケーションの配備と実行」で説明したように、これらのアプリケーションが、Web アプリケーションに使用される Restaurant Bean と Customerreview Bean に同じ JNDI 検査を使用するためです。

アプリケーションの配備を取り消す手順については、89 ページの「エンティティ Bean のテストアプリケーションの配備取り消し」を参照してください。

注 – テストアプリケーションなど、データベースにアクセスする J2EE アプリケーションを配備する前に、PointBase サーバーが実行されていることを確認してください。また、 Sun ONE Application Server 7 インスタンスが稼働していること、IDE のデフォルトのアプリケーションサーバーになっていることを確認してください。9 ページの「Sun ONE Application Server 7 がデフォルトサーバーであることの確認」を参照してください。

DGApp アプリケーションを配備するには、以下の操作を行います。

1. **配備されているすべての DigingGuide テストアプリケーションの配備を取り消します。**

   この方法については、89 ページの「エンティティ Bean のテストアプリケーションの配備取り消し」を参照してください。必ずアプリケーションサーバーの再起動も行ってください。

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

   「進捗モニター」ウィンドウに、配備プロセスの実行状況が示されます。

3. **アプリケーションが配備されていることを確認します。**

   「進捗モニター」ウィンドウに、配備プロセスの進捗状況が示されます。出力ウィンドウ上のサーバーインスタンスのログファイルのタブに、進捗メッセージが表示されます。アプリケーションが正常に配備されると、完了のメッセージが表示されます。

   エクスプローラの「実行の表示」ウィンドウに *servername* (*server-hostname*:*server-port-number*) ノードが表示されます。

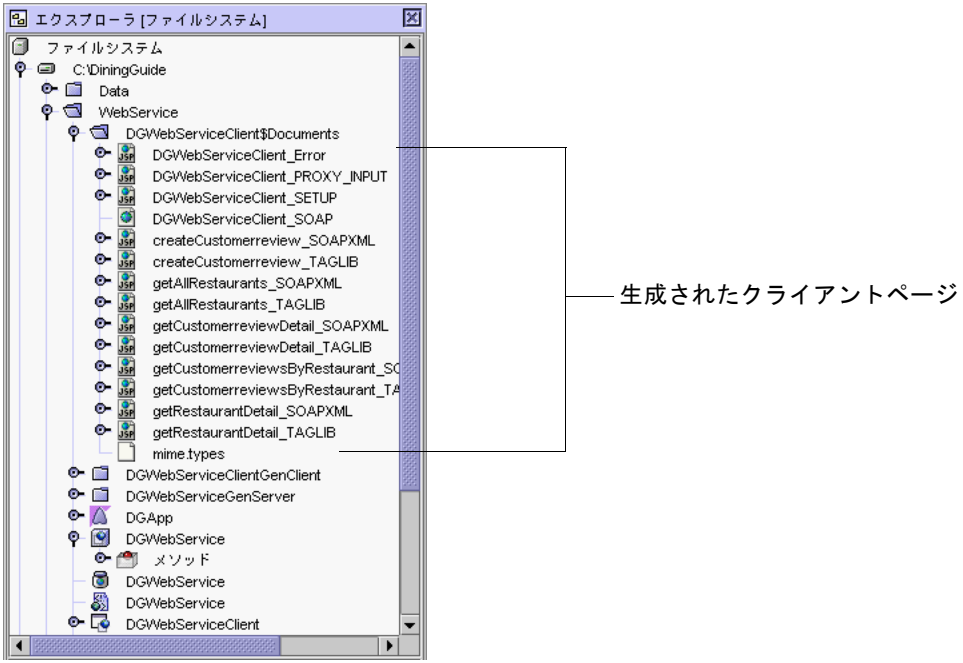
4. **「WebService」パッケージの下の「DGWebServiceClient$Documents」ノードを展開します。**

   次のサポート用項目が作成されていることが分かります。

   - タグライブラリを使用して表示の書式を整える JSP ページ (メソッドごとに 1 つ)
   - 返された SOAP メッセージを表示するための JSP ページ (メソッドごとに 1 つ)
   - 開始 (welcome) ページを構成する一群の JSP ページ
   - JSP エラーページ

   これらのページの詳細は、『Web サービスのプログラミング』を参照してください。

   エクスプローラの表示は次のようになります。



これらのファイルは、「DiningGuideWebServiceClient」ノードの下の「生成されたドキュメント」ノードの下でも参照されています。

テストアプリケーションを使用した Web サービスのテスト

クライアントと Web サービスの間で SOAP 要求と応答がどのようにやりとりされるかについての詳細は、『Web サービスのプログラミング』を参照してください。このマニュアルは、Sun ONE Studio 5 ポータルのドキュメントサイト (`http://sun.co.jp/software/sundev/jde/documentation/`) から入手できます。

注 – 管理サーバーおよび PointBase サーバーが稼働していることを確認してください。また、アプリケーションサーバーがデフォルトのアプリケーションサーバーになっていることも確認してください。

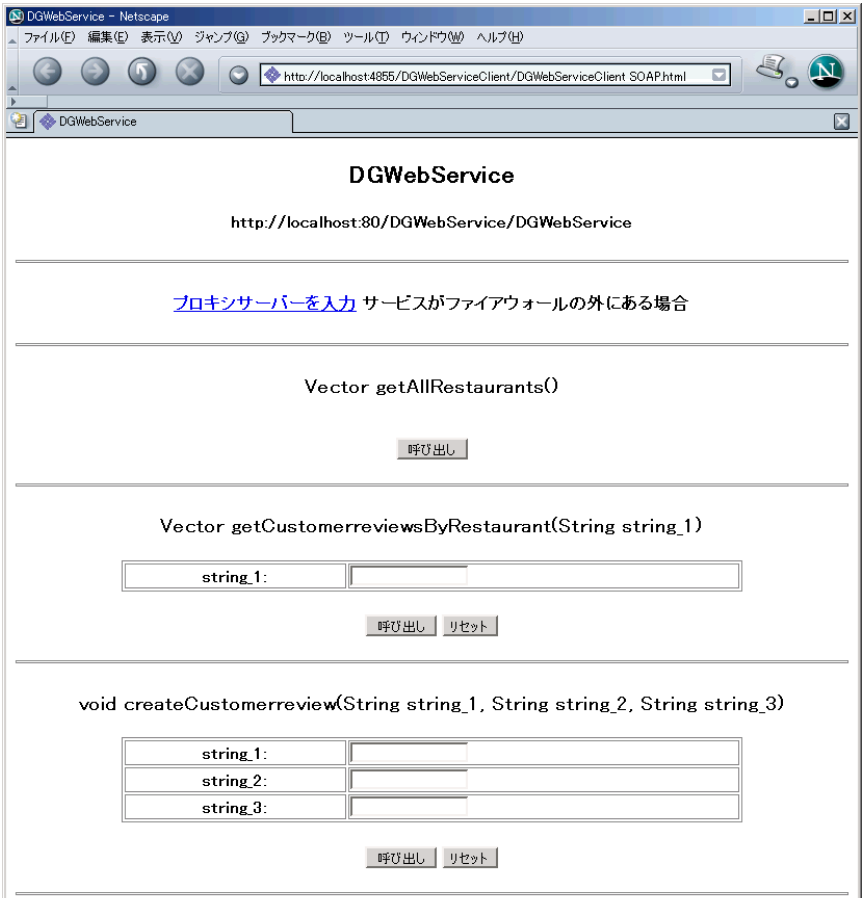
Web サービスをテストするには、次の操作を行います。

1. エクスプローラで「DGWebServiceClient」ノード () を右クリックして、「配備」を選択します。

   DGWebServiceClient が配備されます。このノードは、エクスプローラの「実行時」区画のアプリケーションサーバーインスタンスの「配備されている Web モジュール」の下に表示されます。

2. 同じノードを右クリックして、「実行」を選択します。

   IDE によってテストクライアントが配備されて、デフォルトの Web ブラウザが起動し、クライアントの生成された開始ページ (DGWebServiceClient_SOAP.html) が表示されます。



   このページから、操作が期待通りに機能するかどうかをテストすることができます。

3. テキストフィールドに **`French Lemon`** と入力して、「呼び出し」ボタンをクリックし、 getCustomerreviewsByRestaurant メソッドをテストします。

SOAP メッセージが作成されて、アプリケーションサーバーに送信されます。DGApp Web サービスは、この SOAP メッセージを DiningGuideManager.getCustomerreviewsByRestaurant メソッドのメソッド呼び出しに変換します。メソッドは生成された JSP ページ (getCustomerreviewsByRestaurant_TAGLIB.jsp) にコレクションを返し、次に示すように書式に従った表に返されたデータを表示します。

このデータには、French Lemon レストランに関して入力されているすべてのレコードが含まれます。PointBase コンソールを起動し、次の SQL 文を実行することによってデータを確認することができます。

```
select * from CustomerReview;
```

結果として、入力した CustomerReview レコードが表示されます。

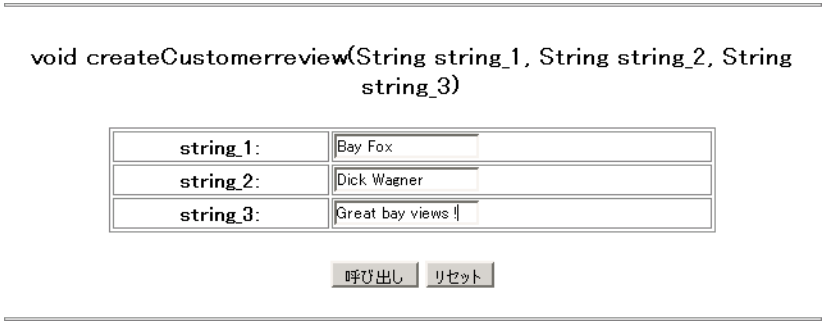
4. SOAP メッセージを表示するには、「SOAP 要求/応答を表示」のリンクをクリックします。

   返されたデータが、XML ラッパーで処理された SOAP メッセージとして表示します。

5. ブラウザの「戻る」ボタンを使用して、開始ページに戻ります。

6. 「restaurantname」フィールドに **Bay Fox** と入力し、残りの 2 つのフィールドに任意のデータを入力して createCustomerreview 操作をテストします。

   次に例を示します。



7. 「呼び出し」ボタンをクリックします。

   このメソッドは、入力パラメータとして複雑な Java オブジェクトを受け取ります。生成された JSP ページである createCustomerreview_SOAP.jsp は、3 つのパラメータの入力を求めます。入力内容は Customerreview オブジェクトに変換されて、SOAP メッセージに渡されます。このメッセージは、アプリケーションサーバーに送信され、そこで Java メソッド呼び出しに戻されて DiningGuideManager EJB コンポーネントに送信されます。ブラウザに、この結果が表示されます。

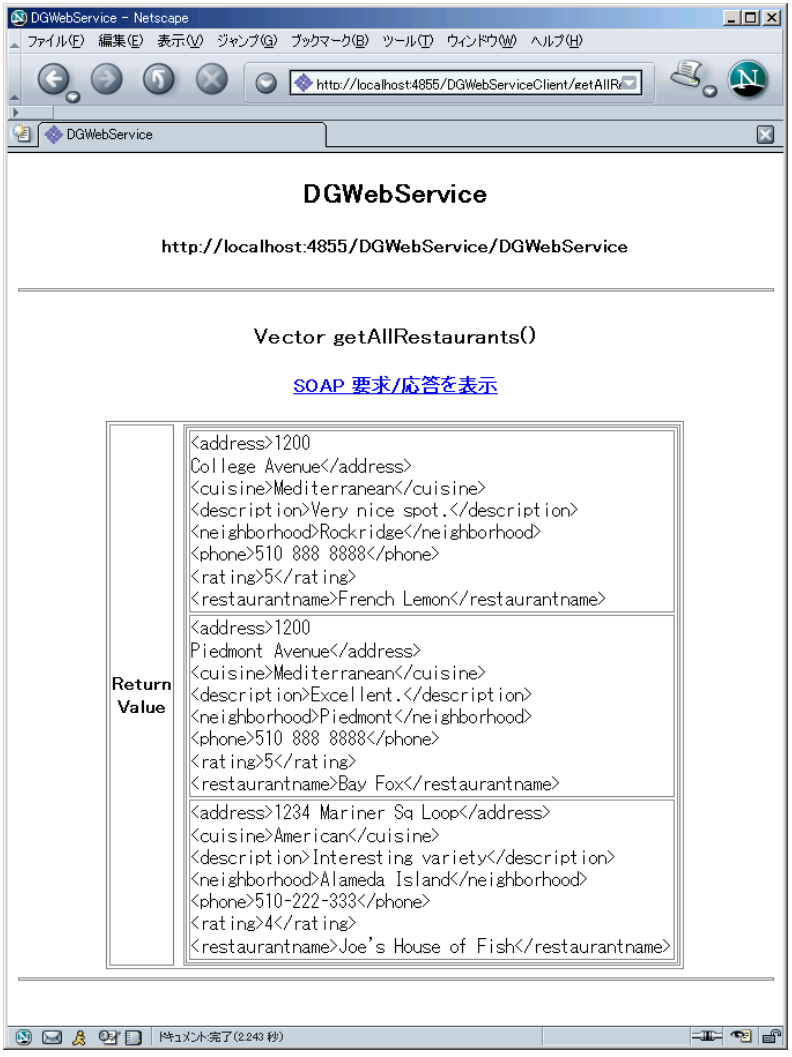
8. 「SOAP 要求/応答を表示」のリンクをクリックします。
   以下のように SOAP 要求および応答が表示されます。

9. ブラウザの「戻る」ボタンを使用して、開始ページに戻ります。
10. 続いて、getAllRestaurants 操作をテストします。開始ページで、その「呼び出し」ボタンをクリックしてください。

Vector getAllRestaurants()

呼び出し

このメソッドは入力パラメータを必要としません。レストランデータのコレクションを返し、そのコレクションが getAllRestaurants_SOAP.jsp ページによって XML 表示されます。



ページの最後のレコードが Restaurant エンティティ Bean のテストで入力した Restaurant レコード (62 ページの「テストクライアントを使用した Restaurant Bean のテスト」を参照) であることに注目してください。

これまでの確認ができれば、DiningGuide アプリケーションの Web サービス が正しく作成されていることになります。第 5 章では、提供されている Swing クライアントを使用して、DiningGuide アプリケーションを実行します。

他の開発者が Web サービスを利用できるようにする

前節では、Web サービスの開発者にとって Web サービスをテストする便利な方法を紹介しました。しかし、所属する組織の他の開発グループ (特にクライアントの開発者) も、Web サービスに対して自分たちの作業内容をテストする必要があります。作成した Web サービス の WSDL ファイルを、他の開発者に簡単に提供することができます。他の開発者は、このファイルから、アプリケーションのクライアントの作成に使用可能なクライアントファイルを生成し、Web サービスに対して自分が作成したクライアントをテストすることができます。他の開発者が Web サービスを利用するためには、配備されている Web サービスの URL が分かっていて、Web サーバーが稼働している必要があります。

他の開発者が Web サービスを利用できるようにするには、次の作業を行います。

1. Web サービスの開発者の作業
   - Web サービスから WSDL ファイルを作成します。
   - Web サービスのユーザーが WSDL ファイルを利用できるようにします。
   - Web サービスのユーザーに配備した Web サービスの URL を教えます。
2. クライアントの開発者 (Web サービスのユーザー) の作業:
   - エクスプローラでマウントしたファイルシステムに WSDL ファイルを追加します。
   - この WSDL から Web サービスのクライアントを作成します。
   - クライアントファイルを生成します。
   - クライアントファイルを基にクライアントを構築します。
   - クライアントファイルの SOAP RPC の URL プロパティとして、Web サービスの URL を指定します。

クライアントファイルを生成すると、アプリケーションの実際のクライアントの開発に必要な JSP ページが生成されます。

WSDL ファイルの生成

アプリケーションの Web サービスを共用できるようにするための第一歩は、Web サービスの WSDL ファイルを生成することです。この作業は、Web サービスの開発者が行います。

Web サービスの WSDL ファイルを生成するには、次の操作を行います。

1. **エクスプローラで 「DGWebService 」ノード () を右クリックして、「WSDL を生成」を選択します。**

   「WebService」パッケージの下に「DGWebService」という名前の WSDL ファイル (緑色の球の入ったアイコンのノード ) が作成されます。

   このファイル (`DGWebService.wsdl`) は使用しているオペレーティングシステムのファイルシステムの `DiningGuide/WebService` ディレクトリにあります。

2. **他の開発チームからこのファイルを利用できるようにします。**

   電子メールにファイルを添付するか、Web サイトで配布してください。

WSDL ファイルからクライアントファイルを作成

アプリケーションの Web サービスを共用できるようにするための第 2 段階の作業は、Web サービスサポート用のすべてのファイルを WSDL ファイルから生成することです。この作業は、クライアントの開発者が行います。

WSDL ファイルから Web サービスファイルとクライアントファイルを生成するには、次の操作を行います。

1. **使用しているオペレーティングシステムのファイルシステムでディレクトリを作成して、「`DGWebService.wsdl`」ファイルをそのディレクトリに保存します。**

   たとえば `c:\wsdlHolder` ディレクトリを作成して、そのディレクトリにファイルをペーストします。

2. **Sun ONE Studio 5のエクスプローラで、このディレクトリをマウントします。**

3. **マウントしたディレクトリを右クリックして、「新規」→「Java パッケージ」を選択します。**

4. **「名前」フィールドに `MyClientPackage` と入力し、「完了」をクリックします。**

   ディレクトリ内に MyClientPackage が表示されます。

5. **エクスプローラで「MyClientPackage」を右クリックして、「新規」→ 「すべてのテンプレート」を選択します。**

6. 新規ウィザードで「Web サービス」ノードを展開し、「Web サービスクライアント」選択して、「次へ」をクリックします。

   「Web サービスクライアント」ページが表示されます。

7. 「名前」フィールドに **`NewClient`** と入力します。

8. 「パッケージ」フィールドが MyClientPackage になっていることを確認します。

9. ソースとして「ローカル WSDL ファイル」オプションを選択して、「次へ」をクリックします。

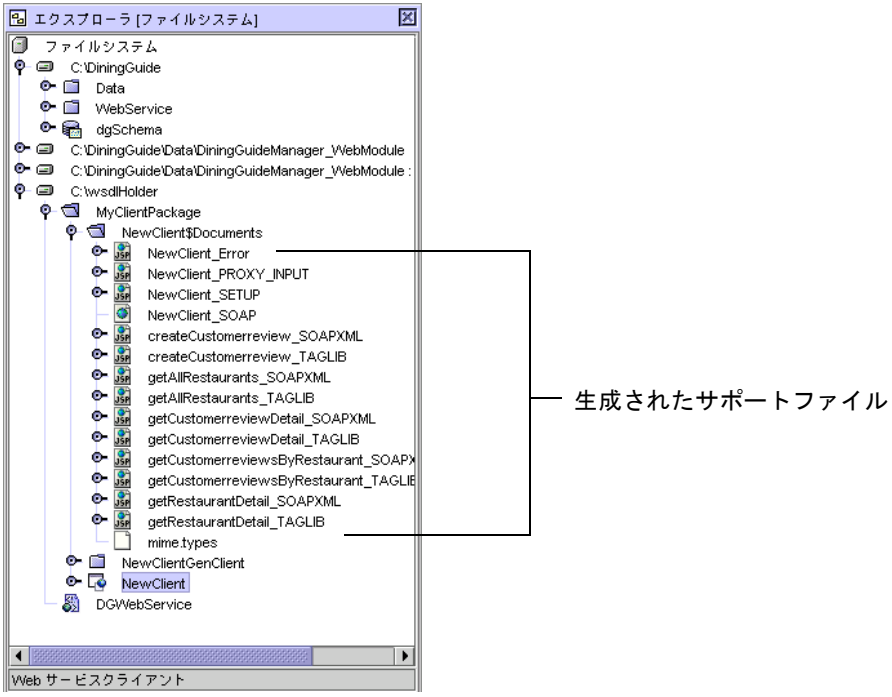
   新規ウィザードの「ローカル WSDL ファイルを選択」ページが表示されます。

10. 手順 1で作成したディレクトリを展開し、DGWebService WSDL ファイルを選択して、「完了」をクリックします。

    エクスプローラにクライアントノード () が表示されます。

11. 「NewClient」 クライアントノードを右クリックして、「クライアントファイルを生成」を選択します。

    エクスプローラに「生成されたドキュメント」ノードが作成されます。開いた状態の「生成されたドキュメント」ノードに、クライアントに必要な JSP ページと開始ページが含まれていることが分かります。



これで、クライアントを使用して、Web サービスをテストすることができます (108 ページの「テストアプリケーションを使用した Web サービスのテスト」を参照)。

多くの場合は、アプリケーションが完成したら、UDDI レジストリに Web サービスを公開し、外部の開発者がそのサービスを利用できるようにします。Sun ONE Studio 5 には、このテストを行うためのシングルユーザー内部 UDDI レジストリと、StockApp という具体例が用意されており、この具体例は、Sun ONE Studio 5 ポータルサイトの Examples and Tutorials ページ (`http://forte.sun.com/ffj/documentation/tutorialsandexamples.html`) から入手することができます。外部 UDDI レジストリへの公開については、『Web サービスのプログラミング』を参照してください。

第5章

チュートリアルアプリケーションのクライアントの作成

このチュートリアルには、第 4 章 で作成した Web サービスと通信する Swing クライアントが用意されています。この章では、その Swing クライアントを使用して、DiningGuide アプリケーションを実行する方法を学びます。

提供のクライアントには、RestaurantTable と CustomerReviewTable という 2 つの Swing クラスが含まれており、 これらのクラスを WebService パッケージに追加し、RestaurantTable クラスを実行することでアプリケーションを実行します。

なお、このクライアントは非常に原始的なもので、Web サービス用に生成したクライアントのメソッドへのアクセス方法を具体的に理解していただく目的に用意されています。

この章の内容は以下のとおりです。

提供コードを使用したクライアントの作成

クライアントクラスは、`DiningGuide.zip` ファイル内に Java クラスファイルとして用意されており、この zip ファイルは Developer Resources ポータルサイトからダウンロードできます。

提供されている 2 つの Java クライアントクラスを、DiningGuide アプリケーションにコピーします。

1. IDE で「DigingGuide」フォルダ内に Client パッケージを作成します。

a. 「DigingGuide」フォルダを右クリックして、「新規」→「Java パッケージ」を選択します。

b. 新規ウィザードの「名前」フィールドに **`Client`** と入力して、「完了」をクリックします。

2. Developer Resources ポータルサイトからダウンロードした `DiningGuide.zip` ファイルを解凍します。

a. Developer Resources ポータルサイトから `DiningGuide.zip` ファイルをダウンロードします。

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

b. ダウンロードしたファイルをローカルディレクトリ (たとえば `/MyZipFiles` ディレクトリ) に解凍します。

3. ファイルシステムコマンドを使用して、2 つのクライアントファイルを DiningGuide ソースファイルから Client パッケージにコピーします。

- Microsoft Windows システムの場合は、ファイルを `Client` フォルダにコピーします。
- Solaris または Linux 環境の場合は、以下のようにコマンドを入力します。

```
$ cp /MyZipFiles/DiningGuide/Client/*.java /DiningGuide/Client
```

4. IDE のエクスプローラで「DiningGuide/Client」パッケージを展開し、2 つの新しいクラスが存在することを確認します。

注 – IDE のフォームエディタで Swing クライアントを作成すると、`.form` ファイルと `.java` ファイルが 1 つずつ作成されます。`.form` ファイルは、フォームエディタで GUI コンポーネントを編集するために使用されます。しかし、RestaurantTable クラスと CustomerReviewTable クラスの `.form` ファイルは提供されていません。このため、フォームエディタでそれらのクラスの GUI コンポーネントを変更することはできません。

チュートリアルアプリケーションの実行

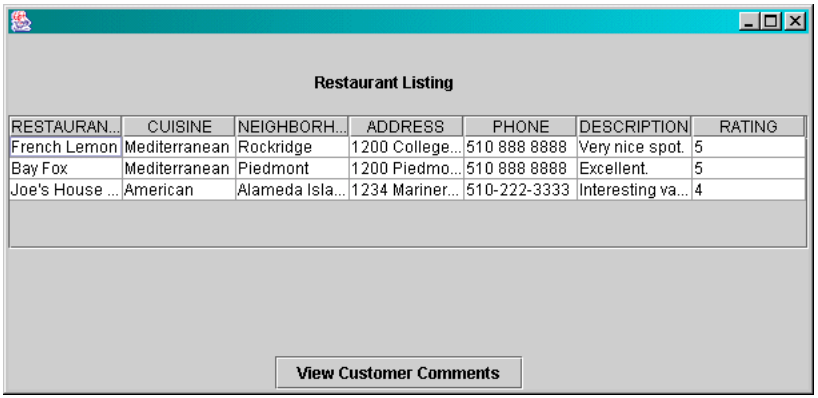
注 – チュートリアルアプリケーションを実行する前に、PointBase サーバーが稼働していることを確認してください。また、 Sun ONE Application Server 7 サーバーが稼働していて、IDE のデフォルトのアプリケーションサーバーになっていることを確認してください。9 ページの「Sun ONE Application Server 7 がデフォルトサーバーであることの確認」を参照してください。

DiningGuide アプリケーションを実行するには、以下の手順で RestaurantTable クラスを実行します。

1. IDE のエクスプローラの「実行時」タブをクリックします。
2. 「サーバーレジストリ」、「インストールされているサーバー」、「Sun ONE Application Server 7」の順にノードを展開します。
3. サーバーインスタンスの「配備されているアプリケーション」サブノードを右クリックします。
4. DGApp アプリケーションが配備されたままであることを確認します。

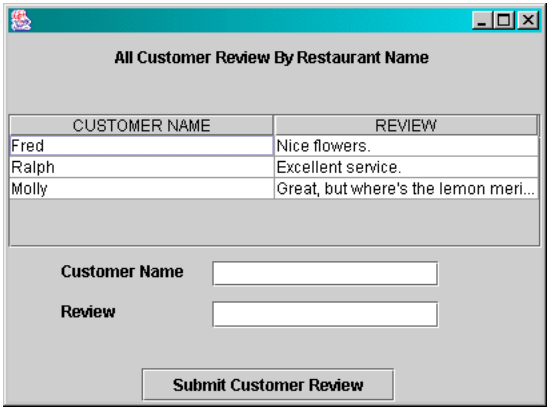
   配備されている場合は、「DGApp」ノードが「配備されているアプリケーション」サブノードの下に表示されています。
5. 配備されていない場合は、106 ページの「テストアプリケーションの配備」の説明にしたがって配備します。
6. エクスプローラの「ファイルシステム」タブで「RestaurantTable」ノードを右クリックして「実行」を選択します。

   IDE が実行モードに切り替わります。「実行」ウィンドウに「Restaurant」ノードが現れ、以下のような「RestaurantTable」ウィンドウが表示されます。



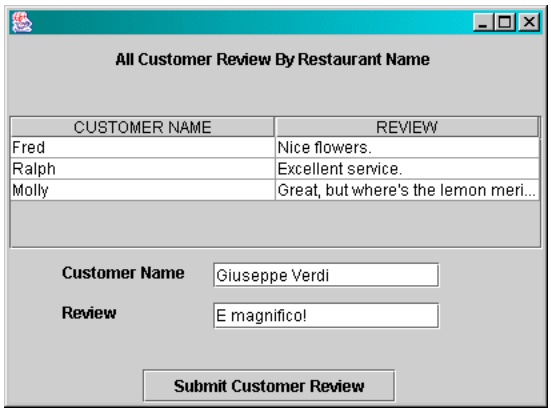
7. 表のレストランをどれか選択して、「View Customer Comments」ボタンをクリックします。

   たとえば、French Lemon レストランを選択してください。
   「CustomerReviewTable」ウィンドウが表示されます。データベースにそのレストランに関するコメントが存在する場合は、以下のように表示されます。存在しない場合は、空の表が表示されます。

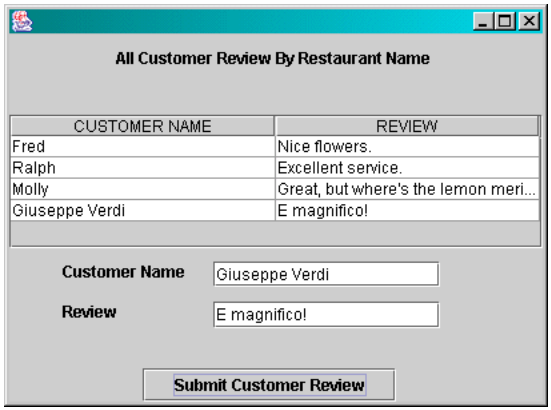


8. 「CUSTOMER NAME」フィールドと「REVIEW」フィールドに何らかを入力して、「Submit Customer Review」ボタンをクリックします。

   次に例を示します。



レコードがデータベースに登録され、同じ「CustomerReviewTable」ウィンドウに次のように表示されます。



9. 17 ページの「ユーザーから見たチュートリアルアプリケーション」で説明している機能を一通り試してみます。

10. どれかウィンドウを閉じることによってアプリケーションを終了します。

    アプリケーションを終了すると、実行ウィンドウに Sun ONE Application Server 7 プロセスがまだ動作中であることが示されます。アプリケーションサーバーを停止する必要はありません。チュートリアルアプリケーションの J2EE アプリケーションのどれかを再配備するか、 この Swing クライアント以外のテストクライアントを再実行すると、この Sun ONE Application Server が自動的に再起動されます。

    IDE を終了すると、アプリケーションサーバーや Web サーバーなどの動作中のプロセスを終了するためのダイアログが表示されます。動作中の各プロセスについて、プロセスを選択して、「タスクを終了」ボタンをクリックしてください。IDE の動作中にプロセスを手動で終了することもできます。このためには、「実行」ウィンドウでそのプロセスのノードを右クリックして、「プロセスを終了」を選択します。

クライアントコードの内容

DiningGuide アプリケーションに組み込んだ 2 つのクライアントクラスは、フォームエディタで作成された Swing コンポーネントおよびアクションと、ソースエディタで作成されたいくつかのメソッドから構成されます。ソースエディタで追加されたメソッドには、クライアントをインスタンス化して、クライアントからそのメソッドを利用できるようにする、重要なタスクが含まれています。

Swing クライアントと Web サービスがどのように対話するのかを理解するため、この節ではクライアントの主なアクションについて説明します。

- 126 ページの「レストランデータの表示」
- 127 ページの「選択されたレストランの利用者コメントデータの表示」
- 130 ページの「新規利用者コメントレコードの作成」

レストランデータの表示

レストランデータの表示は、以下のようにクライアントをインスタンス化して、その getAllRestaurants メソッドを呼び出す、RestaurantTable クラスのメソッドによって行われます。

1. RestaurantTable.getAllRestaurants メソッドがクライアントをインスタンス化して、クライアントの getAllRestaurants メソッド (レストランデータをフェッチするメソッド) を呼び出し、フェッチされたレストランデータを 1 つの Vector として返します。

```
private Vector getAllRestaurants() {
    Vector restList = new Vector();
    try {
        WebService.DGWebServiceClientGenClient.DGWebService  service1 = new
        WebService.DGWebServiceClientGenClient.DGWebService_Impl();
        WebService.DGWebServiceClientGenClient.DGWebServiceServantInterface
port =
        service1.getDGWebServiceServantInterfacePort();

        restList = (java.util.Vector)port.getAllRestaurants();
    }
    catch (Exception ex) {
        System.err.println("Caught an exception." ); );
        ex.printStackTrace();
    }
    return restList;
}
```

2. RestaurantTable コンストラクタが返されたレストランデータを restaurantList 変数に書き込み、RestaurantTable.putDataToTable を呼び出します。

```
public RestaurantTable() {
    initComponents();
    restaurantList=getAllRestaurants();
    putDataToTable();
}
```

3. The RestaurantTable.putDataToTable メソッドは Vector を反復処理し、表を生成します。

```
private void putDataToTable() {
    Iterator j=restaurantList.iterator();
    while (j.hasNext()) {
        RestaurantDetail ci = (RestaurantDetail)j.next();
        String strRating = null;
        String[] str ={ci.getRestaurantname(), ci.getCuisine(),
ci.getNeighborhood(), ci.getAddress(), ci.getPhone(),
ci.getDescription(), strRating.valueOf(ci.getRating()),
        };
        TableModel.addRow(str);
    }
}
```

4. RestaurantTable.Main メソッドが Swing jTable コンポーネントとして表を表示します。

```
public static void main(String args[]) {
    new RestaurantTable().show();
}
```

選択されたレストランの利用者コメントデータの表示

利用者コメントデータの表示は、RestaurantTable のボタンコンポーネントのアクションによって CustomerReviewTable がインスタンス化されることで始まります。CustomerReviewTable のメソッドはクライアントのメソッドを利用して利用者コメントデータをフェッチし、そのデータで表を生成します。そして RestaurantTable のボタンコンポーネントのアクションによって、その表が表示されます。

1. 利用者コメントデータを取得するために、RestaurantTable のボタンがクリックされると、RestaurantTable.jButton1ActionPerformed メソッドが CustomerReviewTable オブジェクトをインスタンス化して、putDataToTable メソッドを呼び出し、選択された列のデータを渡します。

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
    int r =jTable1.getSelectedRow();
    int c = jTable1.getSelectedColumnCount();
    String i =(String)TableModel.getValueAt(r,0);
    CustomerReviewTable crt = new CustomerReviewTable();
    crt.putDataToTable(i);
    crt.show();
    System.out.println(i);
}
```

2. CustomerReviewTable.putDataToTable メソッドが CustomerReviewTable.getCustomerReviewByName メソッドを呼び出して、選択されたレストラン名を渡し、返された Vector を customerList 変数に代入します。

```
public void putDataToTable(java.lang.String restaurantname) {
    RestaurantName = restaurantname;
    java.util.Vector customerList
=getCustomerReviewByName(restaurantname);
    Iterator j=customerList.iterator();
    while (j.hasNext()) {
        CustomerreviewDetail ci = (CustomerreviewDetail)j.next();
        String[] str = {ci.getCustomername(),ci.getReview()
        };
        TableModel.addRow(str);
    }
}
```

3. The CustomerReviewTable.getCustomerReviewByName メソッドが、必要に応じてクライアントをインスタンス化し、その getCustomerreviewsByRestaurant メソッドを呼び出して、選択されたレストラン名を渡します。

```
private Vector getCustomerReviewByName(java.lang.String restaurantname) {
    Vector custList = new Vector();
    try{
        WebService.DGWebServiceClientGenClient.DGWebService  service2 = new
        WebService.DGWebServiceClientGenClient.DGWebService_Impl();
        WebService.DGWebServiceClientGenClient.DGWebServiceServantInterface
port = service2.getDGWebServiceServantInterfacePort();

        custList =(java.util.Vector)port.getCustomerreviewsByRestaurant(restaurantname);
}
    catch (Exception ex) {
        System.err.println("Caught an exception." ); );
        ex.printStackTrace();
    }
    return custList;
}
```

4. コメントデータが CustomerReviewTable.putDataToTable メソッドに渡され、このメソッドによってデータが反復処理されて、そのデータで表が生成されます。

```
public void putDataToTable(java.lang.String restaurantname) {
    RestaurantName = restaurantname;
    java.util.Vector customerList
=getCustomerReviewByName(restaurantname);
    Iterator j=customerList.iterator();
    while (j.hasNext()) {
        CustomerreviewDetail ci = (CustomerreviewDetail)j.next();
        String[] str = {ci.getCustomername(),ci.getReview()
        };
        TableModel.addRow(str);
    }
}
```

5. RestaurantTable.jButton1ActionPerformed メソッドが表を表示します。

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent
evt)
{
    int r =jTable1.getSelectedRow();
    int c = jTable1.getSelectedColumnCount();
    String i =(String)TableModel.getValueAt(r,0);
    CustomerReviewTable crt = new CustomerReviewTable();
    crt.putDataToTable(i);
    crt.show();
    System.out.println(i);
}
```

新規利用者コメントレコードの作成

「Customer Review」ウィンドウで名前とコメントを入力して、「Submit Customer Review」ボタンをクリックすると、以下のようにして、CustomerReviewTable の jButton1ActionPerformed メソッドが、クライアントのメソッドを利用してデータベースにそのコメントレコードを作成、「Customer Review」ウィンドウを再表示します。

1. CustomerReviewTable のボタンがクリックされて利用者コメントレコードが送信されると、CustomerReviewTable.jButton1ActionPerformed メソッドが必要に応じて新しいクライアントをインスタンス化して、その createCustomerreview メソッドを呼び出し、レストラン名と利用者名、コメントデータを渡します。

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        WebService.DGWebServiceClientGenClient.DGWebService  service1 = new
        WebService.DGWebServiceClientGenClient.DGWebService_Impl();
        WebService.DGWebServiceClientGenClient.DGWebServiceServantInterface
port = service1.getDGWebServiceServantInterfacePort();

        port.createCustomerreview(RestaurantName,
        customerNameField.getText(),reviewField.getText());
    }
    catch (Exception ex) {
        System.err.println("Caught an exception." ); );
        ex.printStackTrace();
    }
    refreshView();
    }
```

2. 同様に jButton1ActionPerformed メソッドが CustomerReviewTable.refreshView メソッドを呼び出します。

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        WebService.DGWebServiceClientGenClient.DGWebService service1 = new
        WebService.DGWebServiceClientGenClient.DGWebService_Impl();
        WebService.DGWebServiceClientGenClient.DGWebServiceServantInterface
port = service1.getDGWebServiceServantInterfacePort();

        port.createCustomerreview(RestaurantName,
        customerNameField.getText(),reviewField.getText());
    }
    catch (Exception ex) {
        System.err.println("Caught an exception." ); );
        ex.printStackTrace();
    }
    refreshView();
    }
```

3. CustomerReviewTable.refreshView メソッドが putDataToTable メソッドを呼び出し、レストラン名を渡します。

```
void refreshView() {
    try{
        while(TableModel.getRowCount()>0) {
            TableModel.removeRow(0);
        }
        putDataToTable(RestaurantName);
        repaint();
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

4. CustomerReviewTable.putDataToTable メソッドが渡されたデータで表を生成します。

```
public void putDataToTable(java.lang.String restaurantname) {
    RestaurantName = restaurantname;
    java.util.Vector customerList
=getCustomerReviewByName(restaurantname);
    Iterator j=customerList.iterator();
    while (j.hasNext()) {
        CustomerreviewDetail ci = (CustomerreviewDetail)j.next();
        String[] str = {ci.getCustomername(),ci.getReview()
        };
        TableModel.addRow(str);
    }
}
```

5. CustomerReviewTable.refreshView メソッドがウィンドウを再表示して、新しいデータを表示します。

```
void refreshView() {
    try{
        while(TableModel.getRowCount()>0) {
            TableModel.removeRow(0);
        }
        putDataToTable(RestaurantName);
        repaint();
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

付録 A

DiningGuide のソースファイル

この付録では、次の DiningGuide コンポーネントのコードをまとめています。

- EJB 層のコンポーネント
  - 135 ページの「RestaurantBean.java のソース」
  - 138 ページの「RestaurantDetail.java のソース」
  - 143 ページの「CustomerreviewBean.java のソース」
  - 146 ページの「CustomerreviewDetail.java のソース」
  - 149 ページの「DiningGuideManagerBean.java のソース」
- クライアントのコンポーネント
  - 153 ページの「RestaurantTable.java のソース」
  - 157 ページの「CustomerReviewTable.java のソース」

これらのソースファイルは、次の Sun ONE Studio 5 Developer Resources ポータルサイトからダウンロード可能な DiningGuide アプリケーションの zip ファイルにも含まれています。

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

参考 – これらのファイルの内容を Sun ONE Studio 5 のソースエディタにカット & ペーストすると、すべての書式設定が失われます。ソースエディタでコードの書式を整えるには、そのコードブロックを選択して、Control-Shift F を押してください。

注 – PDF からコピーした折り返し行はそれぞれ独立した行になり、コンパイラはある種の無意味な文と解釈します。コード中のそうした行の前には、「ソースエディタで次の 2 行を 1 行にする」というコメントを挿入してあります。またソースエディタは、簡単に見つけて修正できるよう、そうした行にコーディングエラーのマークを付けます。

ソースエディタは行末のキャリッジリターンを読み取れないため、Solaris や Linux をお使いの場合には、これらのファイルからコピーすることはお勧めしません。ソースファイルを見るには、DiningGuide の zip ファイルを解凍し、IDE 内の解凍されたディレクトリにマウントしてください。

RestaurantBean.java のソース

```
package Data;

import javax.ejb.*;


public abstract class RestaurantBean implements javax.ejb.EntityBean {

    private javax.ejb.EntityContext context;

    /**
     * @see javax.ejb.EntityBean#setEntityContext(javax.ejb.EntityContext)
     */
    public void setEntityContext(javax.ejb.EntityContext aContext) {
        context=aContext;
    }

    /**
     * @see javax.ejb.EntityBean#ejbActivate()
     */
    public void ejbActivate() {

    }

    /**
     * @see javax.ejb.EntityBean#ejbPassivate()
     */
    public void ejbPassivate() {

    }

    /**
     * @see javax.ejb.EntityBean#ejbRemove()
     */
    public void ejbRemove() {

    }
```

```
    /**
     * @see javax.ejb.EntityBean#unsetEntityContext()
     */
    public void unsetEntityContext() {
        context=null;
    }

    /**
     * @see javax.ejb.EntityBean#ejbLoad()
     */
    public void ejbLoad() {

    }

    /**
     * @see javax.ejb.EntityBean#ejbStore()
     */
    public void ejbStore() {

    }

    public abstract java.lang.String getRestaurantname();
    public abstract void setRestaurantname(java.lang.String restaurantname);

    public abstract java.lang.String getCuisine();
    public abstract void setCuisine(java.lang.String cuisine);

    public abstract java.lang.String getNeighborhood();
    public abstract void setNeighborhood(java.lang.String neighborhood);

    public abstract java.lang.String getAddress();
    public abstract void setAddress(java.lang.String address);

    public abstract java.lang.String getPhone();
    public abstract void setPhone(java.lang.String phone);

    public abstract java.lang.String getDescription();
    public abstract void setDescription(java.lang.String description);

    public abstract int getRating();
    public abstract void setRating(int rating);
```

```
    public java.lang.String ejbCreate(java.lang.String restaurantname,
java.lang.String cuisine, java.lang.String neighborhood, java.lang.String
address, java.lang.String phone, java.lang.String description, int rating)
throws javax.ejb.CreateException {
        if (restaurantname == null) {
            //ソースエディタで次の 2 行を 1 行にする
            throw new javax.ejb.CreateException("The restaurant name is
required.");
        }
        setRestaurantname(restaurantname);
        setCuisine(cuisine);
        setNeighborhood(neighborhood);
        setAddress(address);
        setPhone(phone);
        setDescription(description);
        setRating(rating);
        return null;
    }

   public void ejbPostCreate(java.lang.String restaurantname, java.lang.String
cuisine, java.lang.String neighborhood, java.lang.String address,
java.lang.String phone, java.lang.String description, int rating) throws
javax.ejb.CreateException {
    }

    public Data.RestaurantDetail getRestaurantDetail() {
        return (new RestaurantDetail(getRestaurantname(),
        getCuisine(), getNeighborhood(), getAddress(), getPhone(),
        getDescription(), getRating()));
    }
}
```

RestaurantDetail.java のソース

```
/*
 * RestaurantDetail.java
 *
 * Created on March 27, 2003, 3:35 PM
 */

package Data;

import java.beans.*;

public class RestaurantDetail extends Object implements java.io.Serializable {

    private static final String PROP_SAMPLE_PROPERTY = "SampleProperty";

    private String sampleProperty;

    private PropertyChangeSupport propertySupport;

    /** Holds value of property restaurantname. */ */
    private String restaurantname;

    /** Holds value of property cuisine. */ */
    private String cuisine;

    /** Holds value of property neighborhood. */ */
    private String neighborhood;

    /** Holds value of property address. */ */
    private String address;

    /** Holds value of property phone. */ */
    private String phone;

    /** Holds value of property description. */ */
    private String description;

    /** Holds value of property rating. */ */
```

```
    private int rating;

    /** Creates new RestaurantDetail */
    public RestaurantDetail() {
        propertySupport = new PropertyChangeSupport( this );
    }

    public RestaurantDetail(java.lang.String restaurantname, java.lang.String
cuisine, java.lang.String neighborhood, java.lang.String address,
java.lang.String phone, java.lang.String description, int rating) {
        System.out.println("Creating new RestaurantDetail");
        setRestaurantname(restaurantname);
        setCuisine(cuisine);
        setNeighborhood(neighborhood);
        setAddress(address);
        setPhone(phone);
        setDescription(description);
        setRating(rating);
    }

    public String getSampleProperty() {
        return sampleProperty;
    }

    public void setSampleProperty(String value) {
        String oldValue = sampleProperty;
        sampleProperty = value;
        propertySupport.firePropertyChange(PROP_SAMPLE_PROPERTY, oldValue,
sampleProperty);
    }


    public void addPropertyChangeListener(PropertyChangeListener listener) {
        propertySupport.addPropertyChangeListener(listener);
    }

   public void removePropertyChangeListener(PropertyChangeListener listener) {
        propertySupport.removePropertyChangeListener(listener);
    }

    /** Getter for property restaurantname.
     * @return Value of property restaurantname.
```

```
     *
     */
    public String getRestaurantname() {
        return this.restaurantname;
    }

    /** Setter for property restaurantname.
     * @param restaurantname New value of property restaurantname.
     *
     */
    public void setRestaurantname(String restaurantname) {
        this.restaurantname = restaurantname;
    }

    /** Getter for property cuisine.
     * @return Value of property cuisine.
     *
     */
    public String getCuisine() {
        return this.cuisine;
    }

    /** Setter for property cuisine.
     * @param cuisine New value of property cuisine.
     *
     */
    public void setCuisine(String cuisine) {
        this.cuisine = cuisine;
    }

    /** Getter for property neighborhood.
     * @return Value of property neighborhood.
     *
     */
    public String getNeighborhood() {
        return this.neighborhood;
    }

    /** Setter for property neighborhood.
     * @param neighborhood New value of property neighborhood.
     *
```

```
     */
    public void setNeighborhood(String neighborhood) {
        this.neighborhood = neighborhood;
    }

    /** Getter for property address.
     * @return Value of property address.
     *
     */
    public String getAddress() {
        return this.address;
    }

    /** Setter for property address.
     * @param address New value of property address.
     *
     */
    public void setAddress(String address) {
        this.address = address;
    }

    /** Getter for property phone.
     * @return Value of property phone.
     *
     */
    public String getPhone() {
        return this.phone;
    }

    /** Setter for property phone.
     * @param phone New value of property phone.
     *
     */
    public void setPhone(String phone) {
        this.phone = phone;
    }

    /** Getter for property description.
     * @return Value of property description.
     *
     */
```

```
    public String getDescription() {
        return this.description;
    }

    /** Setter for property description.
     * @param description New value of property description.
     *
     */
    public void setDescription(String description) {
        this.description = description;
    }

    /** Getter for property rating.
     * @return Value of property rating.
     *
     */
    public int getRating() {
        return this.rating;
    }

    /** Setter for property rating.
     * @param rating New value of property rating.
     *
     */
    public void setRating(int rating) {
        this.rating = rating;
    }
}
```

CustomerreviewBean.java のソース

```
package Data;

import javax.ejb.*;

public abstract class CustomerreviewBean implements javax.ejb.EntityBean {

    private javax.ejb.EntityContext context;


    /**
     * @see javax.ejb.EntityBean#setEntityContext(javax.ejb.EntityContext)
     */
    public void setEntityContext(javax.ejb.EntityContext aContext) {
        context=aContext;
    }

    /**
     * @see javax.ejb.EntityBean#ejbActivate()
     */
    public void ejbActivate() {

    }

    /**
     * @see javax.ejb.EntityBean#ejbPassivate()
     */
    public void ejbPassivate() {

    }

    /**
     * @see javax.ejb.EntityBean#ejbRemove()
     */
    public void ejbRemove() {

    }
```

```
    /**
     * @see javax.ejb.EntityBean#unsetEntityContext()
     */
    public void unsetEntityContext() {
        context=null;
    }

    /**
     * @see javax.ejb.EntityBean#ejbLoad()
     */
    public void ejbLoad() {

    }

    /**
     * @see javax.ejb.EntityBean#ejbStore()
     */
    public void ejbStore() {

    }

    public abstract java.lang.String getRestaurantname();
    public abstract void setRestaurantname(java.lang.String restaurantname);

    public abstract java.lang.String getCustomername();
    public abstract void setCustomername(java.lang.String customername);

    public abstract java.lang.String getReview();
    public abstract void setReview(java.lang.String review);

    public Data.CustomerreviewKey ejbCreate(java.lang.String restaurantname,
java.lang.String customername, java.lang.String review) throws
javax.ejb.CreateException {
        if ((restaurantname == null) || (customername == null)) {
            //ソースエディタで次の 2 行を 1 行にする
            throw new javax.ejb.CreateException("Both the restaurant name and
customer name are required.");
        }
        setRestaurantname(restaurantname);
        setCustomername(customername);
        setReview(review);
        return null;
```

```
    }

   public void ejbPostCreate(java.lang.String restaurantname, java.lang.String
customername, java.lang.String review) throws javax.ejb.CreateException {
    }

    public Data.CustomerreviewDetail getCustomerreviewDetail() {
        return (new CustomerreviewDetail(getRestaurantname(),
        getCustomername(), getReview()));
    }
}
```

CustomerreviewDetail.java のソース

```
 * CustomerreviewDetail.java
 *
 * Created on March 27, 2003, 3:35 PM
 */

package Data;

import java.beans.*;

public class CustomerreviewDetail extends Object implements
java.io.Serializable {

    private static final String PROP_SAMPLE_PROPERTY = "SampleProperty";

    private String sampleProperty;

    private PropertyChangeSupport propertySupport;

    /** Holds value of property restaurantname. */ */
    private String restaurantname;

    /** Holds value of property customername. */ */
    private String customername;

    /** Holds value of property review. */ */
    private String review;

    /** Creates new CustomerreviewDetail */
    public CustomerreviewDetail() {
        propertySupport = new PropertyChangeSupport( this );
    }

    public CustomerreviewDetail(java.lang.String restaurantname,
java.lang.String customername, java.lang.String review) {
        System.out.println("Creating new CustomerreviewDetail");
        setRestaurantname(restaurantname);
        setCustomername(customername);
```

```
        setReview(review);
    }

    public String getSampleProperty() {
        return sampleProperty;
    }

    public void setSampleProperty(String value) {
        String oldValue = sampleProperty;
        sampleProperty = value;
        propertySupport.firePropertyChange(PROP_SAMPLE_PROPERTY, oldValue,
sampleProperty);
    }

    public void addPropertyChangeListener(PropertyChangeListener listener) {
        propertySupport.addPropertyChangeListener(listener);
    }

    public void removePropertyChangeListener(PropertyChangeListener listener) {
        propertySupport.removePropertyChangeListener(listener);
    }

    /** Getter for property restaurantname.
     * @return Value of property restaurantname.
     *
     */
    public String getRestaurantname() {
        return this.restaurantname;
    }

    /** Setter for property restaurantname.
     * @param restaurantname New value of property restaurantname.
     *
     */
    public void setRestaurantname(String restaurantname) {
        this.restaurantname = restaurantname;
    }

    /** Getter for property customername.
     * @return Value of property customername.
     *
```

```
     */
    public String getCustomername() {
        return this.customername;
    }

    /** Setter for property customername.
     * @param customername New value of property customername.
     *
     */
    public void setCustomername(String customername) {
        this.customername = customername;
    }

    /** Getter for property review.
     * @return Value of property review.
     *
     */
    public String getReview() {
        return this.review;
    }

    /** Setter for property review.
     * @param review New value of property review.
     *
     */
    public void setReview(String review) {
        this.review = review;
    }
}
```

DiningGuideManagerBean.java のソース

```
package Data;

import javax.ejb.*;
import javax.naming.*;

public class DiningGuideManagerBean implements javax.ejb.SessionBean {
    private javax.ejb.SessionContext context;
    private RestaurantHome myRestaurantHome;
    private CustomerreviewHome myCustomerreviewHome;

    /**
     * @see javax.ejb.SessionBean#setSessionContext(javax.ejb.SessionContext)
     */
    public void setSessionContext(javax.ejb.SessionContext aContext) {
        context=aContext;
    }

    /**
     * @see javax.ejb.SessionBean#ejbActivate()
     */
    public void ejbActivate() {

    }

    /**
     * @see javax.ejb.SessionBean#ejbPassivate()
     */
    public void ejbPassivate() {

    }

    /**
     * @see javax.ejb.SessionBean#ejbRemove()
     */
    public void ejbRemove() {
```

```
    }

    /**
     * See section 7.10.3 of the EJB 2.0 specification
     */
    public void ejbCreate() {
        System.out.println("Entering DiningGuideManagerEJB.ejbCreate()");
        Context c = null;
        Object result = null;
        if (this.myRestaurantHome == null) {
            try {
                c = new InitialContext();
                result = c.lookup("java:comp/env/ejb/Restaurant");
                myRestaurantHome =
(RestaurantHome)javax.rmi.PortableRemoteObject.narrow(result,
RestaurantHome.class);
            }
            catch (Exception e) {System.out.println("Error: "+
            e); }
        }
        Context crc = null;
        Object crresult = null;
        if (this.myCustomerreviewHome == null) {
            try {
                crc = new InitialContext();
                result = crc.lookup("java:comp/env/ejb/Customerreview");
                myCustomerreviewHome =
(CustomerreviewHome)javax.rmi.PortableRemoteObject.narrow(result,
CustomerreviewHome.class);
            }
            catch (Exception e) {System.out.println("Error: "+
            e); }
        }

    }

    public java.util.Vector getAllRestaurants() {
        System.out.println("Entering
DiningGuideManagerEJB.getAllRestaurants()");
        java.util.Vector restaurantList = new java.util.Vector();
        try {
```

```
            java.util.Collection rl = myRestaurantHome.findAll();
            if (rl == null) { restaurantList = null; }
            else {
                RestaurantDetail rd;
                java.util.Iterator rli = rl.iterator();
                while ( rli.hasNext() ) {
                    rd =((Restaurant)rli.next()).getRestaurantDetail();
                    System.out.println(rd.getRestaurantname());
                    System.out.println(rd.getRating());
                    restaurantList.addElement(rd);
                }
            }
        }
        catch (Exception e) {
            //ソースエディタで次の 2 行を 1 行にする
            System.out.println("Error in
DiningGuideManagerEJB.getAllRestaurants(): " + e);
        }
        //ソースエディタで次の 2 行を 1 行にする
      System.out.println("Leaving DiningGuideManagerEJB.getAllRestaurants()");
        return restaurantList;
    }

    public java.util.Vector getCustomerreviewsByRestaurant(java.lang.String
restaurantname) {
        System.out.println("Entering
DiningGuideManagerEJB.getCustomerreviewsByRestaurant()");
        java.util.Vector reviewList = new java.util.Vector();
        try {
            java.util.Collection rl =
myCustomerreviewHome.findByRestaurantName(restaurantname);
            if (rl == null) { reviewList = null; }
            else {
                CustomerreviewDetail crd;
                java.util.Iterator rli = rl.iterator();
                while ( rli.hasNext() ) {
                  crd = ((Customerreview)rli.next()).getCustomerreviewDetail();
                    System.out.println(crd.getRestaurantname());
                    System.out.println(crd.getCustomername());
                    System.out.println(crd.getReview());
                    reviewList.addElement(crd);
                }
```

```
            }
        }
        catch (Exception e) {
            //ソースエディタで次の 2 行を 1 行にする
            System.out.println("Error in
DiningGuideManagerEJB.getCustomerreviewsByRestaurant(): " + e);
        }
        //ソースエディタで次の 2 行を 1 行にする
        System.out.println("Leaving
DiningGuideManagerEJB.getCustomerreviewsByRestaurant()");
        return reviewList;
    }

    public void createCustomerreview(java.lang.String restaurantname,
java.lang.String customername, java.lang.String review) {
        System.out.println("Entering
DiningGuideManagerEJB.createCustomerreview()");
        try {
            Customerreview customerrev =
myCustomerreviewHome.create(restaurantname, customername,
            review);
        } catch (Exception e) {
            //ソースエディタで次の 2 行を 1 行にする
            System.out.println("Error in
DiningGuideManagerEJB.createCustomerreview(): " + e);
        }
        //ソースエディタで次の 2 行を 1 行にする
        System.out.println("Leaving
DiningGuideManagerEJB.createCustomerreview()");
    }

    public Data.RestaurantDetail getRestaurantDetail() {
        return null;
    }

    public Data.CustomerreviewDetail getCustomerreviewDetail() {
        return null;
    }
}
```

RestaurantTable.java のソース

```
/*
 * RestaurantTable.java
 *
 * Created on March 12, 2003, 4:29 PM
 */

package Client;
import javax.swing.table.*;
import java.util.*;
import WebService.DGWebServiceClientGenClient.*;
/**
 *
 * @author  administrator
 */
public class RestaurantTable extends javax.swing.JFrame {
    /** Creates new form RestaurantTable */
    public RestaurantTable() {
        initComponents();
        restaurantList=getAllRestaurants();
        putDataToTable();
    }
    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code.  The content of this method is
     * always regenerated by the Form Editor.
     */
    private void initComponents() {//GEN-BEGIN:initComponents
        jButton1 = new javax.swing.JButton();
        jScrollPane1 = new javax.swing.JScrollPane();
        jTable1 = new javax.swing.JTable();
        jLabel1 = new javax.swing.JLabel();
        getContentPane().setLayout(new
        org.netbeans.lib.awtextra.AbsoluteLayout());
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                exitForm(evt);
            }
```

```
        });
        jButton1.setText("View Customer Comments");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });
        getContentPane().add(jButton1, new
        org.netbeans.lib.awtextra.AbsoluteConstraints(200, 240, -1, -1));
        TableModel = (new javax.swing.table.DefaultTableModel(
        new Object [][] {
        },
        new String [] {
            "RESTAURANT NAME", "CUISINE", "NEIGHBORHOOD", "ADDRESS", "PHONE",
            "DESCRIPTION", "RATING"
        }
        ) {
            Class[] types = new Class [] {
                java.lang.String.class, java.lang.String.class,
                java.lang.String.class,

java.lang.String.class,java.lang.String.class,java.lang.String.class,java.lang
                .String.class
            };
            public Class getColumnClass(int columnIndex) {
                return types [columnIndex];
            }
        });
        jTable1.setModel(TableModel);
        jScrollPane1.setViewportView(jTable1);
        getContentPane().add(jScrollPane1, new
        org.netbeans.lib.awtextra.AbsoluteConstraints(0, 60, 600, 100));
        jLabel1.setText("Restaurant Listing");
        getContentPane().add(jLabel1, new
        org.netbeans.lib.awtextra.AbsoluteConstraints(230, 20, 110, 30));
        pack();
    }//GEN-END:initComponents
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton1ActionPerformed
        int r =jTable1.getSelectedRow();
        int c = jTable1.getSelectedColumnCount();
```

```
        String i =(String)TableModel.getValueAt(r,0);
        CustomerReviewTable crt = new CustomerReviewTable();
        crt.putDataToTable(i);
        crt.show();
        System.out.println(i);
    }//GEN-LAST:event_jButton1ActionPerformed
    /** Exit the Application */
    private void exitForm(java.awt.event.WindowEvent evt)
{//GEN-FIRST:event_exitForm
        System.exit(0);
    }//GEN-LAST:event_exitForm
    private void putDataToTable() {
        Iterator j=restaurantList.iterator();
        while (j.hasNext()) {
            RestaurantDetail ci = (RestaurantDetail)j.next();
            String strRating = null;
            String[] str =

{ci.getRestaurantname(),ci.getCuisine(),ci.getNeighborhood(),ci.getAddress(),
             ci.getPhone(),ci.getDescription(),
             strRating.valueOf(ci.getRating()),
            };
            TableModel.addRow(str);
        }
    }
    private Vector getAllRestaurants() {
        Vector restList = new Vector();
        try {
            WebService.DGWebServiceClientGenClient.DGWebService  service1 = new
             WebService.DGWebServiceClientGenClient.DGWebService_Impl();
           WebService.DGWebServiceClientGenClient.DGWebServiceServantInterface
port
            =
            service1.getDGWebServiceServantInterfacePort();
            restList = (java.util.Vector)port.getAllRestaurants();
        }
        catch (Exception ex) {
            System.err.println("Caught an exception." ); );
            ex.printStackTrace();
        }
        return restList;
    }
```

```
    private Vector getCustomerreviewByRestaurant(java.lang.String
    restaurantname) {
        Vector reviewList = new Vector();
        try {
            WebService.DGWebServiceClientGenClient.DGWebService  service2 = new
             WebService.DGWebServiceClientGenClient.DGWebService_Impl();
           WebService.DGWebServiceClientGenClient.DGWebServiceServantInterface
port
             =
             service2.getDGWebServiceServantInterfacePort();
             reviewList =

(java.util.Vector)port.getCustomerreviewsByRestaurant(restaurantname);
        }
        catch (Exception ex) {
            System.err.println("Caught an exception." ); );
            ex.printStackTrace();
        }
        return reviewList;
    }
    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        new RestaurantTable().show();
    }
    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton jButton1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTable jTable1;
    private javax.swing.JLabel jLabel1;
    // End of variables declaration//GEN-END:variables
    private DefaultTableModel TableModel;
    private java.util.Vector restaurantList = null;
}
```

CustomerReviewTable.java のソース

```
/*
 * CustomerReviewTable.java
 *
 * Created on March 12, 2003, 4:29 PM
 */

package Client;
import javax.swing.table.*;
import java.util.*;
import WebService.DGWebServiceClientGenClient.*;
/**
 *
 ** @author administrator
 */
public class CustomerReviewTable extends javax.swing.JFrame {
    /** Creates new form CustomerReviewTable */
    public CustomerReviewTable() {
        initComponents();
    }
    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code.  The content of this method is
     * always regenerated by the Form Editor.
     */
    private void initComponents() {//GEN-BEGIN:initComponents
        jScrollPane1 = new javax.swing.JScrollPane();
        jTable1 = new javax.swing.JTable();
        jButton1 = new javax.swing.JButton();
        customerNameLabel = new javax.swing.JLabel();
        customerNameField = new javax.swing.JTextField();
        reviewLabel = new javax.swing.JLabel();
        reviewField = new javax.swing.JTextField();
        jLabel1 = new javax.swing.JLabel();
        getContentPane().setLayout(new
        org.netbeans.lib.awtextra.AbsoluteLayout());
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
```

```
                exitForm(evt);
            }
        });
        TableModel = (new javax.swing.table.DefaultTableModel(
        new Object [][] {
        },
        new String [] {
            "CUSTOMER NAME", "REVIEW"
        }
        ) {
            Class[] types = new Class [] {
                java.lang.String.class,java.lang.String.class
            };
            public Class getColumnClass(int columnIndex) {
                return types [columnIndex];
            }
        });
        jTable1.setModel(TableModel);
        jScrollPane1.setViewportView(jTable1);
        getContentPane().add(jScrollPane1, new
        org.netbeans.lib.awtextra.AbsoluteConstraints(0, 60, 400, 100));
        jButton1.setText("Submit Customer Review");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });
        getContentPane().add(jButton1, new
        org.netbeans.lib.awtextra.AbsoluteConstraints(100, 250, 190, -1));
        customerNameLabel.setText("Customer Name");
        getContentPane().add(customerNameLabel, new
        org.netbeans.lib.awtextra.AbsoluteConstraints(40, 170, -1, -1));
        getContentPane().add(customerNameField, new
        org.netbeans.lib.awtextra.AbsoluteConstraints(153, 170, 170, -1));
        reviewLabel.setText("Review");
        getContentPane().add(reviewLabel, new
        org.netbeans.lib.awtextra.AbsoluteConstraints(40, 200, 80, -1));
        getContentPane().add(reviewField, new
        org.netbeans.lib.awtextra.AbsoluteConstraints(153, 200, 170, 20));
        jLabel1.setText("All Customer Review By Restaurant Name");
        getContentPane().add(jLabel1, new
```

```
        org.netbeans.lib.awtextra.AbsoluteConstraints(80, 10, 240, -1));
        pack();
    }//GEN-END:initComponents
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{//GENFIRST:event_jButton1ActionPerformed
        try {
            WebService.DGWebServiceClientGenClient.DGWebService  service1 = new
             WebService.DGWebServiceClientGenClient.DGWebService_Impl();
            WebService.DGWebServiceClientGenClient.DGWebServiceServantInterface
             port =
             service1.getDGWebServiceServantInterfacePort();
             port.createCustomerreview(RestaurantName,
             customerNameField.getText(),reviewField.getText());
        }
        catch (Exception ex) {
             System.err.println("Caught an exception." ); );
             ex.printStackTrace();
        }
        refreshView();
    }//GEN-LAST:event_jButton1ActionPerformed
    void refreshView() {
        try {
            while(TableModel.getRowCount()>0) {
                TableModel.removeRow(0);
            }
            putDataToTable(RestaurantName);
            repaint();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
    /** Exit the Application */
    private void exitForm(java.awt.event.WindowEvent evt)
{//GENFIRST:event_exitForm
        System.exit(0);
    }//GEN-LAST:event_exitForm
    public void putDataToTable(java.lang.String restaurantname) {
        RestaurantName = restaurantname;
       java.util.Vector customerList =getCustomerReviewByName(restaurantname);
        Iterator j=customerList.iterator();
```

```
        while (j.hasNext()) {
            CustomerreviewDetail ci = (CustomerreviewDetail)j.next();
            String[] str = {ci.getCustomername(),ci.getReview()
            };
            TableModel.addRow(str);
        }
    }
    private Vector getCustomerReviewByName(java.lang.String restaurantname) {
        Vector custList = new Vector();
        try {
            WebService.DGWebServiceClientGenClient.DGWebService  service2 = new
            WebService.DGWebServiceClientGenClient.DGWebService_Impl();
            WebService.DGWebServiceClientGenClient.DGWebServiceServantInterface
            port =
            service2.getDGWebServiceServantInterfacePort();
            custList =

(java.util.Vector)port.getCustomerreviewsByRestaurant(restaurantname);
        }
        catch (Exception ex) {
            System.err.println("Caught an exception." ); );
            ex.printStackTrace();
        }
        return custList;
    }
    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        new CustomerReviewTable().show();
    }
    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JLabel reviewLabel;
    private javax.swing.JButton jButton1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextField customerNameField;
    private javax.swing.JTable jTable1;
    private javax.swing.JLabel customerNameLabel;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JTextField reviewField;
    // End of variables declaration//GEN-END:variables
```

```
    private DefaultTableModel TableModel;
    private java.lang.String RestaurantName = null;
    //private java.util.Vector restaurantList = null;
}
```

付録B

DiningGuide のデータベーススクリプト

この付録では、DiningGuide アプリケーション用のデータベーススクリプトをまとめています。

PointBase データベース用のスクリプト

```
drop table CustomerReview;
drop table Restaurant;

create table Restaurant(
       restaurantName varchar(80),
       cuisine        varchar(25),
       neighborhood   varchar(25),
       address        varchar(30),
       phone          varchar(12),
       description    varchar(200),
       rating         tinyint,
constraint pk_Restaurant primary key(restaurantName));

create table CustomerReview(
       restaurantName varchar(80) not null references
Restaurant(restaurantName),
       customerName   varchar(25),
       review         varchar(200),
constraint pk_CustomerReview primary key(CustomerName, restaurantName));

insert into Restaurant (restaurantName, cuisine, neighborhood, address, phone,
description, rating) values ('French Lemon','Mediterranean','Rockridge','1200
College Avenue','510 888 8888','Very nice spot.',5);
insert into Restaurant (restaurantName, cuisine, neighborhood, address, phone,
description, rating) values ('Bay Fox','Mediterranean','Piedmont','1200
Piedmont Avenue','510 888 8888','Excellent.',5);

insert into CustomerReview (restaurantName, customerName, review) values
('French Lemon','Fred','Nice flowers.');
insert into CustomerReview (restaurantName, customerName, review) values
('French Lemon','Ralph','Excellent service.');
```

Oracle データベース用のスクリプト

```
drop table CustomerReview;
drop table Restaurant;

create table Restaurant(
       restaurantName varchar(80),
       cuisine        varchar(25),
       neighborhood   varchar(25),
       address        varchar(30),
       phone          varchar(12),
       description    varchar(200),
       rating         number(1,0),
constraint pk_Restaurant primary key(restaurantName));
grant all on Restaurant to public;

create table CustomerReview(
       restaurantName varchar(80) not null references
Restaurant(restaurantName),
       customerName   varchar(25),
       review         varchar(200),
constraint pk_CustomerReview primary key(CustomerName, restaurantName));
grant all on CustomerReview to public;

insert into Restaurant (restaurantName, cuisine, neighborhood, address, phone,
description, rating) values ('French Lemon','Mediterranean','Rockridge','1200
College Avenue','510 888 8888','Very nice spot.',5);
insert into Restaurant (restaurantName, cuisine, neighborhood, address, phone,
description, rating) values ('Bay Fox','Mediterranean','Piedmont','1200
Piedmont Avenue','510 888 8888','Excellent.',5);

insert into CustomerReview (restaurantName, customerName, review) values
('French Lemon','Fred','Nice flowers.');
insert into CustomerReview (restaurantName, customerName, review) values
('French Lemon','Ralph','Excellent service.');

commit;
```

付録C

Oracle データベースによるチュートリアルの作成

この付録では、Oracle データベースを使用して DiningGuide チュートリアルを作成、実行する手順を説明します。この付録の内容は次のとおりです

- この後の Oracle データベースとのデータベース接続の設定
- 173 ページの「データベース表の作成」
- 175 ページの「Oracle データベースによる EJB コンポーネントの作成」
- 176 ページの「Oracle データベースによる Web サービスの作成」

注 – このマニュアルには、「DiningGuide アプリケーションファイル」の名前を参照している個所が出てきます。DiningGuide アプリケーションファイルには、チュートリアルアプリケーションの完成版と、そのアプリケーションの実行方法を説明した readme ファイル、および必要なデータベース表を作成するための SQL スクリプトファイルが含まれます。これらのファイルを 1 つの zip 形式のファイルにまとめたものが Sun ONE Studio 5 Developer Resources ポータル (`http://forte.sun.com/ffj/documentation/tutorialsandexamples.html`) にあり、ダウンロードすることができます。

Oracle データベースとのデータベース接続の設定

アプリケーションサーバー環境で必要な JDBC 関連の作業を行うことによって、Sun ONE Application Server 7 が Oracle データベースに接続できるように設定します。次の作業を行います。

- データベースの JDBC ドライバを有効にする
- 接続プールの作成

- JDBC データソースの作成
- JDBC 持続マネージャの作成

Oracle の Type 4 JDBC ドライバを有効にする

JDBC ドライバを有効にするということは、Sun ONE Studio 5 および Sun ONE Application Server 7 のクラスパスにドライバのライブラリを追加することを意味します。このためには、Oracle Type 4 JDBC ドライバライブラリ (`classes12.zip`) が必要です。このドライバは、Oracle のポータルサイトからダウンロードできます。IDE を起動する前に、JDBC Type 4 ドライバを Sun ONE Studio 5 のプログラムファイルにコピーします。

Oracle の Type 4 JDBC ドライバを有効にする

1. **Oracle Type 4 ドライバライブラリを *s1studio-install-directory*`/lib/ext` ディレクトリにコピーします。**

   たとえば、`classes12.zip` ファイルを `c:\Sun\studio5_se\lib\ext` にコピーします。

注 – Sun ONE Studio 5 のホームディレクトリに書き込むには、スーパーユーザーまたは管理者特権が必要です。

---

2. **IDE を再起動します。**
3. **エクスプローラの「実行時」区画でアプリケーションサーバーインスタンスを選択します。**

   このインスタンスには *app-server-name* (*app-server-host*:*app-server-port*) の形式のラベルが付いています。たとえばデフォルトのサーバーは server1 (localhost:4848)、非特権ユーザーのサーバーは MyServer (localhost:4855) といったラベルです。

4. **アプリケーションサーバーインスタンスのプロパティを表示します。**

   通常、「プロパティ」ウィンドウはエクスプローラの下にあります。ノードを選択すると、ウィンドウにプロパティが表示されます。ウィンドウがない場合は、サーバーインスタンスのノードを右クリックして、「プロパティ」を選択します。

5. **「クラスパス接頭辞」プロパティのプロパティエディタを開きます。**

   プロパティの値フィールドをクリックして省略符号ボタン (...) を表示し、再度そのボタンをクリックします。「クラスパス接頭辞」エディタウィンドウが表示されます。

6. **「JAR/ZIP を追加」ボタンをクリックします。**

   「JAR ファイルを追加」ファイル検索を使用して、`classes12.zip` を検索します。

7. **`classes12.zip` ファイルを選択して、「了解」をクリックします。**

8. 「了解」を選択して「プロパティエディタ」ウィンドウを閉じます。

IDE と Oracle サーバーの接続

チュートリアルの JDBC 接続リソースまたは EJB を作成するには、IDE を Oracle データベースに接続する必要があります。これらのコンポーネントを作成する前に接続しても、作成の途中で接続してもかまいません。以下は、事前にデータベースに接続する方法を示しています。

1. Oracle サーバーが稼働していることを確認します。
2. エクスプローラの「実行時」区画で「データベース」ノードを展開し、その「ドライバ」サブノードを展開します。

   Oracle thin というノードが表示されます。

   Oracle JDBC ドライバが正しく有効にされていない場合、このノードには赤いストライクが付いています。その場合は、168 ページの「Oracle の Type 4 JDBC ドライバを有効にする」の手順に従ってください。
3. このノードを右クリックして、「接続」を選択します。

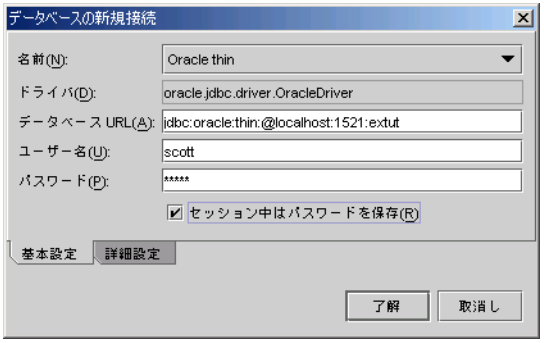
   「新規データベース接続」ダイアログが表示されます。
4. 「名前」フィールドで Oracle thin が選択状態であることを確認します。
5. 「データベース URL」「ユーザー」「パスワード」に適切なプロパティ値を入力します。

   たとえば、データベースがローカルにインストールされ、SID が「extut」で、デフォルトの Oracle ログインのユーザー名「scott」とパスワード「tiger」を使用している場合は、次の値を入力します。(1521 は標準の Oracle ポート番号です。)

名前	値
URL	jdbc:oracle:thin:@localhost:1521:extut
ユーザー	scott
パスワード	tiger

6. 「セッション中はパスワードを保存」を有効にします。

   「新規データベース接続」ダイアログが次のようになります。



7. 「了解」をクリックします。
8. 「ドライバ」ノードを閉じます。

   jdbc:oracle:thin:@*hostname*:1521:*sid* [*Username* on *Password*] という形式のラベルの「Oracle thin ドライバ」ノードが表示されます。
9. このノードを展開し、その「表」サブノードを展開します。

   データベース内の、RESTAURANT、CUSTOMERREVIEW などの表が表示されます。

JDBC 接続プールの作成

JDBC 接続プールを作成して、システム内のビジネスオブジェクトがデータベースアクセスを共有できるようにするには、まず、データベースに関係する情報を使用して JDBC 接続プールを定義して、Sun ONE Application Server 7に登録します。

注 – この作業に進む前に、管理サーバーとアプリケーションサーバーの両方が稼働していることを確認してください (3 ページの「ソフトウェアの起動」を参照)。

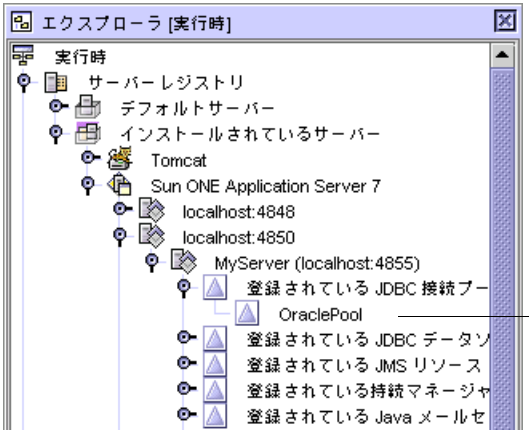
このチュートリアル用の Oracle JDBC 接続リソースを作成するには、次のようにします。

1. エクスプローラの「実行時」区画で「サーバーレジストリ」、「インストールされているサーバー」、「Sun ONE Application Server 7」の順にノードを展開します。
2. 「登録されていない JDBC 接続プール」ノードを右クリックし、「新規 JDBC 接続プールを追加」を選択します。

   新規 JDBC 接続プールウィザードが開きます。
3. JDBC 接続プール名として `OraclePool` を入力します。
4. 「既存の接続から抽出」オプションを有効にします。

5. プルダウンメニューから Oracle thin という文字列を選択します。
6. 「次へ」、そして「完了」をクリックします。
   このリソースを登録するかどうかを問い合わせるウィンドウが表示されます。
7. 「登録」ボタンをクリックします。
   「JDBC 接続プールの登録」ダイアログが表示されます。
8. 一覧から登録するサーバーインスタンスを選択して、「登録」をクリックします。
   接続プールが登録されると、正常に登録されたことを通知するメッセージが表示されます。

9. 「閉じる」ボタンをクリックしてウィンドウを閉じます。
   登録された OraclePool 接続プールが表示されます。



OraclePool 接続プールが表示されない場合は、「登録されている JDBC 接続プール」ノードを右クリックし、「リストを再表示」を選択します。

JDBC データソースの作成

JDBC データソース (JDBC リソースともいう) では、 getConnection() メソッドを使用してデータベースに接続することができます。データソースを作成する前に、管理サーバーとアプリケーションサーバーの両方が稼働していることを確認してください。

JDBC データソースを作成するには、次のようにします。

1. エクスプローラの「実行時」区画で「サーバーレジストリ」、「インストールされているサーバー」、「Sun ONE Application Server 7」の順にノードを展開します。
2. 「登録されていない JDBC データソース」ノードを右クリックし、「新規データソースを追加」を選択します。

   新規 JDBC リソースウィザードが開きます。
3. 「既存の JDBC データソースを使用」オプションを有効にして、一覧から `OraclePool` を選択します。
4. JNDI 名として **`jdbc/jdbc-oracle`** を入力し、「有効化」フィールドで true を選択します。
5. 「完了」をクリックします。

   このリソースを登録するかどうかを問い合わせるウィンドウが表示されます。
6. 「登録」ボタンをクリックします。

   「JDBC データソースの登録」ダイアログが表示されます。
7. 一覧から登録するサーバーインスタンスを選択して、「登録」をクリックします。

   データソースが登録されると、正常に登録されたことを通知するメッセージが表示されます。
8. 「閉じる」ボタンをクリックしてウィンドウを閉じます。

   「登録されている JDBC データソース」の下に「jdbc/jdbc-oracle」ノードが表示されます。表示されない場合は、「JDBC データソース」ノードを右クリックして、「リストを再表示」を選択します。

JDBC 持続マネージャの作成

持続マネージャは、コンテナにインストールされているエンティティ Bean の持続性を担当するコンポーネントです。持続マネージャを作成する前に、管理サーバーとアプリケーションサーバーの両方が稼働していることを確認してください。

JDBC 持続マネージャを作成するには、次のようにします。

1. エクスプローラの「実行時」区画で「サーバーレジストリ」、「インストールされているサーバー」、「Sun ONE Application Server 7」の順にノードを展開します。
2. 「登録されていない持続マネージャ」ノードを右クリックし、「持続マネージャを追加」を選択します。

   新規持続マネージャウィザードが開きます。
3. 「既存の JDBC データソースを使用」オプションを有効にして、一覧から `jdbc/jdbc-oracle` を選択します。
4. JNDI 名として **`jdo/OraclePm`** を入力し、「有効化」フィールドで true を選択します。
5. 「完了」をクリックします。

   このリソースを登録するかどうかを問い合わせるウィンドウが表示されます。
6. 「登録」ボタンをクリックします。

   「持続マネージャの登録」ダイアログが表示されます。
7. 一覧から登録するサーバーインスタンスを選択して、「登録」をクリックします。

   持続マネージャが登録されると、正常に登録されたことを通知するメッセージが表示されます。
8. 「閉じる」ボタンをクリックしてウィンドウを閉じます。

   「登録されている持続マネージャ」ノード下に「jdo/OraclePm」ノードが表示されます。表示されない場合は、ノードを右リックして、「リストを再表示」を選択します。

データベース表の作成

DiningGuide チュートリアルでは、データベース表を 2 つ使用します。これらのデータベース表は、Oracle Server データベースに手動で作成する必要があります。この後の手順では、付属の SQL スクリプトを使用して表を作成する方法を示します。Microsoft Windows ユーザーは、付録 B に提供されている SQL スクリプトをコピー、ペーストして、表を作成することができます。Solaris および Linux ユーザーは、DiningGuide アプリケーションファイルにあるスクリプトの `diningguide_ora.sql` を利用できます。

Microsoft Windows システムでチュートリアルの表を Oracle データベースにインストールするには、次のようにします。

1. 「スタート」→「プログラム」→「Oracle (実際のバージョン)」→「Application Development」→「SQL Plus」の順に選択し、Oracle コンソールを開きます。

2. **ユーザー名とパスワードを入力して SQL Plus にログインします。**

   たとえば、Oracle のデフォルトのインストールの場合は、ユーザー名 (scott) とパスワード (tiger) を使用します。

3. **SQL のプロンプトが表示されたら、 のスクリプトをコピーし、プロンプトの横にペーストします。**

参考 – 最初の 2 つの DROP 文は省きます。これらの文では、まだ作成されていない表が参照されるので、実行した場合、エラーが発生します。ただし、これらの DROP 文は、後でスクリプトを再実行して表を初期化するときには有用です。

Solaris または Linux 環境でチュートリアル用データベースをインストールするには、次のようにします。

1. **Developer Resources ポータルサイトからダウンロードした `DiningGuide.zip` ファイルを解凍します。**

   たとえば、`/MyZipFiles` ディレクトリに解凍します。

2. **コマンドプロンプトで、次のように入力します。**

```
$ cd your-unzip-dir/DiningGuide/db
$ sqlplus db-userid/db-password@db-servicename @diningguide_ora.sql
```

例 :

```
$ cd /MyZipFiles/DiningGuide/db
$ sqlplus scott/tiger@MyDB @diningguide_ora.sql
```

2 つの DROP 文でエラーが発生しますが、無視してかまいません。

Oracle データベースによる EJB コンポーネントの作成

この節では、Oracle データベースを使用して EJB 層を作成するために必要な作業を、第 3 章に対する変更点としてまとめています。対応する節とその変更点は、表 C-1 に示す通りです。

表 C-1　第 3 章に対する Oracle 用の変更点

節	変更点
36 ページの「チュートリアルの表のデータベーススキーマの作成」 の手順 2b	1. 「ドライバ」フォルダを展開し、「Oracle thin」ノードを右クリックして、「新規データベース接続」を選択します。 2. データベースで　Oracle URL、ユーザー名、およびパスワードを指定します。 3. 「セッション中はパスワードを保存」を有効にして、「了解」をクリックします。 4. 「ドライバ」フォルダを閉じます。Oracle 接続が表示されます。
同じ節の手順 5b	一覧から Oracle 接続を選択します。
38 ページの「Restaurant エンティティ Bean の作成」	Oracle スキーマを使用して Restaurant エンティティ Bean を作成します。
41 ページの「Customerreview エンティティ Bean の作成」	Oracle スキーマを使用して Customerreview エンティティ Bean を作成します。
59 ページの「Sun ONE Application Server 7 プラグインへのデータベース情報の指定」の手順 9	手順 9: 「JNDI 名」フィールドに `jdo/OraclePM` と入力します。 手順 10: Oracle データベース用のユーザー名とパスワードを入力します。
66 ページの「データベースへの追加内容の確認」	1. SQLPlus を起動します。 2. データベースにログインします。 3. SQL 文を入力します。
67 ページの「Customerreview Bean のテストクライアントの作成」	手順 8: 「JNDI 名」フィールドに `jdo/OraclePM` と入力します。 手順 9: Oracle データベース用のユーザー名とパスワードを入力します。
86 ページの「Sun ONE Application Server 7 プラグインへのデータベース情報の指定」	手順 9: 「JNDI 名」フィールドに `jdo/OraclePM` と入力します。 手順 10: Oracle データベース用のユーザー名とパスワードを入力します。

Oracle データベースによる Web サービスの作成

この節では、Oracle データベースを使用して Web サービスを作成するために必要な作業を、第 4 章に対する変更点としてまとめています。対応する節とその変更点は、表 C-2 に示す通りです。

表 C-2　第 4 章に対する Oracle 用の変更点

節	変更点
108 ページの「テストアプリケーションを使用した Web サービスのテスト」の手順 2	上記の説明に従って SQLPlus を起動してデータベースにログインし、レコードの挿入をテストします。

索引

E

J

N

O

P

R

S

W

あ

い

え

て

は

ひ

ふ

ほ

も