J2EE モジュールおよびアプリケーションのアセンブルと実行

Forte™ for Java™ プログラミングシリーズ

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900 U.S.A.
650-960-1300

Part No. 816-2846-01
2001 年 10 月　Revision A

Send comments about this document to: docfeedback@sun.com

原典 :　*Assembling and Executing J2EE™ Modules and Applications*
Part No: 816-1402-10
Revision A

Please
Recycle

Adobe PostScript

目次

図目次

はじめに

このマニュアルは、Forte for Java プログラミングシリーズの『J2EE モジュールおよびアプリケーションのアセンブルと実行』です。このマニュアルでは、Forte™ for Java™ によってWeb アプリケーションと J2EE アプリケーションをアセンブルし、配備して実行する方法について説明します。

お読みになる前に

このマニュアルは、Forte for Java IDE によってアプリケーションをアセンブル、配備、または実行するすべてのユーザーを対象としています。最初の章は、アセンブルと配備という J2EE の概念について説明していて、アセンブルと配備に関する一般的な理解を求めるすべてのユーザーに役立ちます。

このマニュアルは、次の事項についての知識を前提としています。

- Java プログラミング言語
- Java 2 Platform, Enterprise Edition に関する概念
- Web サーバーソフトウェアおよびアプリケーションサーバーソフトウェア

このマニュアルには、J2EE の概念に関する一般的な知識が必要ですが、これは次のリソースから得ることができます。

- *Sun BluePrints™ Design Guidelines for J2EE*
  `www.java.sun.com/j2ee/blueprints`
- *Java 2 Platform Enterprise Edition Specification* - `www.java.sun.com/products`
- *Java 2 Enterprise Edition Developer's Guide* -
  `www.java.sun.com/j2ee/j2sdkee/devguide1_2_1.pdf`

- *Java Servlet Specification, v2.2* - `www.java.sun.com/products/servlet/index.html`
- *JavaServer Pages Specification*, v1.1 - `www.java.sun.com/products/jsp/index.html`

内容の紹介

第 1 章 では、アセンブルと配備に関する J2EE の概念を要約します。また、モジュールとアプリケーションの J2EE ユニットを示して、モジュールおよびアプリケーション配備記述子について説明します。

第 2 章 では、モジュールとアプリケーションを Forte for Java IDE でアセンブルする方法を説明します。特に、モジュールとアプリケーションのプロパティシートを使用して、モジュールとアプリケーションの配備記述子を設定する方法について説明します。

第 3 章 では、アセンブルされたアプリケーションを配備して実行する方法を説明します。特に、モジュールとアプリケーションのプロパティシートを使用して、モジュールとアプリケーションの配備記述子を設定する方法について説明します。

付録 A では、Forte for Java IDE が Web サーバーおよびアプリケーションサーバーとの対話に使用するメカニズムを説明します。ここには、配備プロセスの詳しい説明を示します。

マルチプラットフォーム対応

このマニュアルで説明される作業は、次のプラットフォームおよびオペレーティングシステムで実行できます。

- Solaris™ 8 *SPARC™ Platform Edition*
- Microsoft Windows 2000, SP2
- Microsoft Windows NT 4.0, SP6
- Red Hat Linux 6.2

このマニュアルに掲載している画面イメージは、すべて Windows 版の Forte for Java ソフトウェアのものです。他のプラットフォームを使用する場合でも、表示上の違いはわずかであるため、内容を理解するのには問題ありません。ほとんどの手順で Forte for Java 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、次のように、Microsoft Windows の「コマンドプロンプトウィンドウ」でのプロンプトと構文が例として示されています。

```
C:¥>cd MyWorkDir¥MyPackage
```

UNIX ® または Linux 環境では、次のようなプロンプトとなり、¥ マーク (またはバックスラッシュ) ではなくスラッシュを使用します。

```
% cd MyWorkDir/MyPackage
```

表記上の規則

表 P-1　表記上の規則

字体または記号	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、コード例を示します。	.login ファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 system%
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示します。	system% **su** Password:
AaBbCc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには rm *filename* と入力します。
『 』	参照する書名を示します。	『コードマネージャ・ユーザーズガイド』を参照してください。
「 」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	sun% **grep '^#define ** **XV_VERSION_STRING'**

関連マニュアル

Forte for Java のマニュアルは、Acrobat Reader (PDF) ファイル、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

次のマニュアルは、Forte for Java のポータルサイト、`docs.sun.com` の Web サイト、およびインターネットオンラインブックストアの Fatbrain.com から入手することができます。

Forte for Java ポータルサイトでのマニュアルの入手先は、`http://www.sun.co.jp/forte/ffj/documentation/index.html` です。`docs.sun.com` の URL は、`http://docs.sun.com` です。Fatbrain.com の URL は、`http://www.fatbrain.com/documentation/sun` です。

- リリースノート (HTML 形式)

  Forte for Java 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。

- インストールガイド (PDF 形式)

  Forte for Java の Edition ごとに用意されています。対応プラットフォームへの Forte for Java 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、Web サーバーやアプリケーションサーバーのインストール、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。

- Forte for Java プログラミングシリーズ (PDF 形式)

  Forte for Java 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』 Part No. 816-2849-01

    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。

  - 『持続プログラミング』 Part No. 816-2850-01

    Forte for Java が提供するさまざまな持続性プログラミングモデルのサポート機能について説明しています。特に、JDBC と透過的な持続性についてを詳細に説明しています。

  - 『Enterprise JavaBeans コンポーネントのプログラミング』 Part No. 816-2845-01

Forte for Java の EJB ビルダーウィザードや、その他のグラフィカルユーザーインタフェースを使用し、Enterprise JavaBeans 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) 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』 Part No. 816-2844-01

  Web サービスモジュールが提供するツールを使用して Web サービスを構築する方法を説明しています。Web サービスは、XML (Extensible Markup Language) 文書の形式で提供されるアプリケーションビジネスサービスであり、HTTP を介して配信されます。

- 『XML データサービス用 JSP のプログラミング』 Part No. 816-2843-01

  Forte for Java Enterprise Service Presentation Toolkit (Forte ESP ツールキット) を使用し、HTML に動的 XML データを組み込む方法を説明しています。

- 『J2EE モジュールおよびアプリケーションのアセンブルと実行』 Part No. 816-2846-01 (このマニュアル)

  EJB モジュールと Web モジュールを組み合わせて J2EE アプリケーションを作成する方法と、J2EE アプリケーションを配備して実行する方法を説明しています。

- Forte for Java チュートリアル (PDF 形式)

  チュートリアルアプリケーションは、ユーザー設定ディレクトリの下の `sampledir/tutorial` ディレクトリにあります。

  - 『Forte for Java, Community Edition チュートリアル』 Part No. 816-2847-01

    Forte for Java, Community Edition のツールを使用し、簡単な J2EE Web アプリケーションを作成する方法を順を追って説明しています。

  - 『Forte for Java, Enterprise Edition チュートリアル』 Part No. 816-2848-01

    Enterprise JavaBeans コンポーネント、アプリケーションテスト機能、Forte for Java Web サービス技術を使用し、アプリケーションを作成する方法を順を追って説明しています。

オンラインヘルプ

オンラインヘルプは、Forte for Java 開発環境内から参照できます。ヘルプキー (Solaris オペレーティング環境では Help キー、Windows および Linux 環境では F1 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

Forte for Java の機能を紹介したプログラム例が、関連する `Readme` ファイルとともに、ユーザー設定ディレクトリの `sampledir/examples` ディレクトリに置かれています。また、Forte for Java のポータルサイトから、Enterprise Edition に固有のサンプルファイルをダウンロードし、それらを `sampledir/examples` ディレクトリに置くこともできます。チュートリアルアプリケーション (『Forte for Java, Community Edition チュートリアル』と『Forte for Java, Enterprise Edition チュートリアル』で説明されているアプリケーションを含む) はすべて、`sampledir/tutorial` ディレクトリに置かれています。

Javadoc

Javadoc 形式のマニュアルは、Forte for Java 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。IDE を起動すると、エクスプローラの Javadoc タブで Javadoc マニュアルを参照できます。

Sun のマニュアルのオンラインでの提供

Sun の各種システムのマニュアルを、次の Web サイトで提供しています。

`http://www.sun.com/products-n-solutions/hardware/docs`

Solaris のマニュアルセットとその他の多くのマニュアルを、次の Web サイトで提供しています。

`http://docs.sun.com`

Sun のマニュアルの注文方法

Sun の製品マニュアルは、Fatbrain.com インターネットブックストアを通じて米国 Sun Microsystems, Inc. に直接注文できます。Fatbrain.com の Sun Documentation Center へは次の URL でアクセスできます。

`http://www.fatbrain.com/documentation/sun`

ご意見の送付先

Sun のマニュアルについてのご意見やご要望をお寄せください。今後のマニュアル作成の参考にさせていただきます。次のアドレスまで電子メールをお送りください。

`docfeedback@sun.com`

電子メールのタイトルに、対象マニュアルの Part No. (このマニュアルの場合は 816-2846-01) を明記してください。

第1章

アセンブルと配備の基本

J2EE™ のエンタープライズアプリケーションに対するアプローチは、トランザクション管理やセキュリティ妥当性検査などの、ほとんどのエンタープライズアプリケーションが必要とする標準機能をエンタープライズ開発者が使用できる実行サービスの形式で提供することによって、開発者の作業を簡略化するものです。

J2EE 開発者は、トランザクション管理コードやセキュリティ妥当性検査コードを記述しなくても大規模なエンタープライズアプリケーションを開発できます (ただし、このようなコードが簡単な場合はコードを記述することもできます)。その代わり開発者は、Java Server Pages™ (JSP™) ソフトウェアや Enterprise JavaBeans コンポーネントなどの J2EE コンポーネントにそのビジネスロジックをカプセル化することによって、実行環境から機能を取得します。これらのコンポーネントは、J2EE 実行環境と対話して、コンポーネント開発者が指定する正しい種類のサービスを取得できます。

Forte for Java プログラミングシリーズの別のマニュアルでは、J2EE コンポーネントを作成して、J2EE 実行サービスを効果的に利用するビジネスロジックを記述する方法を説明しています。このマニュアルでは、個々の J2EE コンポーネントをとりあげて、それらを結合し、J2EE 実行環境に配信するプロセスについて説明します。主な 2 つのプロセスは次のとおりです。

- **アセンブル**。コンポーネントを、完全なビジネス機能を実行する、より大きなユニットに結合する
- **配備**。アセンブルされたアプリケーションを J2EE 実行環境に配信する

この章では、アセンブルと配備についての J2EE の概念と用語を示します。第 2 章 では、Forte™ for Java™ IE を使用して J2EE モジュールとアプリケーションをアセンブルする方法を説明します。第 3 章 では、IDE 内からアプリケーションを配布する方法を説明します。付録 A は、IDE が Web サーバーおよびアプリケーションサーバーと対話して IDE 内からの配備を可能にするために使用する機構を説明します。

J2EE コンポーネント

J2EE プラットフォームには、いくつかの異なる種類のコンポーネントが定義されて指定されており、それぞれがエンタープライズアプリケーションで異なるロールを持っています。これらのコンポーネントには、次のものがあります。

- JSP ページ。動的 HTML 応答を生成するために使用される Java 言語ロジックを含む HTML ページ
- サーブレット。HTTP 入力を読み取り、Java 言語ロジックによってそれを処理し、HTML 応答を動的に生成できる Java クラス
- Enterprise Java Beans™ (EJB™)。トランザクション処理機能、セキュリティ機能、およびほかのエンタープライズ機能を提供するコンポーネント

J2EE モジュール

J2EE モジュールは、オンラインショッピングカートの管理などの認識可能なビジネス機能を実行し、これらの機能の実行に必要な J2EE コンポーネントを含みます。ビジネス機能が非常に単純な場合、このモジュールは、単一の JSP ページまたは単一の Enterprise Bean にまで単純化できます。一般的には、モジュールには複数の J2EE コンポーネントと、ビジネス機能を実現するためにともに動作する、Java クラスおよび画像ファイルなどの静的リソースが含まれます。

すべてのモジュールに配備記述子があります。これは、モジュールを識別し、モジュールの内容をリストして、モジュールに必要な実行サービスを指定する XML (Extensible Markup Language) です。実行環境は、その配備記述子を持つモジュールを実行ユニットとして認識し、個々のコンポーネントではなくモジュールに対してサービスを割り当てます。たとえば、セキュリティロールはモジュールレベルで宣言されて、モジュール内のすべてのコンポーネントがロールのリストを使用できます。

J2EE プラットフォームは、いくつかの異なる種類のモジュールを定義しています。IDE は、以下で説明する Web モジュールと EJB モジュールの開発をサポートします。

Web モジュール

Web モジュールには、次のものを含めることができます。

- J2EE Web コンポーネント (JSP ページとサーブレット)
- アプレット
- カスタムタグライブラリ
- 静的コンテンツ。静的 HTML ページ、画像、サウンドファイルなど

図 1-1 は、典型的な Web モジュールを示しています。

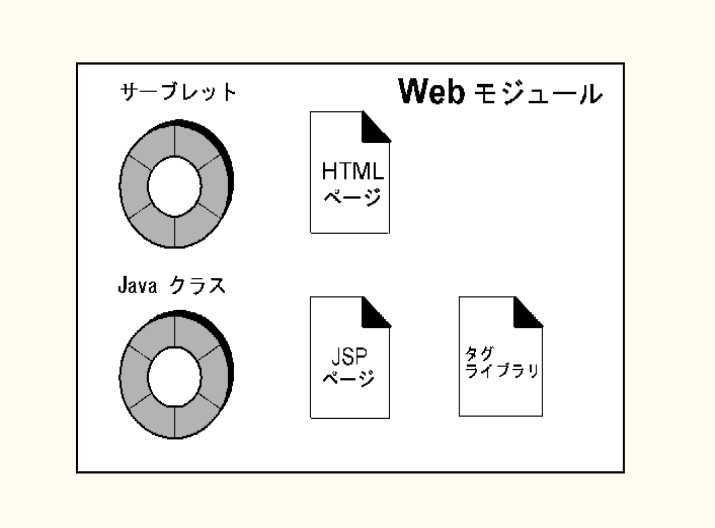


図 1-1　典型的な Web モジュール

Web モジュールの機能

Web モジュールには、表示機能とビジネスロジック機能の両方があります。Web モジュールは HTTP 要求を受け取って、Web ブラウザを実行するユーザーに形式化された HTML ページを返すことができます。また、JSP ページとサーブレットにビジネスロジックを含めて、受信する HTTP 要求を処理し、HTTP 応答を動的に生成することもできます。

多くのビジネスアプリケーションでは Enterprise Bean の大容量のトランザクション機能およびセキュリティ機能は不要であり、Web モジュールが完全なアプリケーションとして機能して、表示機能とビジネスロジック機能の両方を実現します。

アプリケーションに Enterprise Bean の機能が必要な場合は、Web モジュールの表示機能を EJB モジュールの処理機能と結合して、J2EE アプリケーションを作成できます。

Web モジュール配備形式

Web モジュールの配備形式は、配備記述子が含まれる WAR (Web アーカイブ) ファイルです。

EJB モジュール

EJB モジュールには、次のものを含めることができます。

- Enterprise JavaBeans コンポーネント
- Java インタフェースとクラス

図 1-2 は、典型的な EJB モジュールを示しています。

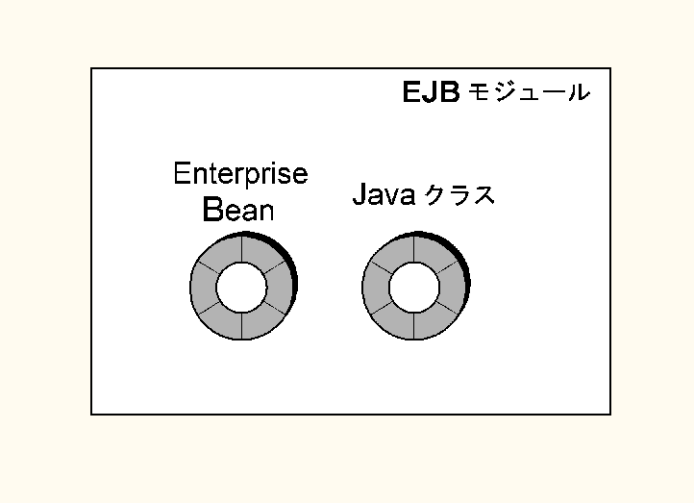


図 1-2　典型的な EJB モジュール

EJB モジュールの機能

EJB モジュールには、表示 (ユーザーインタフェース) 機能がありませんが、大容量のセキュリティ保護されたトランザクション処理機能はあります。EJB モジュールは通常、Web モジュールなど、表示機能のないほかのタイプのモジュールと結合されます。これらのモジュールはアセンブルされて J2EE アプリケーションになります。

EJB モジュール配備形式

EJB モジュールの配備形式は、モジュールの配備記述子が含まれる EJB JARファイルです。

J2EE モジュールのアセンブル

モジュールをアセンブルすると、一連のコンポーネントが、J2EE 実行環境で実行できて、J2EE 実行サービスを利用できる配備可能なユニットになります。

このアセンブルプロセスは比較的自由に行えます。通常は、ビジネスロジックを開発するコンポーネントプロバイダが、関連するコンポーネントをアセンブルして、認識可能なビジネス機能を実行するモジュールに変換します。ただし、コンポーネントプロバイダの役割はさまざまです。プロバイダによっては、いくつかのモジュールを開発して、それらを J2EE アプリケーションにアセンブルし、アプリケーション自体を配備するという作業すべてを実行します。また、ほかの組織で使用されるモジュールを開発するプロバイダもいます。これらのプロバイダは、開発したモジュールの最終的な使用法についてはほとんど知りません。

コンポーネントプロバイダがアセンブル時に入手できる情報が多いほど、配備記述子により多くの情報を追加できます。また、上記で説明した後者のプロバイダのように、コンポーネントプロバイダの役割が制限されているほど、プロジェクトの後の段階で完了する必要がある、定義されていない配備記述子の要素が多くなります。

この節では、開発プロジェクト間のこのような違いを念頭に置いて、モジュールをアセンブルするための主要なステップについて説明します。

1. モジュールを宣言してその内容を示す配備記述子を用意します。

   Forte for Java IDE では、モジュールノードを作成して、コンポーネントをモジュールに追加して、配備記述子のこの部分を用意します。

2. ビジネスロジックに密接に関連する配備記述子に要素を追加します。Web モジュールでは、この作業にはコンポーネント初期設定パラメータの設定が含まれます。EJB モジュールでは、この作業にはリソース参照の設定、モジュール内のほかの Enterprise Bean に対する参照のリンク、およびモジュールでの Enterprise Bean のトランザクション動作の定義が含まれます。

   Forte for Java IDE では、コンポーネントとモジュールのプロパティシートを処理して、配備記述子にこれらの要素を追加します。

   この時点で、ビジネスロジックが定義されます。モジュール内のコンポーネントは必要に応じてトランザクションによって相互に対話し、モジュールの作成目的であるビジネス機能を実現できます。これらのステップはビジネス機能に影響するため、通常はコンポーネントプロバイダが実行します。たとえばプロバイダは、あるモジュール向けに一連の Enterprise Bean を開発した場合、どの Bean が相互のメソッドを呼び出す必要があるかがわかります。また、それに必要なトランザクション動作の種類もわかります。

   残りのステップでは、モジュールをアプリケーションにおけるロールに合わせて設定するか、または特定の実行環境に合わせて調整します。コンポーネントプロバイダは、これらのステップを完了するのに必要な情報を持っていない場合もあります。このため通常これらのステップは、モジュールを J2EE アプリケーションにアセンブルする担当者か、またはこれを配備する担当者に託されます。

3. 配備記述子を使用して、モジュール内のコンポーネントをほかのモジュールのコンポーネントにリンクします。

   これは、別のモジュールにある Enterprise Bean メソッドを呼び出す Web コンポーネントおよび EJB コンポーネントの両方に適用されます。詳細については、7 ページの「EJB 参照」を参照してください。

4. 配備記述子を使用して、実行環境に合わせてモジュールを調整します。

   ここでは、モジュールと J2EE 実行環境間の多数の異なる対話が行われます。これらの対話のいくつかは、サーバー製品に特定のものです。Forte for Java IDE では、コンポーネント、モジュール、およびプロパティシートによって、配備記述子要素と非標準サーバー特定要素の両方が求められます。

Web モジュールのアセンブルの問題

この節では、Web モジュールをアセンブルするときに生じるいくつかの問題について述べます。

Web ページの URL

エンドユーザーは、Web ブラウザを起動してモジュールの開始ページを開き、Web モジュール内のビジネスロジックを使用します。このページ、およびモジュール内にあるほかのページの URL は、モジュールが Web サーバーに配備されたときに作成されます。これらの URL は、次の配備記述子タグから得られます。

- サーブレットコンテキスト
- サーブレットマッピング

EJB 参照

ある Web モジュールが EJB モジュール内の Enterprise Bean のメソッドを呼び出す場合、その Web モジュールは、JNDI および RMI の両方を使用するリモートアクセス機構を使用する必要があります。Web モジュールは、特定の種類の Enterprise Bean への参照 (そのホームインタフェースおよびリモートインタフェースの種類によって Enterprise Bean を指定) を宣言する必要があります。コンポーネントプロバイダは、次の参照を宣言する必要があります。

```
<ejb-ref>
    <description />
    <ejb-ref-name>ejb/greeter</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>samples.helloworld.ejb.GreeterHome</home>
    <remote>samples.helloworld.ejb.Greeter</remote>
    <ejb-link>TheGreeter</ejb-link>
  </ejb-ref>
```

Web モジュールの EJB 参照は、その Web モジュールが、参照に指定された種類の Enterprise Bean を持つ EJB モジュールを含む J2EE アプリケーションにアセンブルされるまで、特定の EJB モジュールおよび Enterprise Bean にリンクすることはできません。これらの参照のリンクは、通常アプリケーションのアセンブル担当者が行います。

EJB モジュールのアセンブルの問題

この節では、EJB モジュールをアセンブルするときに生じるいくつかの問題について述べます。

EJB 参照

Enterprise Bean メソッドを呼び出すために、呼び出し元は、JNDI および RMI の両方を使用するリモートアクセス機構を使用する必要があります。呼び出し元モジュールは、特定の種類の Enterprise Bean (ホームインタフェースおよびリモートインタフェースの種類によって指定) への参照を呼び出し先モジュールに宣言しておく必要があります。この参照メカニズムは、Enterprise Bean メソッドの呼び出しすべてが、J2EE 実行環境のトランザクションおよびセキュリティサービスに確実に認識されるようにします。これらの EJB 参照は、呼び出し元モジュールの開発者が作成する必要があります。

Enterprise Bean が同じモジュールにある別の Enterprise Bean のメソッドを呼び出すときにも、同じメカニズムが使用されます。

コンポーネントプロバイダは、モジュール内で使用されるすべての参照を宣言する必要があります。コンポーネントプロバイダは、モジュール内でリンク可能なすべての参照をリンクする必要もあります。ただし、別のモジュール内の Enterprise Bean への参照は、そのモジュールがアセンブルされて、指定した種類の Enterprise Bean を持つ別のモジュールを含むアプリケーションにならないとリンクできません。

その他の参照

EJB モジュールは、EJB 参照だけでなく、モジュールを配備する前にアセンブル作業を必要とする次の参照を含む場合もあります。

- **環境エントリ参照**は、Enterprise Bean のソースコードではなく配備記述子に指定できる名前付き値への参照です。環境エントリは、配備サイトのポリシーまたは手順に依存する値に使用されます。当座預金口座を処理する Enterprise Bean は、料金が課せられることなく顧客に許可される超過貸し出し回数を示す環境エントリを使用します。このような場合、環境エントリ値はポリシーが認識されると設定できます。これは、モジュールのアセンブル時よりも、アプリケーションのアセンブル時やアプリケーションの配備時に設定する方が一般的です。配備記述子タグは、環境エントリ参照を値にリンクするために使用されます。

- **リソースファクトリ参照**は、特定データベース向けに設定された接続マネージャを作成する接続マネージャファクトリや、特定セットの持続可能クラス向けに持続マネージャを作成するために設定された持続マネージャファクトリなどの、特定の配備環境に合わせてリソースを作成するコンポーネントへの参照です。この種類のリ

ソースは一般に配備環境に固有であるため、リソースファクトリ参照は通常配備時にリンクされます。配備記述子タグは、参照を特定のリソースファクトリにリンクするために使用されます。

- **セキュリティロール参照。**プログラム式セキュリティを使用する Enterprise Bean の開発者は、一般的に開発時には、最終的な配備環境での実際のセキュリティロール名が何になるかを知りません。したがって、 Enterprise Bean 開発者は、セキュリティチェックコードでセキュリティロール参照 (「admin」など) を使用します。これらの参照は、後で配備環境によって指定された実際のセキュリティロールにリンクされます (「admin」ロール参照は 「sysadmin」ロールにリンクできます)。配備記述子タグは、セキュリティ参照を実際のセキュリティロールにリンクするために使用されます。

宣言型セキュリティ

J2EE 実行環境では、各メソッドへのアクセスが許可されたセキュリティロールを宣言することによって、Enterprise Bean メソッドと Web リソースへのアクセスを制御できます。この情報は配備記述子タグに指定され、実行環境は必要なすべての妥当性検査を実行します。宣言型セキュリティを設定するには、次のことを実行します。

1. セキュリティロールを宣言します。

   モジュール配備記述子は、モジュールで実際に使用されるセキュリティロールを宣言します。典型的なセキュリティロール宣言を次に示します。

```
<security-role>
    <description>SecurityAdministrator</description>
    <role-name>SecAdmin</role-name>
</security-role>
```

2. Web モジュールのセキュリティ制約を識別します。

モジュールの配備記述子を使用して、どのセキュリティロールがどの Web リソースにアクセスできるかを指定します。制約が追加された後は、指定されたロールだけが Web リソースにアクセスできます。セキュリティ制約の典型的なタグを次に示します。

```
<security-constraint>
    <web-resource-collection>
      <web-resource-name>HelloServlet</web-resource-name>
    </web-resource-collection>
    <auth-constraint>
      <role-name>SecAdmin</role-name>
</auth-constraint>
</security-constraint>
```

3. EJB モジュールのメソッドアクセス権を識別します。

   モジュールの配備記述子を使用して、どのセキュリティロールがどの Enterprise Bean メソッドを呼び出すことができるかを指定します。指定されたロールだけがメソッドを呼び出すことができます。メソッドアクセス権のタグは、Web モジュールセキュリティ制約のタグに似ています。

トランザクション管理

J2EE 実行環境は、それを要求する J2EE コンポーネントでのトランザクションを管理できます。これは、コンテナ管理によるトランザクションと呼ばれます。単一のメソッド呼び出しを各自のトランザクションとして処理することから、相互のメソッドを呼び出して分散トランザクションに参加する一連のコンポーネントを指定するにいたるまで、さまざまなトランザクション管理方針を使用できます。トランザクション管理方針を指定するには、モジュールの配備記述子を使用してください。

トランザクション管理方針はビジネスロジック動作の一部であるため、コンポーネントプロバイダは一般に、コンポーネントをモジュールにアセンブルするときにこの方針を設定します。

オーバーライド

モジュールレベルの配備記述子で使用されるタグの多くは、モジュールを J2EE アプリケーションにアセンブルするときにアプリケーションレベルの記述子によってオーバーライドできます。オーバーライド機構は次のように動作します。

- アプリケーションが配備されると、J2EE 実行環境はまずアプリケーションレベルの配備記述子を読み取って、検出したタグすべてをそこに適用します。
- 実行環境はさらにモジュールレベルの配備記述子を読み取って、アプリケーションレベルの配備記述子に存在しなかったタグをすべて適用します。あるタグがアプリケーションレベルの配備記述子で検出された場合、モジュールレベル配備記述子のそのタグの出現は無視されます。

この結果、アプリケーション配備記述子でタグを使用すると、モジュールレベル配備記述子にある同じタグはすべてオーバーライドされます。アプリケーションレベルでタグを使用した場合、モジュールレベル配備記述子は変更されませんが無視されます。

たとえば、環境エントリについて考えます。EJB モジュールの開発者が、ポンドのオンス数が実際の配備サイトでのポリシーによって異なることに気づいた場合を想定します。この開発者は、デフォルト値 16 を設定して、アプリケーションのアセンブル担当者にモジュールを引き継ぎました。アプリケーションのアセンブル担当者にもサイト固有の知識がなかったため、アセンブルされたアプリケーションは、引き続きデフォルト値が有効なままで配備担当者に引き継がれました。

顧客のサイトでは、配備者がサイトの正しい値が 15 であることを知っていたため、アプリケーションレベル配備記述子に、モジュールレベル記述子に設定された値をオーバーライドするタグを追加しました。

Web アプリケーション

J2EE コンポーネントからアセンブルできるもっとも単純な種類のアプリケーションは、`web.xml` 配備記述子を持つ単一の Web モジュールである、Web アプリケーションです。この種類のアプリケーションは、Web モジュール専用環境 (Web サーバー) に配備できます。

Forte for Java IDE を使用すると、複数の Web モジュールを Web モジュールグループとしてアセンブルして、Web 専用環境に配備することもできます。この詳細については、『Web コンポーネントのプログラミング』を参照してください。

J2EE アプリケーション

J2EE アプリケーションは 1 つまたは複数のモジュールから構成されます。これは、Web モジュールにも EJB モジュールにもできます。J2EE アプリケーションにはアプリケーションレベル配備記述子の `application.xml` があり、アプリケーションレベルで設定されたすべてのタグが含まれます。また、アプリケーションレベルでタグが検出されない場合に使用されるモジュールレベル配備記述子の複製もあります。

アプリケーションレベル配備記述子には、モジュール記述子になかったいくつかのタグが含まれます。図 1-3 は、1 つの Web モジュールと 1 つの EJB モジュールから構成される典型的な J2EE アプリケーションを示しています。

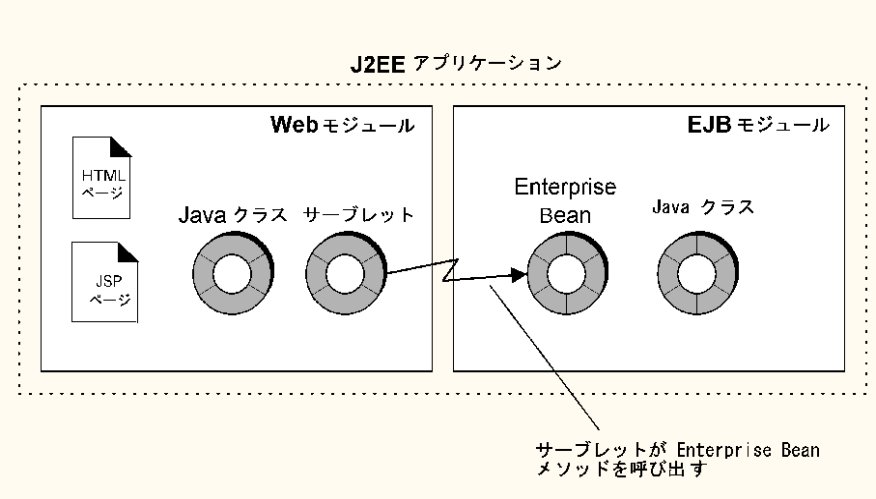


図 1-3　典型的な J2EE アプリケーション

J2EE アプリケーションは、EAR ファイルでの配備向けにパッケージ化されていて、application.xml 配備記述子ファイル、モジュール配備記述子の複製、およびモジュールの JAR ファイルと WAR ファイルが含まれます。

J2EE アプリケーションのアセンブル

アプリケーションのアセンブルとは、J2EE モジュールを J2EE アプリケーションに結合することです。モジュールのアセンブルと同様、どのアプリケーションのアセンブル作業も、開発チームと開発プロセスの性質によって決まります。主要なステップは次のとおりです。

1. アプリケーションを宣言してその内容を示す配備記述子を用意します。

   次に、アプリケーションレベル配備記述子の単純な例を示します。アプリケーションには、1 つの EJB モジュールと 1 つの Web モジュールが含まれます。

```
<application>
  <display-name>hello</display-name>
  <description>J2EE Application hello</description>
  <module>
    <ejb>greeter_module.jar</ejb>
    <alt-dd>greeter_module.xml</alt-dd>
  </module>
  <module>
    <web>
      <web-uri>helloweb.war</web-uri>
      <context-root>/helloweb_module</context-root>
    </web>
    <alt-dd>helloweb.xml</alt-dd>
  </module>
</application>
```

2. 配備記述子を使用して、Web モジュールの EJB 参照を EJB モジュールの Enterprise Bean にリンクするなど、異なるモジュールのコンポーネント間の対話を定義します。

   詳細については、7 ページの「EJB 参照」を参照してください。

3. 配備記述子を使用して、環境に合わせてアプリケーションを調整し、実行サービスを要求します。

   これは、モジュールの調整プロセスに似ています。実際には、実行する必要がある調整作業の量は、モジュールレベル配備記述子に追加された情報量によって決まります。

アセンブル機能と配備機能が別々だと、調整の一部を実行できない場合があります。この場合は、一部の動作を定義しないまま配備担当者にアプリケーションを配信します。

配備

配備とは、Web アプリケーションまたは J2EE アプリケーションを配備可能な形式で Web サーバーまたはアプリケーションサーバーに渡すプロセスをいいます。サーバーには通常、このプロセス用の配備ツールがあります。J2EE 配備担当者が配備されるアプリケーションまたはモジュールを識別し、サーバーはアーカイブファイルを配備されたアプリケーションの独自のパスにコピーします。アプリケーションをアセンブルする主要ステップは次のとおりです。

1. Web アプリケーションまたは J2EE アプリケーションを、配備する環境に合わせて調整します。
2. 配備先の Web サーバーまたはアプリケーションサーバーに必要なサーバー固有情報を提供します。

   ほとんどの Web サーバーとすべてのアプリケーションサーバーでは、標準的な J2EE 配備記述子タグが示す以上に詳しい情報が必要です。これらの情報は、配備記述子外部のサーバーに対して、サーバーベンダーによって定義された内容と形式を持つファイルで提供されます。これらのファイルを用意する必要があります。
3. サーバーの配備ツールを使用して、配備プロセスを開始します。

これらの作業はすべて Forte for Java IDE から実行できます。IDE には、非標準のサーバー固有情報を要求するプロパティシートがあります。また、WAR ファイルや EAR ファイルを作成して、それらを自動的にサーバーの配備ツールに渡す配備コマンドもあります。

Web アプリケーションまたは Web ベースの表示が行われる J2EE アプリケーションがサーバーに配備されたら、サーバーの機能を使用してアプリケーションを実行します。Web ブラウザを起動して、アプリケーションの開始ページを開きます。このページ、およびアプリケーション内にあるほかのページの URL は、Web モジュールの配備記述子にあるサーブレットコンテキストとサーブレットマッピングによって決定されます。

第2章

J2EE モジュールとアプリケーションのアセンブル

第 1 章 では、Forte for Java IDE が J2EE モジュールおよびアプリケーションをエクスプローラウィンドウでノードとして表示し、これらのノードのコンテキストメニューとプロパティシートを使用してアセンブル作業を実行できることを説明しました。この章では、これらのメニューとプロパティシートについて詳しく説明します。また、モジュールとアプリケーションのアセンブルに必要なステップも示します。

エクスプローラウィンドウでのモジュールとアプリケーションの表示

J2EE モジュールとアプリケーションは、IDE のエクスプローラウィンドウのノードとして表示されます。モジュールを示すノードにはモジュール内のコンポーネントを示すサブノードがあり、アプリケーションを示すノードにはアプリケーション内のモジュールを示すサブノードがあります。したがって、エクスプローラを使用して、処理しているモジュールとアプリケーションの内容を検討できます。

Web モジュール

Web モジュールのノードとサブノードは、モジュール内の個々のファイルを表します。Web モジュールには標準的なディレクトリ構造 (詳細については、『Web コンポーネントのプログラミング』を参照) があり、この構造はエクスプローラウィンドウに表示されます。図 2-1 は、エクスプローラに表示された Web モジュールを示しています。

Web モジュールの最上位ノードは、Web モジュールの最上位ディレクトリを表します。IDE がディレクトリを Web モジュールとして認識するには、これをエクスプローラウィンドウのファイルシステムとしてマウントする必要があります。Web モジュールディレクトリが別のファイルシステムのサブディレクトリとして表示された場合、IDE はこれを Web モジュールとして認識しません。この詳細は、『Web コンポーネントのプログラミング』でも説明されています。

最上位ノードには、`WEB-INF` ディレクトリを示すサブノードがあります。`WEB-INF` ディレクトリには、JAR 形式の Web コンポーネントに使用される `lib` サブディレクトリを示すサブノードと、`.java` 形式のすべての Web コンポーネントに使用される「クラス」サブディレクトリを示すサブノードがあります。`WEB-INF` には、モジュールの配備記述子ファイルを表す `web.xml` サブノードもあります。これは、Web モジュールの標準的なディレクトリ構造です。

この例の Web モジュールには、JSP ページ `myNewJSP`、HTML ページ `index.html`、およびタグライブラリ `myTagLib` を示すノードもあります。これらのノードは、Web コンポーネントプロバイダによって追加されたコンポーネントとリソースを表します。これらのノードに加えて、「クラス」ディレクトリには、サーブレットクラス `myNewServlet` を示すノードが含まれます。これは、コンポーネントプロバイダによって追加されたもう 1 つのリソースです。

図 2-1　Web モジュールノードとそのサブノード

この Web モジュールの表示は、特定のディレクトリとその内容に対応します。配備記述子 (`web.xml` ファイル) は、ソースコードに含まれます。

EJB モジュール

EJB モジュールは、Web モジュールとは異なる方法で表示されます。EJB モジュールの最上位ノードは、特定のディレクトリとその内容を表しません。代わりに、EJB モジュールノードは、モジュールの配備記述子を表します。これは、1 つのディレクトリ、または異なる複数のファイルシステム内にある多数のディレクトリに存在する Enterprise Bean のリストとして機能します。配備記述子は、コンポーネントのソースコードの所在を指定します。

EJB モジュールを 「論理」ノードによって表すと、異なるディレクトリの Enterprise Bean を 1 つの EJB モジュールに結合できます。また、これは配備記述子の構成情報をソースコードとは別個に維持します。EJB モジュールを配備すると、配備記述子ファイルが生成されて、モジュール内に含まれるコンポーネントの `.class` ファイルが、ファイルシステム内のどこにあってもそこから EJB JAR ファイルにコピーされます。

図 2-2 は、エクスプローラウィンドウに表示された EJB モジュールを示しています。このモジュールには、モジュールに含まれている 3 つの Enterprise Bean を示すサブノードがあります。これらの各 Enterprise Bean は異なるディレクトリにある可能性があります。1 つの Enterprise Bean が、ファイルシステム内の異なるいくつかの場所に存在する可能性もあります。たとえば、 Enterprise Bean のインタフェースのソースコードがあるディレクトリに存在し、これらのインタフェースを実現するクラスが別のディレクトリに存在する場合もあります。

図 2-2　EJB モジュールノードとそのサブノード

J2EE アプリケーション

J2EE アプリケーションも論理ノードによって表されます。EJB モジュールノードと同様、J2EE アプリケーションの最上位ノードも単一のディレクトリまたはファイルシステムを表しません。代わりに、これはアプリケーションレベルの配備記述子を表し、アプリケーションを構成するモジュールのリストとして機能します。これらのモジュールのソースコードは、複数のディレクトリまたはファイルシステムに存在します。

IDE は、アプリケーションレベルの配備記述子をソースコードとは別に維持して、同じソースコードを複数の J2EE アプリケーションで使用できるようにします。アプリケーションを配備する (またはアプリケーションの .ear ファイルを生成する) 場合にのみ、IDE はすべての .class ファイルをコピーして、それらを .ear ファイルの配備記述子と関連付けます。

図 2-3 は、エクスプローラに表示された J2EE アプリケーションを示しています。図 2-1 と 図 2-2 のモジュールは、アプリケーションに追加されて、アプリケーションノードのサブノードによって表されています。

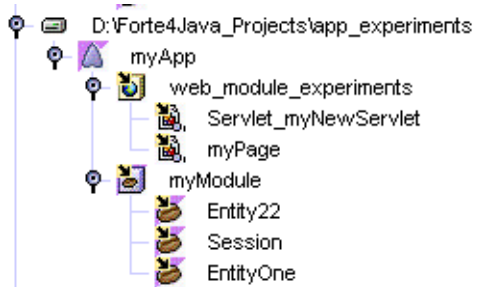


図 2-3　配備記述子の J2EE アプリケーションノードとそのプロパティシートの表示

J2EE モジュールまたはアプリケーションを表すすべての IDE ノードにプロパティシートがあります。このプロパティシートには、モジュールまたはアプリケーションの配備記述子にあるタグに対応するプロパティがあります。つまり、テキストエディタを使用して XML 配備記述子を編集して形式化する代わりに、プロパティシートで処理できます。

Web モジュールの場合、配備記述子はファイルとして存在し、このファイル (web.xml) はエクスプローラウィンドウに表示されます。これは図 2-1 で確認できます。つまり、配備記述子に指定した値はソースファイルに関連付けられます。

EJB モジュールまたは J2EE アプリケーションの場合、配備記述子ファイルは、モジュールを配備するか (EAR ファイルが必要)、EAR ファイルを生成するまで生成されません。これらの処理によって配備記述子ファイルが生成されて、.class ファイルの特定のコピーとともに保存されます。

以下の節では、モジュールおよびアプリケーションノードとそのプロパティシートの処理方法について説明します。

Web モジュールのアセンブル

Web モジュールは、Web コンポーネントとほかのリソースのセットです。モジュール内のコンポーネントが、必要なサービスを Web サーバーまたはアプリケーションサーバーから取得するには、モジュールを意味のある方法でアセンブルする必要があります。つまり、モジュールは、コンポーネントを識別して実行に必要な実行サービスを指定する配備記述子を必要とします。

モジュールのビジネスロジックを記述するコンポーネントプロバイダは、このいくつかを実行する必要があります。コンポーネントプロバイダは、初期化パラメータの設定、サーブレットマッピング、環境エントリ参照、および類似の内容の設定を担当します。

セキュリティの設定などのその他の情報は、配備環境に詳しい担当者が設定する必要があります。コンポーネントプロバイダがこの作業を行う場合もありますが、コンポーネントプロバイダがこの作業を行わずに、アセンブル担当者または配備担当者が行う場合もあります。

この節では、Web モジュールをアセンブルするときに生じる問題について述べます。特定の Web モジュールをアセンブルするときにとる処置は、開発プロセスにおける役割と、処理中のモジュールの内容によって異なります。このため、Web モジュールをアセンブルするステップごとの手順を示すことはできません。アセンブル担当者が、各モジュールで必要なものが何かを判断する必要があります。処理中のモジュールに必要なものを判断するのに役立つアセンブルプロセスの基本的な手順を次に示します。

1. 新しい Web モジュールを作成します。この手順は、『Web コンポーネントのプログラミング』に記載されています。
2. Web コンポーネントとその他のリソースをモジュールに追加します。この手順も、『Web コンポーネントのプログラミング』に記載されています。
3. モジュールを分析して、コンポーネント内のビジネスロジックがユニットとして機能できるようにするアセンブル作業を実行します。

   a. コンポーネントプロバイダが担当する場合は、初期化パラメータの設定などのモジュールのビジネスロジックを完成させるアセンブル作業を実行する必要があります。さらに、モジュールに追加したビジネスロジックに関する知識を元にして、この節で説明するどのステップが必要であり、どのステップを実行できるかを判断します。

      Web モジュールに必要なアセンブル作業も、モジュールがスタンドアロン Web アプリケーションとして使用されるか、または J2EE アプリケーション内のモジュールとして使用されるかによって異なります。Web モジュールプロパティの多くは、アプリケーションサーバーでのみ使用されます。これらのプロパティは、モジュールがスタンドアロン Web アプリケーションとして配備された場合に何も影響しません。

   b. 別の人によって開発されたモジュールを J2EE アプリケーションに組み込むか、または別の人によって開発された Web モジュールをスタンドアロン Web アプリケーションとして配備する場合は、そのモジュールを分析して、追加のアセンブル作業が必要かどうかを判断する必要があります。コンポーネントプロバイダは、その作業のいくつかを実行していない場合があります。

次の節では、Web モジュールで実行できるアセンブル作業について説明します。各作業に関する情報は、各自のモジュールに実行する必要がある作業を判断するのに役立ちます。

サーブレットコンテキストの設定

ユーザーは、URL がマップされた Web モジュールリソースにアクセスできます。Web リソースの URL の一般的な形式は次のとおりです。

```
http://hostname:port/servlet_context/servlet/servlet_name
```

サーバールートとリソース名の間のその位置を見ればわかるように、サーブレットコンテキストは、Web モジュール内のリソースと同じサーバーインスタンスに配備されたほかの Web モジュール内のソースを区別しています。サーブレットコンテキストを提供する手順は、Web モジュールがスタンドアロン Web アプリケーションとして使用されるか、または J2EE アプリケーションの Web モジュールとして使用されるかによって異なります。

- スタンドアロン Web アプリケーションとして使用されるモジュールをアセンブルする場合は、モジュールのプロパティとしてサーブレットコンテキストを指定します。サーブレットコンテキストを指定するには、`WEB-INF` ノードの「コンテキストルート」プロパティを使用します。
- 後で J2EE アプリケーションに組み込まれるモジュールをアセンブルする場合は、アプリケーションレベルでサーブレットコンテキストを指定します。60 ページの「Web モジュールへのサーブレットコンテキストの設定」 を参照してください。

図 2-4 は、サーブレットコンテキストを `myWebApp` に設定する Web アプリケーションの「コンテキストルート」プロパティを示しています。このアプリケーションにおける Web リソースの URL の形式は次のとおりです。

```
http://hostname:port/myWebApp/servlet/servlet_name
```

図 2-4　Web アプリケーション (`WEB-INF`) のプロパティ

サーブレットコンテキストを指定しないと、デフォルトで / に設定されて、Web モジュールのリソースの URL は次に示す一般的な形式になります。

```
http://hostname:port/servlet/servlet_name
```

サーブレットの設定

モジュールにサーブレットが含まれる場合は、サーブレットの動作を制御するプロパティを処理する必要があります。

デフォルトのサーブレットプロパティ

IDE でサーブレットを作成すると、デフォルトの配備記述子エントリを使用して作成されます。たとえば、myNewServlet を作成すると、次の配備記述子タグエントリが作成されます。

```
<servlet>
    <servlet-name>Servlet_myNewServlet</servlet-name>
    <display-name>Servlet myNewServlet</display-name>
    <description>Default configuration created for
servlet.</description>
    <servlet-class>myNewServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Servlet_myNewServlet</servlet-name>
    <url-pattern>/servlet/myNewServlet</url-pattern>
</servlet-mapping>
```

この画面は、「サーブレット」プロパティエディタに表示されます (このプロパティエディタを開くには、モジュールの web.xml ファイルを右クリックしてから、コンテキストメニューで「プロパティ」 > 「サーブレット」> 「...」 ボタンという一連のコマンドを選択します)。図 2-5 では、「サーブレット」プロパティエディタには、myNewServlet のデフォルト値が表示されています。

図 2-5　「サーブレット」プロパティエディタ

これらのデフォルトのタグエントリを使用する Web モジュールを配備すると、ユーザーは次のURL にある `myServlet` にアクセスできます。

```
http://hostname:port/servlet_context/servlet/Servlet_myNewServlet
```

「セキュリティロール参照」フィールドについては、29 ページの「セキュリティの設定」を参照してください。「初期パラメータ」および「起動時に読み込み」フィールドは、サーブレットのビジネスロジックに使用されます。

代替サーブレットマッピング

モジュールのエンドユーザーまたはモジュール内の別の Web コンポーネントに、代替 URL にあるサーブレットにアクセスさせる場合は、このサーブレットに代替 URL をマップする必要があります。

これは、「サーブレットマッピング」プロパティエディタで行います (このプロパティエディタを開くには、モジュールの `web.xml` ファイルを右クリックしてから、コンテキストメニューで 「プロパティ」 > 「サーブレットマッピング」 > 「...」ボタンという一連のコマンドを選択します)。「サーブレットマッピング」プロパティエディタには、モジュール内のサーブレットとそれらにマップされた URL が一覧表示されます。

マッピングを追加する場合は、「追加」ボタンをクリックして表示されたダイアログを使用して新しい URL パターンを追加し、それを既存のサーブレットにマップします。図 2-6 は、`myNewServlet` に対する 2 つのマッピングと、デフォルトマッピン

グ、および my_url_pattern への 2 番目のマッピングを示す「サーブレットマッピング」プロパティエディタを示しています。必要であれば、デフォルトマッピングを選択して「削除」ボタンをクリックし、プロパティエディタと配備記述子から削除できます。

図 2-6　「サーブレットマッピング」プロパティエディタ

この代替マッピングを設定したら、サーブレットを次の URL で実行できます。

```
http://hostname:port/servlet_context/servlet/myAlternateName
```

インポートされたサーブレット

サーブレットをモジュール内に作成するのではなく、Web モジュールにインポートした場合は、デフォルトの配備記述子プロパティがありません。「サーブレット」プロパティエディタおよび「サーブレットマッピング」プロパティエディタを使用して、これらのプロパティを設定する必要があります。

ソースコード形式 (.java ファイル) でサーブレットをインポートした場合は、これを WEB-INF/class ディレクトリに入れます。アーカイブ形式 (.war ファイル) でサーブレットをインポートした場合は、WEB-INF/lib ディレクトリに入れます。

JSP ページの設定

アセンブルする Web モジュールに JSP ページコンポーネントが含まれる場合は、JSP ページの動作を制御するプロパティを処理する必要があります。

デフォルトの JSP ページプロパティ

IDE で JSP ページを作成した場合、自動的に作成される配備記述子エントリはありません。このことは、ビジネスロジックがプログラム方式で JSP ページにアクセスする場合には問題になりません。たとえば、次のコードは JSP ページ `myNewJSP` を使用するサーブレットからの例です。このコードは、JSP ページの実際のファイル名 (`myNewJSP.jsp`) を指定しています。

```
...
response.setContentType("text/html");
        RequestDispatcher dispatcher;
      dispatcher = getServletContext().getRequestDispatcher ("/myNewJSP.jsp");
        dispatcher.include(request, response);
...
```

JSP への URL マッピング

ユーザーが URL によってアクセスできるように JSP ページを設定するには、サーブレットマッピングによく似たマッピングを設定する必要があります。

最初に、JSP ページのサーブレット名を指定します。これは、「JSP ファイル」プロパティエディタで行います (このプロパティエディタを開くには、web.xml ファイルを右クリックしてから、コンテキストメニューで 「プロパティ」 > 「配備」タブ > 「JSP ファイル」 > 「...」ボタンという一連のコマンドを選択します)。

「追加」ボタンを選択すると、JSP ページファイルを選択してサーブレット名を割り当てるためのダイアログが開きます。図 2-7 は、サーブレット名 `myPage` が `myNewJSP.jsp` にマッピングされた後の「JSP ファイル」プロパティエディタを示しています。

図 2-7　「JSP ファイル」プロパティエディタ

次のステップでは、サーブレット名に URL パターンをマップします。これは、「サーブレットマッピング」エディタで行います。図 2-8 は、URL パターンがサーブレット名 myPage にマップされた後の「サーブレットマッピング」プロパティエディタを示しています。

図 2-8　「サーブレットマッピング」プロパティエディタ

これで JSP ファイル myNewJSP.jsp によって定義された JSP ページは、次の URL でアクセスできます。

```
http://hostname:port/servlet_context/myJSP_Page
```

タグライブラリの設定

IDE でタグライブラリを作成した場合、自動的に作成される配備記述子エントリはありません。タグライブラリを使用する JSP ページコンポーネントにタグライブラリを表示するには、タグライブラリマッピングを設定してから、JSP ページでそのマッピングを参照する必要があります。

最初のステップでは、タグライブラリに名前を割り当てて、その名前を URI にマッピングします。これは、「タグライブラリ」プロパティエディタで行います (このプロパティエディタを開くには、`web.xml` ノードを右クリックしてから、「プロパティ」 > 「タグライブラリ」 > 「...」 ボタンを選択します)。図 2-9 は、`MyTagLib` が `/mytaglib` URI にマップされた後の「タグライブラリ」プロパティエディタを示しています。

図 2-9　「タグライブラリ」プロパティエディタ

タグライブラリに URI をマップすると、モジュール内の Web コンポーネントはその URI によってタグライブラリを参照できます。たとえば、`MyTagLib` でタグを使用する JSP ページは、その `<body>` タグの次に、以下のものを必要とします。

```
<%@taglib uri="mytaglib" prefix="mt" %>
```

JSP ページは、次のように、タグライブラリで個々のタグを実行できます。

```
<mt:myHelloTag />
```

開始ファイルの設定

IDE で Web モジュールを作成する場合、配備記述子は開始ファイルのデフォルト値を指定します。図 2-10 は、デフォルトの開始ファイル名を示す「開始ファイル」プロパティエディタを示しています (このプロパティエディタを開くには、「開始ファイル」プロパティをクリックして、「...」 ボタンをクリックします)。ユーザーがモ

ジュールの基本 URL にアクセスすると、Web コンテナはモジュールディレクトリにあるこれらのファイルを検索します。最初に検出されたファイルが開始ページとして表示されます。

モジュールの開始ファイルを作成するもっとも簡単な方法は、これらのデフォルト名を持つファイルを作成して、それをモジュールに追加するというものです。たとえば、図 2-10 は、`index.html` ファイルを含む Web モジュールを示しています。

図 2-10 「開始ファイル」プロパティエディタ

モジュールの開始ページに異なるファイルを使用する場合は、プロパティエディタを使用してそのファイルを指定します。ファイルを追加するには、「項目」フィールドにその名前を入力して「追加」ボタンをクリックします。その他のボタンを使用すると、ファイルを並べ替えたり、リストから削除したりできます。図 2-11 は、サーブレット GreeterServlet が開始ファイルとして指定された後のプロパティエディタを示しています。

図 2-11 変更された「開始ファイル」プロパティエディタ

エラーページの設定

モジュールにエラーページを指定する場合は、モジュールの配備記述子にそれらのページを指定する必要があります。これは、「エラーページ」プロパティエディタで行います (このプロパティエディタを開くには、web.xml アイコンを右クリックしてから、コンテキストメニューで 「プロパティ」 > 「配備」タブ > 「エラーページ」 > 「...」 ボタンを選択します)。

エラーは、HTTP エラーコードまたは Java 例外クラスに分類できます。エディタには、エラーカテゴリごとに 1 つずつ、2 つの「追加」ボタンがあります。どちらかのカテゴリに、エラーを指定して、それをページにマップします。図 2-12 は、エラーページが HTTP エラーコード 404 に割り当てられた後のプロパティエディタを示しています。

図 2-12 「エラーページ」プロパティエディタ

セキュリティの設定

一般に J2EE モデルは、セキュリティがモジュールレベルで設定されているものと想定しています。これは、ビジネスロジックを設計および開発するときに、コンポーネントプロバイダがセキュリティを考慮するものと想定します。コンポーネントプロバイダの多くは、ビジネスロジックを利用するユーザーのロールと、ロールごとに使用が許されている機能について一般的に理解しています。

人材データを処理する Web モジュールのコンポーネントプロバイダについて考えます。このプロバイダは、その人材情報を維持するためにすべての社員から使用可能でなければならない Web リソースと、人材関連事務、監督、監査などの各ロールからだけ使用できる Web リソースを認識しています。

コンポーネントプロバイダと初期モジュールアセンブル担当者は、これらのロールを表す汎用セキュリティロールを設定して、これらを作成した Web リソースにマップできます。これらの汎用ロールは、後で配備環境の実際のユーザー名とグループ名にマップできます。このマッピングは、おそらくアプリケーションのアセンブル時、またはその後アプリケーションが本番環境に配備されるときに、アプリケーションレベルで実行できます。

また、プロジェクトによっては、コンポーネントプロバイダおよび初期モジュールアセンブル担当者が、セキュリティをビジネスロジックにどのように適用すればよいかがわからない場合があります。このような場合、担当者はセキュリティロールの設定を開発プロセスのもっと後の段階にまで延期できます。たとえば、アプリケーションのアセンブル担当者がモジュールレベルの配備記述子を処理してから、モジュールをアプリケーションにアセンブルすることができます。

以下の節では、モジュールレベルでセキュリティロールを設定して、それらを Web リソースにマップする方法を説明します。

Web モジュールの宣言型セキュリティは、Web リソースへのセキュリティロールのマッピングから構成されます。この作業を行うには、次のことが必要です。

1. セキュリティロールを宣言します。
2. 保護対象の「Web リソース」を定義します。Web リソースはモジュール内の URI です。
3. セキュリティロールを Web リソースにマップします。これにより、マップされたロールに、指定 Web リソースへのアクセス権が付与されます。

セキュリティの内容説明については、第 1 章を参照してください。

セキュリティロールの宣言

セキュリティロールは、「セキュリティロール」プロパティエディタで宣言します (このプロパティエディタを開くには、`web.xml` ノードを右クリックしてから、コンテキストメニューで 「プロパティ」 >「セキュリティ」タブ >「セキュリティロー

ル」>「...」ボタンを選択します)。「追加」ボタンを選択すると、新しいセキュリティロールを宣言するためのダイアログが開きます。「編集」および「削除」ボタンを選択すると、すでに宣言されているロールを処理できます。

図 2-13 は、Me および EveryoneElse の 2 つのロールが宣言された後の「セキュリティロール」プロパティエディタを示しています。

図 2-13 「セキュリティロール」プロパティエディタ

Web リソースへのセキュリティロールのマッピング

「セキュリティ制限」プロパティエディタでは、Web リソースを定義して、それらへのアクセスを承認されたロールを指定できます (このプロパティエディタを開くには、web.xml ノードを右クリックしてから、コンテキストメニューで「プロパティ」>「セキュリティ」タブ >「セキュリティ制限」>「...」ボタンを選択します) 。「追加」ボタンを選択すると、Web リソースを定義するためのダイアログが開きます。

図 2-14 は、URL パターン /servlet/myAlternateName を Web リソースとして定義する方法を示します (URL パターンと Web コンポーネントにマップする方法については、22 ページの「サーブレットの設定」と 24 ページの「JSP ページの設定」を参照してください)。

図 2-14 「Web リソースコレクション」ダイアログ

Web リソースは、URL パターンまたはそれらのサブセットに関連するすべての HTTP メソッドとして定義できます。各自で定義した Web リソースをセキュリティロールにマップすることもできます。このダイアログを完了すると、情報の概要が「セキュリティ制限」プロパティエディタに表示されます。図 2-15 は、Web リソースが `myNewServlet` という名前で設定された後の「セキュリティ制限」プロパティエディタを示しています。

図 2-15 「セキュリティ制限」プロパティエディタ

「編集」および「削除」ボタンを使用すると、プロパティエディタに表示されるセキュリティ制約をどれでも変更できます。

プログラム可能なセキュリティ

モジュール内の Web コンポーネントがプログラム可能なセキュリティを使用している場合は、セキュリティチェックコードで使用されるセキュリティロール参照を、モジュールレベルで宣言されたセキュリティロールにマップする必要があります。

プログラム可能なセキュリティ機能を使用する Web コンポーネントには、ユーザーの資格に直接アクセスするコードが含まれており、コンテナの宣言型セキュリティメカニズムが実行する以上の検証を実行します。次に例を示します。

```
...
context.isCallerInRole(roleRefMe);
...
```

ロールはモジュールレベルで定義されるため、このコンポーネントレベルのコードが記述された時点ではおそらく明らかにされていません。したがって、このコードは、後で実際のセキュリティロールにマップされるセキュリティロール参照 (`roleRefMe`) を使用します。このマッピングは、「編集サーブレット」ダイアログで行います。このダイアログを開くには「サーブレット」プロパティエディタでロール参照を使用するサーブレットを選択し、「編集」をクリックします。図 2-16 は、参照 `roleRefMe` がロール `Me` にマップされた「編集サーブレット」ダイアログを示しています。

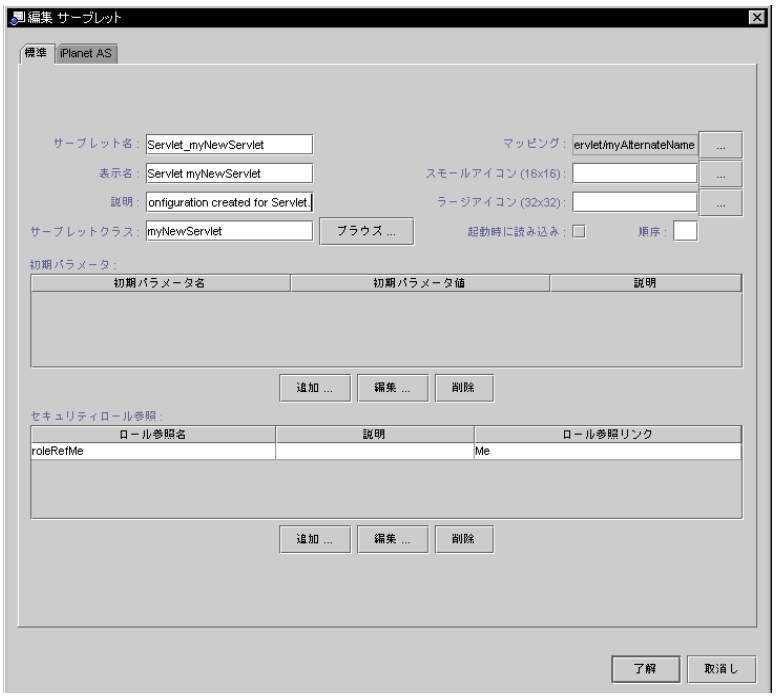


図 2-16 「編集サーブレット」ダイアログ

モジュールレベルでロールを宣言してから、このようなマッピングを実行する必要があります。

EJB 参照の設定

Enterprise Bean のメソッドを EJB モジュールで呼び出す Web コンポーネントは、EJB 参照によって呼び出しを行います。EJB 参照には次の 2 つの部分があります。

- JNDI ルックアップ。Enterprise Bean メソッドを呼び出す Web コンポーネントは、JNDI 命名機能を使用して名前付き Enterprise Bean へのリモート参照を取得し、このリモート参照を使用して、Enterprise Bean のメソッドの 1 つを呼び出します。
- Web モジュールの配備記述子の参照を示すエントリ。このエントリは、J2EE 実行環境への参照を宣言します。この実行環境は JNDI 機能との対話を管理します。

EJB 参照の JNDI ルックアップコード

Enterprise Bean メソッドを呼び出す Web コンポーネントは、次の例に示すようなルックアップコードを含む必要があります。

```
try {
    // Obtain Initial Context--Opens Communication With JNDI Naming:
    Context ic = new InitialContext();
    // Request Lookup of Enterprise Bean Name--Returns Remote Reference:
    Object obj = ic.lookup("java:comp/env/ejb/GreeterBean");
    // Narrow the Remote Reference to the Home Object Type
    HomeType myHome=(HomeType) javax.rmi.PortableRemoteObject.narrow(obj,
                                                    HomeType.class);
}
catch(Exception e) {
    System.out.println(e.toString());
    e.printStackTrace();
    return;
}
// Use the Home to Create the Remote
myGreeterRemote = myGreeterHome.create();
// Use the Remote to Execute Enterprise Bean Business Method
String theMessage = myGreeterRemote.getGreeting();
```

コメントは、各ステップの内容を要約しています。このコードは、ホームインタフェースの名前を指定して Enterprise Bean の種類を指定していますが、特定の Enterprise Bean は指定していません。この指定は、配備記述子エントリによって行われます。

EJB 参照の配備記述子エントリ

ほかの配備記述子エントリと同様、このエントリもプロパティエディタで設定します。この場合は、「EJB 参照」プロパティエディタです (このプロパティエディタを開くには、web.xml ノードを右クリックしてから、コンテキストメニューで 「プロパティ」 > 「参照」タブ > 「EJB 参照」 > 「...」ボタンという一連のコマンドを選択します)。

「追加」ボタンをクリックすると、新しい参照を追加するためのダイアログが開きます。参照を追加するには、ルックアップコードの JNDI 名に一致する参照名、ホームインタフェースの種類、およびリモートインタフェースの種類を指定する必要があります。図 2-17 は、これらのフィールドが入力された「編集 EJB 参照」ダイアログを示しています。

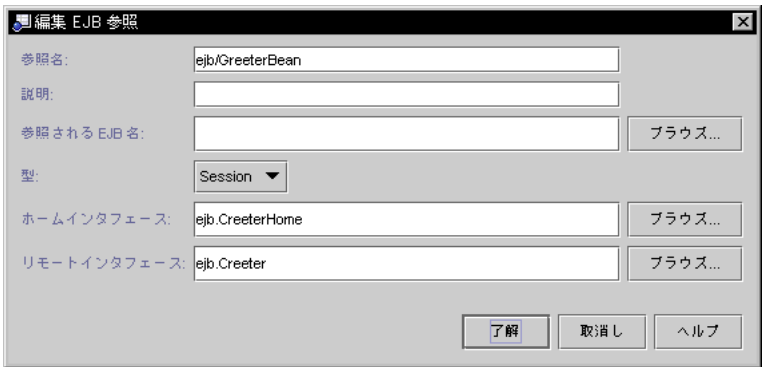


図 2-17 「編集 EJB 参照」ダイアログ

この時点で、Web モジュールだけをアセンブルしていて、同時に Web モジュールを J2EE アプリケーションにアセンブルする予定がない場合は、リンク可能な特定の Enterprise Bean へのアクセス権はありません。「了解」をクリックしてダイアログを閉じて、リンクしていない状態で参照を終了します。図 2-18 は、EJB 参照が設定されたがリンクされていない「EJB 参照」プロパティエディタを示しています。

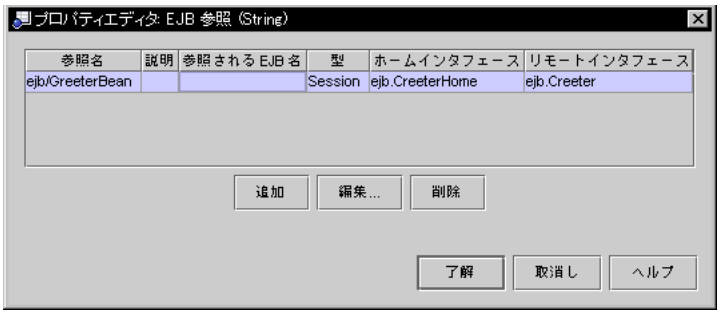


図 2-18 参照がリンクされていない状態のプロパティエディタ

EJB 参照をリンクするには、Web モジュールを、参照に指定されたホームインタフェースおよびリモートインタフェースを実装する Enterprise Bean を含む J2EE アプリケーションにアセンブルする必要があります。Web モジュールを J2EE アプリケーションにアセンブルする場合は、参照を編集してから (参照を選択して「編集」ボタ

ンをクリックする)、これらのインタフェースを実装する Enterprise Bean をブラウズできます。図 2-19 は、EJB 参照がリンクされた 「EJB 参照」プロパティエディタを示しています。

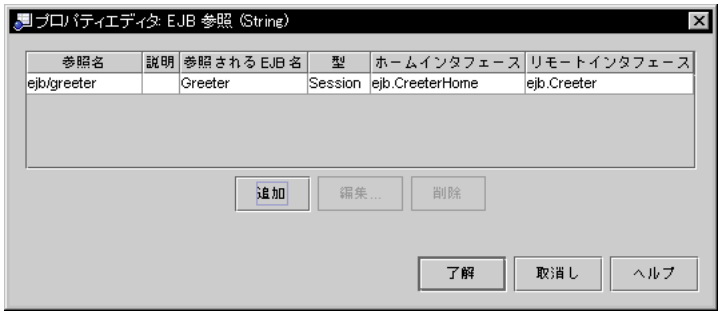


図 2-19 参照がリンクされた状態のプロパティエディタ

リソース参照の設定

Web モジュールは、データベースなどの外部のリソースにアクセスする場合、リソース参照によってアクセスを行います。たとえば、Web モジュールが人材データベースにアクセスする場合、実際に使用されるデータベースは配備環境によって異なります。これは、配備記述子で特定のデータベースに関連付けることができるリソース参照を使用して処理されます。

リソース参照の JNDI ルックアップ

Enterprise Bean メソッドを呼び出す Web コンポーネントは、次の例に示すようなルックアップコードを含む必要があります。

```
try {
    // Obtain Initial Context--Opens Communication With JNDI Naming:
    Context ic = new InitialContext();
    // Request Lookup of Resource--In This Example a JDBC Datasource:
    javax.sql.DataSource hrDB = (javax.sql.DataSource)
                                ic.lookup("java:comp/env/jdbc/Local_HR_DB");
}
catch(Exception e) {
    System.out.println(e.toString());
    e.printStackTrace();
    return;
}
```

リソース参照の配備記述子エントリ

ほかの種類の参照と同様、リソース参照もプロパティエディタで設定します。リソース参照の場合は、「リソース参照」プロパティエディタです (このプロパティエディタを開くには、`web.xml` ノードを右クリックしてから、コンテキストメニューで「プロパティ」 >「参照」タブ >「リソース参照」 >「...」ボタンという一連のコマンドを選択します)。

「追加」ボタンをクリックすると、新しい参照を追加するためのダイアログが開きます。参照を追加するには、ルックアップコードの JNDI 名に一致する参照名、リソースの種類および承認の種類を指定する必要があります。アプリケーションのアセンブル担当者および配備担当者が各環境で正しいリソースを見つけるのに役立つテキストを「説明」フィールドに追加してください。図 2-20 は、これらのフィールドが入力された「リソース参照」プロパティエディタを示しています。

図 2-20 「リソース参照」プロパティエディタ

EJB 参照とは違って、リソース参照は、アセンブルまたは配備時にほかの J2EE コンポーネントにリンクされません。代わりに名前付きリソースが、個別に環境に配備されます。Web コンポーネントは、JNDI ルックアップを使用して、実行時にこれらのリソースへの参照を取得します。この場合、特定のデータベース向けに構成された JDBC データソースが、`Local_HR_DB` という名前で個別に環境に配備されます。

環境エントリ参照の設定

環境エントリには次の 2 つの部分があります。

1. JNDI ルックアップ。環境エントリを使用する Web コンポーネントは、JNDI 命名機能を使用してエントリの値をルックアップします。

2. 環境エントリを示すエントリと、Web モジュールの配備記述子にあるその値。このエントリは、J2EE 実行環境への参照を宣言します。

環境エントリ参照の JNDI ルックアップ

環境エントリ値を使用する Web コンポーネントは、次の例に示すようなコードを必要とします。

```
try {
    // Obtain Initial Context--Opens Communication With JNDI Naming:
    Context ic = new InitialContext();
    // Request Lookup of Environment Entry Named "Cache Size":
    Integer cacheSize = ic.lookup("java:comp/env/CacheSize");
}
catch(Exception e) {
    System.out.println(e.toString());
    e.printStackTrace();
    return;
}
```

コメントは、各行の内容を要約しています。

環境エントリ参照の配備記述子エントリ

ほかの種類の参照の配備記述子エントリと同様、このエントリもプロパティエディタで設定します。この場合は、「環境エントリ」プロパティエディタです (このプロパティエディタを開くには、`web.xml` ノードを右クリックしてから、コンテキストメニューで 「プロパティ」 >「参照」タブ >「環境エントリ」 >「...」ボタンという一連のコマンドを選択します)。

「追加」ボタンをクリックすると、新しい参照を追加するためのダイアログが開きます。参照を追加するには、Web コンポーネントコードの JNDI 名に一致する参照名、データの種類および初期値を指定する必要があります。「説明」を使用すると、アプリケーションのアセンブル担当者および配備担当者が各環境に正しい値を指定するのに役立ちます。図 2-21 は、これらのフィールドが入力された「環境エントリ」プロパティエディタを示しています。

図 2-21 「環境エントリ」プロパティエディタ

EJB モジュールのアセンブル

EJB モジュールは、EJB コンポーネントと、J2EE 実行環境に配備して実行できるその他のリソースのセットです。モジュールを正常に実行するには、正しくアセンブルする必要があります。アセンブルを行うと、モジュールをアプリケーションサーバーごとに識別して、モジュールに必要な実行サービスを要求する配備記述子が生成されます。

ほとんどの場合、コンポーネントプロバイダがモジュールの最初のアセンブルを実行し、モジュールの作成、コンポーネントの追加、参照の設定などを行います (リンクは行わない場合もあります)。

コンポーネントプロバイダは、モジュールを完全にはアセンブルできない場合があります。コンポーネントプロバイダは、モジュールが使用される J2EE アプリケーションがどれであるかがわからないために、完成後のアプリケーションで決定される値を指定できないことがあります。また、コンポーネントプロバイダは、モジュールが最終的に配備されるアプリケーションサーバーがわからず、非標準のサーバー固有のプロパティの値を指定できないこともあります。

このため、EJB モジュールをアセンブルする標準的な手順はありません。EJB モジュールをアセンブルするときに行う処置は、アセンブルするコンポーネントと開発プロジェクトにおけるコンポーネントプロバイダの役割によって異なります。次に参考となるガイドラインを示します。

1. コンポーネントプロバイダは新しい EJB モジュールを作成します。

2. コンポーネントプロバイダはこのモジュールに Enterprise Bean を追加します。

3. コンポーネントプロバイダは、アセンブル作業を実行して、組み込まれる Enterprise Bean のビジネスロジックがユニットとして機能できるようにします。これらの作業は次のとおりです。

   a. Enterprise Bean で使用されるすべての参照が正しく設定されたかどうかを確認する。

   b. すべての EJB 参照をモジュールの Enterprise Bean にリンクする。

   c. コンテナ管理によるトランザクションを定義する。

   d. 汎用セキュリティロールを設定して、これらのロールにセキュリティロール参照をマップする。

   コンポーネントプロバイダがこれらのアセンブル作業の一部を実行できない場合、その作業は延期できます。たとえば、最初のアセンブル担当者がモジュールのセキュリティ要件を知らない場合は、この作業を後に残すことができます。

以下の節では、EJB モジュールに対して実行できるアセンブル作業について説明します。各作業について示された情報は、各自のモジュールに対して実行する必要がある作業を判断するのに役立ちます。

EJB モジュールの作成

次に示すいくつかの方法があります。

- Enterprise Bean ノードを右クリックしてから、コンテキストメニューの「新規 EJB モジュール」を選択します。すると、モジュールの名前を指定して、ファイルシステム内での位置を選択するためのダイアログが開きます。新しいモジュールを表すノードが選択したディレクトリに作成されて、作成を開始した Enterprise Bean が新しいモジュールに組み込まれます。
- ディレクトリノードを右クリックしてから、コンテキストメニューで「新規」>「J2EE」>「EJB モジュール」という一連のコマンドを選択します。すると、モジュールの名前を指定するためのダイアログが開きます。新しいモジュールを表すノードが、選択したディレクトリに作成されます。

新しいモジュールを表すノードが、選択したディレクトリに作成されます。つまり、モジュールを記述する情報、モジュールの配備記述子を生成するために最終的に使用される情報がこのディレクトリに保存されます。モジュールのコンポーネントのソースコードは、このディレクトリにはコピーされません。

モジュールのソースコードすべてを単一のファイルシステムに保存した場合は、そのファイルシステムの最上位にモジュールノードを配置できます。モジュールのソースコードが異なる複数のファイルシステムにあって、複数の開発者が所有している場合は、ソースコードを含むファイルシステムとは別に、モジュールとアプリケーションだけを含むディレクトリセットを作成します。

EJB コンポーネントおよびその他のリソースのモジュールへの追加

モジュールを作成したら、モジュールノードを右クリックして「EJB の追加...」を選択し、そこに Enterprise Bean を追加できます。すると、ダイアログが表示されて、ここで Enterprise Bean のためにマウントしたファイルシステムすべてをブラウズできます。Enterprise Bean を選択して「了解」をクリックすると、Enterprise Bean を表すノードがモジュールノードの下に追加されます。モジュールに属する Enterprise Bean すべてを追加するまで、このコマンドを使用し続けることができます。Enterprise Bean をモジュールに追加すると、IDE はほかの種類のリソースに対する Enterprise Bean の依存関係すべてに注意して、これらもモジュールに自動的に取り込みます。

モジュールに追加する Enterprise Bean のソースコードは、モジュールノードを保持するディレクトリにはコピーされません。

EJB モジュールの分析

ロールによっては、作成したばかりのモジュールまたは別の開発者から取得したモジュールを検討して、これを分析してから J2EE アプリケーションにアセンブルします。いずれの場合も、必要なアセンブル作業を判断し、これらの作業のうち、モジュールのビジネスロジック、アプリケーション全体、および配備環境に関する各自の知識によって実行できるものがどれかを判定する必要があります。モジュールに関して確認する点には次のものがあります。

- 配備記述子エントリがモジュール内のすべての参照について設定されているかどうか。
- モジュール内のほかの Enterprise Bean に対する EJB 参照がすべてリンクされているかどうか。

- 汎用セキュリティロールがモジュールに設定されているかどうか。モジュールの Enterprise Bean 内のセキュリティロール参照がこれらの汎用セキュリティロールにリンクされているかどうか。メソッドアクセス権がこれらの汎用セキュリティロールにマップされているかどうか。
- コンテナ管理によるトランザクションが定義されているかどうか。

別の開発者から取得したモジュールを処理する場合は、これらの項目の状態をチェックしてから、モジュールをアプリケーションにアセンブルする方法を判断する必要があります。

EJB 参照の設定

モジュールに追加した Enterprise Bean が別の Enterprise Bean のメソッドを呼び出す場合は、EJB 参照が必要です。参照には次の 2 つの部分があります。

- JNDI ルックアップ。別の Enterprise Bean のメソッドを呼び出す Enterprise Bean は、JNDI 命名機能を使用して、2 番目の Enterprise Bean に対するリモート参照を取得します。これはさらに、このリモート参照を使用して、2 番目の Enterprise Bean のメソッドを呼び出します。
- EJB モジュールの配備記述子の参照を示すエントリ。このエントリは、J2EE 実行環境への参照を宣言します。この実行環境は JNDI 機能との対話を管理します。

EJB 参照の JNDI ルックアップコード

最初の Enterprise Bean は、次の例に示すようなルックアップコードを含む必要があります。

```
try {
    // Obtain Initial Context--Opens Communication With JNDI Naming:
    Context ic = new InitialContext();
    // Request Lookup of Enterprise Bean Name--Returns Remote Reference:
    Object obj = ic.lookup("java:comp/env/ejb/GreeterBean");
    // Narrow the Remote Reference to the Home Object Type
    HomeType myHome = (HomeType) javax.rmi.PortableRemoteObject.narrow(obj,
                                                        HomeType.class);
}
catch(Exception e) {
    System.out.println(e.toString());
    e.printStackTrace();
```

```
    return;
    }
// Use the Home to Create the Remote
myGreeterRemote = myGreeterHome.create();
// Use the Remote to Execute Enterprise Bean Business Method
String theMessage = myGreeterRemote.getGreeting();
```

コメントは、各ステップの内容を要約しています。このコードは、ホームインタフェースの名前を指定して Enterprise Bean の種類を指定していますが、特定の Enterprise Bean は指定していません。この指定は、配備記述子エントリによって行われます。

EJB 参照の配備記述子エントリ

ほかの配備記述子エントリと同様、このエントリもプロパティエディタで設定します。この場合は、「EJB 参照」プロパティエディタです (このプロパティエディタを開くには、モジュールノードを右クリックしてから、コンテキストメニューで「プロパティ」>「参照」タブ >「EJB 参照 」>「...」ボタンという一連のコマンドを選択します)。

「追加」ボタンをクリックすると、新しい参照を追加するためのダイアログが開きます。参照を追加するには、ルックアップコードの JNDI 名に一致する参照名、ホームインタフェースの種類、およびリモートインタフェースの種類を指定する必要があります。図 2-22 は、これらのフィールドが入力された「編集 EJB 参照」ダイアログを示しています。

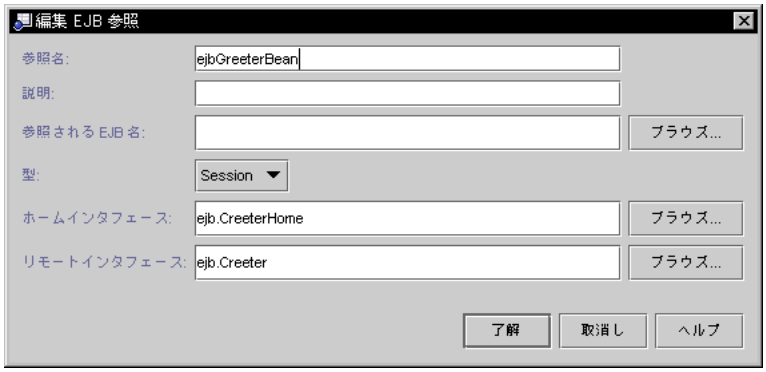


図 2-22 「編集 EJB 参照」ダイアログ

この時点で、参照に指定された Enterprise Bean がモジュールに含まれない場合は、「了解」をクリックしてダイアログを閉じて、参照をリンクされていない状態で残します。図 2-23 は、EJB 参照が設定されたがリンクされていない「EJB 参照」プロパティエディタを示しています。

処理中のモジュールが、参照に指定された Enterprise Bean が別の EJB モジュールに存在するように設計されている場合は、参照をリンクされていない状態のままで残すことができます。参照は、2 つのモジュールが J2EE アプリケーションにアセンブルされるとリンクされます。図 2-23 は、リンクされていない参照を示しています。

図 2-23 参照がリンクされていない状態のプロパティエディタ

参照に指定された Enterprise Bean が処理中の EJB モジュールに取り込まれたら、参照をリンクすることができます。「参照される EJB 名」フィールドの隣にある「ブラウズ...」ボタンをクリックしてください。すると、参照に指定されたインタフェースを実装する、モジュール内のすべての Enterprise Bean を一覧表示するダイアログが開きます。正しい Enterprise Bean を選択して「了解」をクリックしてください。図 2-24 は、EJB 参照がリンクされた後の「EJB 参照」プロパティエディタを示しています。

図 2-24 参照がリンクされた状態のプロパティエディタ

リソースファクトリ参照の設定

EJB モジュール内の Enterprise Bean のどれかが外部リソースにアクセスする場合は、リソース参照を EJB レベルで設定する必要があります。たとえば、EJB モジュールは人材データベースを処理するように設計されていますが、アクセスする実際のデータベースは配備環境によって異なります。この状態は、リソースファクトリによって処理できます。

データソース (`javax.sql.DataSource` のインスタンス) は、モジュールの外部に構成されていて、JNDI 名によって実行環境に配備されます。EJB モジュールのコードは名前によってデータソースにアクセスし、これを使用して正しいデータベースとの接続を取得します。

リソース参照は、次の 2 つの部分から構成されます。

- JNDI ルックアップ。リソースを使用する Enterprise Bean は、JNDI 命名機能を使用して、リソースに対するリモート参照を取得します。さらにこれは、この参照を使用して、リソースから必要なものを取得します。
- モジュールの配備記述子の参照を示すエントリ。このエントリは、J2EE 実行環境への参照を宣言します。この実行環境は JNDI 機能との対話を管理します。

これは通常コンポーネントプロバイダによって完成されますが、特定の実行環境に向けてモジュールを構成する場合は、この情報を検討する必要があります。

リソースファクトリ参照の JNDI ルックアップ

名前付きリソースを使用する必要がある Enterprise Bean は、次の例に示すようなコードを含む必要があります。

```
try {
    // Obtain Initial Context--Opens Communication With JNDI Naming:
    Context ic = new InitialContext();
    // Request Lookup of Resource--In This Example a JDBC Datasource:
    javax.sql.DataSource hrDB = (javax.sql.DataSource)
                                ic.lookup("java:comp/env/jdbc/Local_HR_DB");
}
catch(Exception e) {
    System.out.println(e.toString());
```

```
    e.printStackTrace();
    return;
}
```

リソースファクトリ参照の配備記述子エントリ

ほかの種類の参照と同様、リソース参照もプロパティエディタで設定します。リソース参照の場合は、「リソース参照」プロパティエディタです (このプロパティエディタを開くには、Enterprise Bean の論理ノードを右クリックしてから、コンテキストメニューで「プロパティ」>「参照」タブ >「リソースファクトリ参照」 >「...」ボタンという一連のコマンドを選択します)。

「編集」ボタンをクリックすると、リソースファクトリ参照を検討するためのダイアログが開きます。参照には、ルックアップコードの JNDI 名に一致する参照名、リソースの種類および承認の種類を指定する必要があります。「説明」に参照の目的を示すと、環境での正しいリソースの検出に役立ちます。図 2-20 は、これらのフィールドが入力された「リソースファクトリ参照」プロパティエディタを示しています。

図 2-25 「リソースファクトリ参照」プロパティエディタ

EJB 参照とは違って、リソース参照は、アセンブルまたは配備時にほかの J2EE コンポーネントにリンクされません。代わりに名前付きリソースが、個別にサーバー環境に配備されます。Enterprise Bean は、JNDI ルックアップを使用して、実行時にこれらのリソースへの参照を取得します。この場合、特定のデータベース向けに構成された JDBC データソースが、`Local_HR_DB` という名前で個別にサーバーの環境に配備されます。これは、Enterprise Bean のサーバー固有のプロパティによって処理できます。詳細については、78 ページの「リソースファクトリ参照」を参照してください。

環境エントリ参照の設定

環境エントリには次の 2 つの部分があります。

- JNDI ルックアップ。環境エントリを使用する Enterprise Bean は、JNDI 命名機能を使用してエントリの値をルックアップします。
- 環境エントリを示すエントリとモジュールの配備記述子にあるその値。このエントリは、J2EE 実行環境への参照を宣言します。この値は、モジュールが EJB JAR 形式でエクスポートされた場合でも配備記述子で変更できます。これにより、配備担当者とアセンブル担当者は、ソースコードにアクセスしないでモジュールの動作を変更できます。

環境エントリ参照の JNDI ルックアップ

環境エントリを使用する Enterprise Bean は、次の例に示すようなコードを必要とします。

```
try {
    // Obtain Initial Context--Opens Communication With JNDI Naming:
    Context ic = new InitialContext();
    // Request Lookup of Environment Entry Named "Cache Size":
    Integer cacheSize = ic.lookup("java:comp/env/CacheSize");
}
catch(Exception e) {
    System.out.println(e.toString());
    e.printStackTrace();
    return;
}
```

コメントは、各行の内容を要約しています。

環境エントリ参照の配備記述子エントリ

ほかの種類の参照の配備記述子エントリと同様、このエントリもプロパティエディタで設定します。この場合は、「環境エントリ」プロパティエディタです (このプロパティエディタを開くには、モジュールノードを右クリックしてから、コンテキストメニューで「プロパティ」>「参照」タブ >「 環境エントリ」 > 「...」ボタンという一連のコマンドを選択します)。

「追加」ボタンを使用すると、新しい参照を追加するためのダイアログが開きます。参照を追加するにはルックアップコードの JNDI 名に一致する参照名、データの型および初期値を指定する必要があります。「説明」を使用すると、アプリケーションの

アセンブル担当者および配備担当者が各環境に正しい値を指定するのに役立ちます。図 2-26 は、これらのフィールドが入力された「環境エントリ」プロパティエディタを示しています。

図 2-26 「環境エントリ」プロパティエディタ

コンテナ管理によるトランザクションの定義

以下の節では、コンテナ管理によるトランザクションの設定について説明します。Bean 管理によるトランザクションについては、『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。

モジュール内の Enterprise Bean のどれかがコンテナ管理によるトランザクションを使用する場合は、考えられる各トランザクションの適用範囲を定義する必要があります。コンテナ管理によるトランザクションによって IDE で作成された Enterprise Bean はすべて、デフォルトで「Required」トランザクション属性に設定されます。図 2-27 は、デフォルト設定を使用する「トランザクション設定」プロパティエディタを示しています。トランザクション属性は Enterprise Bean レベルで設定されて、Enterprise Bean のメソッドすべてによって継承されます (このプロパティエディタを開くには、モジュールノードを右クリックしてから、コンテキストメニューで「プロパティ」>「トランザクション設定」>「...」ボタンを選択します)。

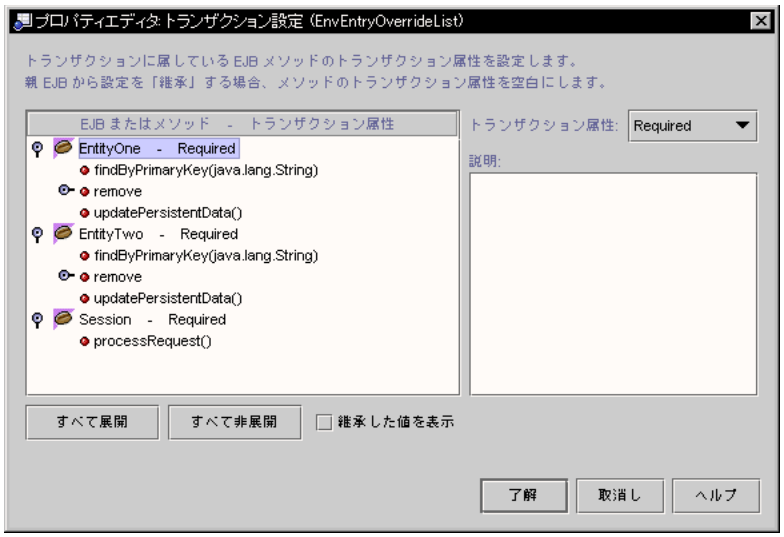


図 2-27 デフォルトのトランザクション属性

メソッドに「Required」トランザクション属性がある場合は、トランザクション形式で実行する必要があります。メソッドがアクティブなトランザクションなしで呼び出されると、コンテナは新しいトランザクションを開始します。メソッドが進行中のアクティブなトランザクションによって呼び出されると、コンテナはアクティブなトランザクションのメソッドを取り込みます。

EJB モジュールのビジネスロジックは、通常いくつかの Enterprise Bean に及んでいます。おそらく処理中のモジュールには、Web モジュールの表示ロジックによって呼び出された 1 つのセッション Bean があり、これらの要求を実行するために、モジュール内のいくつかのエンティティ Bean のメソッドを呼び出しています。図 2-28 は、このような種類のトランザクションを示しています。

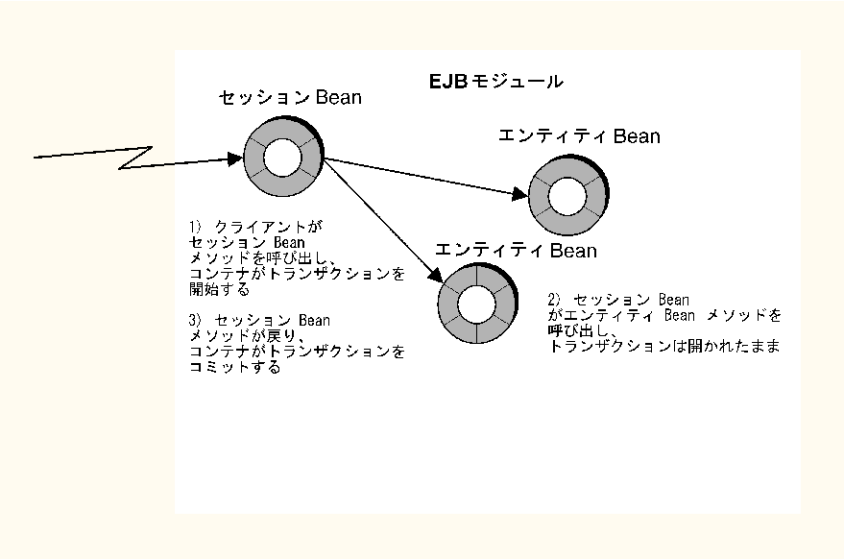


図 2-28 複雑なトランザクション

コンテナにこれらのトランザクション境界を認識させて、エンティティ Bean によって実行されるすべての作業を単一のトランザクションとして処理させてコミットさせるか、またはロールバックさせる必要があります。コンテナにこの処理を指示するには、トランザクションに関与する Enterprise Bean のトランザクション属性を変更します。表示層から受け取った新しい各要求を新しいトランザクションにする必要があるため、セッション Bean のトランザクション属性を「RequiresNew」に変更します。2 つのエンティティ Bean で呼び出されたメソッドのトランザクション属性を「Mandatory」に変更します。これは、メソッドを進行中のトランザクションによって呼び出す必要があることを示します。図 2-29 は、これらの変更が行われた後の「トランザクション設定」プロパティエディタを示しています。

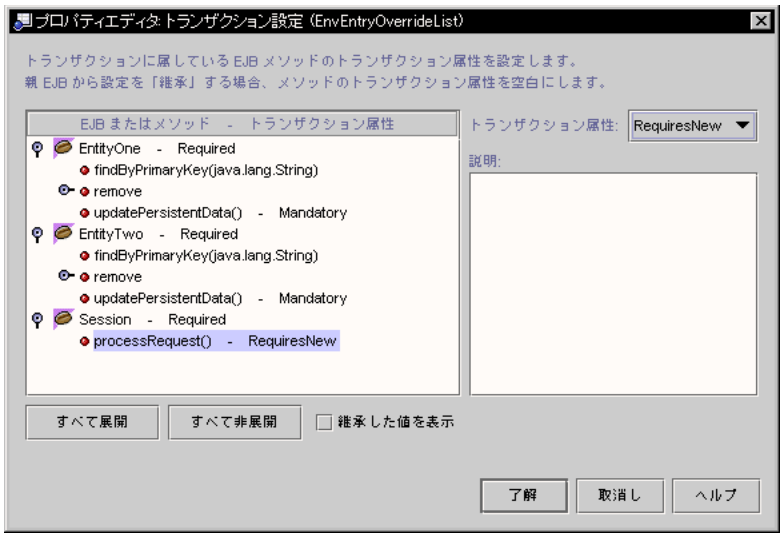


図 2-29 変更されたトランザクション設定

トランザクション属性はメソッドレベルで変更されています。このエディタは、Enterprise Bean レベルの「Required」ラベルと、変更されたメソッドの「Mandatory」または「RequiresNew」ラベルによってこのことを示しています。これらの属性は、各メソッドを選択して、「トランザクション属性」フィールドの値を変更することによって変更されています。

処理する各 EJB モジュールでは、処理するモジュールのビジネスロジックを分析して、このロジックによって暗黙指定される異なるトランザクションモデルを判断する必要があります。さらに、これらのトランザクションに関与する Enterprise Bean (またはそのメソッド) のトランザクション属性を設定して、これらのトランザクションモデルを実現します。

セキュリティの設定

一般に J2EE モデルは、セキュリティがモジュールレベルで設定されているものと想定しています。これは、ビジネスロジックを設計して開発するときに、コンポーネントプロバイダがセキュリティを考慮するものと想定します。コンポーネントプロバイダの多くは、ビジネスロジックを利用するユーザーのロールと、各リソースへのアクセスが許されているロールについて一般的に理解しています。

コンポーネントプロバイダの場合、 コンポーネントプロバイダは、人材データにアクセスするいくつかの Enterprise Bean を開発して、これらを EJB モジュールにアセンブルします。コンポーネントプロバイダは、その人材情報を維持するためにすべての社員から使用可能でなければならないデータと、人材関連事務、監督、監査、などの各ロールからだけを使用できるデータを認識しています。

コンポーネントプロバイダと初期モジュールアセンブル担当者は、これらのロールを表す汎用セキュリティロールを設定して、これらのデータにアクセスするメソッドにそのロールをマップできます。これらの汎用ロールは、後で配備環境の実際のユーザー名とグループ名にマップできます。この 2 番目のマッピングは、おそらくアプリケーションのアセンブル時、またはその後アプリケーションが本番環境に配備されるときに、アプリケーションレベルで実行できます。

また、プロジェクトによっては、コンポーネントプロバイダおよび初期モジュールアセンブル担当者が、セキュリティをビジネスロジックにどのように適用すればよいかがわからない場合があります。このような場合、担当者はセキュリティの設定を開発プロセスのもっと後の段階にまで延期できます。たとえば、アプリケーションのアセンブル担当者がモジュールレベルのセキュリティを処理してから、モジュールをアプリケーションにアセンブルすることができます。

EJB モジュールのセキュリティ設定は、Enterprise Bean メソッドへのセキュリティロールのマッピングから構成されます。この作業を行うには、次のことが必要です。

1. モジュールのサービスを使用できるユーザーのカテゴリを表す汎用セキュリティロールを宣言します。
2. これらのセキュリティロールを、モジュールの Enterprise Bean メソッドにマップします。これにより、どのロールがどのメソッドにアクセスできるかが決まります。
3. モジュールに、プログラム可能なセキュリティ機能を使用する Enterprise Bean が含まれる場合は、これらの Enterprise Bean で使用されるセキュリティロール参照をセキュリティロールにマップします。

以下の節では、これらのアセンブル作業を実行する方法を説明します。セキュリティの内容説明については、第 1 章を参照してください。

セキュリティロールの宣言

セキュリティロールは、「モジュールロールの編集」ダイアログで宣言します (このダイアログを開くには、モジュールノードを右クリックしてから、コンテキストメニューで「プロパティ」>「セキュリティロール」>「...」ボタンを選択します。「セキュリティロール」プロパティエディタが開いたら、「モジュールロールの編集」ボタンをクリックします)。

図 2-30 は、`Me` および `EveryoneElse` の 2 つのロールが宣言された後の「モジュールロールの編集」ダイアログを示しています。

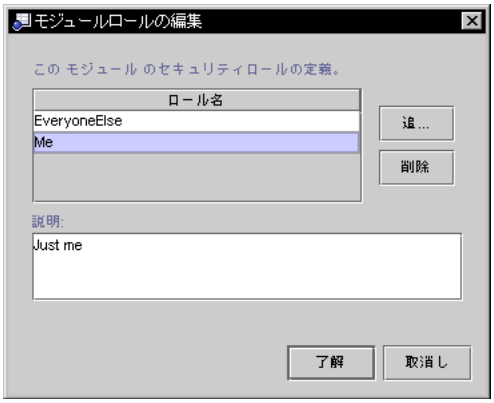


図 2-30 「セキュリティロール」プロパティエディタ

メソッドアクセス権へのセキュリティロールのマッピング

モジュールのセキュリティロールを宣言したら、各ロールに対して、モジュール内の特定の Enterprise Bean を実行する権限をロールに付与できます。これは「メソッドアクセス権」プロパティエディタで行います (このプロパティエディタを開くには、モジュールに含まれる Enterprise Bean を表すノードを右クリックして、コンテキストメニューから「プロパティ」>「メソッドのアクセス権 」>「...」ボタンを選択します)。プロパティエディタは表であり、 Enterprise Bean の各メソッドを示す行と、モジュールに宣言された各セキュリティロールを示す列が表示されます。図 2-31 は、前の節で宣言された 2 つのセキュリティロールによって、プロパティエディタがどのようになるかを示しています。

図 2-31 「メソッドのアクセス権」プロパティエディタ

ロールに対してメソッドの実行権を付与するには、ロールの列を検出します。ロールが実行権を持つすべてのメソッドについて、この列を下に移動し、そのメソッドの行にチェックマークをつけます。図 2-31 では、`EveryoneElse` ロールには `create()`、`getEJBHome()`、`getEJBMetaData()`、および `getGreeting()` の実行権があり、`Me` ロールにはメソッドのアクセス権がありません。

このエディタにはいくつかのショートカットがあります。リストされたメソッドすべてに対するアクセス権をロールに対して付与するには、そのロールの列で任意の場所をクリックして、「列を設定」ボタンをクリックします。ロールからすべてのアクセス権を削除するには、ロールの列で任意の場所をクリックして「列を消去」をクリックします。

「列を設定」ボタンをクリックすると、Enterprise Bean の現在定義されているすべてのメソッドに対するアクセス権が付与されます。ロールに対して、現在のすべてのメソッドおよび今後追加されるすべてのメソッドに対するアクセス権を付与するには、そのロールの列に移動して、最初の行である `*()` というラベルの付いた行をクリックします。

プログラム可能なセキュリティ

モジュール内の Enterprise Bean がプログラム可能なセキュリティを使用している場合は、セキュリティチェックコードで使用されるセキュリティ参照を、モジュールレベルで宣言されたセキュリティロールにマップする必要があります。

プログラム可能なセキュリティ機能を使用する Enterprise Bean には、ユーザーの資格に直接アクセスするコードが含まれており、コンテナの宣言型セキュリティ機構が実行する以上の検証を実行します。次に例を示します。

```
...
context.isCallerInRole(everyOne);
...
```

セキュリティロールはモジュールレベルで定義されるため、この Enterprise Bean コードが記述された時点ではおそらく明らかになっていません。したがって、このコードは、モジュールに Enterprise Bean を追加するときに実際のセキュリティロールにマップされる、セキュリティロール参照 (everyOne) を使用します。このマッピングは、「セキュリティロール」プロパティエディタで行います。図 2-32 は、セキュリティ参照がリンクされていない状態のプロパティエディタを示しています。

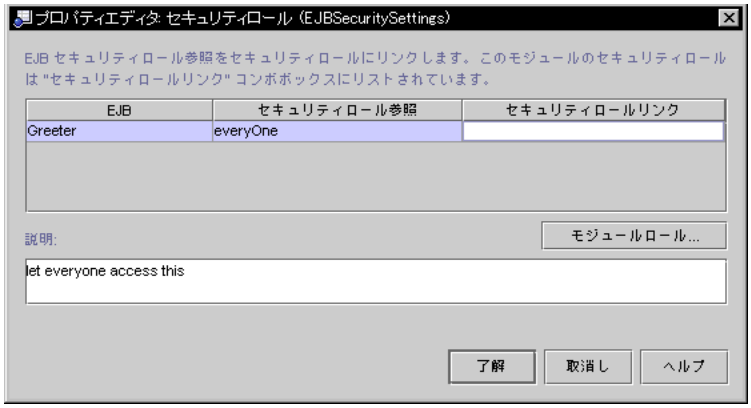


図 2-32 「セキュリティロール」プロパティエディタ

参照をマップするには、「セキュリティロールリンク」フィールドをクリックしてロールを選択します。図 2-33 は、everyOne リンクがマップされた後のプロパティエディタを示しています。

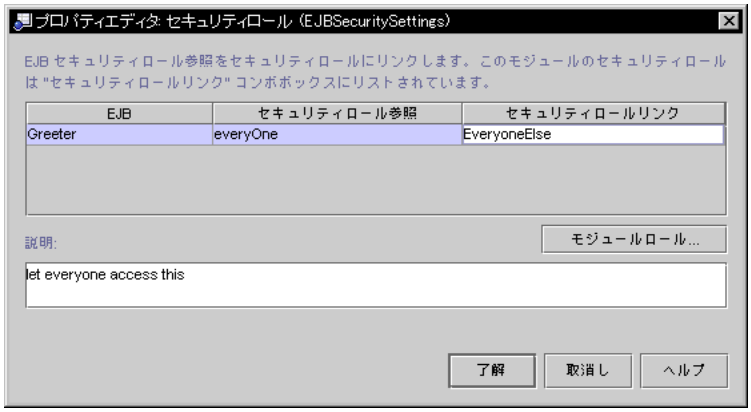


図 2-33 リンクされたセキュリティロール参照

J2EE アプリケーションのアセンブル

EJB モジュールは、Web モジュールと、J2EE 実行環境に配備して実行できるその EJB コンポーネントのセットです。アプリケーションを正常に実行するには、正しくアセンブルする必要があります。アセンブルを行うと、モジュールをアプリケーションサーバーごとに識別して、モジュールに必要な実行サービスを要求する配備記述子が生成されます。

J2EE アプリケーションは、それ自体がモジュールレベルでアセンブルされているモジュールからアセンブルされます。個々のモジュールがアセンブルされている程度は異なります。同様に、アプリケーションのアセンブル担当者がアプリケーションのビジネスロジックをどの程度理解しているかも異なります。アセンブル担当者がビジネスロジックの一部またはすべてを開発している場合もあれば、アセンブル担当者のアプリケーションに関する知識が、モジュール開発者提供の資料に基づいている場合もあります。

コンポーネントプロバイダは、アプリケーションアセンブル担当者に引き渡す前にモジュールを完全にはアセンブルできない場合があります。また、コンポーネントプロバイダは、モジュールが使用される J2EE アプリケーションがどれであるかがわからないために、完全なアプリケーションで決定される値を指定できないことがあります。コンポーネントプロバイダは、モジュールが最終的に配備されるアプリケーションサーバーがわからず、非標準のサーバー固有のプロパティの値を指定できないこともあります。

このため、J2EE アプリケーションをアセンブルする標準的な手順はありません。アプリケーションをアセンブルするときにとる処置は、開発プロジェクトにおける役割と、処理中のモジュールの内容によって異なります。アプリケーションをアセンブルするためのいくつかのガイドラインを次に示します。

1. アプリケーションアセンブル担当者が、アセンブルするモジュールを収集します。
2. アプリケーションアセンブル担当者が各モジュールを評価して、モジュールレベルのアセンブル作業をさらに実行する必要があるかどうかを判断します。詳細については、19 ページの「Web モジュールのアセンブル」 または 42 ページの「EJB モジュールの分析」 を参照してください。アプリケーションアセンブル担当者は、必要なモジュールレベルのアセンブル作業をすべて実行します。
3. アプリケーションアセンブル担当者は、新しい J2EE アプリケーションを作成します。
4. アプリケーションアセンブル担当者は、Web モジュールと EJB モジュールをアプリケーションに追加します。
5. アプリケーションアセンブル担当者は、結合されたモジュールがユニットとして機能できるようにする、その他のアプリケーションアセンブル作業を実行します。これらの作業は次のとおりです。
   a. すべての EJB 参照がリンクされていることを確認する。
   b. 環境エントリのコンポーネントレベルまたはモジュールレベルの設定を無効にする。
   c. モジュールレベルのセキュリティロールをアプリケーションレベルのロールにマップする。

   アプリケーションアセンブル担当者が最終的な配備環境を知らない場合、ステップ b と c は、実際の配備プロセスまで延期できます。

以下の節では、J2EE アプリケーションモジュールで実行できるアセンブル作業について説明します。各作業に関する情報は、各自のアプリケーションに実行する必要がある作業を判断するのに役立ちます。

J2EE アプリケーションの作成

次に示すいくつかの方法があります。

- EJB モジュールを右クリックしてから、コンテキストメニューの「新規アプリケーション」を選択します (または、Web モジュールの `WEB-INF` ノードを右クリックしてから、「新規 J2EE アプリケーション」を選択します)。すると、モジュールの名前を指定して、ファイルシステム内での位置を選択するためのダイアログが開きます。新しいアプリケーションを表すノードが、選択したディレクトリに作成されて、開始したモジュールが新しいアプリケーションに組み込まれます。
- ディレクトリノードを右クリックしてから、コンテキストメニューで「新規」>「J2EE」>「アプリケーション」という一連のコマンドを選択します。すると、アプリケーションの名前を指定するためのダイアログが開きます。新しいアプリケーションを表すノードが、選択したディレクトリに作成されます。

新しいアプリケーションが、選択したディレクトリ内のノードによって表されます。つまり、アプリケーションを記述する情報、アプリケーションの配備記述子を生成するために最終的に使用される情報がこのディレクトリに保存されます。アプリケーションのコンポーネントのソースコードは、このディレクトリにはコピーされません。

アプリケーションのソースコードすべてを単一のファイルシステムに保存した場合は、そのファイルシステムの最上位にアプリケーションノードを配置できます。アプリケーションのソースコードが異なる複数のファイルシステムにあって、複数の開発者が所有している場合は、ソースコードを含むファイルシステムとは別に、アプリケーションとモジュールだけを含むディレクトリセットを作成します。このようなファイルシステムは、ソースコードのディレクトリ構造とは異なるアプリケーションの構造を確認するのに役立ちます。

アプリケーションへのモジュールの追加

アプリケーションを作成したら、アプリケーションノードを右クリックして、「モジュールの追加」を選択して、そこにモジュールを追加できます。すると、ダイアログが表示されて、モジュールのためにマウントしたファイルシステムすべてをブラウズして、アプリケーションに追加できます。モジュールを選択して「了解」をクリックすると、モジュールを表すノードがアプリケーションノードの下に追加されます。

- Web モジュールを追加するには、モジュールの `WEB-INF` ノードを選択します。
- EJB モジュールを追加するには、モジュールノードを選択します。

アプリケーションに属するモジュールすべてを追加するまで、このコマンドを使用し続けることができます。アプリケーションに追加するモジュールのソースコードは、アプリケーションノードを保持するディレクトリにはコピーされません。例については、図 2-3を参照してください。

アプリケーションの分析

ロールによって、作成したモジュールまたはほかの開発者から取得したモジュールのどちらかを J2EE アプリケーションにアセンブルします。いずれの場合も、必要なアプリケーションレベルのアセンブル作業を判断し、これらの作業のうち、各モジュールのビジネスロジック、アプリケーション全体、および配備環境に関する各自の知識によって実行できるものがどれかを判定する必要があります。アプリケーションに関して確認する点には次のものがあります。

- モジュールのアセンブル時に延期された作業があって、モジュールをアプリケーションにアセンブルする前に完了する必要があるかどうか。ヘルプについては、42 ページの「EJB モジュールの分析」を参照してください。
- 1 つのモジュールから、リンクする必要がある別のモジュールへの EJB 参照があるかどうか。
- アプリケーションレベルでオーバーライドする必要がある環境値があるかどうか。
- 重複する名前を持つ異なるモジュールに Enterprise Bean があるかどうか。
- 単一のロールで解釈処理する必要があるモジュールレベルのセキュリティロールがあるかどうか。
- モジュールレベルのセキュリティロールを、配備環境で使用されるロールにマッピングできるかどうか。
- アプリケーションのいずれかの Web モジュールに Web コンテキスト (サーブレットコンテキスト) を設定する必要があるかどうか。

Web モジュールへのサーブレットコンテキストの設定

ユーザーは、URL がマップされた Web モジュールリソースにアクセスできます。Web リソースの URL の一般的な形式は次のとおりです。

```
http://hostname:port/servlet_context/servlet/servlet_name
```

サーバールートとリソース名にはさまれた位置情報を見ればわかるように、サーブレットコンテキストは、アプリケーション内のリソースと同じサーバーインスタンスに配備されたほかのアプリケーション内のリソースを区別しています。サーブレットコンテキストは、取り込まれた Web モジュールのプロパティシートで設定します (取り込まれた Web モジュールを表すアプリケーションのサブノードを右クリックして、「プロパティ」 > 「Web コンテキスト」を選択します)。図 2-34 は、Web コンテキストプロパティに値 `myContext` が指定されたプロパティシートを示しています。プロパティの名前は「Web コンテキスト」と指定されていますが、各自で指定した値はサーブレットコンテキストとして使用されています。

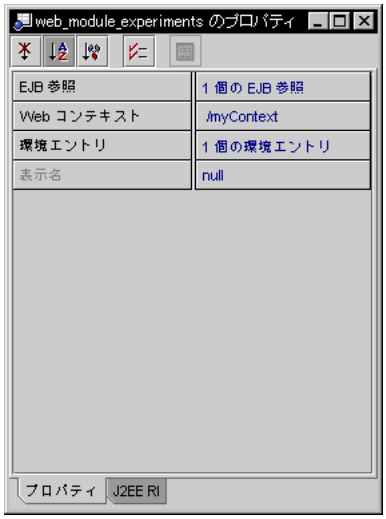


図 2-34 取り込まれた Web モジュールのプロパティシート

この値を使用した場合、アプリケーションの Web リソースの URL 形式は次のようになります。

```
http://hostname:port/myContext/servlet/servlet_name
```

サーブレットコンテキストを指定しないと、デフォルトで / に設定されて、アプリケーションの Web リソースの URL は次のような一般的な形式になります。

```
http://hostname:port/servlet/servlet_name
```

EJB 参照の解釈処理

アプリケーションに追加したモジュールのどれかに、別のモジュールの Enterprise Bean に対する参照がある場合は、これらの参照をリンクする必要があります。これは、アプリケーションの「EJB 参照」プロパティエディタで行います (このエディタを開くには、アプリケーションノードを右クリックしてから、コンテキストメニューで「プロパティ」>「EJB 参照」>「...」ボタンを選択します)。図 2-35 は、1 つの EJB 参照を持つプロパティエディタを示しています。

図 2-35 「EJB 参照」プロパティエディタ

アプリケーション内の参照は、モジュール (例では `helloweb`) と参照名 (JNDI ルックアップコードと配備記述子に表示される名前) によって識別されます。参照がリンクされていない場合、その「値」フィールドは空になります。参照をリンクするには、「値」フィールドをクリックします。参照に指定されたインタフェースに一致する Enterprise Bean のリストが表示されます。

注 - 参照がすでに同じモジュールの Enterprise Bean にリンクされている場合は、Enterprise Bean 名が「値」フィールドに表示されます。

環境エントリのオーバーライド

アプリケーションに環境エントリが含まれる場合は、それらに設定された値をモジュールレベルで変更できます。これは、アプリケーションの「環境エントリ」プロパティエディタで行います (このエディタを開くには、アプリケーションノードを右クリックしてから、コンテキストメニューで「プロパティ」>「 環境エントリ」>「...」ボタンを選択します)。図 2-36 は 1 つの環境エントリを持つこのプロパティエディタを示しています。

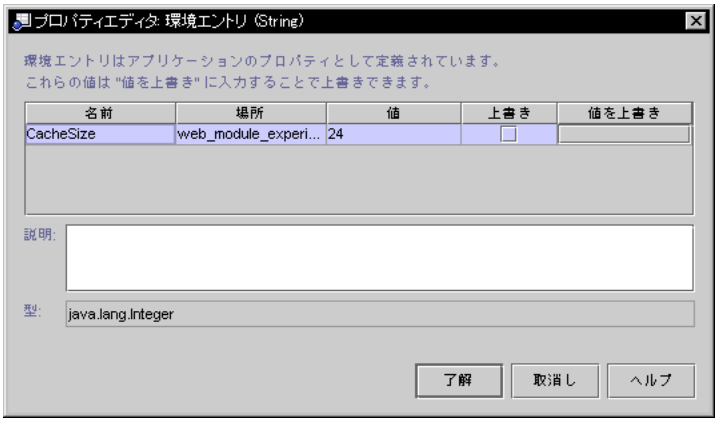


図 2-36 「環境エントリ」プロパティエディタ

アプリケーションで検出された環境エントリは、名前 (JNDI ルックアップコードと配備記述子で使用されるものと同じ名前) とモジュールによって識別されます。モジュールレベルで設定された値は「値」フィールドに表示されます。この値をオーバーライドするには、「上書き」列のチェックボックスをクリックして、「値を上書き」フィールドにオーバーライド値を指定します。

アプリケーションレベルのセキュリティの設定

アプリケーションアセンブル担当者に対して、新しくアセンブルされたアプリケーションは次のいずれかの状態で示されます。

- アプリケーション内の 1 つまたは複数のモジュールにセキュリティロールが定義されていない場合は、モジュールレベルでロールを定義する必要があります。29 ページの「セキュリティの設定」と 52 ページの「セキュリティの設定」を参照してください。

- セキュリティが汎用ロールによってモジュールレベルで設定されている場合は、類似のロールに対して異なるモジュールの異なる名前が指定されています。この場合は、同等のロールすべてを同じアプリケーションレベルのロールにマッピングする必要があります。
- モジュールレベルでセキュリティを設定したコンポーネントプロバイダが配備環境を知っていて、モジュールレベルのロールが相互にマップされている場合は、追加のマッピングを何も行う必要はありません。

追加マッピングが必要な場合は、アプリケーションの「セキュリティロール」プロパティエディタで行います (このエディタを開くには、アプリケーションノードを右クリックしてから、コンテキストメニューで「プロパティ」>「セキュリティロール」>「...」ボタンを選択します)。図 2-37 は、2 つのセキュリティロールがモジュールレベルで定義されているプロパティエディタを示しています。

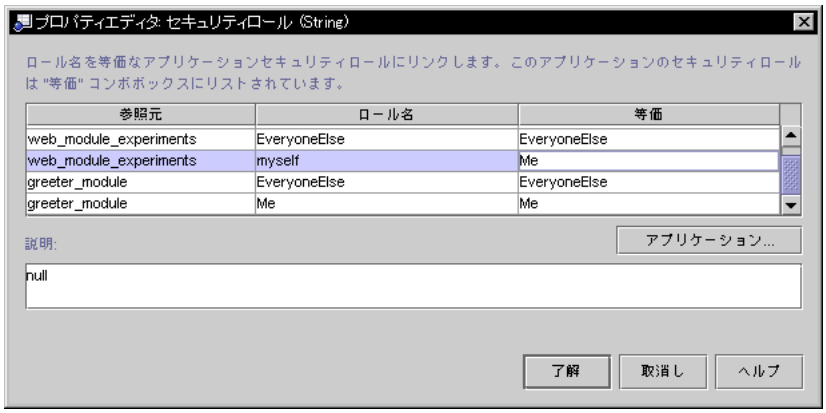


図 2-37 アプリケーションレベルの「セキュリティロール」プロパティエディタ

モジュールレベルで宣言されたセキュリティロールは、ダイアログの最初の 2 つの列に表示されます。各ロールは、そのモジュールと名前によって識別されます。モジュールレベルロールごとに、IDE はデフォルトのアプリケーションレベルロールを作成しますが、この名前はモジュールレベルロールと同じです。アプリケーションレベルロールは、「等価」列に表示されます。

ロールのこのエディタによってロールの違いを解釈処理する方法は 2 つあります。図 2-37 の例は、2 つのモジュールを持つアプリケーションを示しています。各モジュールには 2 つのセキュリティロールがあります。違いは 1 つで、`web_module_experiments` のロール名は `myself` ですが、`greeter_module` のロール名は `Me` です。これらのロールは等価であるため、1 つのアプリケーションレベルロールだけが必要です。図では、この違いはロール `myself` をロール `Me` に再

マップすることによって解釈処理されています。実際にはここで、Me および myself の両方のモジュールレベルロールが、同じアプリケーションレベルロール Me にマップされています。

完全に新しいロールをアプリケーションレベルで作成して、複数のモジュールレベルロールをそれにマップすることもできます。たとえば、アプリケーションのモジュールの 1 つに sa という名前のロールがあって、もう 1 つのモジュールに sadmin という名前のロールがある場合に、この違いを、新しいアプリケーションレベルロール sysadmin を作成することによって解釈処理するとします。この作業を行うには、「アプリケーションロールの編集」ボタンをクリックして、アプリケーションレベルのロールを宣言するためのダイアログを開きます。

sysadmin ロールを宣言したら、「セキュリティロール」プロパティエディタに戻ります。「等価」列をクリックして、両方のモジュールレベルロールを再マップします。これにより、アプリケーションレベルロールが表示されます。sysadmin ロールを選択してください。

配備記述子の表示と編集

通常、モジュールおよびアプリケーションのプロパティシートを使って、配備記述子の内容を制御します。プロパティを設定して、配備記述子の内容を設定します。IDE を使用すると、モジュールおよびアプリケーションの実際の XML 配備記述子を表示できます。

配備記述子の表示

J2EE アプリケーション、取り込まれた Web モジュール、および取り込まれた EJB モジュールの配備記述子を表示できます。配備記述子を表示するには、ノードを右クリックして、コンテキストメニューの「配備記述子の表示」を選択します。配備記述子はソースエディタに読み取り専用モードで表示されます。

EJB モジュール配備記述子の編集

EJB モジュール配備記述子は編集できます。この作業を行うには、EJB モジュールノードを右クリックします。

第3章

J2EE モジュールとアプリケーションの実行

Forte for Java の配備および実行機能を使用すると、エンタープライズアプリケーションを対話形式で開発できます。適切な Web サーバーまたはアプリケーションサーバーがインストールされていることを前提として、各担当者または各チームは、Web アプリケーションまたは J2EE アプリケーションの開発とアセンブル、その配備、テスト目的での実行、ソースコードやコンポーネントプロパティの修正、再配備と再テストなどを行うことができます。再アセンブルは、テストでアセンブルの問題が生じなかった場合には必要ありません。

実際の配備では、Forte for Java 実行機能は、サーバーに用意されている配備ツールの代替手段にはなりません。アプリケーションのテストを終了したら、WAR ファイルまたは EAR ファイルを生成して、サーバーの配備ツールを使用して配備できます。

この章では、アセンブルされた Web アプリケーションおよび J2EE アプリケーションを IDE 内から配備して、Web ブラウザを介してテスト目的で実行する方法について説明します。IDE には、コンポーネントレベルでテストを実行するための機能もあります。これらの機能は、『Web コンポーネントのプログラミング』、『Enterprise JavaBeans コンポーネントのプログラミング』、および『Web サービスのプログラミング』に記載されています。

エクスプローラウィンドウへのサーバーの表示

Web アプリケーションまたは J2EE アプリケーションをサーバーに配備するには、サーバーと対話する必要があります。このプロセスを簡略化するために、Forte for Java は、Web サーバーおよびアプリケーションサーバーをノードとしてエクスプローラウィンドウに表示します。

エクスプローラウィンドウのほかのノードと同様、これらのサーバーノードにもプロパティシートとコンテキストメニューコマンドがあり、IDE 内からのサーバーとの対話を管理するのに役立ちます。この節では、サーバーノードを特定して説明します (これらのノードを表示する機構については、付録 Aを参照してください)。

サーバーレジストリノード

最上位にはサーバーレジストリノードがあります。このノードは、ほかのサーバー関連ノードをグループ化するものです。このノードには、独自のコマンドやプロパティがありません。図 3-1 は、デフォルトのサブノードを持つサーバーレジストリを示しています。

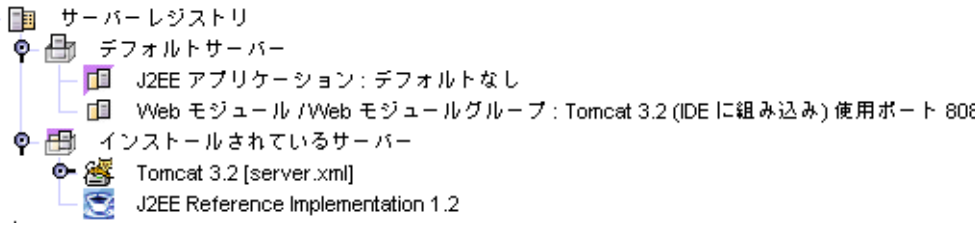


図 3-1　サーバーレジストリとデフォルトのサブノード

「インストールされているサーバー」ノード

このノードは、ほかのノードをグループ化するために存在します。このノードには、独自のコマンドやプロパティがありません。図 3-1 は、デフォルトのサブノードを持つ「インストールされているサーバー」ノードを示しています。

サーバー製品ノード

これらのノードは、「インストールされているサーバー」ノードのサブノードです。これらの各ノードは、IDE によって認識される Web サーバーまたは アプリケーションサーバー製品を表します (これらのノードは、実際には指定されたサーバーノードと対話可能な IDE プラグインモジュールの存在を表します。サーバープラグインの詳細については、付録 Aを参照してください)。図 3-1は、デフォルトインストール後の IDE と、Tomcat Web サーバーおよび J2EE RI アプリケーションサーバーを持つ「インストールされているサーバー」ノードを示しています。

サーバー製品がインストールされて、対応するサーバー製品ノードがエクスプローラウィンドウに表示されると、IDE はサーバーのインスタンスを認識して、それにアプリケーションを配備できます。これらの各ノードにはコンテキストメニューとプロパティシートがありますが、各ノードの機能は、サーバー製品とプラグインモジュールによって決まります。

手順はサーバー製品によって異なりますが、通常、サーバーを使用するには、これらのいずれかのサーバーを構成して、サーバー製品の特定のインストールを認識します。次に例を示します。

- Tomcat サーバーは製品とともにインストールされるため、サーバーの位置はすでにわかっています。Tomcat 3.2 を起動してそこに配備するために、追加構成は必要ありません。
- `J2EE Reference Implementation 1.2` ノードは、Reference Implementation の (J2SDKEE) インストールディレクトリを指定して構成する必要があります。ノードを右クリックして「プロパティ」を選択してください。RIHome プロパティを選択して、Reference Implementation インストールディレクトリを入力します。この作業を行うと、ノードを使用して RI サーバーのインスタンスを作成、起動、停止できます。

IDE でサーバー製品を設定する方法については、『Forte for Java, Enterprise Edition インストールガイド』を参照してください。

サーバーインスタンスノード

アプリケーションは、サーバー製品の実行インスタンスに配備します。IDE 内からの配備を行うには、インストールされているサーバーのインスタンスを起動して、そのインスタンスを表すノードを作成する必要があります。

この手順はサーバー製品によって異なります。J2EE RI の場合は、`J2EE Reference Implementation 1.2` ノードを右クリックして、コンテキストメニューから「サーバーインスタンスの追加」を選択します。これにより、J2EE RI サーバープログラムが起動して、新しいサーバーインスタンスを表すサブノードが追加されます。図 3-2 は、この手順で Reference Implementation サーバーインスタンスを追加した後のサーバーレジストリを示しています。追加したサーバーインスタンスは、「`RI インスタンス1`」となっています。

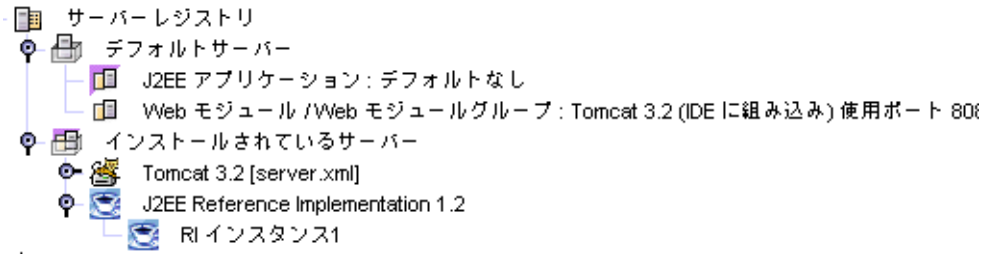


図 3-2　J2EE RI のサーバーインスタンスノード

使用しているサーバー製品が何であっても、同様の作業を行う必要があります。

デフォルトのサーバーノード

デフォルトの Web サーバーとデフォルトのアプリケーションサーバーを指定できます。デフォルトのサーバーを指定すると、ほかのサーバーを指定しない限り、そのデフォルトのサーバーにアプリケーションが配備されます。配備するアプリケーションごとにターゲットのサーバーを指定することもできます。

サーバーインスタンスをデフォルトのサーバーにするには、そのサーバーインスタンスノードを右クリックして、コンテキストメニューから「デフォルトとして設定」を選択します。図 3-3 は、`RI インスタンス1` がデフォルトのアプリケーションサーバーに指定された後のデフォルトのサーバーノードを示しています。

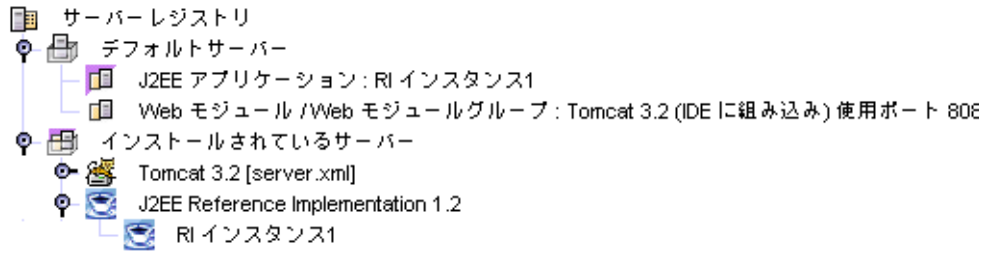


図 3-3　RI インスタンス1 は、デフォルトのアプリケーションサーバーです。

サーバー固有のプロパティ

IDE には、サーバープロパティとサーバーインスタンスノードだけでなく、異なるサーバー製品で必要な非標準プロパティを設定する方法もあります。サーバープラグインモジュールをインストールすると、エクスプローラウィンドウの「実行時」タブにサーバー製品ノードが追加される方法については既に説明しました。さらに、プラグインをインストールすると、コンポーネント、モジュール、およびアプリケーションのプロパティシートにタブが追加されます。たとえば、図 3-4 は、「Tomcat 3.2 」タブのあるプロパティシートを示しています。

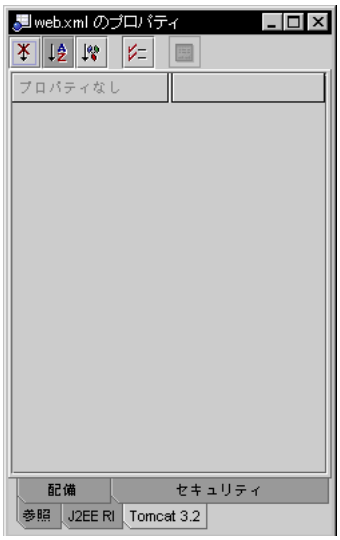


図 3-4　Web モジュールの非標準「プロパティ」タブ

これらのタブを使用して、特定サーバーの非標準 (J2EE 標準で不要な)プロパティに関する情報を指定します。

Web アプリケーションの実行

この節では、IDE 内から Web アプリケーションを実行するためのガイドラインをいくつか示します。

1. アセンブルされた Web アプリケーションから始めます。アプリケーションのアセンブルが完全かどうかを検討します。特に、モジュールレベルのセキュリティロールとアプリケーションレベルのロールのマッピングを確認します。この作業は、意図的に配備段階まで延期されている場合があります。

2. 配備の構成に役立つ、標準の配備記述子プロパティをすべて設定します。これらのプロパティはこの章で説明しています。

3. 使用する予定のサーバーに、非標準のサーバー固有のプロパティがないかを確認します。たとえば、アプリケーションプロパティシートには、追加のセキュリティロールマッピングを可能にする J2EE RI サーバーのタブがあります。これらのプロパティは、アプリケーションプロパティシートだけでなく、コンポーネントやモジュールのプロパティシートにも表示されます。

4. IDE 内から配備して実行する場合は、配備先のサーバーインスタンスが必要です。サーバーインスタンスノードに詳しくない場合は、詳細について 69 ページの「サーバーインスタンスノード」を参照してください。サーバー製品をインストールする必要がある場合は、『Forte for Java, Enterprise Edition インストールガイド』を参照してください。

5. ターゲットの Web サーバーインスタンスを選択するか、またはデフォルトの Web サーバーインスタンスを使用します。

6. 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」を選択して、アプリケーションを実行します。ブラウザが起動して、Web アプリケーションの開始ページが開きます。

IDE の外部に配備する準備ができたら、WAR ファイルを生成して、サーバーの配備ツールを使用して配備できます。詳細については、80 ページの「Web モジュールまたは Web アプリケーションのエクスポート」を参照してください。

非標準プロパティの設定

Web アプリケーションのプロパティシートを開きます (web.xml ノードを右クリックしてから、コンテキストメニューの「プロパティ」を選択します)。インストールされているサーバーノードすべてのタブがあります。図 3-4 は、Tomcat 3.2 サーバーのタブを示しています。

Web アプリケーションには、現在、Tomcat 3.2 および J2EE Reference Implementation 1.2 サーバーの非標準プロパティはありません。インストールして設定するほかのサーバー製品には非標準プロパティが存在する場合があります。

Web アプリケーションの配備

Web アプリケーションを 2 つの独立した操作として配備して実行する場合は、アプリケーションの WEB-INF ノードを右クリックして、コンテキストメニューの「配備」を選択します。アプリケーションは Web サーバーに配備されます。

IDE の出力ウィンドウで配備の進捗に関するメッセージを確認してください。配備が完了したら、アプリケーションを実行できます。

配備された Web アプリケーションの実行

実行コマンドを使用した場合は、ブラウザが自動的に起動して、アプリケーションの開始ページが開きます。配備と実行を別々に行なった場合は、ブラウザを各自で起動して、アプリケーションの Web リソースの 1 つに対応する URL を開くことができます。

開始ページを設定した場合は、URL は次に示す例のようになります。これは、デフォルトポート 8080 のローカルマシンで実行される Tomcat 3.2 に配備した場合を想定しています。

```
http://localhost:8080/myWebApp/index.html
```

myWebApp は、アプリケーションのコンテキストルートまたはサーブレットコンテキストであり、WEB-INF プロパティシートに設定されています。このプロパティの詳細については、20 ページの「サーブレットコンテキストの設定」を参照してください。

サーブレットなどの特定のリソースを実行する場合、URL は次に示す例のようになります。

```
http://localhost:8080/myWebApp/servlet/myServlet
```

J2EE アプリケーションの実行

この節では、IDE 内から J2EE アプリケーションを実行するためのガイドラインを示します。

1. アセンブルされた J2EE アプリケーションから始めます。アプリケーションのアセンブルが完全かどうかを検討します。特に、モジュールレベルのセキュリティロールとアプリケーションレベルのロールのマッピングを確認します。この作業は、意図的に配備段階まで延期されている場合があります。

2. 配備の構成に役立つ標準の配備記述子プロパティをすべて設定します。これらのプロパティはこの章で説明しています。

3. 使用する予定のサーバーに、非標準のサーバー固有のプロパティがないかを確認します。たとえば、アプリケーションプロパティシートには、サーバー環境で使用されるユーザー名とグループ名へのアプリケーションセキュリティロールのマップを可能にする J2EE RI サーバーのタブがあります。

4. IDE 内から配備して実行する場合は、配備先のサーバーインスタンスが必要です。サーバーインスタンスノードに詳しくない場合は、詳細について 69 ページの「サーバーインスタンスノード」を参照してください。サーバー製品をインストールして構成する必要がある場合は、『Forte for Java, Enterprise Edition インストールガイド』を参照してください。

5. ターゲットのアプリケーションサーバーインスタンスを選択するか、またはデフォルトのアプリケーションサーバーインスタンスを使用します。

6. アプリケーションを右クリックして、コンテキストメニューから「配備」を選択します。

7. アプリケーションの Web リソースにマップされた URL を開いて、Web ブラウザに配備されたアプリケーションを実行します。

IDE の外部に配備する準備ができたら、EAR ファイルを生成して、サーバーのツールを使用して配備できます。詳細については、82 ページの「J2EE アプリケーションのエクスポート」を参照してください。

標準アプリケーションプロパティの設定

このダイアログを開くには、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で「EJB 名を表示」を選択します。

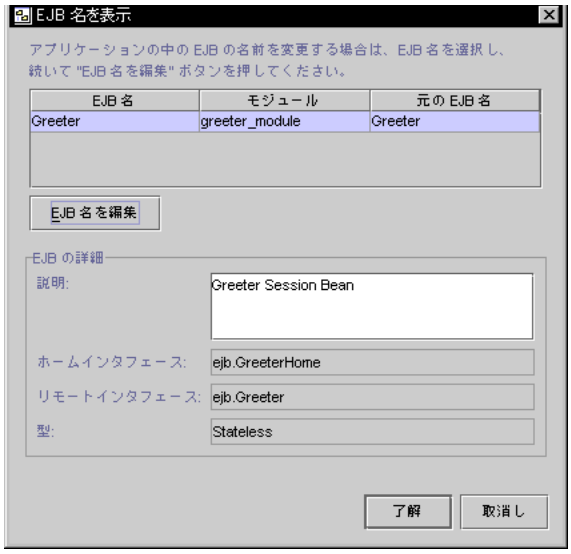


図 3-5　EJB 名のマッピング

非標準アプリケーションプロパティの設定

アプリケーションノードのプロパティシートを開きます (ノードを右クリックして、コンテキストメニューの「プロパティ」を選択します)。プロパティシートには、インストールされている各サーバーノードのタブがあります。図 3-6 は、J2EE Reference Implementation サーバーのタブを示しています。

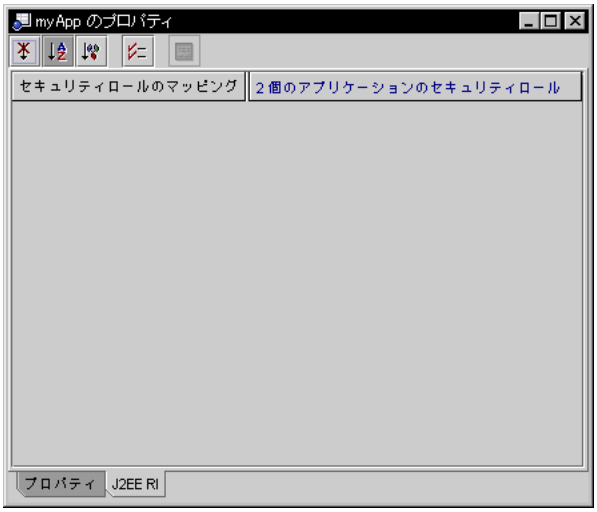


図 3-6　サーバー固有のプロパティタブ

各サーバー製品には、独自の非標準プロパティがあります。サーバーに配備するには、まずこれらの非標準プロパティの値を指定する必要があります。この節では、J2EE RI サーバーの非標準プロパティについて説明しています。ほかのサーバー製品でも類似した作業を実行します。

サーバー固有のセキュリティのマッピング

配備中のアプリケーションで使用されるセキュリティロールを、サーバーの環境に定義された実際のユーザーとグループにマップできます。たとえば、アプリケーションにはシステム管理者のロールである `sysadmin` があり、配備先サーバーインスタンスのシステム管理者の実際のユーザー名にこのロールをマップするとします。この作業は、「セキュリティロールのマッピング」プロパティエディタで行います (このエディタを開くには、アプリケーションノードを右クリックして、「プロパティ」>「セキュリティロールのマッピング」>「...」ボタンを選択します)。

図 3-7 は、アプリケーションに定義されている 3 つのセキュリティロールを表示するこのプロパティエディタを示しています。これらのセキュリティロールの 1 つにユーザー名またはグループ名をマップするには、セキュリティロールを選択して、「追加」ボタンの 1 つをクリックします。開かれたダイアログを使用して、セキュリティロールにマップするユーザー名またはグループ名を指定します。

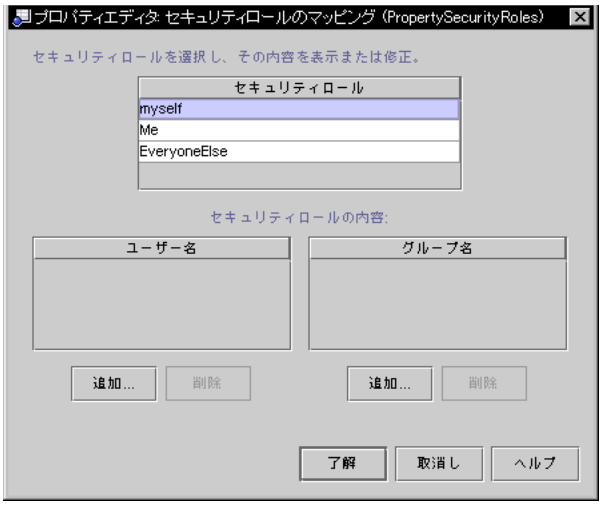


図 3-7　J2EE RI サーバーの「セキュリティロールのマッピング」プロパティエディタ

コンテナ管理による持続性

配備する J2EE アプリケーションにコンテナ管理によるエンティティ Bean が含まれる場合は、それらのエンティティ Bean に対していくつかの SQL が生成されています。この SQL は、エンティティ Bean がデータベースと対話するときに使用されます。エンティティ Bean は通常、特定のデータベーススキーマによって作成され、SQL はこのスキーマに対して生成されます。SQL の表の名前、列の名前などはすべて、元のスキーマと同じものが使用されます。これは通常、コンポーネントプロバイダの役割です。

元のデータベースまたはその正確な複製にアクセスできるアプリケーションサーバーに対してアプリケーションを配備する場合、SQL は問題なく実行されます。アプリケーションを配備、実行してテストする場合は、通常問題ありません。ただし、エンティティ Bean が別の組織によって開発されていて、元のスキーマに正確に一致しないデータベースに対してこれを実行する場合は、生成された SQL を編集する必要があります。

生成された SQL と関連プロパティを表示するには、コンテナ管理によるエンティティ Bean を右クリックして、「プロパティ」>「J2EE RI」タブを選択します。図 3-8 は、このプロパティシートの例を示しています。

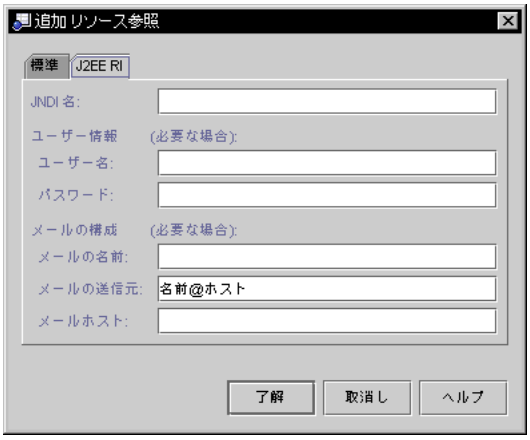
図 3-8　コンテナ管理によるエンティティ Bean のサーバー固有の SQL

コンテナ管理によるエンティティ Bean とこれらのプロパティの編集については、『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。

リソースファクトリ参照

リソースファクトリ参照を使用すると、アプリケーションで、JNDI ルックアップを持つデータベースなどのリソースを検索できます。データベースを検索するために、アプリケーションは、特定データベース向けに構成された `javax.sql.DataSource` のインスタンスへの参照を調べます。データソースインスタンスから、アプリケーションは特定データベースへの接続を取得します。

第 2 章 では、Web コンポーネントまたは Enterprise Bean コンポーネントに JNDI ルックアップおよび配備記述子を設定する方法を説明しました (詳細については、37 ページの「リソース参照の設定」または 46 ページの「リソースファクトリ参照の設定」を参照してください)。「J2EE RI」非標準プロパティタブを使用すると、アプリケーションがルックアップによって取得するリソースを構成できます。このタブに定義するリソースは、アプリケーションサーバーによって配備されます。



サーバーインスタンスの選択

アプリケーションサーバープロパティのデフォルト値は「デフォルトアプリケーションサーバー」であり、デフォルトアプリケーションサーバーとして選択されたサーバーも、プロパティエディタを開いて、アプリケーションに別のサーバーを選択できることを示しています。

J2EE アプリケーションの配備

Web アプリケーションを配備するには、アプリケーションのノードを右クリックして、コンテキストメニューの「配備」を選択します。アプリケーションはデフォルトの Web サーバーに配備されます。

IDE の出力ウィンドウで配備の進捗に関するメッセージを確認してください。配備が完了したら、アプリケーションを実行できます。

配備された J2EE アプリケーションの実行

ブラウザを起動して、アプリケーションの Web リソースの 1 つに対応する URL を開きます。開始ページを設定した場合は、URL は次に示す例のようになります。この例は、デフォルトポート 8000 のローカルマシンで実行される J2EE Reference Implementation 1.2 に配備した場合を想定しています。

```
http://localhost:8000/myWebContext/index.html
```

myWebContext は、アプリケーションの Web コンテキストであり、取り込まれた Web モジュールのプロパティシートに設定されています。このプロパティの詳細については、60 ページの「Web モジュールへのサーブレットコンテキストの設定」を参照してください。

サーブレットなどの特定のリソースを実行する場合、URL は次に示す例のようになります。

```
http://localhost:8000/myWebContext/servlet/myServlet
```

myServlet は、サーブレットクラスにマッピングされた名前です。サーブレットへの URI のマッピングについては、22 ページの「サーブレットの設定」を参照してください。

アーカイブファイルのエクスポート

アプリケーションのテストを終了したら、Forte for Java IDE 外部での配備に向けて、これを WAR ファイルまたは EAR ファイルにエクスポートできます。

Web モジュールまたは Web アプリケーションのエクスポート

Web アプリケーションと Web モジュールは、WAR (Web アーカイブ) ファイルとしてエクスポートされます。アプリケーションまたはモジュールをエクスポートする前に、エクスポートファイルに取り込まれる項目のリストを検討できます。この作業は、WEB-INF ノードを右クリックして「WAR の内容を表示」を選択して実行します。図 3-9 は、「WAR の内容」ダイアログの例を示しています。

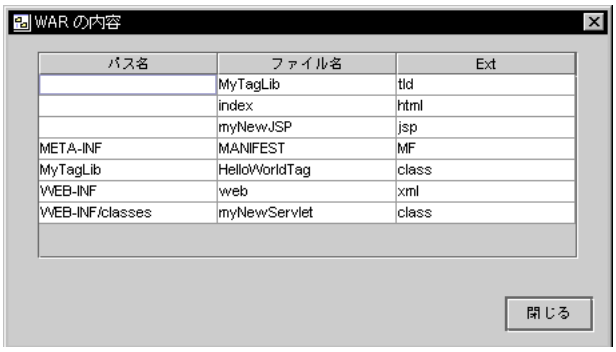


図 3-9　表示された WARの内容

Web モジュールまたはアプリケーションをエクスポートするには、`WEB-INF` ノードを右クリックして「WAR ファイルをエクスポート」を選択します。すると、WAR ファイルを生成する前にその位置を指定するためのダイアログが開きます。

EJB モジュールのエクスポート

EJB モジュールは、EJB JAR ファイルとしてエクスポートされます。モジュールをエクスポートする前に、エクスポートファイルに取り込まれる項目のリストを検討できます。この作業は、モジュールノードを右クリックして、「EJB Jar の内容を計算」を選択して行います。図 3-10 は、EJB Jar の内容を表すダイアログの例を示しています。

図 3-10 EJB JAR ファイルの内容

EJB モジュールには、「追加のファイル」および「除外されるライブラリ Jar ファイル」という 2 つのプロパティがあり、EJB JAR ファイルの内容を管理するために使用できます。これは通常、コンポーネントプロバイダによって行われます。

モジュールをエクスポートするには、モジュールノードを右クリックして「EJB Jar ファイルをエクスポート」を選択します。すると、EJB JAR ファイルを生成する前にその位置を指定するためのダイアログが開きます。

J2EE アプリケーションのエクスポート

J2EE アプリケーションは、EAR (エンタープライズアーカイブ) ファイルとしてエクスポートされます。J2EE アプリケーションをエクスポートするプロセスで、アプリケーションのモジュール向けの WAR ファイルまたは EJB JAR ファイルが生成されます。J2EE アプリケーションをエクスポートする前に、前の節の手順を使用して、モジュールレベルの WAR ファイルおよび EJB JAR ファイルの内容を検討できます。

J2EE アプリケーションをエクスポートするには、アプリケーションノードを右クリックして「アプリケーション EAR ファイルをエクスポート」を選択します。すると、EAR ファイルを生成する前にその位置を指定するためのダイアログが開きます。

付録 A

Forte for Java による J2EE モジュールおよびアプリケーション配備のサポート

前の章では、J2EE モジュールとアプリケーションの配備について簡単に説明しました。この章では、Forte for Java IDE 内から J2EE モジュールまたはアプリケーションを配備して実行する機構について詳しく説明します。

反復開発のサポート

IDE の開発機能は、エンタープライズアプリケーションの反復開発をサポートしています。適切な Web サーバーまたはアプリケーションサーバーがインストールされていることを前提として、各担当者または各チームは、Web モジュールや J2EE アプリケーションの開発とアセンブル、その配備、テスト目的での実行、ソースコードやコンポーネントプロパティの修正、再配備と再テストなどを行うことができます。再アセンブルは、テストでアセンブルの問題が生じた場合以外は必要ありません。

実際の配備では、Forte for Java 配備機能は、サーバーに用意されている配備ツールの代替手段にはなりません。開発のこの段階に達したら、WAR または EAR ファイルとしてアプリケーションをエクスポートして、サーバーが提供するツールによってこれを配備します。

この章では、Forte for Java IDE と Web サーバーまたはアプリケーションサーバーの対話について詳しく説明します。ここでは、IDE の配備コマンドを使用するときに何が起こるかについて説明します。配備機能の動作を理解すると、この機能を効果的に使用できます。配備機能の実際の使用手順は、第 3 章に説明されています。

サーバープラグインの概念

配備とは、配備可能な形式のモジュールまたはアプリケーションを J2EE 実行環境に配信することをいいます。実行環境は、Web サーバーまたはアプリケーションサーバーの形式を使用します。特定のサーバーに配備するために、IDE は有効なコマンドをサーバーの配備ツールに対して実行する必要があります。これに加えて、ほとんどのサーバーには、J2EE 標準配備記述子だけでなく非標準プロパティが必要であり、IDE はこれらのプロパティを提供できる必要があります。

IDE が各種の Web サーバーおよびアプリケーションサーバーに配備できるようにするために、サーバープラグインという概念が開発されました。プラグインとは、IDE と特定のサーバー製品間の対話を管理する IDE モジュールをいいます。アプリケーションを配備する場合、その配備先のサーバーを選択します。IDE は適切なプラグインを使用して、各配備コマンドを処理します。これにより、サーバーの配備ツールに適したコマンドを生成して、サーバーに渡すファイルに適切な非標準プロパティファイルを取り込むことができます。図 A-1 は、この手順を示しています。

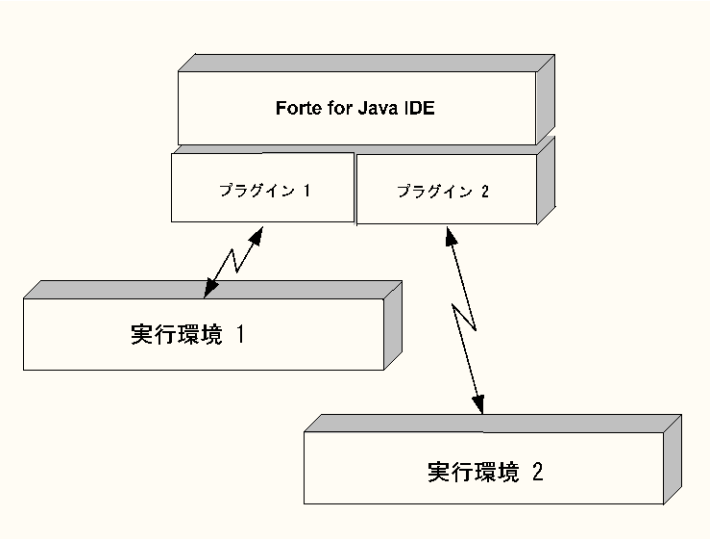


図 A-1　IDE が J2EE 実行環境と通信できるようにするサーバープラグイン

アプリケーションを配備するアプリケーション開発者に対して、プラグインは次の機能を提供します。

- IDE のエクスプローラウィンドウにおけるプラグインの表示。各プラグインは、サーバー製品ノードで表されます。開発者は、サーバー製品のインストールディレクトリによって、サーバー製品ノードを構成します。サーバー製品ノードの表示と使用法については、68 ページの「サーバー製品ノード」を参照してください。
- サーバー製品ノードのサブノードとしての、実行サーバーインスタンスの表示。開発者は、配備ターゲットとして、エクスプローラウィンドウに表示された任意のサーバーインスタンスを選択できます。サーバーインスタンスノードの表示と使用法については、69 ページの「サーバーインスタンスノード」を参照してください。
- コンポーネント、モジュール、およびプロパティシートのサーバー固有のタブ。これらのタブには、サーバー製品に必要な非標準プロパティが表示され、サーバー製品に必要な値の入力が開発者に要求されます。サーバー固有のプロパティタブの詳細については、72 ページの「非標準プロパティの設定」を参照してください。
- 選択したサーバーに合わせて配備コマンドを処理する機構。この処理の詳細については、次の節で説明します。

プラグインを使用する配備プロセス

1. IDE をインストールする場合は、使用する Web サーバーまたはアプリケーションサーバーと適切なプラグインをインストールします (一部のサーバーとプラグインはデフォルトでインストールされています。詳細については、『Forte for Java, Enterprise Edition インストールガイド』を参照してください)。
2. アプリケーションのビジネスロジックに合わせて J2EE コンポーネントを開発します。
3. コンポーネントをモジュールに、続けてアプリケーションにアセンブルします。プロパティシートを使用して、J2EE 標準配備記述子要素とサーバーに必要な非標準要素を指定します。
4. アプリケーションがアセンブルされたら、ターゲットサーバーインスタンスを特定します。

5. IDE の配備コマンドを使用して、配備プロセスを開始します。

   第 3 章 には、配備に向けてアプリケーションを準備して、配備コマンドを発行する手順が示されています。この章では、主にコマンド発行後に何が起こるかについて説明します。

6. IDE は、アプリケーションの WAR ファイルまたは EAR ファイルを作成するために必要なすべてのファイルを特定します。

   これには、配備記述子で特定された J2EE コンポーネントと、これらのファイルで使用される Java クラスまたは静的リソースが含まれます。IDE は、コンポーネントのファイル依存関係をすべて特定します。

7. IDE は、アプリケーションの配備先であるサーバー製品を特定します。

8. プラグインは、WAR ファイルまたは EAR ファイル用のファイルの妥当性検査を行います。

9. IDE は、WAR ファイルまたは EAR ファイルをアプリケーション向けに生成します。これには、J2EE 配備記述子、サーバー固有のタブを持つ個別ファイル、およびリモートメソッドの呼び出しに必要なスタブまたはスケルトンクラスが含まれます。

10. プラグインは WAR ファイルまたは EAR ファイルをサーバーに渡します。

    サーバー製品によって、プラグインは同じアプリケーションの初期の配備を自動的にクリーンアップするか、またはサーバーインスタンスにすでに配備されているアプリケーションとの衝突を解釈処理しようとします。

11. サーバーが引き継いで、配備記述子とサーバー固有の配備ファイルを読み取り、独自の標準に従って WAR ファイルまたは EAR ファイルを配備します。

この後、開発者は Web ブラウザを起動して、サーバーで実行される配備されたアプリケーションへの HTTP 接続を開くことができます。開発者が Web アプリケーションを実行する場合、IDE は Web ブラウザを自動的に起動して、アプリケーションの開始ページを開きます。

Web モジュールおよび J2EE アプリケーション以外のコンポーネントの配備

Web モジュールと J2EE アプリケーションだけが、サーバーに実際に配備して実行できる項目です。ただし、開発中のビジネスロジックの小さなユニットのテストが必要な場合があります。Forte for Java IDE では、より小さいビジネスロジックのユニットを含むモジュールとアプリケーションを作成することによって、これらのコンポーネントを配備して実行することができます。また、いくつかのタイプのコンポーネントに対して、テストクライアントを生成することもできます。これらの機能については、『Web コンポーネントのプログラミング』および『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。

索引

あ

い

え

か

く

こ

さ