BEA WebLogic Server™

J2EE アプリケーションのパッケージ化とデプロイ

著作権

限定的権利条項

商標または登録商標

J2EE アプリケーションのパッケージ化とデプロイ

パート番号	マニュアルの日付	ソフトウェアのバージョン
なし	2001 年 11 月 1 日	BEA WebLogic Server バージョン 6.1 SP1

目次

1 J2EE アプリケーションのパッケージ化とデプロイ

J2EE アプリケーションをデプロイした経験のある方は、アプリケーション サーバによってそのプロセスが異なることを理解しているでしょう。WebLogic Server™ では、そのプロセスはツールによって自動化されます。

このチュートリアルでは、チュートリアルの Web サイトからダウンロードできる銀行業務アプリケーションのサンプルを使用して、組み込みツールを使ってデプロイメント記述子を生成する方法、J2EE アプリケーションのサンプルをパッケージ化してサーバにデプロイする方法について説明します。このチュートリアルでは、そのプロセスをユーザのアプリケーションに応用できるように解説しています。

デプロイする手順は以下のとおりです。

- 手順 1 : アプリケーション ディレクトリの設定
- 手順 2 : 記述子の生成
- 手順 3 : Web 記述子の編集
- 手順 4 : EJB 記述子の編集
- 手順 5 : application.xml の編集
- 手順 6 : 接続プールの設定
- 手順 7 : 展開されたアプリケーションのデプロイ
- 手順 8 : アプリケーションのパッケージ化

展開形式またはアーカイブされた形式の選択

展開形式またはアーカイブされた形式のいずれでも、WebLogic Server にアプリケーションをデプロイできます。展開形式は完全なディレクトリ構造を持ちます。これに対し、アーカイブされた形式には、J2EE 仕様で定義された WAR、EJB JAR、および EAR ファイルなどの標準的な J2EE パッケージが含まれます。

以下のような場合は、完全なディレクトリ構造を持つ展開形式でアプリケーションをデプロイします。

- アプリケーションのテスト中である場合。
- サーバ上でアプリケーションの実行中に、静的ファイル（JSP または HTML ファイル）を変更する可能性がある場合。
- WebLogic Server をアプリケーション サーバとして使用し、他のサーバ（Apache HTTP サーバなど）を Web サーバとして使用する場合。
- エンタープライズ Bean を変更および再コンパイルして、ホット デプロイ機能を使用して再デプロイする場合。

アーカイブされた形式は、アプリケーションをテストまたは変更する必要がない場合に適しています。標準的な J2EE アーカイブでアプリケーションをデプロイすると、以下のような利点があります。

- すべての J2EE エンタープライズ アプリケーション コンポーネント（WAR、EJB JAR など）は、1 つのアプリケーションの一部とみなされます
- RMI 呼び出しが高速になり、EJB のローカル インタフェースが使用されるため、パフォーマンスが向上します。
- アーカイブされたアプリケーションは、それぞれ独自のクラス ローダに隔離されます（クラス ローダ の詳細については、「手順 8 : アプリケーションのパッケージ化」を参照してください）。

アプリケーションを展開形式またはアーカイブされた形式でデプロイする場合、またはテスト用に展開形式を使用してからプロダクション用にアーカイブされた形式を使用する場合のいずれでも、最初の手順はデプロイメント ディレクトリを設定することです。

手順 1 : アプリケーション ディレクトリの設定

アプリケーション ディレクトリは、WebLogic Server ディレクトリの外部に、独自のディレクトリとして配置しておきます。そのディレクトリで、プロダクション サーバにデプロイする前に、開発、構築、およびテストを行います。アプリケーション ディレクトリには、開発領域とデプロイメント領域の両方が必要です。

図 1-1 のように、アプリケーション ディレクトリを設定します。ここでは、標準的な J2EE アプリケーションのディレクトリ構造を使用します。

図 1-1 アプリケーション ディレクトリ

```
banking\                    .. またはユーザのアプリケーション ディレクトリ名を
使用する ..
  META-INF\
  application.xml
  dev\
    web\                   ..jsp および html ファイル、サーブレット、画像、およ
び関連ファイル ..
    ejb\                   .. エンタープライズ Bean ソース ファイル ..
  deploy\
    web\                   ..jsp、html、および画像 ファイル ..
      WEB-INF\
        web.xml
        weblogic.xml
        classes\           .. コンパイル済みサーブレット ..
        lib\               .. サード パーティ ライブラリ、タグ ライブラリ ..
    ejb\                   .. コンパイル済み Bean ..
      META-INF\
        ejb-jar.xml
        weblogic-ejb-jar.xml
        weblogic-cmp-rdbms-jar.xml
    ear\
```

この構造は以下について示します。

- 展開形式またはアーカイブされた形式のどちらでデプロイする場合も、同じディレクトリ構造とデプロイメント記述子を使用します（「手順 8 : アプリケーションのパッケージ化」で説明するように、EAR ファイルを作成するときに META-INF ディレクトリを移動します）。

- アプリケーションを開発し、そのソース ファイルを WebLogic Server ディレクトリの外部のディレクトリに格納します。
- Web アプリケーション用とエンタープライズ Bean モジュール用に別々のディレクトリを使用します。これにより、ツールを使用した記述子の生成が簡単になり、アーカイブ ファイル作成用のディレクトリ構造も準備されます。
- アプリケーションを構築し、コンパイル済みファイルをアプリケーション ディレクトリに格納できます。同じコンピュータに格納されたサーバ上でテストする場合は、格納したファイルを、そのディレクトリからデプロイします。

ベスト プラクティス : WAR、EJB JAR、および EAR パッケージを別々のステージング ディレクトリで構築します。これにより、一部のクラスを更新して再デプロイする必要がある場合に、パッケージごとに処理することができます。

デプロイメント記述子

必要なデプロイメント記述子には、J2EE 記述子と WebLogic Server に固有の記述子があります。これらの記述子は、J2EE アプリケーションの最も一般的な部分に必要です。

コンポーネント	記述子	タイプ
Web アプリケーション (または WAR ファイル)	`web.xml` `weblogic.xml`	J2EE WebLogic

EJB コンポーネント (または EJB JAR ファイル)	`ejb-jar.xml`
	`weblogic-ejb-jar.xml`
	`weblogic-cmp-rdbms-jar.xml`
		J2EE
		WebLogic
		WebLogic

J2EE アプリケーション (EAR ファイル)	`application.xml`	J2EE

アプリケーションにリソース アダプタがある場合、またはスタンドアロンの Java クライアント アプリケーションを使用する場合は、追加の記述子が必要です。このチュートリアルでは説明しませんが、リソース アダプタ用の記述子は ra.xml および weblogic-ra.xml、クライアント アプリケーション用の記述子は application-client.xml と client-application-runtime.xml です。詳細については、こちらを参照してください。

アプリケーションのコンパイル

ディレクトリを設定したら、エンタープライズ Bean とサーブレットを正しい場所にコンパイルする必要があります。エンタープライズ Bean クラスは、それらのヘルパー クラス、リモート スタブ クラス、およびスケルトン クラスとともに、deploy\ejb に生成されます。このことはステージング ディレクトリを設定する際に重要です。後で EJB JAR ファイルを作成するときに、エンタープライズ Bean とヘルパー クラスだけをコンパイルし、ejbc ユーティリティを使用して、.RMI スタブおよびスケルトン クラスを追加します。

Web アプリケーションでは、以下のことが必要です。

- サーブレットは *myapp*\deploy\web\WEB-INF\classes にコンパイルします（サーブレットをこのディレクトリに単純に移動しないでください）。
- JSP ファイル、HTML ファイル、および画像を deploy\web の最上位ディレクトリに移動します。

次に、エンタープライズ Bean を *myapp*\deploy\ejb にコンパイルします。開発用ディレクトリに構築スクリプトを格納して、デプロイメント ディレクトリにクラスをコンパイルすると、構築スクリプトで一度に多数のクラスを簡単にコンパイルできます。

コード リスト 1-1　Windows 上の構築スクリプトによるコンパイル

```
@REM EJB、サーブレット、RMI クラスをコンパイル

javac -d ..\deploy\ejb ejb\Account.java ejb\AccountBean.java
  ejb\AccountHome.java ejb\RMILogger.java ejb\RMILoggerImpl.java
 ejb\BankConstants.java ejb\ProcessingErrorException.java
  ejb\Client.java

javac -d ..\deploy\web\WEB-INF\classes web\BankAppServlet.java
```

```
java weblogic.rmic -d ..\deploy\ejb
  examples.tutorials.migration.banking.RMILoggerImpl
```

javac の -d オプションには、コンパイル済みクラスの格納場所を指定します。

クラスがコンパイルされる順序を確認します。エンタープライズ Bean クラスは、サーブレット クラスよりも先にコンパイルされます。サーブレットのコンパイルにはエンタープライズ Bean クラスが必要です。したがって、コンパイル中に BankAppServlet がコンパイル済みエンタープライズ Bean を見つけられるように、その場所を CLASSPATH に追加する必要があります。

最初に setEnv スクリプトを実行して、WebLogic Server 用アプリケーションをコンパイルするための標準の CLASSPATH を設定します。

```
cd WL_HOME\config\mydomain
setEnv
```

次に、サーブレットのソース ファイル（banking\dev\web）から、サーブレットが必要とするコンパイル済みエンタープライズ Bean クラス（banking\ejb\deploy）への相対パスを追加します。

```
set CLASSPATH=..\deploy\ejb;%CLASSPATH%
```

CLASSPATH を調整したら、構築スクリプトを実行して、アプリケーション クラスをデプロイメント ディレクトリにコンパイルできます。

実践編：アプリケーション ディレクトリの設定

手順 2：記述子の生成

次の手順は、デプロイメント記述子の生成です。J2EE 記述子の記述方法は理解していても、WebLogic 記述子の記述方法は知らないかもしれません。

DDInit ツールでは、アプリケーション コンポーネントのすべての記述子（J2EE 記述子 および WebLogic 記述子）を生成します。たとえば、Web アプリケーション コンポーネントの場合、このツールで web.xml および weblogic.xml が生成されます。したがって、J2EE 記述子の記述方法をよく理解している場合でも、すべての記述子を生成するか記述する必要があります。

記述子を生成するツールは以下のとおりです。

- war.DDInit。web.xml および weblogic.xml デプロイメント記述子用。
- ejb.DDInit。EJB 1.1 Bean の ejb-jar.xml、weblogic-ejb-jar.xml、および weblogic-cmp-rdbms-jar.xml 用。
- ejb20.DDInit。EJB 2.0 Bean の ejb-jar.xml、weblogic-ejb-jar.xml、および weblogic-cmp-rdbms-jar.xml 用。

このリリース（WebLogic Server 6.1SP1）では、J2EE アプリケーションの最上位の記述子である application.xml は生成できません。このため、以下の 2 つの選択肢があります。

- 提供される application.xml 記述子をカスタマイズします（「Samples and Tutorials」から banking.zip をダウンロードします）。
- 自分で application.xml を記述します（詳細については、こちらを参照してください）。

DDInit ツールを使用するには、コマンド ウィンドウを開き、デプロイメント ディレクトリより 1 レベル上のディレクトリに移動します（この例では、ejb および web の上の banking\deploy です）。次に、以下のいずれかのコマンドを入力します。

```
java weblogic.ant.taskdefs.war.DDInit directoryName
java weblogic.ant.taskdefs.ejb.DDInit directoryName
```

生成された記述子は、適切な WEB-INF および META-INF ディレクトリに格納されます。

記述子を再生成する場合

DDInit ツールでは、既存の記述子を常に上書きします。アプリケーションで、エンタープライズ Bean、JSP ファイル、またはサーブレットを追加または変更する場合は、記述子を再生成および再編集する必要があります。

それ以外では、アプリケーションで使用するデータベースのテーブルまたはカラムを変更する場合に、既存の記述子を編集する必要があります。これは、そのテーブルまたはカラムを使用するすべてのアプリケーションに影響します。HTTP セッションや JSP コンパイルのパラメータなど、サーバによるアプリケーションの実行方法に影響する値を変更する場合にも、記述子を編集する必要があります。

実践編 : 記述子の生成

手順 3 : Web 記述子の編集

記述子を生成したら、検証機能付き XML エディタ（たとえば、BEA XML エディタ）で編集する必要があります。最初に編集する記述子は、`web\WEB-INF` ディレクトリの **`web.xml`** および **`weblogic.xml`** です。

XML エディタを使用する利点は、ファイル内の特 定の場所で有効な XML 要素がわかり、ファイルを検証すると、不適切な要素や構文の場所が示されることです。これにより、アプリケーションをデプロイする前に、記述子が適切かどうかを確認できます。

図 1-2　BEA XML エディタでの有効な XML 要素の挿入

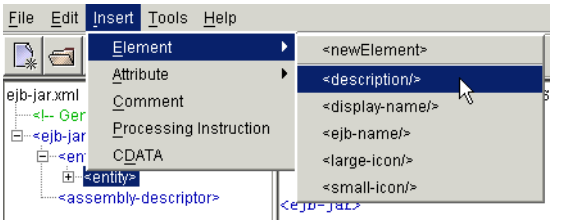
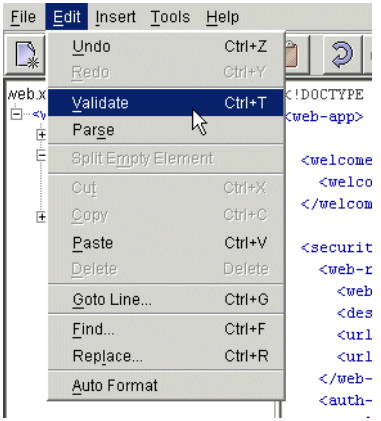


図 1-3　XML ファイルの検証



ベスト プラクティス：XML を解析および検証するツールでデプロイメント記述子を編集します。これにより、XML 構文エラーを早期に捕捉して、アプリケーションを正常にデプロイすることができます。

web.xml の編集

ここでは、生成された web.xml および weblogic.xml を確認して編集します。web.xml スキーマは Java サーブレット仕様 2.3 および BEA のマニュアルで定義されています。

コード リスト 1-2 は、生成された web.xml を示します。

コード リスト 1-2　生成された web.xml

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.2//EN" "http://java.sun.com/j2ee/dtds/web-
app_2.2.dtd">

<web-app>
  <servlet>
    <servlet-name>BankAppServlet</servlet-name>
    <servlet-class>examples.tutorials.migration.banking.
    BankAppServlet
    </servlet-class>
  </servlet>

  <servlet>
    <servlet-name>error</servlet-name>
    <jsp-file>error.jsp</jsp-file>
  </servlet>

  <servlet>
    <servlet-name>AccountDetail</servlet-name>
    <jsp-file>AccountDetail.jsp</jsp-file>
  </servlet>

  <servlet-mapping>
    <servlet-name>BankAppServlet</servlet-name>
    <url-pattern>/BankAppServlet</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>error</servlet-name>
    <url-pattern>/error</url-pattern>
  </servlet-mapping>
```

```
  <servlet-mapping>
    <servlet-name>AccountDetail</servlet-name>
    <url-pattern>/AccountDetail</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>login.html</welcome-file>
  </welcome-file-list>

  <security-constraint>
    <display-name></display-name>
    <web-resource-collection>
      <web-resource-name>My secure resources</web-resource-name>
      <description>Resources to be placed under security
          control.</description>
      <url-pattern>/private/*.jsp</url-pattern>
      <url-pattern>/private/*.html</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>guest</role-name>
      </auth-constraint>
    </security-constraint>

  <security-role>
    <description>The role allowed to access our
      content</description>
  <role-name>guest</role-name>
  </security-role>
</web-app>
```

web.xml では、以下の要素に注意します。

- <servlet>
- <servlet-mapping>
- <display-name>

<servlet> および <servlet-mapping> 要素では、ユーザがその URL で直接サーブレットにアクセスできるように、サーブレット クラスを URL にマップします。この動作でよい場合は、<servlet> と <servlet-mapping> の組み合わせは、サーブレットを示すものにしておきます。この動作にしない場合は、組み合わせを削除します。

<display-name> 要素は、ツールでの Web アプリケーション名を示しますが、通常は値がありません。DOCTYPE ヘッダで、Web アプリケーション バージョン 2.2 DTD を使用することが示されている場合、<display-name> 要素があるとファイルが検証されなくなります。その場合は、<display-name> を削除します。

コード リスト 1-3　サンプル銀行業務アプリケーション用の編集済み web.xml

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.2//EN" "http://java.sun.com/j2ee/dtds/web-
app_2.2.dtd">

<web-app>
  <welcome-file-list>
    <welcome-file>login.html</welcome-file>
  </welcome-file-list>

  <security-constraint>
    <web-resource-collection>
      <web-resource-name>My secure resources</web-resource-name>
      <description>Resources to be placed under security control.
      </description>
      <url-pattern>/private/*.jsp</url-pattern>
      <url-pattern>/private/*.html</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>guest</role-name>
    </auth-constraint>
  </security-constraint>

<security-role>
    <description>The role allowed to access our content
    </description>
    <role-name>guest</role-name>
</security-role>

</web-app>
```

実践編 : web.xml の編集

weblogic.xml の編集

weblogic.xml 記述子では、アプリケーションで、サーバが HTTP セッションと JSP コンパイルを処理する方法についての情報を示すパラメータ（名前と値の組み合わせ）を指定します（詳細については、こちらを参照してください）。

このリリースでは、生成された weblogic.xml の DOCTYPE 文を、次の文で置き換える必要があります。

```
<!DOCTYPE weblogic-web-app PUBLIC "-//BEA Systems, Inc.//DTD Web
Application 6.1//EN" "http://www.bea.com/servers/wls610/dtd/
weblogic-web-jar.dtd">
```

この DOCTYPE 文では、XML エディタが記述子を検証できるように、正しいバージョンの DTD を使用しています。アプリケーションに合わせて値を編集する必要がなければ、記述子の他の部分はそのままにしておきます。

banking.zip で配布された weblogic.xml のバージョンを使用して、アプリケーションに合わせて編集することもできます。

コード リスト 1-4 適切な weblogic.xml

```
<!DOCTYPE weblogic-web-app PUBLIC "-//BEA Systems, Inc.//DTD Web
Application 6.1//EN" "http://www.bea.com/servers/wls610/dtd/
weblogic-web-jar.dtd">

<weblogic-web-app>
  <session-descriptor>
    <session-param>
      <param-name>URLRewritingEnabled</param-name>
      <param-value>true</param-value>
    </session-param>
    <session-param>
      <param-name>InvalidationIntervalSecs</param-name>
      <param-value>60</param-value>
    </session-param>
    <session-param>
      <param-name>PersistentStoreType</param-name>
      <param-value>memory</param-value>
    </session-param>
    <session-param>
      <param-name>TimeoutSecs</param-name>
      <param-value>3600</param-value>
    </session-param>
  </session-descriptor>

  <jsp-descriptor>
    <jsp-param>
      <param-name>compileCommand</param-name>
      <param-value>javac</param-value>
    </jsp-param>
    <jsp-param>
      <param-name>precompile</param-name>
      <param-value>false</param-value>
    </jsp-param>
    <jsp-param>
      <param-name>workingDir</param-name>
      <param-value>C:\TEMP\</param-value>
    </jsp-param>
    <jsp-param>
```

```
      <param-name>keepgenerated</param-name>
      <param-value>true</param-value>
    </jsp-param>
    <jsp-param>
      <param-name>pageCheckSeconds</param-name>
      <param-value>5</param-value>
    </jsp-param>
  </jsp-descriptor>
</weblogic-web-app>
```

手順 4 : EJB 記述子の編集

生成された EJB 記述子は ejb\META-INF ディレクトリに格納されます。ejb-jar.xml を確認しますが、変更はそれほど必要ありません。**ただし、weblogic-ejb-jar.xml と weblogic-cmp-rdbms-jar.xml は、必ず編集する必要があります。**通常、これらのファイルではデータベース スキーマまたはエンタープライズ Bean に固有の値が必要です。

ejb-jar.xml の編集

ejb-jar.xml （**エンタープライズ JavaBeans 仕様**で定義される）で確認する要素は、<ejb-name> と <assembly-descriptor> です。

例として、コード リスト 1-5 に、サンプル銀行業務アプリケーション用の生成された ejb-jar.xml を示します。

コード リスト 1-5　生成された ejb-jar.xml

```
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 1.1//EN' 'http://java.sun.com/j2ee/dtds/ejb-
jar_1_1.dtd'>


<!-- Generated XML! -->
<ejb-jar>
  <enterprise-beans>
    <entity>
      <ejb-name>AccountBean</ejb-name>
```

```
      <home>examples.tutorials.migration.banking.AccountHome
      </home>
      <remote>examples.tutorials.migration.banking.Account
      </remote>
      <ejb-class>examples.tutorials.migration.banking.AccountBean
      </ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>java.lang.String</prim-key-class>
      <reentrant>False</reentrant>
      <cmp-field>
          <field-name>accountId</field-name>
      </cmp-field>
      <cmp-field>
          <field-name>balance</field-name>
      </cmp-field>
      <cmp-field>
          <field-name>accountType</field-name>
      </cmp-field>
          <primkey-field>accountId</primkey-field>
    </entity>
  </enterprise-beans>

<assembly-descriptor>
</assembly-descriptor>

</ejb-jar>
```

<ejb-name> の値は、アプリケーションの各エンタープライズ Bean の名前です。<ejb-name> の値は、デプロイメント記述子の内部でのみ使用されますが、各記述子で一致している必要があります。生成された <ejb-name> をそのまま使用することをお勧めします。変更する場合は、weblogic-ejb-jar.xml および weblogic-cmp-rdbms-jar.xml でも、同様に変更する必要があります。

また、<assembly-descriptor> 要素が空である点にも注意してください。つまり、この要素のデフォルト値が使用されます。空の <assembly-descriptor> 要素には、以下の意味が含まれます。

```
<assembly-descriptor>
  <container-transaction>
    <method>
      <description>container managed</description>
      <ejb-name>AccountBean</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>*</method-name>
    </method>
  <trans-attribute>Required</trans-attribute>
  </container-transaction>
</assembly-descriptor>
```

つまり、<assembly-descriptor> のデフォルト値では、この Bean に含まれるすべてのメソッドのトランザクション属性 Required が指定されています。<ejb-jar.xml> の <ejb-name> を変更する場合、あるいは、<trans-attribute> または <method-name> に異なる値を指定する場合は、<assembly-descriptor> の値のみ編集する必要があります。

たとえば、withdraw メソッドに異なるトランザクション属性を指定する場合、次のように <assembly-descriptor> を記述できます。

```
<assembly-descriptor>
  <container-transaction>
    <method>
      <description>container managed</description>
      <ejb-name>AccountBean</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>withdraw</method-name>
    </method>
  <trans-attribute>Mandatory</trans-attribute>
  </container-transaction>
</assembly-descriptor>
```

実践編 : ejb-jar.xml の編集

weblogic-ejb-jar.xml の編集

weblogic-ejb-jar.xml 記述子（詳細については、こちらを参照してください）では、WebLogic Server に固有のエンタープライズ Bean の動作を記述します。weblogic-ejb-jar.xml には、ユニークな WebLogic 要素が多く含まれます。

サンプル銀行業務アプリケーションでも、ユーザのアプリケーションでも、以下の要素を探す必要があります。

- <ejb-name>。ejb-jar.xml 内の値と同じ値を指定します。
- <is-modified-method-name>。この WebLogic 要素は、EJB 1.1 CMP Bean の場合にのみ追加します（EJB 2.0 CMP Bean では追加しません）。
- <type-version>。エンタープライズ Bean の適切なバージョン（5.1.0 または 6.0）を使用します。
- <jndi-name>。<ejb-name> が使用されていることを確認します。

サンプル銀行業務アプリケーションでは、ejb.DDInit ツールにより、コード リスト 1-6 に示す weblogic-ejb-jar.xml ファイルが生成されます。

コード リスト 1-6　生成された weblogic-ejb-jar.xml

```
<!DOCTYPE weblogic-ejb-jar PUBLIC '-//BEA Systems, Inc.//DTD
WebLogic 6.0.0 EJB//EN' 'http://www.bea.com/servers/wls600/dtd/
weblogic-ejb-jar.dtd'>

<!-- Generated XML! -->

<weblogic-ejb-jar>
  <weblogic-enterprise-bean>
    <ejb-name>AccountBean</ejb-name>
      <entity-descriptor>
          <persistence>
            <persistence-type>
            <type-identifier>WebLogic_CMP_RDBMS</type-identifier>
            <type-version>5.1.0</type-version>
            <type-storage>META-INF/weblogic-cmp-rdbms-jar.xml
            </type-storage>
            </persistence-type>
            <persistence-use>
            <type-identifier>WebLogic_CMP_RDBMS</type-identifier>
            <type-version>5.1.0</type-version>
            </persistence-use>
          </persistence>
      </entity-descriptor>

      <jndi-
name>examples.tutorials.migration.banking.AccountHome
      </jndi-name>
  </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

この場合、<ejb-name> 要素は正しいため、他の記述子のこの要素は変更しません。<type-version> の値は 5.1.0 または 6.0 です（この場合は、サンプル アプリケーションの Bean に 5.1.0 を適用しています）。WebLogic Server バージョン 6.1 を使用する場合でも、<type-version> には 6.1 の値を指定できません。このことは、後で EJB コンポーネントを EJB JAR ファイルにパッケージ化し、サーバにデプロイする前に ejbc を使用して EJB JAR を確認する際に、重要になります。

次に、サンプル アプリケーションでは CMP 1.1 Bean を使用するため、<persistence> 要素内に <is-modified-method-name> 要素を追加する必要があります。<is-modified-method-name> では、Bean の保存時に WebLogic Server によって呼び出される Bean のメソッドを指定します。たとえば、<is-modified-method-name> は次のようになります。

```
<persistence>
<is-modified-method-name>isModified</is-modified-method-name>
```

また、パッケージ名の代わりに <ejb-name> を使用するように、<jndi-name> の値を調整します。

```
<jndi-name>AccountBean.AccountHome</jndi-name>
```

編集した weblogic-ejb-jar.xml ファイルは コード リスト 1-7 のようになります。

コード リスト 1-7　サンプル銀行業務アプリケーションの編集済み weblogic-ejb-jar.xml

```
<!DOCTYPE weblogic-ejb-jar PUBLIC '-//BEA Systems, Inc.//DTD
WebLogic 6.0 EJB//EN' 'http://www.bea.com/servers/wls600/dtd/
weblogic-ejb-jar.dtd'>

<weblogic-ejb-jar>
<weblogic-enterprise-bean>
  <ejb-name>AccountBean</ejb-name>
  <entity-descriptor>
    <persistence>
      <is-modified-method-name>isModified
      </is-modified-method-name>
      <persistence-type>
            <type-identifier>WebLogic_CMP_RDBMS</type-identifier>
            <type-version>5.1.0</type-version>
            <type-storage>META-INF/weblogic-cmp-rdbms-jar.xml
            </type-storage>
      </persistence-type>
      <persistence-use>
          <type-identifier>WebLogic_CMP_RDBMS</type-identifier>
          <type-version>5.1.0</type-version>
      </persistence-use>
    </persistence>
  </entity-descriptor>

  <jndi-name>AccountBean.AccountHome</jndi-name>
    </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

実践編 : weblogic-ejb-jar.xml の編集

weblogic-cmp-rdbms-jar.xml の編集

weblogic-cmp-rdbms-jar.xml 記述子（詳細については、こちらを参照してください）では、エンティティ Bean によるデータベース テーブルおよびカラムへのアクセス方法を記述します。この記述子はエンティティ Bean 用にだけ生成され、セッション Bean 用には生成されません。

この記述子で確認が必要な主な要素は finder です（コード リスト 1-8 を参照）。finder 要素には、Bean で使用され、データベース クエリで更新される必要があるファインダ メソッドが指定されます。

コード リスト 1-8　生成された weblogic-cmp-rdbms-jar.xml

```
<!DOCTYPE weblogic-rdbms-jar PUBLIC  '-//BEA Systems, Inc.//DTD
WebLogic 6.0.0 EJB 1.1 RDBMS Persistence//EN' 'http://www.bea.com/
servers/wls600/dtd/weblogic-rdbms11-persistence-600.dtd'>

<!-- Generated XML! -->

<weblogic-rdbms-jar>
  <weblogic-rdbms-bean>
    <ejb-name>AccountBean</ejb-name>
    <pool-name>AccountBeanPool</pool-name>
    <table-name>AccountBeanTable</table-name>
    <field-map>
      <cmp-field>accountId</cmp-field>
      <dbms-column>accountIdColumn</dbms-column>
    </field-map>
    <field-map>
      <cmp-field>balance</cmp-field>
      <dbms-column>balanceColumn</dbms-column>
    </field-map>
    <field-map>
      <cmp-field>accountType</cmp-field>
      <dbms-column>accountTypeColumn</dbms-column>
    </field-map>
    <finder>
      <finder-name>findByPrimaryKey</finder-name>
      <finder-param>java.lang.String</finder-param>
      <finder-query><![CDATA[(= 1 1)]]></finder-query>
    </finder>
    <finder>
      <finder-name>findAccount</finder-name
      <finder-param>double</finder-param>
      <finder-query><![CDATA[(= 1 1)]]></finder-query>
    </finder>
    <finder>
      <finder-name>findBigAccounts</finder-name>
```

```
      <finder-param>double</finder-param>
      <finder-query><![CDATA[(= 1 1)]]></finder-query>
    </finder>
    <finder>
      <finder-name>findNullAccounts</finder-name>
      <finder-query><![CDATA[(= 1 1)]]></finder-query>
    </finder>
    </weblogic-rdbms-bean>
</weblogic-rdbms-jar>
```

<finder> 要素は、EJB ホーム インタフェースのファインダ メソッド シグネチャを、データを取得するデータベース クエリと関連付けます。

<finder> 要素には、マークアップとして解釈されないようにテキストのブロックをマークするための CDATA 属性があります。各 CDATA 属性には、データベースに固有の、WebLogic クエリ言語によるデータベース クエリが必要です。

ただし、<findByPrimaryKey> という名前のメソッドの <finder> 要素がある場合は、ほとんどの場合に受け入れられるデフォルトのクエリがサーバで生成されるため、この要素を削除できます。

コード リスト 1-9　編集済み weblogic-cmp-rdbms-jar.xml

```
<!DOCTYPE weblogic-rdbms-jar PUBLIC  '-//BEA Systems, Inc.//DTD
WebLogic 6.0.0 EJB 1.1 RDBMS Persistence//EN' 'http://www.bea.com/
servers/wls600/dtd/weblogic-rdbms11-persistence-600.dtd'>

<weblogic-rdbms-jar>
  <weblogic-rdbms-bean>
    <ejb-name>AccountBean</ejb-name>
    <pool-name>AccountBeanPool</pool-name>
    <table-name>AccountBeanTable</table-name>
    <field-map>
      <cmp-field>accountId</cmp-field>
      <dbms-column>accountIdColumn</dbms-column>
    </field-map>
    <field-map>
      <cmp-field>balance</cmp-field>
      <dbms-column>balanceColumn</dbms-column>
    </field-map>
    <field-map>
      <cmp-field>accountType</cmp-field>
      <dbms-column>accountTypeColumn</dbms-column>
    </field-map>
    <finder>
      <finder-name>findAccount</finder-name>
      <finder-param>double</finder-param>
      <finder-query><![CDATA[(= balance $0)]]></finder-query>
    </finder>
    <finder>
```

```
      <finder-name>findBigAccounts</finder-name>
      <finder-param>double</finder-param>
      <finder-query><![CDATA[(> balance $0)]]></finder-query>
    </finder>
    <finder>
      <finder-name>findNullAccounts</finder-name>
      <finder-query><![CDATA[(isNull accountType)]]></finder-
query>
    </finder>
  </weblogic-rdbms-bean>
</weblogic-rdbms-jar
```

実践編 : weblogic-cmp-rdbms-jar.xml の編集

手順 5 : application.xml の編集

必要な最後の記述子は、J2EE アプリケーション全般について記述する application.xml です。WebLogic Server 6.1SP1 では、application.xml を生成できません。ただし、このチュートリアルで提供されるサンプルの application.xml 記述子を使用して、アプリケーションに合わせて編集できます。編集は非常に簡単です。

application.xml の主な目的は、J2EE アプリケーションの Web モジュールと EJB モジュールの場所を指定することです。J2EE 1.2 を使用する場合、application.xml には、次のように DOCTYPE 定義が含まれている必要があります（全体を 1 行で入力）。

```
<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE
Application 1.2//EN" "http://java.sun.com/j2ee/dtds/
application_1_2.dtd">
```

J2EE 1.3 の場合は、次のようになります。

```
<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE
Application 1.2//EN" "http://java.sun.com/dtd/
application_1_3.dtd">
```

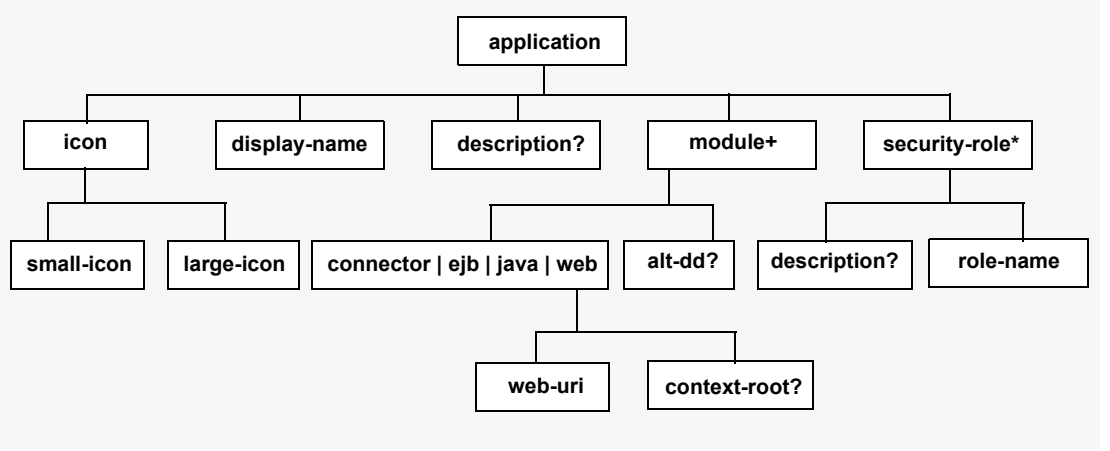
DOCTYPE に続けて、いくつかの要素が必要です。

- 他のすべての要素を含む <application> 要素

- <application> 内の <icon>、<display-name>、および <description> 要素（ツールによって使用される）
- <ejb>、<web>、<connector>、および <java> 要素が含まれる <module> 要素（アプリケーションのモジュールに対応）

要素または値を追加する場合に参考となるように、図 1-4 に application.xml のスキーマを示します。

図 1-4　application.xml の構造



この章の最初にある 図 1-1 に示すディレクトリ構造を使用すると、アプリケーションを展開形式でデプロイする場合、application.xml ファイルは コード リスト 1-10 のようになります。アプリケーションを EAR ファイルにパッケージ化する場合は、内容が異なります。

コード リスト 1-10　サンプルの application.xml 記述子

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE application PUBLIC '-//Sun Microsystems, Inc.//DTD J2EE
Application 1.2//EN' 'http://java.sun.com/j2ee/dtds/
application_1_2.dtd'>

<application>
  <display-name></display-name>
```

```
  <module>
    <ejb>\ejb</ejb>
  </module>
  <module>
    <web>
      <web-uri>\web</web-uri>
      <context-root>banking</context-root>
    </web>
  </module>
</application>
```

<context-root> の値は、アプリケーションにアクセスするために使用する URL の一部として、ホスト名とポート名の後、アクセスされるファイルまたはサーブレットの名前の前に置かれます。

```
http://localhost:7001/banking/login.html
```

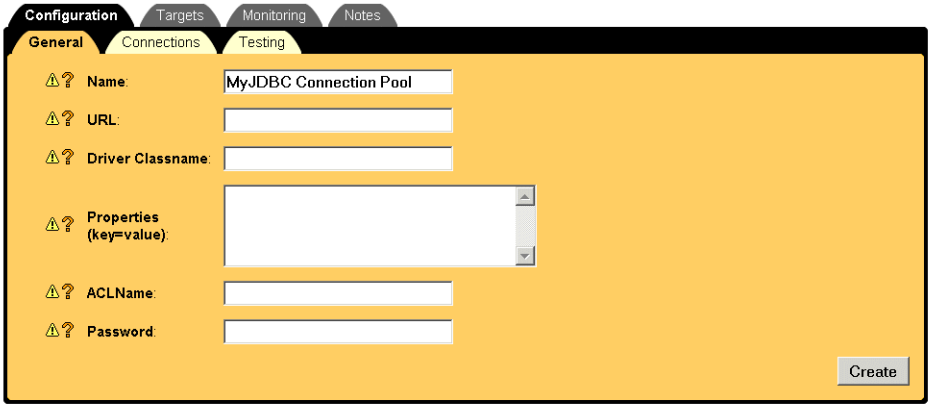
実践編 : application.xml の編集

手順 6：接続プールの設定

J2EE アプリケーションで複数の層を使用してデータベースにアクセスする場合、アプリケーションをデプロイする前に接続プールを設定する必要があります。接続プールとは、アプリケーションからデータベースに接続するためにユーザが使用する JDBC 接続のグループに名前を付けたものです。

接続プールは Administration Console を使用してグラフィカルに設定できます。Administration Console では、*WL_HOME*\config\mydomain にある config.xml ファイルに、アプリケーションのエントリを追加します。

図 1-5　Administration Console での接続プールの設定



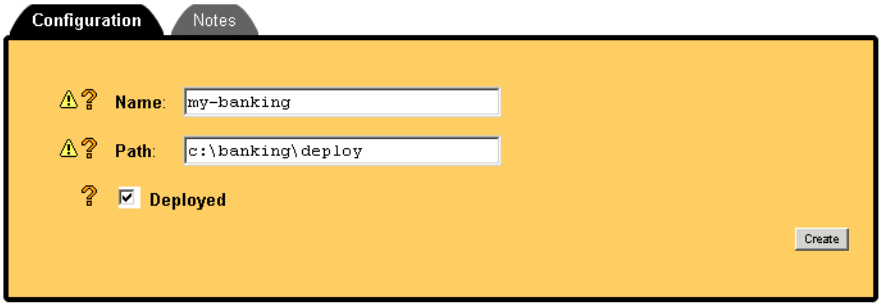
実践編：接続プールの設定

手順 7：展開されたアプリケーションのデプロイ

接続プールを設定したら、Administration Console を使用して、展開形式でアプリケーションをデプロイできます。このチュートリアルでは、1 つのサーバにデプロイする方法を説明します。

この手順では、アプリケーションは開発用ディレクトリにそのまま格納しておきます。デプロイおよびテスト後に、ソース コードを変更して再コンパイルする場合があるためです。サーバが稼動していれば、Administration Console を使用して簡単にデプロイできます。

図 1-6　Administration Console からのデプロイ

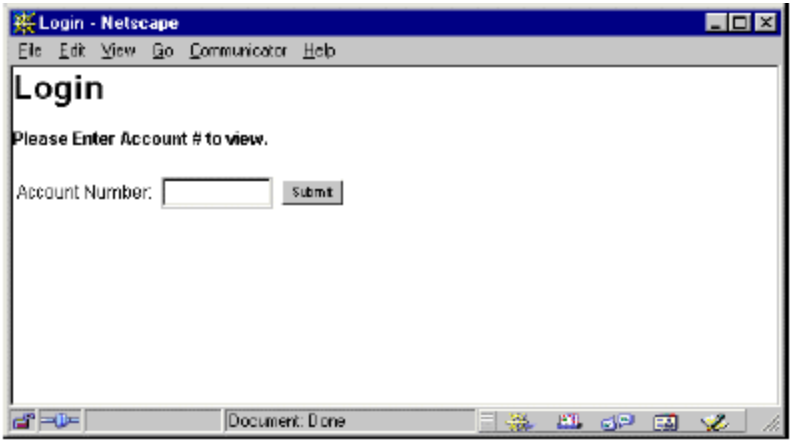


他には、以下のようなデプロイ方法があります。

- 自動デプロイメント。アプリケーションを `config\somedomain\applications` ディレクトリに格納し、開発モードでサーバを起動すると、自動的にデプロイされます。
- アプリケーションを格納するドメインの **`config.xml`** を編集します。
- **`weblogic.deploy`** ユーティリティを使用します。

正常にデプロイされたら、そのアプリケーションを実行できます。Web クライアントがある場合は、application.xml にある <context-root> の値を使用して、Web ブラウザからアプリケーションを起動します。たとえば、Web ブラウザに **http://localhost:7001/banking/login.html** と入力して、サンプル銀行業務アプリケーションを起動します。

図 1-7　銀行業務アプリケーションの起動



アプリケーションをデプロイしたら、再デプロイするかどうか、いつ再デプロイするかについて決定します。テスト中の場合は、頻繁に再デプロイする可能性があります。また、アプリケーションが誤って再デプロイされるのを防ぐこともできます。

再デプロイメントを制御するには、REDEPLOY という名前のデプロイメント ディレクトリに空のテキスト ファイルを作成します。これで、以下のように再デプロイメントを制御できます。

- たとえば、スペースを追加してファイルを変更するか、UNIX システムの場合は、ファイルに対して touch コマンドを使用すると、アプリケーションが再デプロイされます。テスト中はこの方法が便利です。
- REDEPLOY ファイルが修正されないようにパーミッションを変更すると、アプリケーションの再デプロイも防止されます。プロダクション用に展開されたアプリケーションをデプロイする場合は、この方法を使用します。

実践編：展開形式のデプロイ

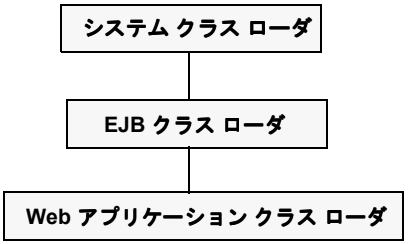
手順 8：アプリケーションのパッケージ化

アプリケーションのアーカイブ ファイルへのパッケージ化は、アプリケーションがテスト済みの最終版になり、プロダクション用に準備が整ったときに行います（実際には、プロダクション用にアーカイブされた形式でデプロイすることに決定した場合）。

アプリケーションを展開形式で正常にデプロイした場合は、アーカイブとしても正常にデプロイできます。EAR ファイルとしてパッケージ化されたアプリケーション コンポーネントをデプロイする主な理由は、WebLogic Server がアプリケーション コンポーネントを 1 つのアプリケーションとして扱えるようにすることです。

EAR ファイルをデプロイすると、サーバでは、EJB JAR ファイル用と WAR ファイル用の、2 つの新しいクラス ローダを作成します。クラス ローダには階層があります。システム クラス ローダはスーパークラスで、EJB クラス ローダはシステム クラス ローダのサブクラスです。また、Web アプリケーション クラス ローダは EJB クラス ローダのサブクラスです。

図 1-8 EAR ファイル用に作成されたクラス ローダ



つまり、WAR ファイル内のクラスは EJB JAR ファイル内のクラスを簡単に見つけることができ、JSP ファイルまたはサーブレットが EJB を呼び出すときに便利です。また、この階層では、EJB 層を再デプロイしなくても、Web アプリケーション（変更される可能性が高い）を再デプロイできます。

ベスト プラクティス：子クラス ローダ内のクラスを参照している親クラス ローダ内のクラスは、パッケージ化しないでください。子クラス ローダは親を参照できますが、親は子を参照できません。WAR クラス ローダは EJB クラス ローダの子です。つまり、WAR ファイルに含まれるファイルは、EJB JAR ファイル内のファイルにアクセスできますが、その逆はできません。

ただし、WAR ファイルと EJB JAR ファイルを個別にデプロイすると、それらは個別のアプリケーションとみなされ、2 つの個別のクラス ローダ階層によって管理されます。WAR パッケージ内のファイルは EJB JAR 内のファイルにアクセスできないため、EJB のホームおよびリモート インタフェースを WAR ファイルにパッケージ化する必要があります。

図 1-9　個別の WAR ファイルと EJB JAR ファイルで作成されたクラス ローダ

WAR および EJB JAR パッケージを構築するには、標準の JDK JAR ツールを使用します。最初に、コンポーネント ディレクトリより 1 レベル上のディレクトリ（このサンプルでは、banking\deploy）に移動します。まず、WAR ファイルをパッケージ化するには、次のコマンドを実行します。

```
jar cvf banking.war -C web .
```

これで、deploy ディレクトリに banking.war が作成されます。次に、EJB JAR ファイルをパッケージ化するには、次のコマンドを使用します。

```
jar cvf banking.jar -C ejb .
```

同様に、deploy ディレクトリに banking.jar が作成されます。

EAR ファイルをパッケージ化するには、展開形式のデプロイに使用したものとは若干異なるディレクトリ構造が必要です。J2EE アプリケーションの EAR ファイルは、WAR ファイルと EJB JAR ファイルから構築されるためです。図 1-1 に示すディレクトリ構造を使用して、ear ディレクトリを追加し、以下のように、アプリケーションの最上位レベルの META-INF ディレクトリとその下の WAR および EJB JAR ファイルを移動します。

```
banking\
  META-INF\
```

```
    application.xml
  dev\
  deploy\
  ear\
    META-INF\
      application.xml
      banking.war
      banking.jar
```

この時点で、application.xml を編集して、新しいアプリケーション構造を反映させる必要があります。XML エディタでファイルを開いて、<ejb> および <web> 要素の値を変更します。

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE application PUBLIC '-//Sun Microsystems, Inc.//DTD J2EE
Application 1.2//EN' 'http://java.sun.com/j2ee/dtds/
application_1_2.dtd'>

<application>
<display-name></display-name>
  <module>
    <ejb>banking.jar</ejb>
  </module>
  <module>
    <web>
      <web-uri>banking.war</web-uri>
      <context-root>banking</context-root>
    </web>
  </module>
</application>
```

次に、J2EE アプリケーションを生成するには、別の JAR コマンドを使用します。deploy\ear ディレクトリに移動したら、次のコマンドを使用できます。

```
jar cvf banking.ear .
```

実践編 : アプリケーションのパッケージ化

手順 9：EAR ファイルのデプロイ

EAR ファイルをプロダクション モードでサーバにデプロイできるように、アプリケーションを EAR ファイルにパッケージ化するまでに、アプリケーションをテストしてプロダクション用に準備しておく必要があります。通常は、別のディレクトリでアプリケーションを記述し、アプリケーションをプロダクション サーバにデプロイする前に、ユーザの WebLogic Server 上でデプロイメントをテストします。

アプリケーションを展開形式でデプロイできた場合は、EAR ファイルを正常にデプロイできます。ここでも、Administration Console を使用できます。

図 1-10　Administration Console からの EAR ファイルのデプロイ

上記のように EAR ファイルをデプロイするには、*WL_HOME*\config\mydomain\config.xml に以下のような行を記述します。

```
<Application Name="banking-ear" Path
  "c:\banking\deploy\banking.ear">

  <EJBComponent Name="\ejb" URI="\ejb"/>
  <WebAppComponent Name="\web" URI="\web"/>

</Application>
```

アプリケーションを config.xml で指定したら、開発モードまたはプロダクション モードのいずれの場合でも、サーバを起動するとアプリケーションが起動されます。config\mydomain\startWebLogic.cmd（UNIX の場合は startWebLogic.sh）を編集すると、起動モードを変更できます。

コード リスト 1-11　startWebLogic スクリプトの編集

```
@rem プロダクション モードの設定。true に設定すると、サーバは
@rem プロダクション モードで起動する。false に設定すると、開発モードで
@rem 起動する。設定しない場合、デフォルトで false になる
set STARTMODE=true
```

実践編 : EAR ファイルのデプロイ

A　実践編：サンプル銀行業務アプリケーションのデプロイ

以下の節では、サンプル銀行業務アプリケーションを WebLogic Server にデプロイする方法を、順を追って説明します。

前提条件

チュートリアルを使用するには、以下のものが必要です。

- WebLogic Server 6.1
  - http://www.beasys.co.jp/evaluation/index.html

  この評価版は、すべての機能が含まれる WebLogic Server Premium Edition です。インストールしている場合は、WebLogic Server Advantage Edition でも、このチュートリアルを実行できます。

  ただし、このチュートリアルでは WebLogic Express を使用することはできません。

- サンプル銀行業務アプリケーション `banking.zip`
  - http://e-docs.bea.com/wls/docs61/samples.html
- BEA XML エディタまたはそれ以外の任意の XML エディタ
  - http://developer.bea.com/tools/utilities.jsp

このチュートリアルの指示は Microsoft Windows に固有のものですが、UNIX プラットフォームにも簡単に適合させることができます。

注意

実践編の手順では、パス名は次のようになります。

`WL_HOME``\config\mydomain`

ここでは、*`WL_HOME`* を WebLogic Server ディレクトリの名前に置き換えます。

アプリケーション ディレクトリの設定

1. チュートリアルの Web サイトから `banking.zip` をダウンロードして、`c:` ドライブに解凍します。
2. `banking` ディレクトリ内に、`META-INF` および `deploy` ディレクトリを以下のように作成します。

```
banking\
  META-INF\
  dev\
    web\
    ejb\
  deploy\
    web\
      WEB-INF\
        classes\
        lib\
      ejb\
        META-INF\
```

Windows で作業している場合、`META-INF` ディレクトリは、`Meta-inf` のように大文字 / 小文字が混在して表示されます。

3. コマンド ウィンドウを開きます。
4. 構築スクリプトを実行するように、シェル環境を設定します。

```
cd c:\
cd WL_HOME\config\mydomain
setEnv
```

5. クラスパスにコンパイル済み EJB クラスの場所を追加します。

```
cd c:\
cd banking\dev
set CLASSPATH=..\deploy\ejb;%CLASSPATH%
```

6. CLASSPATH の値をチェックします。

```
echo %CLASSPATH%
```

次のように表示されます。

```
..\deploy\ejb;C:\weblogic61sp1\jdk131\lib\tools.jar;C:\weblogi
c61sp1\wlserver6.1\lib\weblogic_sp.jar;C:\weblogic61sp1\wlserve
r6.1\lib\weblogic.jar;.
```

7. 構築スクリプトを実行して、サーブレットとエンタープライズ Bean クラスを適切な場所にコンパイルします。

```
build
```

8. JSP ファイル、HTML ファイル、および画像を deploy ディレクトリにコピーします。

```
cd ..\dev
copy images\* ..\..\deploy\web
cd html
copy * ..\..\deploy\web
```

JSP および HTML ファイルは、deploy\web\html ではなく deploy\web に移動することに注意してください。

記述子の生成

1. deploy ディレクトリに移動します。

```
cd ..
cd deploy
```

2. Web アプリケーション記述子を生成します。

   ```
   java weblogic.ant.taskdefs.war.DDInit web
   ```

   これで、web.xml および weblogic.xml 記述子が生成され、web\WEB-INF に格納されます。

3. EJB 記述子を生成します。

   ```
   java weblogic.ant.taskdefs.ejb.DDInit ejb
   ```

   これで、ejb-jar.xml、weblogic-ejb-jar.xml、および weblogic-cmp-rdbms-jar.xml が生成され、ejb\META-INF に格納されます。

web.xml の編集

1. web.xml が含まれるディレクトリに移動します。

   ```
   cd web\WEB-INF
   ```

2. 生成された web.xml のバックアップを作成します。

   ```
   cp web.xml c:\tmp\web.xml
   ```

3. 任意の XML エディタを開きます。
4. XML エディタで、banking\deploy\web\WEB-INF\web.xml を開きます。
5. ファイルの最初にある <!-- Generated XML! --> という行を削除します。
6. <display-name> 要素を削除します。
7. ユーザがサーブレットに直接アクセスしないように、<servlet> および <servlet-mapping> 要素を 3 つとも削除します。
8. ファイルを保存して検証します。

weblogic.xml の編集

1. 生成された weblogic.xml のバックアップを作成します。

   ```
   cp weblogic.xml c:\tmp\weblogic.xml
   ```

2. XML エディタで、生成された weblogic.xml を開きます。
3. ファイルの最初にある <!-- Generated XML! --> という行を削除します。
4. 既存の DOCTYPE 文を次の文で置き換えます。全体を 1 行で入力します。

   ```
   <!DOCTYPE weblogic-web-app PUBLIC "-//BEA Systems, Inc.//DTD Web
   Application 6.1//EN"
   "http://www.bea.com/servers/wls610/dtd/weblogic-web-jar.dtd">
   ```

5. ファイルを保存して検証します。

weblogic-ejb-jar.xml の編集

1. 生成された weblogic-ejb-jar.xml のコピーを作成します。

   ```
   cp weblogic-ejb-jar.xml c:\tmp\weblogic-ejb-jar.xml
   ```

2. XML エディタで、生成された weblogic-ejb-jar.xml を開きます。
3. ファイルの最初にある <!-- Generated XML! --> という行を削除します。
4. <persistence> の後に、次の要素を挿入します。

   ```
   <is-modified-method-name>isModified</is-modified-method-name>
   ```

5. Java パッケージ名の代わりに ejb-name の値を使用するように、jndi-name の値を変更します。

   ```
   <jndi-name>AccountBean.AccountHome</jndi-name>
   ```

6. ファイルを保存して検証します。

weblogic-cmp-rdbms-jar.xml の編集

1. 生成された weblogic-cmp-rdbms-jar.xml のコピーを作成します。

   ```
   cp weblogic-cmp-rdbms-jar.xml c:\tmp\weblogic-cmp-rdbms-jar.xml
   ```

2. XML エディタで weblogic-cmp-rdbms-jar.xml を開きます。
3. ファイルの最初にある <!-- Generated XML! --> という行を削除します。
4. findByPrimaryKey を検索します。この文字列が含まれる <finder> 要素を削除します。
5. findAccount を検索して、<finder-query> の値を次のように変更します。

   ```
   <![CDATA[(= balance $0)]]>
   ```

6. findBigAccounts を検索して、<finder-query> の値を次のように変更します。

   ```
   <![CDATA[(> balance $0)]]>
   ```

7. findNullAccounts を検索して、<finder-query> の値を次のように変更します。

   ```
   <![CDATA[(isNull accountType)]]>
   ```

8. ファイルを保存して検証します。

application.xml の編集

1. サンプル銀行業務アプリケーションで、c:\banking\dev\application.xml を
   c:\banking\deploy\META-INF\application.xml にコピーします。
2. XML エディタで application.xml を開きます。

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <!DOCTYPE application PUBLIC '-//Sun Microsystems, Inc.//DTD
   J2EE Application 1.2//EN'
   'http://java.sun.com/j2ee/dtds/application_1_2.dtd'>
   ```

```
<application>
   <display-name></display-name>
   <module>
     <ejb>\ejb</ejb>
   </module>
   <module>
     <web>
       <web-uri>\web</web-uri>
       <context-root>banking</context-root>
     </web>
   </module>
</application>
```

3. EJB および Web モジュール ディレクトリの名前を使用するように、<ejb> および <web-uri> の値を編集します。
4. アプリケーションにアクセスするためにユーザによって URL に挿入される名前を使用するように、<context-root> の値を編集します。
5. ファイルを保存して検証します。

接続プールの設定

1. WebLogic Server を起動します。

   ```
   cd c:\
   cd WL_HOME\config\mydomain
   startWebLogic
   ```

2. Administration Console を起動するには、Web ブラウザを開いて、次の URL に移動します。

   ```
   http://localhost:7001/console
   ```

   または、サーバをインストールしたときに異なるホスト名とリスン ポートを指定した場合は、それを使用します。

3. 左ペインで、[**JDBC ｜接続プール**] をクリックします。
4. 右ペインで、[**新しい JDBC Connection Pool のコンフィグレーション**] をクリックします。
5. 以下の値を入力します。

[**名前**]weblogic.jdbc.connectionPool.demoPool
[**URL**]
jdbc:cloudscape:demo
[**ドライバ クラス名**]
COM.cloudscape.core.JDBCDriver
[**プロパティ**]
user=none
password=none
server=none
[**ACLName**]
weblogic.allow.reserve.jdbc.connectionPool.
demoPool=everyone

6. [**作成**] をクリックします。
7. [**接続**] をクリックします。
8. 以下の値を入力します。

   [**初期容量**] 1
   [**最大容量**] 2
   [**増加容量**] 1

9. [**適用**] をクリックします。
10. アプリケーションをデプロイできるように、Administration Console をそのまま起動しておきます。

展開形式のデプロイ

1. Administration Console の左ペインで、[**デプロイメント | アプリケーション**] をクリックします。
2. 右ペインで、[**新しい Application のコンフィグレーション**] をクリックします。
3. アプリケーションの名前、およびパス名の c:\banking\deploy を入力します。
4. [**デプロイ**] ボックスがチェックされていることを確認します。
5. [**作成**] をクリックします。

6. 左ペインの [**デプロイメント**] の下にアプリケーションが表示されていることを確認します。
7. c:\banking\deploy で、REDEPLOY という名前の空のテキスト ファイルを作成します。
8. アプリケーションを再デプロイする可能性がある場合は、REDEPLOY に対するパーミッションが read-write に設定されていることを確認します。
9. サンプル銀行業務アプリケーションを起動するには、Web ブラウザを開いて、次の URL に移動します。
   http://localhost:7001/banking/login.html.

アプリケーションのパッケージ化

1. deploy ディレクトリに戻ります。

   ```
   cd c:\
   cd banking\deploy
   ```

2. WAR ファイルを作成して、Web アプリケーションより 1 レベル上のディレクトリに格納します。

   ```
   jar cvf banking.war -C web .
   ```

3. WAR ファイルの内容をチェックして、適切であること確認します。

   ```
   jar tf banking.war

   META-INF/
   META-INF/MANIFEST.MF
   AccountDetail.jsp
   error.jsp
   login.html
   WEB-INF/
   WEB-INF/web.xml
   WEB-INF/weblogic.xml
   WEB-INF/lib/
   WEB-INF/classes/
   WEB-INF/classes/examples/
   WEB-INF/classes/examples/tutorials/
   WEB-INF/classes/examples/tutorials/migration/
   WEB-INF/classes/examples/tutorials/migration/banking/
   WEB-INF/classes/examples/tutorials/migration/banking
   ```

```
  BankAppServlet.class
images/
images/BEA_Button_Final_web.gif
```

4. 最初の EJB JAR ファイルを作成します。

   ```
   jar cvf banking.jar -C ejb .
   ```

5. JAR ファイルの内容をチェックします。

   ```
   jar tf banking.jar
   ```

   JAR ファイルの内容は以下のようになります。

   ```
   META-INF/
   META-INF/MANIFEST.MF
   META-INF/ejb-jar.xml
   META-INF/weblogic-cmp-rdbms-jar.xml
   META-INF/weblogic-ejb-jar.xml
   examples/
   examples/tutorials/
   examples/tutorials/migration/
   examples/tutorials/migration/banking/
   examples/tutorials/migration/banking/Account.class
   examples/tutorials/migration/banking/AccountBean.class
   examples/tutorials/migration/banking/AccountHome.class
   examples/tutorials/migration/banking/BankConstants.class
   examples/tutorials/migration/banking/Client.class
   examples/tutorials/migration/banking/
    ProcessingErrorException.class
   examples/tutorials/migration/banking/RMILogger.class
   examples/tutorials/migration/banking/RMILoggerImpl.class
   ```

6. EAR ファイルを構築するための新しいディレクトリを作成します。

   ```
   mkdir ear
   ```

7. WAR ファイルと EJB JAR ファイルをそこに移動します。

   ```
   move banking.war ear
   move banking.jar ear
   ```

8. アプリケーションの `META-INF` ディレクトリも `ear` に移動します。

   ```
   move META-INF ear
   ```

9. XML エディタで、`ear\META-INF\application.xml` を開きます。

10. WAR ファイルと EJB JAR ファイルを使用するように module 要素を編集します。

```
<module>
   <ejb>banking.jar</ejb>
</module>
<module>
   <web>
     <web-uri>banking.war</web-uri>
     <context-root>banking</context-root>
   </web>
</module>
```

11. EAR ファイルを構築します。

```
jar cvf banking.ear .
```

12. EAR ファイルの内容をチェックします。

```
jar tf banking.ear

META-INF/
META-INF/MANIFEST.MF
banking.war
banking.jar
META-INF/application.xml
```

EAR ファイルのデプロイ

1. Web ブラウザを開いて、Administration Console を起動します（**http://localhost:7001/console** に移動するか、サーバをインストールしたときに指定したホスト名とリスン ポート名を使用します）。
2. 右ペインの [**デプロイメント**] の下にある [**アプリケーション**] をクリックします。
3. [**新しい Application のコンフィグレーション**] をクリックします。
4. アプリケーション名とパス名を入力します。
5. [**デプロイ**] をチェックします。
6. [**作成**] をクリックします。

7. Web ブラウザを開いて **http://localhost:7001/banking/login.html** に移動します。
8. アプリケーションをテスト済みで、プロダクション モードでデプロイする準備が整っている場合は、ドメイン ディレクトリに移動します。

   ```
   cd c:\
   cd WL_HOME\config\mydomain
   ```

   mydomain を置き換えて別のドメインを使用できます。
9. テキスト エディタを使用して startWebLogic.cmd を開きます。
10. set STARTMODE を検索します。サーバをプロダクション モードで起動するために、値が true であることを確認します。
11. ファイルを保存して閉じます。
12. banking.ear を *WL_HOME*\config\mydomain\applications にコピーします。
13. サーバをプロダクション モードで起動します。startWebLogic

    ドメインのアプリケーション ディレクトリに配置されているため、サンプル アプリケーションが自動的にデプロイされます。