Borland AppServer™ 6.7

Development Plug-in for Eclipse ガイド

Borland Software Corporation
20450 Stevens Creek Blvd., Suite 800
Cupertino, CA 95014 USA
www.borland.com

BAS67WTPGuide
2006 年 12 月

Borland®

目次

第 1 章

Borland AppServer の概要

Borland AppServer（AppServer）は、企業環境において分散エンタープライズ アプリケーションの開発、デプロイメント、管理を行うための、サービスやツールのセットです。

AppServer は J2EE 1.4 標準の先進実装製品であり、EJB 2.1、JMS 1.1、Servlet 2.4、JSP 2.0、CORBA 2.6、XML、SOAP などの最新の業界標準技術をサポートします。ボーランドは、2 つのバージョンの AppServer を提供しており、これには、「Java メッセージング サービス（JMS）」に対する最先端のエンタープライズ メッセージング ソリューション（Tibco と OpenJMS）がそれぞれ同梱されています。ユーザーは、AppServer で必要とする機能やサービスのレベルを選択することができ、それらを変更する必要がある場合には、ライセンスをアップグレードすることにより容易に対応できます。

AppServer を利用することにより、J2EE 1.4 プラットフォーム標準を実装した分散 Java/CORBA アプリケーションを安全にデプロイし、さまざまな側面から管理することができます。

AppServer では、インストールごとのサーバー インスタンスの数は無制限です。そのため、同時接続ユーザーの数は無制限です。

AppServer は次のコンポーネントを備えています。

- J2EE 1.4 の実装。
- Apache Web Server バージョン 2.2.3。
- Borland Security。AppServer のセキュリティのためのフレームワークを提供します。
- 先進の集中管理型 JMS 管理ソリューション（Tibco および OpenJMS）。AppServer に同梱されています。
- 分散コンポーネントのための強力な管理ツール群。AppServer の外部で開発されたアプリケーションも含まれます。

AppServer の機能

AppServer では次の機能が提供されます：

- BAS プラットフォームに対するサポート（AppServer に対してサポートされているプラットフォームのリストについては、http://support.borland.com/kbcategory.jspa?categoryID=389 を参照してください）。
- クラスタリング トポロジーに対する完全サポート。
- VisiBroker ORB インフラストラクチャとのシームレスな統合。
- Borland JBuilder 統合開発環境（IDE）との統合。
- 他のボーランド製品（Borland Optimizeit Profiler や ServerTrace など）との統合の強化。
- AppServer により、既存のアプリケーションを Web サービスとして公開したり、新しいアプリケーションや追加 Web サービスと統合することができます。Borland Web サービスは、Apache Axis 1.2 テクノロジー（SOAP 1.2 をサポートする次世代 Apache SOAP サーバー）をベースとしています。

Borland AppServer のドキュメント

AppServer 関連のドキュメントには次のものがあります：

- **『Borland AppServer インストール ガイド』**：AppServer をネットワーク上にインストールする方法について説明されています。これは、Windows、UNIX の各オペレーティング システムに精通しているシステム管理者の方を対象に書かれています。
- **『Borland AppServer 開発者ガイド』**：運用環境における分散オブジェクト ベース アプリケーションのパッケージング、デプロイメント、管理についての詳細情報が記載されています。
- **『Borland 管理コンソール ユーザーズ ガイド』**：Borland 管理コンソール GUI の使用方法についての情報が記載されています。
- **『Borland セキュリティ ガイド』**：VisiSecure for VisiBroker for Java や VisiSecure for VisiBroker for C++ など、AppServer のセキュリティを確保するためのボーランドのフレームワークについて説明されています。
- **『Borland VisiBroker for Java 開発者ガイド』**：Java による VisiBroker アプリケーションの開発方法について説明されています。本書により VisiBroker ORB の設定と管理、プログラミング ツールの使用方法に精通できるよう、記載されています。また、IDL コンパイラ、スマート エージェント、ロケーション サービス、ネーミング サービス、イベント サービス、オブジェクト アクティベーション デーモン（OAD）、サービス品質（QoS: Quality of Service）、インターフェース リポジトリについても説明されています。
- **『Borland VisiBroker VisiTransact ガイド』**：OMG オブジェクト トランザクション サービス仕様に対するボーランドの実装、および、ボーランドのトランザクション サービス統合コンポーネントについて説明されています。

通常、ドキュメントにアクセスするには、AppServer 製品と共にインストールされるヘルプ ビューアを使用します。ユーザーは、スタンドアロンのヘルプ ビューアから、もしくは AppServer GUI ツールから、ヘルプを参照することができます。どちらの場合も、独立したウィンドウ内にヘルプ ビューアが起動されるため、ナビゲーション ペインを利用できるだけでなく、ナビゲーションや印刷のためのヘルプ ビューアのメイン ツールバーも利用することができます。ヘルプ ビューアのナビゲーション ペインには、すべての AppServer ドキュメントや参考ドキュメントの目次、インデックス、包括的な検索を実行できるページがあります。

PDF 形式の『Borland AppServer 開発者ガイド』や『Borland 管理コンソール ユーザーズ ガイド』は、http://info.borland.com/techpubs/appserver より入手可能です。

AppServer オンライン ヘルプ トピックへのアクセス

オンライン ヘルプにアクセスするには（次のいずれかの方法を利用）:

Windows の場合

- [スタート | すべてのプログラム | Borland AppServer | Help Topics] を選択。
- または、Web ブラウザを起動し、`<AppServer_Home>/doc/index.html` を開く。

UNIX の場合

- Web ブラウザを起動し、`<AppServer_Home>/doc/index.html` を開く。

AppServer GUI ツールから AppServer オンライン ヘルプ トピックへのアクセス

AppServer GUI ツールからオンライン ヘルプにアクセスするには（次のいずれかの方法を利用）:

- Borland 管理コンソールから、[Help | Help Topics] を選択。
- Borland デプロイメント ディスクリプタ エディタ（DDEditor）から、[Help | Help Topics] を選択。

ドキュメント表記規則

AppServer のドキュメントでは、文中の特定の部分を表すために、次の表に示す書体や記号を使用しています :

表記規則	用途
ボールド	新規の用語およびドキュメント名に使用されます。
`computer`	ユーザーやアプリケーションが提供する情報、 サンプル コマンドライン、 およびコードです。
`bold computer`	本文では、 ユーザーが入力する情報を示します。サンプル コードでは、 重要な文章を強調表示します。
[ ]	省略可能な項目であることを示します。
...	直前の引数が繰り返し可能であることを示します。
|	二者択一であることを示します。

プラットフォームの表記規則

AppServer のドキュメントでは、プラットフォーム固有の情報を表すために、次の記号を使用しています :

記号	意味
Windows	サポートされているすべての Windows プラットフォーム
Win2003	Windows 2003 のみ
WinXP	Windows XP のみ
Win2000	Windows 2000 のみ
UNIX	サポートされているすべての UNIX プラットフォーム
Solaris	Solaris のみ

Borland サポートへのお問い合わせ

ボーランド社は各種のサポート オプションを提供しています。それらには、インターネット上からの無償サービスもあり、大規模な情報データベースを検索したり、他のボーランド製品ユーザーからの情報を得たりすることが可能です。また、ボーランド製品のインストールに関するサポートから、有償のコンサルタント レベルのサポート、および高レベルなアシスタンスに至るまでの複数のカテゴリから、電話サポートの種類を選択できます。

ボーランドのサポート サービスについての詳細情報の入手や、実際にテクニカル サポートへお問い合わせいただくには、Web サイト http://support.borland.com を参照の上、製品をお使いになっている地域を選択してください。

ボーランド社のサポートへの連絡にあたっては、次の情報をご用意ください。

- 名前
- 会社名およびサイト ID
- 電話番号
- ユーザー ID（米国のみ）
- オペレーティング システムおよびバージョン
- ボーランド製品名およびバージョン
- 適用済みのパッチまたはサービス パック
- クライアントの言語とそのバージョン（使用している場合）
- データベースとそのバージョン（使用している場合）
- 発生した問題の詳細な内容と経緯
- 問題を示すログファイル
- 発生したエラー メッセージまたは例外の詳細な内容

オンライン リソース

ネットワーク上の次のサイトから情報を得ることができます。

ワールド ワイド ウェブ： http://www.borland.com
オンライン サポート： http://support.borland.com （ユーザー ID が必要）

ワールド ワイド ウェブ

http://www.borland.com は、定期的にご確認ください。AppServer 製品チームによる、ホワイト ペーパー、競合製品の分析、FAQ への回答、サンプル アプリケーション、更新ソフトウェア、更新ドキュメント、および新旧製品に関する情報が掲載されています。

特に、次の URL を確認されることをお勧めします：

- http://www.borland.com/downloads/download_appserver.html （AppServer ソフトウェアおよびその他のファイル）
- http://support.borland.com （AppServer FAQ）

Borland ニュースグループ

AppServer を対象とした数多くのスレッド化されたディスカッション グループに参加することができます。Enterprise Server やその他のボーランド製品に関する、ユーザー主体のニュースグループへ参加するには、http://www.borland.com/newsgroups を参照してください。

メモ これらのニュースグループはユーザーによって管理されているものであり、ボーランド社の公式サイトではありません。

第 2 章

Borland AppServer Development Plug-in のインストール

この章では、Borland AppServer (BAS) Development Plug-in for Eclipse のインストール方法について説明します。

インストールの前に

Borland AppServer Development Plug-in for Eclipse をインストールする前に、以下のソフトウェアがインストール済みであることを確認してください。

- Borland AppServer
- Eclipse 3.2（`http://download.eclipse.org/webtools/downloads/drops/R1.5/R-1.5.0-200606281455/`）
- WTP 1.5（`http://download.eclipse.org/webtools/downloads/drops/R1.5/R-1.5.0-200606281455/`）
- XDoclet 1.2.3（`http://sourceforge.net/project/showfiles.php?group_id=31602`）。`xdoclet-bin-1.2.3.zip` という名前のファイルをダウンロードします。
- JDK 5.0

BAS Development Plug-in のインストール

Borland AppServer Development Plug-in for Eclipse をインストールするには：

1. Eclipse を起動している場合は、プロジェクトを保存し、終了します。
2. <APPSERVER_HOME>/etc/eclipse ディレクトリから、以下のフォルダをコピーします。
   a. com.borland.bas.jst.server
   b. com.borland.enterprise.ui
   c. com.borland.enterprise.util
   d. com.borland.visibroker.sdk.core
3. これらのフォルダを <ECLIPSE_HOME>/plugins フォルダに貼り付けます。
4. Eclipse を起動します。

注意 Eclipse の起動後に、[Loading Class and Dependency File] ダイアログ ボックスにエラーが表示される場合は、<ECLIPSE_HOME>\configuration\org.eclipse.osgi から bundledata.<x> ファイルを削除してください。ここで、<x> は、bundledata ファイルのバージョン番号を表します。

メモ Eclipse で、XDoclet ビルダが有効になっており、XDoclet の設定が正しいことを確認してください。Eclipse で XDoclet ランタイムを設定するには、[ウィンドウ | 設定 ...] を選択し、[XDoclet] をクリックします。

第 3 章

概要

この章では、Borland AppServer Development Plug-in for Eclipse の概要について説明します。

はじめに

Borland AppServer Development Plug-in for Eclipse を利用すると、以下のことが可能になります。

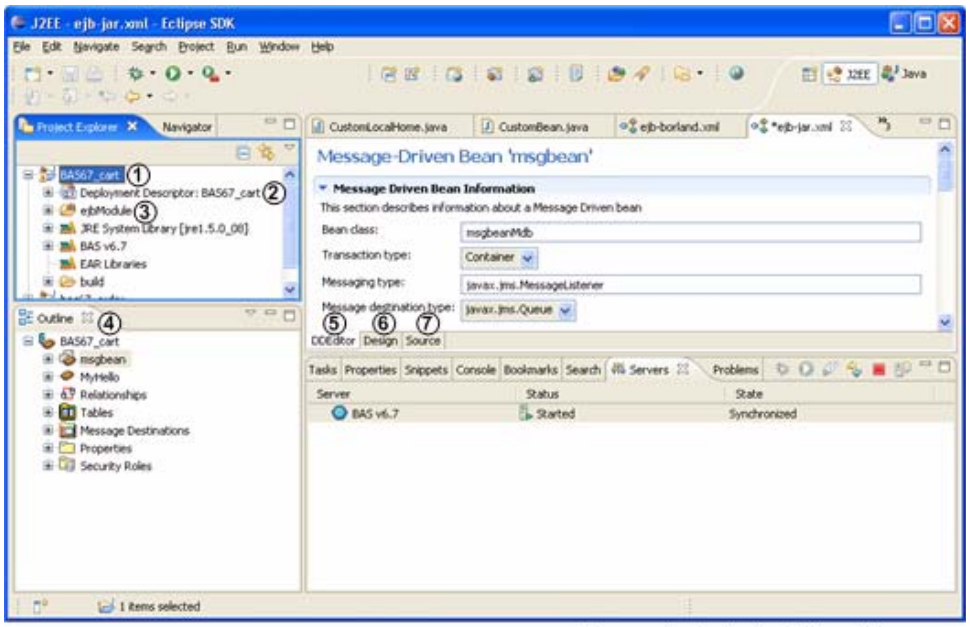
- Eclipse で Borland AppServer をインストール済みのランタイム環境として追加すること。Borland AppServer を新しいサーバーとして追加すると、Eclipse から Borland AppServer の起動と停止ができます。
- EJB プロジェクトでの作業。EJB プロジェクトでの作業の詳細については、14 ページの「EJB プロジェクトでの作業」を参照してください。
- Web プロジェクトでの作業。Web プロジェクトでの作業の詳細については、35 ページの「Web プロジェクトでの作業」を参照してください。
- EAR プロジェクトでの作業。EAR プロジェクトでの作業の詳細については、45 ページの「EAR プロジェクトでの作業」を参照してください。
- アプリケーション クライアント プロジェクトでの作業。アプリケーション クライアント プロジェクトでの作業の詳細については、49 ページの「アプリケーション クライアント プロジェクトでの作業」を参照してください。

ユーザー インターフェイスの理解

Borland AppServer Development Plug-in には、以下のビューがあります。

- プロジェクト エクスプローラ ビュー
- 標準のデプロイメント デスクリプタ ファイル用のエディタ ビュー。このエディタ ビューには、以下のビューが含まれています。
  - デプロイメント デスクリプタのエディタ ビュー
  - アウトライン ツリー ビュー

以下の画面は、Borland AppServer Development Plug-in for Eclipse をインストールした後の Eclipse のユーザー インターフェイスです。



① EJB プロジェクト
② 論理デプロイメント デスクリプタ ノード（ejb-jar.xml へのマップ）
③ ソース ディレクトリ
④ デプロイメント デスクリプタのアウトライン ツリー ビュー
⑤ BAS デプロイメント デスクリプタの エディタ ビュー
⑥ XML ツリー エディタ ビュー
⑦ XML エディタ ビュー

プロジェクト エクスプローラ ビュー

プロジェクト エクスプローラ ビューには、プロジェクトに関連するさまざまなファイルが表示されます。また、このビューを利用して、プロジェクトの検証、デプロイメント、エクスポートができます。

プロジェクト構造の理解

EJB プロジェクト

EJB プロジェクトには以下が含まれます。

- `META-INF` フォルダ。ここには、標準およびベンダーのデプロイメント デスクリプタ ファイルが含まれます。
  - `ejb-jar.xml`
  - `ejb-borland.xml`
- XDoclet 用のマージ ファイル。これらのマージ ファイルは、標準のデプロイメント デスクリプタ ファイル（`ejb-jar.xml`）にカスタム データをマージするために、Borland AppServer XDoclet Builder によって呼び出されます。
  - `session-beans.xml`
  - `entity-beans.xml`
  - `message-driven-beans.xml`

メモ これらの XDoclet 用のマージ ファイルは、変更しないことをお勧めします。

- `Backup` ディレクトリ。重要なビジネス ロジックは、しばしば抽象 Bean クラスに実装されます。EJB を削除する際に、貴重なコードが失われないように、Boroland AppServer Development Plug-in は、抽象 Bean クラスを `Backup` ディレクトリにバックアップします。

Web プロジェクト

Web プロジェクトには以下が含まれます。

- `WebContent\WEB-INF` ディレクトリ。ここには、標準およびベンダーのデプロイメント デスクリプタ ファイルが含まれます。
  - `web.xml`
  - `web-borland.xml`

DDEditor ビュー

DDEditor ビューは、デプロイメント デスクリプタ成果物を構成したり編集するための、ユーザー フレンドリなインターフェイスを提供します。DDEditor ビューを有効にするには、[DDEditor] タブをクリックします。

デプロイメント デスクリプタのアウトライン ツリー ビュー

デプロイメント デスクリプタのアウトライン ツリー ビューには、デプロイメント デスクリプタのナビゲーション ツリーが表示されます。デプロイメント デスクリプタのアウトライン ツリー ビューを表示するには、[ウィンドウ | ビューの表示 | アウトライン] を選択します。このビューを利用すると、以下の操作が可能になります。

- さまざまなデプロイメント デスクリプタ ファイルのモデリング。

  EJB プロジェクトのモデリングの詳細については、21 ページの「Bean 成果物のモデリング」を参照してください。

  Web プロジェクトのモデリングの詳細については、38 ページの「Web 成果物のモデリング」を参照してください。

  EAR プロジェクトのモデリングの詳細については、46 ページの「EAR 成果物のモデリング」を参照してください。

  アプリケーション クライアントのモデリングの詳細については、51 ページの「アプリケーション クライアント成果物のモデリング」を参照してください。
- プロジェクトの検証。プロジェクトの検証の詳細については、11 ページの「プロジェクトの検証」を参照してください。
- プロジェクトのエクスポート。プロジェクトのエクスポートの詳細については、12 ページの「プロジェクトのエクスポート」を参照してください。
- プロジェクトの再ロード。Borland AppServer Development Plug-in for Eclipse を利用すると、物理的なデプロイメント デスクリプタ ファイルに基づいて、デプロイメント デスクリプタ ツリーを再ロードできます。

  プロジェクトを再ロードするには、アウトライン ビューで、プロジェクトを右クリックし、[Reload] を選択します。
- プロジェクトの保存。Borland AppServer Development Plug-in for Eclipse を利用すると、暗黙の保存操作を実行できます。このオプションは、メモリ内のデータをディスクに永続化します。

  プロジェクトを保存するには、アウトライン ビューで、プロジェクトを右クリックし、[Save] を選択します。
- プロジェクトの修復。Borland AppServer Development Plug-in for Eclipse を利用すると、プロジェクトを修復できます。プロジェクトを修復するには、アウトライン ビューで、プロジェクトを右クリックし、[Fix Project] を選択します。

  [Fix Project] オプションを選択すると、Borland AppServer Development Plug-in は以下のタスクを実行します。

  - Borland AppServer 機能を、インポートされたプロジェクトに追加します。
  - プロジェクトに対する Borland AppServer 機能を修復します。たとえば、Borland AppServer Builder や Borland AppServer の特性。
  - プロジェクト用のマージ ファイル エントリを修復します。
  - メモリ内のデプロイメント デスクリプタ データをディスクに永続化します。
- Borland AppServer Development Plug-in for Eclipse のヘルプの表示。Borland AppServer Development Plug-in for Eclipse のヘルプを表示するには、アウトライン ツリー ビューで、プロジェクト ノードを右クリックし、[Borland AppServer Project Help] を選択します。

第 4 章

プロジェクトでの作業

この章では、Borland AppServer Development Plug-in を利用して、プロジェクトを作成、検証、エクスポート、再ロード、保存、および修復する方法について説明します。

プロジェクトの作成

新規プロジェクトを作成したり、既存のソース コードとデプロイメント デスクリプタ ファイルを利用してプロジェクトを作成できます。

EJB プロジェクトの作成については、14 ページの「EJB プロジェクトの作成」を参照してください。

Web プロジェクトの作成については、35 ページの「Web プロジェクトの作成」を参照してください。

EAR プロジェクトの作成については、45 ページの「EAR プロジェクトの作成」を参照してください。

アプリケーション クライアント プロジェクトの作成については、49 ページの「クライアント プロジェクトの作成」を参照してください。

プロジェクトの検証

Borland AppServer Development Plug-in for Eclipse を利用すると、デプロイメント デスクリプタ ファイルを検証したり、プロジェクトがデプロイメントできる状態かどうかを確認できます。このメニュー オプションは、Borland のデプロイメント デスクリプタ ファイルと標準のデプロイメント デスクリプタ ファイルを検証します。プロジェクトを検証すると、プロジェクトの検証中に検出されたエラーや警告が [Validation] ダイアログに表示されます。

プロジェクトを検証するには :

1. プロジェクト エクスプローラ ビューで、プロジェクトを右クリックし、[Borland | Validate] を選択します。

   または

   アウトライン ビューで、プロジェクトを右クリックし、[Validate] を選択します。

2. [Validation] ダイアログ ボックスに表示された検証結果を確認し、[OK] をクリックします。

プロジェクトのエクスポート

Borland AppServer Development Plug-in for Eclipse を利用すると、スタブやスケルトンを含む Borland AppServer モジュールを、アーカイブとしてエクスポートできます。

プロジェクトをエクスポートするには：

1. プロジェクト エクスプローラ ビューで、プロジェクトを右クリックし、[Borland ｜ Export As] を選択します。

   または

   アウトライン ビューで、プロジェクトを右クリックし、[Export] を選択します。
2. [EJB Module] ドロップダウン リストからエクスポートする EJB プロジェクトを選択します。
3. [Destination] フィールドに、その EJB モジュールの完全なパスと JAR ファイル名を入力します。
4. エクスポート先の JAR ファイルにソース ファイルを含めるには、[Export source files] チェック ボックスをオンにします。
5. 省略可能：既存の JAR ファイルにエクスポートしているときに、上書きに関する警告を表示したくない場合は、[Overwrite existing file] を選択します。
6. ［Finish］をクリックします。

プロジェクトの再ロード

Borland AppServer Development Plug-in for Eclipse を利用すると、デプロイメント デスクリプタ情報を再ロードしたり、物理的なデプロイメント デスクリプタ ファイルに基づいてプロジェクト全体を再ビルドできます。

プロジェクトを再ロードするには、アウトライン ビューで、プロジェクトを右クリックし、[Reload] を選択します。

プロジェクトの保存

Borland AppServer Development Plug-in for Eclipse を利用すると、暗黙の保存操作を実行できます。このオプションは、メモリ内のデータをディスクに永続化します。

プロジェクトを保存するには、アウトライン ビューで、プロジェクトを右クリックし、[Save] を選択します。

プロジェクトの修復

Borland AppServer Development Plug-in for Eclipse を利用すると、プロジェクトを修復できます。[Fix Project] オプションを選択すると、Borland AppServer Development Plug-in は以下のタスクを実行します。

- プロジェクトに対する Borland AppServer 機能を修復します。たとえば、Borland AppServer Builder や Borland AppServer の特性。
- プロジェクト用のマージ ファイル エントリを修復します。
- メモリ内のデプロイメント デスクリプタ データをディスクに永続化します。

プロジェクトを修復するには、アウトライン ビューで、プロジェクトを右クリックし、[Fix Project] を選択します。

第 5 章

EJB のモデリング

この章では、Borland AppServer Development Plug-in for Eclipse を利用して、EJB プロジェクトでの作業、Bean での作業、および EJB 成果物のモデリングを行う方法について説明します。

プロジェクト構造の理解

EJB プロジェクトには以下が含まれます。

- `META-INF` フォルダ。ここには、標準およびベンダーのデプロイメント デスクリプタ ファイルが含まれます。
  - `ejb-jar.xml`
  - `ejb-borland.xml`
- XDoclet 用のマージ ファイル。これらのマージ ファイルは、標準のデプロイメント デスクリプタ ファイル（`ejb-jar.xml`）にカスタム データをマージするために、Borland AppServer XDoclet Builder によって呼び出されます。
  - `session-beans.xml`
  - `entity-beans.xml`
  - `message-driven-beans.xml`
  - `relationship.xml`
  - `assembly-descriptor.xml`

メモ これらの XDoclet 用のマージ ファイルは、変更しないことをお勧めします。

- `Backup` ディレクトリ。ユーザーが EJB を削除する際に、Boroland AppServer Development Plug-in は、抽象 Bean クラスを `Backup` ディレクトリにバックアップします。

EJB プロジェクトでの作業

EJB プロジェクトの作成

新規の EJB プロジェクトを作成したり、既存のソース コードとデプロイメント デスクリプタ ファイルを利用して EJB プロジェクトを作成できます。

新規 EJB プロジェクトの作成

1 ［ファイル｜新規｜プロジェクト ...］を選択します。
［新規プロジェクト］ダイアログ ボックスが表示されます。

2 [EJB ｜ BAS EJB Project] を選択し、[次へ >] をクリックします。

3 [プロジェクト名] フィールドに、この EJB プロジェクトの名前を入力します。

4 [ターゲット・ランタイム] ドロップダウン リストから Borland AppServer 6.7 のランタイム ターゲットを選択します。たとえば、BAS v6.7。

5 ［終了］をクリックします。

既存のソース コードとデプロイメント デスクリプタ ファイルを利用した EJB プロジェクトの作成

1 ［ファイル｜新規｜プロジェクト ...］を選択します。
［新規プロジェクト］ダイアログ ボックスが表示されます。

2 [EJB ｜ BAS EJB Project] を選択し、[次へ >] をクリックします。

3 [プロジェクト名] フィールドに、この EJB プロジェクトの名前を入力します。

4 [ターゲット・ランタイム] ドロップダウン リストから BAS v6.7 を選択します。

5 この EJB プロジェクトを EAR プロジェクトに追加したい場合は、[EAR にプロジェクトを追加] をオンにして、[EAR プロジェクト名] ドロップダウン リストから EAR プロジェクト名を選択するか、[新規 ...] をクリックして、新規の EAR プロジェクトを作成します。EAR プロジェクトの作成については、45 ページの「EAR プロジェクトの作成」を参照してください。

6 ［終了］をクリックします。

7 既存のデスクリプタ ファイルを `<Workspace_Home>\<projectname>\<ejbModule>\META-INF` ディレクトリにコピーします。`<Workspace_Home>` は、このプロジェクト用の Eclipse のワークスペース ディレクトリ、`<projectname>` は手順 3 で指定したプロジェクトの名前、`<ejbModule>` はソース ディレクトリを表します。

8 既存のソース コードを `<Workspace_Home>\<projectname>\<ejbModule>` ディレクトリにコピーします。`<Workspace_Home>` は、このプロジェクト用の Eclipse のワークスペース ディレクトリ、`<projectname>` は手順 3 で指定したプロジェクトの名前、`<ejbModule>` はソース ディレクトリを表します。

9 プロジェクトを再ロードしてビルドするには、プロジェクトを選択し、[ファイル｜更新] を選択します。

10 プロジェクト エクスプローラ ビューで、作成したプロジェクトをダブルクリックします。

11 プロジェクト エクスプローラ ビューで、そのプロジェクトに属するデプロイメント デスクリプタをダブルクリックします。

12 アウトライン ビューで、ルート ノードを展開します。

13 ルート ノードを右クリックし、[保管] を選択し、デプロイメント デスクリプタ情報を確実にマージ ファイルに保存します。

Bean での作業

セッション Bean の作成

1 プロジェクト エクスプローラ ビューで、[デプロイメント記述子] ノードをダブルクリックします。

2 セッション Bean を作成するには、アウトライン ビューで、プロジェクト名を右クリックし、[New Session Bean] を選択します。

3 [プロジェクト] ドロップダウン リストから、新しいセッション Bean を含める予定のプロジェクトを選択します。

4 [フォルダー] フォルダに、新しい Bean 用のフォルダを入力します。

5 [Java パッケージ] フィールドに、新しい Bean のパッケージ名を入力します。

メモ 慣習に従って、パッケージ名は小文字で始めます。

6 [クラス名] フィールドに、エンタープライズ Bean の名前を入力します。

メモ 慣習に従って、クラス名は大文字で始めます。また、クラス名の接尾辞として "Bean" を使用します。Bean 名には Unicode 文字を使用できますが、エンタープライズ Bean パッケージとエンタープライズ Bean に関連するクラスに対しては、Unicode 文字はサポートされていません。

7 ［次へ］をクリックします。

8 [EJB 名] フィールドに、エンタープライズ Bean クラスの名前を入力します。

9 [宛先 JNDI 名] フィールドに、実行時のエンタープライズ Bean を検索するためにサーバーが使用する論理名を入力します。

10 [表示名] フィールドに、エンタープライズ Bean の短縮名を入力します。この名前はツールで使われます。

11 ［説明］フィールドに、Bean の説明を入力します。

12 [宛先] ドロップダウン リストから、この新しい Bean の宛先タイプを選択します。Bean の状態タイプの更新については、20 ページの「セッション Bean のセッション タイプの更新」を参照してください。

13 [トランザクション・タイプ] ドロップダウン リストから、"Container" を選択します。

14 ［終了］をクリックします。

エンティティ Bean の作成

1. プロジェクト エクスプローラ ビューで、[Deployment Descriptor] ノードをダブルクリックします。
2. アウトライン ビューで、プロジェクトを右クリックし、[New Entity Bean] を選択します。
3. [プロジェクト] ドロップダウン リストから、新しいエンティティ Bean を含む予定のプロジェクトを選択します。
4. [フォルダー] フォルダに、新しい Bean 用のフォルダを入力します。
5. [Java パッケージ] フィールドに、新しい Bean のパッケージ名を入力します。

メモ　慣習に従って、パッケージ名は小文字で始めます。

6. [クラス名] フィールドに、エンタープライズ Bean の名前を入力します。

メモ　慣習に従って、クラス名は大文字で始めます。また、クラス名の接尾辞として "Bean" を使用します。Bean 名には Unicode 文字を使用できますが、エンタープライズ Bean パッケージとエンタープライズ Bean に関連するクラスに対しては、Unicode 文字はサポートされていません。

7. ［次へ］をクリックします。
8. [EJB 名] フィールドに、エンタープライズ Bean クラスの名前を入力します。
9. [スキーマ] フィールドに、この Bean の抽象スキーマを指定するスキーマ名を入力します。
10. [表示名] フィールドに、エンタープライズ Bean の短縮名を入力します。この名前はツールで使われます。
11. ［説明］フィールドに、Bean の説明を入力します。
12. [CMP バージョン] ドロップダウン リストから、"2.x" を選択します。
13. この新しい Bean に対応するユースケースを選択します。
    a. [テーブルから属性をインポート] は、CMP エンティティ Bean 属性がデータベース テーブルからインポートされることを指定します。
       1. ［次へ］をクリックします。
       2. 利用可能な接続の定義を選択するには、[選択可能な接続の定義] リストの接続をクリックし、[次へ] をクリックします。

          新規の JDBC 接続の定義を作成するには、以下を実行します。
          - ［新規 ...］をクリックします。
          - 新規接続ウィザードにある [接続パラメーター] ページの必須 JDBC 接続パラメータを指定します。
          - データベース マネージャ、JDBC ドライバを選択し、その他の接続詳細情報を指定します。JDBC 接続フィルタを指定するには、[フィルターを使用不可にする] チェックボックスをオフにし、適切な接続フィルタを指定します。
          - ［終了］をクリックします。

    b. [Define new attributes] は、CMP エンティティ Bean 属性がユーザー定義であることを指定します。［Next］をクリックします。
       1. エンティティ Bean の CMP 属性を作成するには、[Add] をクリックします。
       2. 属性の名前を指定するには、[Name] フィールドをクリックし、名前を入力します。
       3. 属性のタイプを指定するには、[Type] フィールドをクリックし、タイプを入力します。
       4. 属性をエンティティ Bean のキー フィールドにするには、[Primary Key] チェックボックスをオンにします。
       5. このエンティティ Bean 用のテーブル名を指定するには、[Table] フィールドに名前を入力します。

6 さらに属性を追加するには、手順 1 から 4 を繰り返します。

14 ［次へ］をクリックします。

15 ［終了］をクリックします。

メッセージ Bean の作成

1 プロジェクト エクスプローラ ビューで、[デプロイメント記述子] ノードをダブルクリックします。

2 新規のメッセージ Bean を作成するには、アウトライン ビューで、プロジェクト名を右クリックし、[New Message Bean] を選択します。

3 [プロジェクト] ドロップダウン リストから、新しいメッセージ Bean を含む予定のプロジェクトを選択します。

4 [フォルダー] フォルダに、新しい Bean 用のフォルダを入力します。

5 [Java パッケージ] フィールドに、新しい Bean のパッケージ名を入力します。

メモ 慣習に従って、パッケージ名は小文字で始めます。

6 [クラス名] フィールドに、エンタープライズ Bean の名前を入力します。

メモ 慣習に従って、クラス名は大文字で始めます。また、クラス名の接尾辞として "Bean" を使用します。Bean 名には Unicode 文字を使用できますが、エンタープライズ Bean パッケージとエンタープライズ Bean に関連するクラスに対しては、Unicode 文字はサポートされていません。

7 ［次へ］をクリックします。

8 [EJB 名] フィールドに、エンタープライズ Bean クラスの名前を入力します。

9 [宛先 JNDI 名] フィールドに、実行時のエンタープライズ Bean を検索するためにサーバーが使用する論理名を入力します。

10 [表示名] フィールドに、エンタープライズ Bean の短縮名を入力します。この名前はツールで使われます。

11 ［説明］フィールドに、Bean の説明を入力します。

12 [宛先] ドロップダウン リストから、この新しい Bean の宛先タイプを選択します。

13 [トランザクション・タイプ] ドロップダウン リストから、"Container" を選択します。

14 ［終了］をクリックします。

Bean の名前変更

1 アウトライン ビューで、セッション Bean、メッセージ Bean、またはエンティティ Bean を右クリックし、[Rename] を選択します。

2 [New name] フィールドに、この Bean の新しい名前を入力します。

3 ［OK］をクリックします。

Bean の削除

1 アウトライン ビューで、セッション Bean、メッセージ Bean、またはエンティティ Bean を右クリックし、[Delete] を選択します。

［Delete Artifact］ダイアログ ボックスが表示されます。

2 ［Yes］をクリックします。

Beans のビルド

個別に Bean をビルドする場合は、XDoclet ビルダを強制的に実行し、抽象 Bean ファイル内の更新内容がさまざなインターフェイスとデプロイメント デスクリプタに公開されるようします。Bean をビルドするには、Bean を右クリックし、[Build] を選択します。

コンテナ管理の関係での作業

関係の作成

1 関係を右クリックし、[New Relation] を選択します。

2 [Relation name] フィールドに、この関係の名前を入力します。

3 [From EJB] ドロップダウン リストと [To EJB] ドロップダウン リストから適切な EJB を選択します。

4 ［OK］をクリックします。

Bean の CMR フィールドの定義

1 アウトライン ビューで、[Relationships] ノードを展開します。

2 自分が作成した EJB 関係を展開します。

3 関係のロールをクリックします。

4 ［Name］フィールドに、CMR フィールドの名前を入力します。

5 [Type] ドロップダウン リストから適切な CMR フィールド タイプを選択します。

6 ［Description］ボックスに、CMR フィールドの説明を入力します。

テーブル リファレンスの設定

1 アウトライン ビューで、[Relationships] ノードを展開します。

2 自分が作成した EJB 関係を展開します。

3 関係のロールを右クリックし、[Configure Table Reference] を選択します。以下の操作が実行できます。

   a テーブル リファレンスの関係の作成。それには、以下の手順を実行します。

      1 左側のテーブルの列を選択します。

      2 右側のテーブルの列を選択します。

      3 ［Link］をクリックします。

      4 ［OK］をクリックします。

   b クロス テーブルの追加。[Add Cross Table] ボタンを使用して、関係のない 2 つのテーブルを関係付けることができる中間テーブルを追加します。これは、しばしば多対多の関係の場合に実行します。

      1 左側のテーブルの列を選択します。

      2 [Add Cross table] をクリックします。

      3 [Cross table] ドロップダウン リストから適切なテーブルを選択します。

      4 右側のテーブルの列を選択します。

      5 [Link Cross Table] をクリックします。

      6 ［OK］をクリックします。

   c リンクの削除

      1 リストからリンクを選択します。

      2 ［Remove Link］をクリックします。

      3 ［OK］をクリックします。

   d クロス テーブルの削除

      1 リストからクロス テーブル リンクを選択します。

      2 [Remove Cross Table] をクリックします。

      3 ［OK］をクリックします。

ユーザー定義のビジネス メソッドの作成

メモ 新規のメソッドは、XDoclet を使用してソース コードを生成する EJB に対してのみ追加できます。

1 アウトライン ビューで、Bean を右クリックし、[new Method] を選択します。

   ［New Method］ダイアログ ボックスが表示されます。

2 適切な情報を入力し、[OK] をクリックします。

エンティティ Bean とセッション Bean のインターフェイス タイプの更新

デフォルトでは、新しいセッション Bean を作成すると、インターフェイス タイプはリモートに設定され、エンティティ Bean を作成すると、インターフェイス タイプはローカルに設定されます。

メモ インターフェイス タイプを更新できるのは、XDoclet を使用してソース コードを生成している場合だけです。

エンティティ Bean とセッション Bean のインターフェイス タイプを更新するには :

1. アウトライン ビューで、Bean を右クリックし、[Interface Type] を選択します。

   ［Change Interface Type］ダイアログ ボックスが表示されます。

2. 適切なインターフェイス タイプを選択します。
3. ［OK］をクリックします。

セッション Bean のセッション タイプの更新

デフォルトでは、新しいセッション Bean を作成すると、セッション タイプは "Stateless" に設定されます。

セッション Bean のセッション タイプを更新するには :

1. アウトライン ビューで、セッション Bean を右クリックし、[Session Type] を選択します。

   ［Change Session Type］ダイアログ ボックスが表示されます。

2. 適切なセッション タイプを選択します。
3. ［OK］をクリックします。

エンティティ Bean の永続性タイプの更新

デフォルトでは、新しいエンティティ Bean を作成すると、永続性タイプは "Container"（コンテナ管理の永続性）に設定されます。

エンティティ Bean の永続性タイプを更新するには :

1. アウトライン ビューで、エンティティ Bean を右クリックし、[Persistence Type] を選択します。

   ［Change Persistence Type］ダイアログ ボックスが表示されます。

2. 適切な永続性タイプを選択します。
3. ［OK］をクリックします。

Bean 成果物のモデリング

成果物の理解

エンティティ Bean の成果物

EJB ローカル リファレンス

ejb-local-ref 要素は、EJB コンテナによってローカルに解決できる EJB リファレンスを表します。

```
<xsd: element name="ejb-local-ref" type="borl:ejb-local-refType" minOccurs="0"
maxOccurs="unbounded"/>

<complexType name="ejb-local-refType">
   <sequence>
      <element name="ejb-ref-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string" minOccurs="0"/>
   </sequence>
</complexType>
```

例

```
<ejb-jar>
    <enterprise-beans>
        <session>
            <ejb-name>clerk</ejb-name>
            <bean-home-name>insurance/remote/clerk</bean-home-name>
            <timeout>5</timeout>
            <ejb-local-ref>
                 <ejb-ref-name>ejb/insurance/claim</ejb-ref-name>
            </ejb-local-ref>
            <resource-ref>
                 <res-ref-name>jms/insurance/ConnectionFactory</res-ref-name>
                 <jndi-name>jms/xacf</jndi-name>
            </resource-ref>
        </session>
        <entity>
              <ejb-name>claim</ejb-name>
              <bean-local-home-name>Claim</bean-local-home-name>
              ...
        </entity>
        ...
     </enterprise-beans>
</ejb-jar>
```

EJB リファレンス

この要素は、Bean によって使用される EJB リファレンスを定義するために使用されます。各 EJB リファレンスには、クライアント アプリケーションによって使用される ejb-ref-name と、それに関連付けられた jndi-name が含まれます。

```
<!ELEMENT ejb-ref (ejb-ref-name, jndi-name?)>
```

例

```
<ejb-ref>
    <ejb-ref-name>ejb/Sort</ejb-ref-name>
    <jndi-name>sort</jndi-name>
</ejb-ref>
```

メッセージの宛先リファレンス

この要素は、エンタープライズ Bean のコンテキスト内の JMS キューやトピックなど、メッセージの宛先リファレンスを定義するために使用されます。各メッセージ宛先リファレンスには、Bean によって使用される message-destination-ref-name と、関連付けられている jndi-name が含まれており、これによって JNDI ルックアップから目的のオブジェクトが解決されます。

```
<complexType name="message-destination-refType">
   <sequence>
      <element name="message-destination-ref-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string"/>
   </sequence>
</complexType>

<xsd: element name="message-destination-ref" type="borl:message-destination-
refType" minOccurs="0" maxOccurs="unbounded"/>
```

例

```
<ejb-jar>
       <enterprise-beans>
          <session>
             ...
             <message-destination-ref>
                 <message-destination-ref-name>jms/StockQueue</message-
destination-ref-name>
                 <jndi-name>jms/queues/Queue1</message-destination-type>
             </message-destination-ref>
             ...
          </session>
          ...
       </enterprise-beans>
   </ejb-jar>
```

リソース環境リファレンス

この要素は、Bean によって使用されるリソース環境リファレンスを定義するために使用されます。各リソース環境リファレンスには、Bean によって使用される res-env-ref-name と、それに関連付けられた jndi-name が含まれます。

```
<!ELEMENT resource-env-ref (res-env-ref-name, jndi-name?)>
```

例

```
<resource-env-ref>
    <res-env-ref-name>jdbc/CheckingDataSource</res-ref-name>
    <jndi-name>file:///net/machine/datasources/OracleDataSource</jndi-name>
</resource-env-ref>
```

リソース リファレンス

この要素は、Bean によって使用されるリソース リファレンスを定義するために使用されます。各リソース リファレンスには、クライアント アプリケーションによって使用される res-ref-name と、それに関連付けられた jndi-name が含まれます。

```
<!ELEMENT resource-ref (res-ref-name, jndi-name?)>
```

例

```
<resource-ref>
    <res-ref-name>jdbc/CheckingDataSource</res-ref-name>
    <jndi-name>file:///net/machine/datasources/OracleDataSource</jndi-name>
</resource-ref>
```

CMP フィールド

<cmp-field> 要素を使用して、基本的なフィールド マッピングが行われます。この要素の子ノードでは、フィールド名とマップ先の対応列を指定します。一般には粗粒度のエンティティ Bean を用いて、より細かい粒度のデータを表す Java クラスを実装します。3 番めの子ノード <cmp-field-map> は、細粒度クラスと、その基になるデータベース表現との間のフィールドのマップを定義し、<column-name> 要素の代わりに使用できます。

```
<!ELEMENT cmp-field (field-name, (cmp-field-map* | column-name),property*)>
```

例

```
<cmp-field>
   <field-name>orderNumber</field-name>
   <column-name>ORDER_NUMBER</column-name>
 </cmp-field>
```

CMP フィールドの再生成

a アウトライン ビューで、エンティティ Bean のノードを選択します。

b エンティティ Bean のノードを右クリックし、[Extract CMP Fields from Bean Class] を選択します。

これにより、この Bean クラスに基づく CMP フィールドが再生成されます。

ファインダ

この要素を使用して、エンティティ Bean が使用する検索メソッドを定義します。検索メソッドを生成する場合、実際には、where 節を持つ SQL select 文を生成することになります。select 文には、どのレコードまたはデータを検索して返すかを指定する節があります。コンテナ管理の永続性の下では、<finder> の子ノードを使って where 節の条件を指定する必要があります。

```
<!ELEMENT finder (method-signature, where-clause, load-state?)>
```

例

```
<finder>
    <method-signature>findByStudent(Student s)</method-signature>
    <where-clause>SELECT course_dept, course_number FROM
                  Enrollment WHERE student  = :s[ejb/Student]</where-clause>
    <load-state>False</load-state>
</finder>
```

プロパティ

この要素は、アーカイブまたはそのコンポーネントに含まれるか、それらから参照されるさまざまなリソースのプロパティ値を指定するために使用されます。各プロパティ エントリでは、対応するサブ要素を使用して、プロパティの名前、型、および値を指定します。

```
<!ELEMENT property (prop-name, prop-type, prop-value)>
```

例

```
<property>
    <prop-name>ejb.cacheCreate</prop-name>
    <prop-type>Boolean</prop-type>
    <prop-value>false</prop-value>
</property>
```

関係

テーブル間の関係を指定するには、<relationships> 要素を使用します。<relationships> 要素内で、ロールのソース（エンティティ Bean）を保持する <ejb-relationship-role> と、関係を保持する <cmr-field> 要素を定義します。デスクリプタは、<table-ref> 要素を使用して、2 つのテーブル（<left-table> と <right-table>）間の関係を指定します。次の基本原則を遵守しなければなりません。方向ごとに 1 つの <ejb-relationship-role> を定義しなければなりません。双方向の関係の場合は、相互に参照している Bean ごとに <ejb-relationship-role> を定義しなければなりません。関係 1 つにつき、許される <table-ref> 要素は 1 つだけです。多対多の関係を定義する場合は、CMP エンジンに、左側のテーブルと右側のテーブルの間の関係をモデル化するクロス テーブルを作成させる必要もあります。これは、<cross-table> 要素を使って実行します。

```
<!ELEMENT relationships (ejb-relation+)>
```

例

```
<relationships>
 <ejb-relation>
  <ejb-relationship-role>
   <relationship-role-source>
    <ejb-name>Customer</ejb-name>
   </relationship-role-source>
   <cmr-field>
    <cmr-field-name>specialInformation</cmr-field-name>
    <table-ref>
     <left-table>
      <table-name>CUSTOMER</table-name>
      <column-list>CUSTOMER_NO</column-list>
     </left-table>
     <right-table>
      <table-name>SPECIAL_INFO</table-name>
      <column-list>CUSTOMER_NO</column-list>
     </right-table>
    </table-ref>
   </cmr-field>
  </ejb-relationship-role>
  <ejb-relationship-role>
   <relationship-role-source>
    <ejb-name>SpecialInfo</ejb-name>
   </relationship-role-source>
  </ejb-relationship-role>
 </ejb-relation>
</relationships>
```

テーブル

CMP 2.x のエンティティ Bean が自分のフィールドにデータを格納するために使用するデータベース テーブルの名前を指定します。

```
<!ELEMENT table (#PCDATA)>
```

例

```
<table>Course</table>
```

メッセージの宛先

この要素は、JMS キューやトピックなどのメッセージの宛先を定義するために使用されます。この宛先は、アプリケーション コンポーネントの標準デスクリプタ内にある 1 つ以上の message-destination-ref または message-driven 要素の message-destination-link に対応します。各メッセージの宛先には、message-destination-link 値に対応する message-destination-name と、関連付けられている jndi-name が含まれており、これによって JNDI ルックアップから宛先オブジェクトが解決されます。

```
<element name="message-destination" type="borl:message-destinationType"
minOccurs="0" maxOccurs="unbounded"/>

<complexType name="message-destinationType">
   <sequence>
      <element name="message-destination-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string"/>
   </sequence>
</complexType>
```

例

```
<ejb-jar>
    ...
    <assembly-descriptor>
        ...
        <message-destination>
            <message-destination-name>myAppQueue</message-destination-name>
            <jndi-name>jms/queues/TibcoQueue1</jndi-name>
        </message-destination>
        ...
    </assembly-descriptor>
</ejb-jar>
```

セキュリティ ロール

アーカイブ内のモジュールで使用されるセキュリティ ロールとデプロイメント ロール（該当する場合）の名前を指定します。

```
<!ELEMENT security-role (role-name, deployment-role?)>
```

例

```
<security-role>
        <role-name>administrator</role-name>
        <deployment-role>administrator</deployment-role>
</security-role>
```

インターフェイス タイプ

エンティティ Bean は、リモート インターフェイスまたはローカル インターフェイスによって、メソッドを公開できます。リモート インターフェイスは、ネットワークを介してほかのリモート コンポーネントに Bean のメソッドを公開します。ローカル インターフェイスは、ローカル クライアント（同じ EJB コンテナにあるクライアント）にだけ Bean のメソッドを公開します。

メモ Bean のインターフェイス タイプを更新するたびに、デプロイメント デスクリプタ ファイルが再生成されます。

セッション Bean の成果物

EJB ローカル リファレンス

ejb-local-ref 要素は、EJB コンテナによってローカルに解決できる EJB リファレンスを表します。

```
<xsd: element name="ejb-local-ref" type="borl:ejb-local-refType" minOccurs="0"
maxOccurs="unbounded"/>

<complexType name="ejb-local-refType">
   <sequence>
      <element name="ejb-ref-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string" minOccurs="0"/>
   </sequence>
</complexType>
```

例

```
<ejb-jar>
    <enterprise-beans>
        <session>
            <ejb-name>clerk</ejb-name>
            <bean-home-name>insurance/remote/clerk</bean-home-name>
            <timeout>5</timeout>
            <ejb-local-ref>
                 <ejb-ref-name>ejb/insurance/claim</ejb-ref-name>
            </ejb-local-ref>
            <resource-ref>
                 <res-ref-name>jms/insurance/ConnectionFactory</res-ref-name>
                 <jndi-name>jms/xacf</jndi-name>
            </resource-ref>
        </session>
        <entity>
              <ejb-name>claim</ejb-name>
              <bean-local-home-name>Claim</bean-local-home-name>
              ...
        </entity>
        ...
     </enterprise-beans>
</ejb-jar>
```

EJB リファレンス

この要素は、Bean によって使用される EJB リファレンスを定義するために使用されます。各 EJB リファレンスには、クライアント アプリケーションによって使用される ejb-ref-name と、それに関連付けられた jndi-name が含まれます。

```
<!ELEMENT ejb-ref (ejb-ref-name, jndi-name?)>
```

例

```
<ejb-ref>
    <ejb-ref-name>ejb/Sort</ejb-ref-name>
    <jndi-name>sort</jndi-name>
</ejb-ref>
```

メッセージの宛先リファレンス

この要素は、エンタープライズ Bean のコンテキスト内の JMS キューやトピックなど、メッセージの宛先リファレンスを定義するために使用されます。各メッセージ宛先リファレンスには、Bean によって使用される message-destination-ref-name と、関連付けられている jndi-name が含まれており、これによって JNDI ルックアップから目的のオブジェクトが解決されます。

```
<complexType name="message-destination-refType">
   <sequence>
      <element name="message-destination-ref-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string"/>
   </sequence>
</complexType>

<xsd: element name="message-destination-ref" type="borl:message-destination-
refType" minOccurs="0" maxOccurs="unbounded"/>
```

例

```
<ejb-jar>
        <enterprise-beans>
           <session>
              ...
              <message-destination-ref>
                  <message-destination-ref-name>jms/StockQueue</message-
destination-ref-name>
                  <jndi-name>jms/queues/Queue1</message-destination-type>
              </message-destination-ref>
              ...
           </session>
           ...
        </enterprise-beans>
    </ejb-jar>
```

リソース環境リファレンス

この要素は、Bean によって使用されるリソース環境リファレンスを定義するために使用されます。各リソース環境リファレンスには、Bean によって使用される res-env-ref-name と、それに関連付けられた jndi-name が含まれます。

```
<!ELEMENT resource-env-ref (res-env-ref-name, jndi-name?)>
```

例

```
<resource-env-ref>
    <res-env-ref-name>jdbc/CheckingDataSource</res-ref-name>
    <jndi-name>file:///net/machine/datasources/OracleDataSource</jndi-name>
</resource-env-ref>
```

リソース リファレンス

この要素は、Bean によって使用されるリソース リファレンスを定義するために使用されます。各リソース リファレンスには、クライアント アプリケーションによって使用される res-ref-name と、それに関連付けられた jndi-name が含まれます。

```
<!ELEMENT resource-ref (res-ref-name, jndi-name?)>
```

例

```
<resource-ref>
    <res-ref-name>jdbc/CheckingDataSource</res-ref-name>
    <jndi-name>file:///net/machine/datasources/OracleDataSource</jndi-name>
</resource-ref>
```

テーブル

CMP 2.x のエンティティ Bean が自分のフィールドにデータを格納するために使用するデータベース テーブルの名前を指定します。

```
<!ELEMENT table (#PCDATA)>
```

例

```
<table>Course</table>
```

メッセージの宛先

この要素は、JMS キューやトピックなどのメッセージの宛先を定義するために使用されます。この宛先は、アプリケーション コンポーネントの標準デスクリプタ内にある 1 つ以上の message-destination-ref または message-driven 要素の message-destination-link に対応します。各メッセージ宛先には、message-destination-link 値に対応する message-destination-name と、関連付けられている jndi-name が含まれており、これによって JNDI ルックアップから宛先オブジェクトが解決されます。

```
<element name="message-destination" type="borl:message-destinationType"
minOccurs="0" maxOccurs="unbounded"/>

<complexType name="message-destinationType">
   <sequence>
      <element name="message-destination-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string"/>
   </sequence>
</complexType>
```

例

```
<ejb-jar>
    ...
    <assembly-descriptor>
        ...
        <message-destination>
            <message-destination-name>myAppQueue</message-destination-name>
            <jndi-name>jms/queues/TibcoQueue1</jndi-name>
        </message-destination>
        ...
    </assembly-descriptor>
</ejb-jar>
```

セキュリティ ロール

アーカイブ内のモジュールで使用されるセキュリティ ロールとデプロイメント ロール（該当する場合）の名前を指定します。

```
<!ELEMENT security-role (role-name, deployment-role?)>
```

例

```
<security-role>
        <role-name>administrator</role-name>
        <deployment-role>administrator</deployment-role>
   </security-role>
```

インターフェイス タイプ

セッション Bean は、リモート インターフェイスまたはローカル インターフェイスによって、メソッドを公開できます。リモート インターフェイスは、ネットワークを介してほかのリモート コンポーネントに Bean のメソッドを公開します。ローカル インターフェイスは、ローカル クライアント（同じ EJB コンテナにあるクライアント）にだけ Bean のメソッドを公開します。

メモ Bean のインターフェイス タイプを更新するたびに、デプロイメント デスクリプタ ファイルが再生成されます。

プロパティ

この要素は、アーカイブまたはそのコンポーネントに含まれるか、それらから参照されるさまざまなリソースのプロパティ値を指定するために使用されます。各プロパティ エントリでは、対応するサブ要素を使用して、プロパティの名前、型、および値を指定します。

```
<!ELEMENT property (prop-name, prop-type, prop-value)>
```

例

```
<property>
    <prop-name>ejb.security.transportType</prop-name>
    <prop-type>Enumerated</prop-type>
    <prop-value>CLEAR_ONLY</prop-value>
  </property>
```

メッセージ Bean の成果物

EJB ローカル リファレンス

ejb-local-ref 要素は、EJB コンテナによってローカルに解決できる EJB リファレンスを表します。

```
<xsd: element name="ejb-local-ref" type="borl:ejb-local-refType" minOccurs="0"
maxOccurs="unbounded"/>

<complexType name="ejb-local-refType">
   <sequence>
      <element name="ejb-ref-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string" minOccurs="0"/>
   </sequence>
</complexType>
```

例

```
<ejb-jar>
    <enterprise-beans>
        <session>
            <ejb-name>clerk</ejb-name>
            <bean-home-name>insurance/remote/clerk</bean-home-name>
            <timeout>5</timeout>
            <ejb-local-ref>
                 <ejb-ref-name>ejb/insurance/claim</ejb-ref-name>
            </ejb-local-ref>
            <resource-ref>
                 <res-ref-name>jms/insurance/ConnectionFactory</res-ref-name>
                 <jndi-name>jms/xacf</jndi-name>
            </resource-ref>
        </session>
        <entity>
              <ejb-name>claim</ejb-name>
              <bean-local-home-name>Claim</bean-local-home-name>
              ...
        </entity>
        ...
     </enterprise-beans>
</ejb-jar>
```

EJB リファレンス

この要素は、Bean によって使用される EJB リファレンスを定義するために使用されます。各 EJB リファレンスには、クライアント アプリケーションによって使用される ejb-ref-name と、それに関連付けられた jndi-name が含まれます。

```
<!ELEMENT ejb-ref (ejb-ref-name, jndi-name?)>
```

例

```
<ejb-ref>
    <ejb-ref-name>ejb/Sort</ejb-ref-name>
    <jndi-name>sort</jndi-name>
</ejb-ref>
```

メッセージの宛先リファレンス

この要素は、エンタープライズ Bean のコンテキスト内の JMS キューやトピックなど、メッセージの宛先リファレンスを定義するために使用されます。各メッセージ宛先リファレンスには、Bean によって使用される message-destination-ref-name と、関連付けられている jndi-name が含まれており、これによって JNDI ルックアップから目的のオブジェクトが解決されます。

```
<complexType name="message-destination-refType">
   <sequence>
      <element name="message-destination-ref-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string"/>
   </sequence>
</complexType>

<xsd: element name="message-destination-ref" type="borl:message-destination-
refType" minOccurs="0" maxOccurs="unbounded"/>
```

例

```
<ejb-jar>
        <enterprise-beans>
           <session>
              ...
              <message-destination-ref>
                  <message-destination-ref-name>jms/StockQueue</message-
destination-ref-name>
                  <jndi-name>jms/queues/Queue1</message-destination-type>
              </message-destination-ref>
              ...
           </session>
           ...
        </enterprise-beans>
    </ejb-jar>
```

リソース環境リファレンス

この要素は、Bean によって使用されるリソース環境リファレンスを定義するために使用されます。各リソース環境リファレンスには、Bean によって使用される res-env-ref-name と、それに関連付けられた jndi-name が含まれます。

```
<!ELEMENT resource-env-ref (res-env-ref-name, jndi-name?)>
```

例

```
<resource-env-ref>
    <res-env-ref-name>jdbc/CheckingDataSource</res-ref-name>
    <jndi-name>file:///net/machine/datasources/OracleDataSource</jndi-name>
</resource-env-ref>
```

リソース リファレンス

この要素は、Bean によって使用されるリソース リファレンスを定義するために使用されます。各リソース リファレンスには、クライアント アプリケーションによって使用される res-ref-name と、それに関連付けられた jndi-name が含まれます。

```
<!ELEMENT resource-ref (res-ref-name, jndi-name?)>
```

例

```
<resource-ref>
    <res-ref-name>jdbc/CheckingDataSource</res-ref-name>
    <jndi-name>file:///net/machine/datasources/OracleDataSource</jndi-name>
</resource-ref>
```

テーブル

CMP 2.x のエンティティ Bean が自分のフィールドにデータを格納するために使用するデータベース テーブルの名前を指定します。

```
<!ELEMENT table (#PCDATA)>
```

例

```
<table>Course</table>
```

メッセージの宛先

この要素は、JMS キューやトピックなどのメッセージの宛先を定義するために使用されます。この宛先は、アプリケーション コンポーネントの標準デスクリプタ内にある 1 つ以上の message-destination-ref または message-driven 要素の message-destination-link に対応します。各メッセージ宛先には、message-destination-link 値に対応する message-destination-name と、関連付けられている jndi-name が含まれており、これによって JNDI ルックアップから宛先オブジェクトが解決されます。

```
<element name="message-destination" type="borl:message-destinationType"
minOccurs="0" maxOccurs="unbounded"/>

<complexType name="message-destinationType">
   <sequence>
      <element name="message-destination-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string"/>
   </sequence>
</complexType>
```

例

```
<ejb-jar>
    ...
    <assembly-descriptor>
        ...
        <message-destination>
            <message-destination-name>myAppQueue</message-destination-name>
            <jndi-name>jms/queues/TibcoQueue1</jndi-name>
        </message-destination>
        ...
    </assembly-descriptor>
</ejb-jar>
```

セキュリティ ロール

アーカイブ内のモジュールで使用されるセキュリティ ロールとデプロイメント ロール（該当する場合）の名前を指定します。

```
<!ELEMENT security-role (role-name, deployment-role?)>
```

例

```
<security-role>
        <role-name>administrator</role-name>
        <deployment-role>administrator</deployment-role>
   </security-role>
```

プロパティ

この要素は、アーカイブまたはそのコンポーネントに含まれるか、それらから参照されるさまざまなリソースのプロパティ値を指定するために使用されます。各プロパティ エントリでは、対応するサブ要素を使用して、プロパティの名前、型、および値を指定します。

```
<!ELEMENT property (prop-name, prop-type, prop-value)>
```

例

```
<property>
    <prop-name>ejb.mdb.maxMessagesPerServerSession</prop-name>
    <prop-type>Integer</prop-type>
    <prop-value>1</prop-value>
</property>
```

成果物インスタンスの作成

1 アウトライン ビューで、成果物を右クリックし、[New <artifact_name>] を選択します。ここで、artifact_name は作成する成果物の名前を表します。たとえば、新しい EJB リファレンスを作成したい場合は、EJB リファレンスを右クリックし、[New EJB Reference] を選択します。

ダイアログ ボックスが表示されます。

2 新規の成果物の名前を入力します。

3 ［OK］をクリックします。

成果物インスタンスの名前変更

1 アウトライン ビューで、名前変更したいインスタンスを含む成果物のノードを展開します。たとえば、EJB リファレンス成果物のインスタンスの名前を変更したい場合は、[EJB References] ノードを展開します。

2 成果物インスタンスを右クリックし、[Rename] を選択します。たとえば、[EJB References] ノードの下に EJB_Reference_1 という名前のインスタンスがある場合は、EJB_Reference_1 を右クリックし、[Rename] を選択する必要があります。

ダイアログ ボックスが表示されます。

3 新しい名前を入力します。

4 ［OK］をクリックします。

エンティティ Bean の削除

1 アウトライン ビューで、エンティティ Bean を右クリックし、[Delete] を選択します。

［Delete Artifact］ダイアログ ボックスが表示されます。

2 ［Yes］をクリックします。

成果物の削除

1 アウトライン ビューで、削除したいインスタンスを含む成果物のノードを展開します。たとえば、EJB リファレンス成果物のインスタンスを削除したい場合は、[EJB References] ノードを展開します。

2 成果物インスタンスを右クリックし、[Delete] を選択します。たとえば、[EJB References] ノードの下に EJB_Reference_1 という名前のインスタンスがある場合は、EJB_Reference_1 を右クリックし、[Delete] を選択します。

［Delete Artifact］ダイアログ ボックスが表示されます。

3 ［Yes］をクリックします。

第 6 章

Web のモデリング

この章では、Borland AppServer Development Plug-in for Eclipse を利用して、Web プロジェクトでの作業、サーブレットでの作業、および Web 成果物のモデリングを行う方法について説明します。

プロジェクト構造の理解

Web プロジェクトには以下が含まれます。

- `WebContent\WEB-INF` ディレクトリ。ここには、標準およびベンダーのデプロイメント デスクリプタ ファイルが含まれます。
  - `web.xml`
  - `web-borland.xml`

Web プロジェクトでの作業

Web プロジェクトの作成

新規の Web プロジェクトを作成したり、既存のソース コードとデプロイメント デスクリプタ ファイルを利用して Web プロジェクトを作成できます。

新規 Web プロジェクトの作成

1. ［ファイル｜新規｜プロジェクト ...］を選択します。
   ［新規プロジェクト］ダイアログ ボックスが表示されます。
2. [Web ｜ BAS Dynamic Web Project] を選択し、[次へ >] をクリックします。
3. [プロジェクト名] フィールドに、この Web プロジェクトの名前を入力します。
4. [ターゲット・ランタイム] ドロップダウン リストから Borland AppServer 6.7 のランタイム ターゲットを選択します。たとえば、BAS v6.7。
5. ［終了］をクリックします。

既存のソース コードとデプロイメント デスクリプタ ファイルを利用した Web プロジェクトの作成

1 ［ファイル｜新規｜プロジェクト ...］を選択します。
［新規プロジェクト］ダイアログ ボックスが表示されます。

2 [Web ｜ BAS Dynamic Web Project] を選択し、[次へ >] をクリックします。

3 [プロジェクト名] フィールドに、この Web プロジェクトの名前を入力します。

4 [ターゲット・ランタイム] ドロップダウン リストから Borland AppServer 6.7 のランタイム ターゲットを選択します。たとえば、BAS v6.7。

5 ［終了］をクリックします。

6 既存のデスクリプタ ファイルを `<Workspace_Home>\<projectname>\WebContent\WEB-INF` ディレクトリにコピーします。`<Workspace_Home>` は、このプロジェクト用の Eclipse のワークスペース ディレクトリ、`<projectname>` は手順 3 で指定したプロジェクトの名前です。

7 既存の JSP ソースと HTML ソースを `<Workspace_Home>\<projectname>\WebContent` ディレクトリにコピーします。`<Workspace_Home>` は、このプロジェクト用の Eclipse のワークスペース ディレクトリ、`<projectname>` は手順 3 で指定したプロジェクトの名前です。

既存の Java ソースを `<Workspace_Home>\<projectname>\Src` ディレクトリにコピーします。`<Workspace_Home>` は、このプロジェクト用の Eclipse のワークスペース ディレクトリ、`<projectname>` は手順 3 で指定したプロジェクトの名前です。

8 プロジェクトを再ロードしてビルドするには、プロジェクトを選択し、[ファイル｜更新] を選択します。

9 プロジェクト エクスプローラ ビューで、作成したプロジェクトをダブルクリックします。

10 プロジェクト エクスプローラ ビューで、そのプロジェクトに属するデプロイメント デスクリプタをダブルクリックします。

11 アウトライン ビューで、ルート ノードを展開します。

12 ルート ノードを右クリックし、[保管] を選択し、デプロイメント デスクリプタ情報を確実にマージ ファイルに保存します。

サーブレットでの作業

サーブレットの作成

1 プロジェクト エクスプローラ ビューで、[デプロイメント記述子] ノードをダブルクリックします。

2 アウトライン ビューで、プロジェクト名を右クリックし、[新規｜サーブレット] を選択します。

3 [プロジェクト] ドロップダウン リストから、新しいサーブレットを含む予定のプロジェクトを選択します。

4 [フォルダー] フィールドに、サーブレット クラスを置くフォルダを入力します。

5 [Java パッケージ] フィールドに、新しいサーブレットのパッケージ名を入力します。

メモ 慣習に従って、パッケージ名は小文字で始めます。

6 [クラス名] フィールドに、サーブレットのクラス名を入力します。

メモ 慣習に従って、クラス名は大文字で始めます。

7 [スーパークラス] フィールドに、このサーブレット クラスのスーパー クラスを入力します。

8 ［次へ］をクリックします。

9 このサーブレットの初期化パラメータを、名前 - 値のペアとして入力します。

10 [URL マッピング] ボックスに、このサーブレットにマッピングされる URL 文字列を入力します。

11 ［次へ］をクリックします。

12 ［終了］をクリックします。

ブラウザ JSP ファイルの追加

1 プロジェクト エクスプローラ ビューで、WebContent フォルダを右クリックし、[新規 ｜ JSP] を選択します。

[新規 Java Server Page] ウィンドウが、フォルダが選択された状態で表示されます。

2 [ファイル名] フィールドに、この JSP ファイルのファイル名を入力します。

メモ ファイル名には、必ず JSP 拡張子を含めてください。

3 新しい JSP ファイルにデフォルト値を適用するには、[終了] をクリックします。

ファイル システム内のファイルにリンクして、パス変数を指定するには、[拡張] をクリックします。

JSP ページの初期内容にテンプレート ファイルを使用するには、手順 4 ～ 6 を実行します。

4 ［次へ］をクリックします。

[JSP テンプレートの選択] ウィンドウが表示されます。

5 [JSP テンプレートの使用] チェック ボックスをオンにし、サンプル テンプレートの 1 つを選択します。

6 ［終了］をクリックします。

サーブレットの名前変更

1 アウトライン ビューで、サーブレットを右クリックし、[Rename] を選択します。

2 [New name] フィールドに、このサーブレットの新しい名前を入力します。

3 ［OK］をクリックします。

サーブレットの削除

1 アウトライン ビューで、サーブレットを右クリックし、[Delete] を選択します。
［Delete Artifact］ダイアログ ボックスが表示されます。

2 ［Yes］をクリックします。

Web 成果物のモデリング

成果物の理解

EJB リファレンス

この要素は、Web アプリケーションによって使用される EJB リファレンスを定義するために使用されます。各 EJB リファレンスには、アプリケーションによって使用される ejb-ref-name と、それに関連付けられた jndi-name が含まれます。

```
<!ELEMENT ejb-ref (ejb-ref-name, jndi-name?)>
```

例

```
<ejb-ref>
    <ejb-ref-name>ejb/Sort</ejb-ref-name>
    <jndi-name>sort</jndi-name>
</ejb-ref>
```

リソース環境リファレンス

この要素は、Web アプリケーションで使用されるリソース環境リファレンスを、JNDI 内の名前にマップするために使用されます。各リソース環境リファレンスには、Bean によって使用される res-env-ref-name と、それに関連付けられた jndi-name が含まれます。

```
<!ELEMENT resource-env-ref (res-env-ref-name, jndi-name?)>
```

例

```
<resource-env-ref>
    <res-env-ref-name>jdbc/CheckingDataSource</res-ref-name>
    <jndi-name>file:///net/machine/datasources/OracleDataSource</jndi-name>
</resource-env-ref>
```

リソース リファレンス

この要素は、Web アプリケーションによって使用されるリソース リファレンスを定義するために使用されます。各リソース リファレンスには、アプリケーションによって使用される res-ref-name と、それに関連付けられた jndi-name が含まれます。

```
<!ELEMENT resource-ref (res-ref-name, jndi-name?)>
```

例

```
<resource-ref>
    <res-ref-name>jdbc/CheckingDataSource</res-ref-name>
    <jndi-name>file:///net/machine/datasources/OracleDataSource</jndi-name>
</resource-ref>
```

メッセージの宛先リファレンス

この要素は、JMS キューやトピックなど、メッセージの宛先リファレンスを定義するために使用されます。各メッセージの宛先リファレンスには、アプリケーションによって使用される message-destination-ref-name と、それに関連付けられた jndi-name が含まれます。

```
<element name="message-destination-ref" type="borl:message-destination-refType"
minOccurs="0" maxOccurs="unbounded"/>


<complexType name="message-destination-refType">
   <sequence>
      <element name="message-destination-ref-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string"/>
   </sequence>
</complexType>
```

例

```
<web-app>
        ...
        <message-destination-ref>
           <message-destination-ref-name>jms/StockQueue</message-destination-
ref-name>
           <jndi-name>jms/queues/Queue1</message-destination-type>
        </message-destination-ref>
        ...
</web-app>
```

メッセージの宛先

この要素は、JMS キューやトピックなどのメッセージの宛先を定義するために使用されます。この宛先は、Web アプリケーション内にある 1 つ以上の message-destination-ref 要素の message-destination-link に対応します。各メッセージの宛先には、message-destination-link 値に一致する message-destination-name と、関連付けられている jndi-name が含まれます。

```
<element name="message-destination" type="borl:message-destinationType"
minOccurs="0" maxOccurs="unbounded"/>

<complexType name="message-destinationType">
   <sequence>
      <element name="message-destination-name" type="xsd:string"/>
      <element name="jndi-name" type="xsd:string"/>
   </sequence>
</complexType>
```

例

```
<web-app>
    ...
    <message-destination>
        <message-destination-name>myAppQueue</message-destination-name>
        <jndi-name>jms/queues/TibcoQueue1</jndi-name>
    </message-destination>
    ...
</web-app>
```

Web デプロイメント パス

Tomcat の server.xml ファイルでは、特定のサービスの下にある 1 つ以上のエンジンの下に、1 つ以上のホストを定義できます。Tomcat コンテナの下の Web アプリケーションのデプロイメント場所を正確に指定したい場合は、この要素を使用します。

```
<web-deploy-path> <!ELEMENT web-deploy-path (service, engine, host)>
```

例

```
<web-deploy-path>
    <service>tomcatX</service>
    <engine>cyrpi</engine>
    <host>it3</host>
</web-deploy-path>
```

セキュリティ ロール

Web アプリケーションのロール（web.xml にある）を、Borland Enterprise Server の deployment-role にマッピングします。

```
<!ELEMENT security-role (role-name, deployment-role?)>
```

例

```
<security-role>
        <role-name>administrator</role-name>
        <deployment-role>administrator</deployment-role>
</security-role>
```

成果物インスタンスの作成

1 アウトライン ビューで、成果物を右クリックし、[New <artifact_name>] を選択します。ここで、artifact_name は作成する成果物の名前を表します。たとえば、新規の EJB リファレンスを作成したい場合は、EJB リファレンスを右クリックし、[New EJB Reference] を選択します。

ダイアログ ボックスが表示されます。

2 新規の成果物の名前を入力します。

3 ［OK］をクリックします。

成果物インスタンスの名前変更

1 アウトライン ビューで、名前変更したいインスタンスを含む成果物のノードを展開します。たとえば、EJB リファレンス成果物のインスタンスの名前を変更したい場合は、[EJB References] ノードを展開します。

2 成果物インスタンスを右クリックし、[Rename] を選択します。たとえば、[EJB References] ノードの下に EJB_Reference_1 という名前のインスタンスがある場合は、EJB_Reference_1 を右クリックし、[Rename] を選択する必要があります。

ダイアログ ボックスが表示されます。

3 新しい名前を入力します。

4 ［OK］をクリックします。

成果物の削除

1 アウトライン ビューで、削除したいインスタンスを含む成果物のノードを展開します。たとえば、EJB リファレンス成果物のインスタンスを削除したい場合は、[EJB References] ノードを展開します。

2 成果物インスタンスを右クリックし、[Delete] を選択します。たとえば、[EJB References] ノードの下に EJB_Reference_1 という名前のインスタンスがある場合は、EJB_Reference_1 を右クリックし、[Delete] を選択します。

［Delete Artifact］ダイアログ ボックスが表示されます。

3 ［Yes］をクリックします。

第 7 章

Eclipse での Borland AppServer Development Plug-in の設定

この章では、Eclipse での Borland AppServer (BAS) Development Plug-in の設定方法について説明します。

Borland AppServer をインストール済みのサーバー ランタイム環境として追加

Borland AppServer Development Plug-in for Eclipse をインストールしたら、Eclipse で、Borland AppServer をインストール済みのサーバー ランタイム環境として追加した後に、新規サーバーを定義する必要があります。

Borland AppServer をインストール済みのサーバー ランタイム環境として追加するには :

1. Eclipse で、[ウィンドウ | 設定 ...] を選択します。
   [設定] ダイアログ ボックスが表示されます。
2. 左ペインから、[サーバー | インストール済みランタイム] を選択します。
3. ［追加 ...］をクリックします。
   [新規サーバー・ランタイム] ダイアログ ボックスが表示されます。
4. [Borland | BAS v6.7] を選択し、[次へ] をクリックします。
5. [Borland AppServer インストール・ディレクトリー] フィールドに、Borland AppServer をインストールしたパスを入力するか、[参照 ...] をクリックしてパスを参照します。
6. すべてのフィールドに指定した値が適切であることを確認し、[終了] をクリックします。
7. BAS v6.7 が、[インストール済みサーバー・ランタイム環境] リストに表示されます。
   ［OK］をクリックします。

Eclipse での新規サーバーの定義

Eclipse で Borland AppServer をサーバーとして定義すると、Eclipse から Borland AppServer の起動と停止ができるようになります。

Eclipse で新規サーバーを定義するには :

1 ［ファイル｜新規｜その他 ...］を選択します。
[新規] ダイアログ ボックスが表示されます。

2 [サーバー｜サーバー] を選択し、[次へ >] をクリックします。

3 [Borland ｜ BAS v6.7] を選択し、[終了] をクリックします。

ローカル Borland AppServer の Eclipse からの起動と停止

1 J2EE パースペクティブ ビューに切り替えます。

2 サーバー ビューを表示します。

3 Borland AppServer サーバーを起動するには、サーバー ビューで、Borland AppServer サーバーを右クリックし、[始動] を選択します。
Borland AppServer サーバーを停止するには、サーバー ビューで、Borland AppServer サーバーを右クリックし、[停止] を選択します。

第 8 章

EAR のモデリング

この章では、Borland AppServer development plug-in for Eclipse を利用して、EAR プロジェクトでの作業、モジュールの依存関係での作業、および EAR 成果物のモデリングを行う方法について説明します。

EAR プロジェクトでの作業

EAR プロジェクトの作成

1 ［ファイル｜新規｜プロジェクト ...］を選択します。
［新規プロジェクト］ダイアログ ボックスが表示されます。

2 [J2EE ｜ BAS Enterprise Application Project] を選択し、[次へ >] をクリックします。

3 [プロジェクト名] フィールドに、この EAR プロジェクトの名前を入力します。

4 [ターゲット・ランタイム] ドロップダウン リストから Borland AppServer 6.7 のランタイム ターゲットを選択します。たとえば、BAS v6.7。

5 ［次へ >］をクリックします。
[プロジェクト・ファセット] 画面が表示されます。

6 ［次へ >］をクリックします。

7 この新しい EAR プロジェクトに対して、J2EE モジュールを選択、選択解除、または追加します。

8 ［終了］をクリックします。

J2EE モジュールの依存関係の追加

1 プロジェクト エクスプローラ ビューで、EAR プロジェクトを右クリックし、[プロパティー] を選択します。

2 左ペインから、[J2EE モジュール依存関係] を選択します。

3 モジュールの依存関係を、選択、選択解除、または追加します。

4 ［適用］をクリックします。

5 ［OK］をクリックします。

EAR 成果物のモデリング

成果物の理解

モジュール

この要素は、同一コンテナタイプ内、またはそのタイプのデプロイメント デスクリプタ内で実行される 1 つ以上のコンポーネントのコレクションを表します。モジュール要素には、サブ要素として ejb、java、web のいずれか、および hosts サブ要素が必要です。

```
<xsd:element name="module" minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:choice>
            <xsd:element name="connector" type="xsd:string"/>
            <xsd:element name="ejb" type="xsd:string"/>
            <xsd:element name="java" type="xsd:string"/>
            <xsd:element name="web">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element name="web-uri" type="xsd:string"/>
                  </xsd:sequence>
               </xsd:complexType>
            </xsd:element>
         </xsd:choice>
            <xsd:element name="hosts" type="xsd:string" minOccurs="0"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

例

```
<application>
   <module>
      <ejb>my-ejb.jar</ejb>
   </module>
   <module>
      <web>
         <web-uri>myweb.war</web-uri>
      </web>
   </module>
   ...
</application>
```

プロパティ

この要素は、実行時にアプリケーションに必要なプロパティ値を指定するために使用されます。各プロパティ エントリでは、対応するサブ要素を使用して、プロパティの名前、型、および値を指定します。

```
<xsd:element name="property" minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="prop-name" type="xsd:string"/>
         <xsd:element name="prop-type" type="xsd:string" minOccurs="0"/>
         <xsd:element name="prop-value" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

例

```
<application>
   <module>
      <ejb>my-ejb.jar</ejb>
   </module>
   <module>
      <web>
         <web-uri>myweb.war</web-uri>
      </web>
   </module>
   <property>
      <prop-name>vbroker.security.disable</prop-name>
      <prop-type>security</prop-type>
      <prop-value>false</prop-value>
   </property>
</application>
```

セキュリティ ロール

アプリケーションのロール（`application.xml` にある）を Borland AppServer の deployment-role にマップします。

```
<xsd:element name="security-role" minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="role-name" type="xsd:string"/>
         <xsd:element name="deployment-role" type="xsd:string" minOccurs="0"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

例

```
<security-role>
   <role-name>administrator</role-name>
   <deployment-role>administrator</deployment-role>
</security-role>
```

成果物インスタンスの作成

1 アウトライン ビューで、成果物を右クリックし、[New <artifact_name>] を選択します。ここで、artifact_name は作成する成果物の名前を表します。たとえば、新しいセキュリティ ロールを作成したい場合は、セキュリティ ロールを右クリックし、[New Security Role] を選択します。

ダイアログ ボックスが表示されます。

2 新規の成果物の名前を入力します。

3 ［OK］をクリックします。

成果物インスタンスの名前変更

1 アウトライン ビューで、名前変更したいインスタンスを含む成果物のノードを展開します。たとえば、セキュリティ ロール成果物のインスタンスの名前を変更したい場合は、[Security Role] ノードを展開します。

2 成果物インスタンスを右クリックし、[Rename] を選択します。たとえば、[Security Role] ノードの下に、Security_Role_1 という名前のインスタンスがある場合、Security_Role_1 を右クリックし、[Rename] を選択します。

ダイアログ ボックスが表示されます。

3 新しい名前を入力します。

4 ［OK］をクリックします。

エンティティ Bean の削除

1 アウトライン ビューで、エンティティ Bean を右クリックし、[Delete] を選択します。

［Delete Artifact］ダイアログ ボックスが表示されます。

2 ［Yes］をクリックします。

成果物の削除

1 アウトライン ビューで、削除したいインスタンスを含む成果物のノードを展開します。たとえば、セキュリティ ロール成果物のインスタンスを削除したい場合は、[Security Role] ノードを展開します。

2 成果物インスタンスを右クリックし、[Delete] を選択します。たとえば、[Security Role] ノードの下に、Security_Role_1 という名前のインスタンスがある場合、Security_Role_1 を右クリックし、[Delete] を選択します。

［Delete Artifact］ダイアログ ボックスが表示されます。

3 ［Yes］をクリックします。

第 9 章

アプリケーション クライアントのモデリング

この章では、Borland AppServer Development Plug-in for Eclipse を利用して、アプリケーション クライアント プロジェクトでの作業、およびアプリケーション クライアント成果物のモデリングを行う方法について説明します。

アプリケーション クライアント プロジェクトでの作業

クライアント プロジェクトの作成

1 ［ファイル｜新規｜プロジェクト ...］を選択します。
［新規プロジェクト］ダイアログ ボックスが表示されます。

2 [J2EE ｜ BAS Application Client Project] を選択し、[次へ >] をクリックします。

3 [プロジェクト名] フィールドに、このアプリケーション クライアント プロジェクトの名前を入力します。

4 [ターゲット・ランタイム] ドロップダウン リストから Borland AppServer 6.7 のランタイム ターゲットを選択します。たとえば、BAS v6.7。

5 ［次へ >］をクリックします。
[プロジェクト・ファセット] 画面が表示されます。

6 ［次へ >］をクリックします。

7 [ソース・フォルダー] フィールドに、ソース フォルダ名を入力します。

8 ［終了］をクリックします。

メモ クライアント アプリケーション（たとえば、サンプル EJB クライアント アプリケーション）を実行したい場合は、サーバー ランタイム スタブ（たとえば、`server.jar`）がクライアント プロジェクトに含まれていることを確認します。

既存のソース コードとデプロイメント デスクリプタ ファイルを利用したクライアント プロジェクトの作成

1 ［ファイル｜新規｜その他 ...］を選択します。
［新規プロジェクト］ダイアログ ボックスが表示されます。

2 [J2EE ｜ BAS Application Client Project] を選択し、[次へ >] をクリックします。

3 [プロジェクト名] フィールドに、このプロジェクトの名前を入力します。

4 [ターゲット・ランタイム] ドロップダウン リストから BAS v6.7 を選択します。

5 このプロジェクトを EAR プロジェクトに追加したい場合は、[EAR にプロジェクトを追加] をオンにして、[EAR プロジェクト名] ドロップダウン リストから EAR プロジェクト名を選択するか、[新規] をクリックして、新規の EAR プロジェクトを作成します。EAR プロジェクトの作成については、45 ページの「EAR プロジェクトの作成」を参照してください。

6 ［終了］をクリックします。

7 既存のデスクリプタ ファイルを `<Workspace_Home>\<projectname>\<clientModule>\META-INF` ディレクトリにコピーします。`<Workspace_Home>` は、このプロジェクト用の Eclipse のワークスペース ディレクトリ、`<projectname>` は手順 3 で指定したプロジェクトの名前、`<clientModule>` はソース ディレクトリを表します。

8 既存のソース コードを `<Workspace_Home>\<projectname>\<clientModule>` ディレクトリにコピーします。`<Workspace_Home>` は、このプロジェクト用の Eclipse のワークスペース ディレクトリ、`<projectname>` は手順 3 で指定したプロジェクトの名前、`<clientModule>` はソース ディレクトリを表します。

9 プロジェクトを再ロードしてビルドするには、プロジェクトを選択し、[ファイル｜更新] を選択します。

10 プロジェクト エクスプローラ ビューで、作成したプロジェクトをダブルクリックします。

11 プロジェクト エクスプローラ ビューで、そのプロジェクトに属するデプロイメント デスクリプタをダブルクリックします。

12 アウトライン ビューで、ルート ノードを展開します。

13 ルート ノードを右クリックし、[保管] を選択し、デプロイメント デスクリプタ情報を確実にマージ ファイルに保存します。

アプリケーション クライアント成果物のモデリング

成果物の理解

EJB リファレンス

この要素は、クライアントによって使用される EJB リファレンスを定義するために使用されます。各 EJB リファレンスには、クライアント アプリケーションによって使用される ejb-ref-name と、それに関連付けられた jndi-name（該当する場合）が含まれます。

```
<xsd: element name="ejb-ref" type="borl:ejb-refType" minOccurs="0"
maxOccurs="unbounded"/>

<xsd:complexType name="ejb-refType">
  <xsd:sequence>
   <xsd:element name="ejb-ref-name" type="xsd:string"/>
   <xsd:element name="jndi-name" type="xsd:string" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

例

```
<ejb-ref>
    <ejb-ref-name>ejb/Sort</ejb-ref-name>
    <jndi-name>sort</jndi-name>
</ejb-ref>
```

メッセージの宛先リファレンス

この要素は、JMS キューやトピックなど、メッセージの宛先リファレンスを定義するために使用されます。各メッセージの宛先リファレンスには、クライアント アプリケーションによって使用される message-destination-ref-name と、それに関連付けられた jndi-name が含まれます。

```
<xsd:element name="message-destination-ref" type="borl:message-destination-
refType" minOccurs="0" maxOccurs="unbounded"/>

<xsd:complexType name="message-destination-refType">
  <xsd:sequence>
   <xsd:element name="message-destination-ref-name" type="xsd:string"/>
   <xsd:element name="jndi-name" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

例

```
<application-client>
        ...
        <message-destination-ref>
           <message-destination-ref-name>jms/StockQueue</message-destination-
ref-name>
           <jndi-name>jms/queues/Queue1</message-destination-type>
        </message-destination-ref>
        ...
</application-client>
```

メッセージの宛先

この要素は、JMS キューやトピックなどのメッセージの宛先を定義するために使用されます。この宛先は、アプリケーション クライアント内にある 1 つ以上の message-destination-ref 要素の message-destination-link に対応します。各メッセージの宛先には、message-destination-link 値に一致する message-destination-name と、関連付けられている jndi-name が含まれます。

```
<xsd:element name="message-destination" type="borl:message-destinationType"
minOccurs="0" maxOccurs="unbounded"/>

<xsd:complexType name="message-destinationType">
  <xsd:sequence>
   <xsd:element name="message-destination-name" type="xsd:string"/>
   <xsd:element name="jndi-name" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

例

```
<application-client>
        ...
        <message-destination>
           <message-destination-name>myAppQueue</message-destination-name>
           <jndi-name>jms/queues/TibcoQueue</jndi-name>
        </message-destination>
        ...
</application-client>
```

リソース環境リファレンス

この要素は、クライアントによって使用されるリソース環境リファレンスを定義するために使用されます。各リソース環境リファレンスには、クライアント アプリケーションによって使用される resource-env-ref-name と、それに関連付けられた jndi-name （該当する場合）が含まれます。resource-env-ref-name 要素は、標準のデプロイメント デスクリプタからリソース環境リファレンスを一意に識別します。

```
<element name="resource-env-ref" type="borl:resource-env-refType" minOccurs="0"
maxOccurs="unbounded"/>

<complexType name="resource-env-refType">
   <sequence>
      <element name="resource-env-ref-name" type="xsd:string"/>
      <element ref="borl:jndi-name"/>
   </sequence>
</complexType>
```

例

```
<application-client>
        ...
        <resource-env-ref>
            <resource-env-ref-name>jms/StockQueue</resource-env-ref-name>
            <jndi-name>jms/Queue1</jndi-name>
        </resource-env-ref>
        ...
</application-client>
```

リソース リファレンス

この要素は、クライアントによって使用されるリソース リファレンスを定義するために使用されます。各リソース リファレンスには、クライアント アプリケーションによって使用される res-ref-name と、それに関連付けられた jndi-name（該当する場合）が含まれます。res-ref-name 要素は、標準のデプロイメント デスクリプタからリソース リファレンスを一意に識別します。

```
<xsd:element name="resource-ref" type="borl:resource-refType" minOccurs="0"
maxOccurs="unbounded"/>

<xsd:complexType name="resource-refType">
  <xsd:sequence>
   <xsd:element name="res-ref-name" type="xsd:string"/>
   <xsd:element name="jndi-name" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

例

```
<application-client>
     ...
        <resource-ref>
           <res-ref-name>jdbc/SavingsDataSource</res-ref-name>
           <jndi-name>jdbc/datasources/Oracle</jndi-name>
        </resource-ref>
  ...
</application-client>
```

成果物インスタンスの作成

1 アウトライン ビューで、成果物を右クリックし、[New <artifact_name>] を選択します。ここで、artifact_name は作成する成果物の名前を表します。たとえば、新規の EJB リファレンスを作成したい場合は、EJB リファレンスを右クリックし、[New EJB Reference] を選択します。

ダイアログ ボックスが表示されます。

2 新規の成果物の名前を入力します。

3 ［OK］をクリックします。

成果物インスタンスの名前変更

1 アウトライン ビューで、名前変更したいインスタンスを含む成果物のノードを展開します。たとえば、EJB リファレンス成果物のインスタンスの名前を変更したい場合は、[EJB References] ノードを展開します。

2 成果物インスタンスを右クリックし、[Rename] を選択します。たとえば、[EJB References] ノードの下に EJB_Reference_1 という名前のインスタンスがある場合は、EJB_Reference_1 を右クリックし、[Rename] を選択する必要があります。

ダイアログ ボックスが表示されます。

3 新しい名前を入力します。

4 ［OK］をクリックします。

エンティティ Bean の削除

1 アウトライン ビューで、エンティティ Bean を右クリックし、[Delete] を選択します。

［Delete Artifact］ダイアログ ボックスが表示されます。

2 ［Yes］をクリックします。

成果物の削除

1 アウトライン ビューで、削除したいインスタンスを含む成果物のノードを展開します。たとえば、EJB リファレンス成果物のインスタンスを削除したい場合は、[EJB References] ノードを展開します。

2 成果物インスタンスを右クリックし、[Delete] を選択します。たとえば、[EJB References] ノードの下に EJB_Reference_1 という名前のインスタンスがある場合は、EJB_Reference_1 を右クリックし、[Delete] を選択します。

［Delete Artifact］ダイアログ ボックスが表示されます。

3 ［Yes］をクリックします。

第 10 章

Eclipse からの Borland AppServer プロジェクトのデプロイメント

ここでは、Eclipse から Borland AppServer プロジェクトをデプロイメントする方法について説明します。

Borland AppServer へのプロジェクトのデプロイメント

Borland AppServer Development Plug-in を利用すると、Eclipse で作成した Borland AppServer プロジェクトを、ローカルまたはリモートの Borland AppServer にデプロイメントできます。

デプロイメントの設定

1. デプロイメントしたい Borland AppServer プロジェクトを右クリックし、[プロパティー] を選択します。
   [プロパティー] ダイアログ ボックスが表示されます。
2. 左ペインから、[Borland AppServer Server Settings] を選択します。
3. サーバーに対応するサーバー名、デプロイメント構成名、デプロイメント パーティション名を設定します。サーバーはリモートまたはローカル マシン上で動作しています。
4. 管理ポートを、そのサーバー用のポートに設定します。
5. ［OK］をクリックします。

Borland AppServer へのプロジェクトのデプロイメント

1 適切なデスクリプタ ファイルに情報が保存されるように、プロジェクトを保存します。

2 デプロイメントしたい Borland AppServer プロジェクトを右クリックし、[Borland | Deploy] を選択します。
[Deploy to Borland AppServer] ダイアログ ボックスが表示されます。

3 左ペインで、Borland AppServer にデプロイメントしたいプロジェクトをダブルクリックします。

4 ［Next］をクリックします。

5 [Deploy] をクリックします。

第 11 章

WTP でのデバッグ

この章では、Borland AppServer にデプロイメントされた J2EE モジュールとやり取りする Java クライアント用のデバッグ環境のセットアップ手順について説明します。サーバー コード（たとえば、サーブレットや Bean）のデバッグは、このリリースではサポートされていません。

Eclipse でのクライアント デバッグのセットアップ

1 [実行 | 構成およびデバッグ ...] を選択します。

2 [リモート Java アプリケーション] を右クリックし、[新規] を選択します。

3 ［Arguments］タブを選択します。

4 [VM Arguments] に、次の行を追加します。

```
Dvbroker.agent.port=<your_osagent_port> -Djava.endorsed.dirs="<path_to_bas/lib/endorsed>"
```

デバッグ モードでのクライアントの実行

1 [実行 | デバッグ] を選択します。

2 クライアント プロファイルを選択します。

索引

記号

B

E

W

ア

イ

カ

キ

コ

サ

セ

ソ

テ

ト

フ

ユ

ロ