BEA WebLogic Server™

WebLogic Builder Online Help

BEA WebLogic Server バージョン 7.0
マニュアルの日付 : 2002 年 6 月
改訂 : 2002 年 6 月 28 日

WebLogic Builder Online Help

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	2002 年 6 月 28 日	BEA WebLogic Server バージョン 7.0

目次

1. WebLogic Builder

1 WebLogic Builder

この章の内容は以下のとおりです。

- WebLogic Builder の起動
- WebLogic Server への J2EE モジュールの移行
- Web アプリケーションの操作
- EJB の操作
- J2EE コンテナの操作
- WebLogic Builder のユーザ インタフェース
- WebLogic Builder を使用した Smart Ticket の移植とデプロイ

WebLogic Builder の仕組み

WebLogic Builder は、アプリケーションのデプロイメント記述子 XML ファイルを視覚的に編集するための環境を提供します。WebLogic Builder では記述子ファイルを表示させて視覚的に編集できるため、テキストでこれらの XML ファイルを編集する必要がありません。

WebLogic Builder の機能

最初に、WebLogic Builder で、任意のアプリケーションのコンパイル済み J2EE コンポーネント (*.class ファイルまたは *.class ファイルを含むモジュール) を選択します。WebLogic Server にデプロイするために必要なデプロイメント記述子ファイルがないか、または問題がある場合、WebLogic Builder は、使用可能な新しい記述子ファイルを生成するかどうかをユーザに確認します。

デプロイメント記述子ファイルがあれば、WebLogic Builder を使用して、デプロイメント記述子の要素や属性を編集できます。たとえば、Web アプリケーションにタグ ライブラリを追加したり、EJB にファインダ メソッドを追加したりできます。

サーバにアプリケーションをデプロイするために、WebLogic Builder を使用してアプリケーションをテストします。

推奨される WebLogic Builder の使用方法

WebLogic Builder では、以下の作業を行うことができます。

- J2EE モジュール用のデプロイメント記述子ファイルの生成
- モジュールのデプロイメント記述子ファイルの編集
- デプロイメント記述子ファイルの表示
- デプロイメント記述子ファイルのコンパイルおよび検証
- モジュールのサーバへのデプロイメント

WebLogic Builder の制限事項

WebLogic Builder には以下の制限事項があります。

- アプリケーションの記述子ファイルに、新しいモジュールを追加することはできません。
- *.class ファイルに対して行われた変更は、そのモジュールを閉じてから再度開いた場合にのみ、認識されます。
- EJB 1.1 Bean 用の記述子の生成に対するサポートは保証されていません。EJB 2.0 を対象としています。
- 検証は EJB のみに行われます。
- 変更後保存していない記述子ファイルと、変更前の記述子ファイルとの差分を、自動的に表示させることはできません。
- 記述子の XML 要素の値をバッチ更新することはできません。
- モジュールの記述子ファイルからコンポーネントを削除することはできません。

- WebLogic Builder 上で開いている状態で、記述子ファイルに変更を加えた場合、Builder ではその変更は認識されません。
- ファイルの管理機能はありません。
- エンティティ Bean 間の関連について生成された XML 表現は 1 対 1 の関連でのみ正確です。すでに記述子のあるエンティティ Bean の場合、Builder は「多」サイドのある関連を更新しません。

WebLogic Builder の起動

[スタート] メニューまたはコマンドラインで、WebLogic Builder を起動します。

[スタート] メニューからの場合は、[WebLogic 7.0] の下の [WebLogic Builder] をダブルクリックします。

コマンドラインの場合は、次のコマンドを使用します。

Windows の場合 :

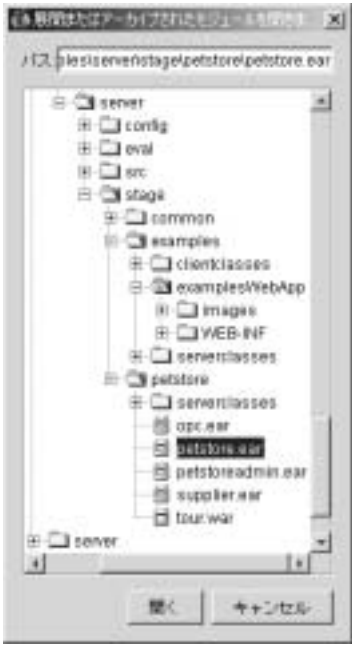
```
startWLBuilder.cmd
```

Unix の場合 :

```
startWLBuilder.sh
```

このコマンドで、環境を設定して WebLogic Builder を起動します。

[ファイル] メニューの [開く] オプションを使用して、モジュール (JAR、EAR、WAR、または展開形式の J2EE モジュール) を開きます。



WebLogic Builder でモジュールを開くと、左側のナビゲーション ツリー画面に、開いたモジュールの記述子ファイルが表示されます。これらのファイルを使用してアプリケーションのコンポーネントを検索および選択します。

右側の画面には、タブ付きのパネルが表示されます。パネル中のフィールドやその他のコントロールを使用してモジュールのデプロイメント記述子の要素を編集します。

左側のナビゲーション ツリー画面でアプリケーションのコンポーネントを選択して、右側のパネルの対応するタブで編集します。

インタフェースの詳細については、「WebLogic Builder のユーザ インタフェース」を参照してください。

WebLogic Server への J2EE モジュールの移行

WebLogic Server のデプロイメント記述子を持たないモジュールを WebLogic Server へ移行するには、[ファイル] メニューの [アーカイブを開く] または [ディレクトリを開く] を使用してモジュールを開きます。

WebLogic Builder では、WebLogic Server に正常にデプロイするために必要なデプロイメント記述子を、モジュールがすべて持っているかどうかをチェックします。必要なデプロイメント記述子がない場合、WebLogic Builder はそれを生成するかどうかをユーザに確認します。同意すると、WebLogic Builder はモジュール内のクラス ファイルを参照して適切なデプロイメント記述子ファイルを作成します。

既存のデプロイメント記述子ファイルが上書きされることはありません。

WebLogic Builder を使用したアプリケーションの WebLogic Server への移行の詳細については、「WebLogic Builder を使用した Smart Ticket の移植とデプロイ」を参照してください。

Web アプリケーションの操作

Web アプリケーションに関する情報については、以下の節を参照してください。

- サーブレット マッピングおよびセキュリティ制約を指定したサーブレットの追加
- ejb-ref/ejb-local-ref および Reference-Description の追加
- リソース参照の追加
- リスナ クラスの追加
- フィルタ マッピングを指定したフィルタの追加
- 11 ページの「マッチ マップ クラスの定義」
- ウェルカム ページおよびエラー ページの設定
- タグ ライブラリの追加
- 仮想ディレクトリの追加

サーブレット マッピングおよびセキュリティ制約を指定したサーブレットの追加

この節では、Web アプリケーションのデプロイメント記述子ファイルにサーブレットを追加し、それらにセキュリティ ロール、制約、および割り当てをコンフィグレーションする方法について説明します。

URL マッピングを指定したサーブレットの追加

デプロイメント記述子ファイルに新しいサーブレットを追加するには、次の手順に従います。

1. ナビゲーション ツリーで、Web アプリケーションの名前の下にある [サーブレット] を選択します。
2. [サーブレット] パネルで、サーブレットを選択して [追加] をクリックします。
3. [一般] タブで、サーブレット名およびサーブレット クラスまたは JSP ファイルを入力します。
4. 必要に応じて、URL パターンを入力して [追加] をクリックすることで、URL マッピング リストにサーブレットに対する URL マッピングを追加します。
5. [OK] をクリックします。

   サーブレットの名前が、ナビゲーション ツリーの [サーブレット] ノードに表示されます。

セキュリティ ロール、制約、および割り当ての追加

セキュリティ ロールにセキュリティ制約および割り当てを追加するには、次の手順に従います。

1. ナビゲーション ツリーの [Web Application] ノードの下で、[セキュリティ ロール] を選択します。

2. 編集パネルで [追加] をクリックし、セキュリティ ロールの名前と説明を入力して、[OK] をクリックします。
3. ナビゲーション ツリーの [Web Application] ノードの下にある [Security Roles Assignments] を選択します。
4. 編集パネルで、[ロール] を選択して [追加] をクリックします。
5. 編集ダイアログで、ロールのメンバーの名前を追加します。
6. [セキュリティ制約] ノードを展開して、ロール ノードを選択します。
7. [リソース / ページ] タブで以下の設定を行います。
   - [Web リソース名]
   - [URL パターン]
   - [HTTP メソッド]
   - 許可されるロールと許可されないロール
   - [転送の保証]
   - [表示名]
8. [ロール] タブで、[リソース / ページ] 設定が許可されるロールを設定します。

ejb-ref/ejb-local-ref および Reference-Description の追加

1. ナビゲーション ツリーで、Web アプリケーションの名前の下にある [サーブレット] ディレクトリを開き、[J2EE 参照] を選択します。
2. [J2EE 参照] 編集パネルで、[EJB 参照] タブを選択して [追加] をクリックします。
3. 編集ダイアログで、EJB に関する以下の項目を指定して [OK] をクリックします。
   - [参照名]
   - [リンク名] (省略可能)

- [EJB タイプ] (Session または Entity)
- [リモート インタフェース]
- [ホーム インタフェース]
- [実行する名前] (省略可能)
- [説明] (省略可能)

`web.xml` に EJB 参照が記述され、[J2EE 参照] 編集パネルの [EJB 参照] タブに表示されます。

リソース参照の追加

`web.xml` および `weblogic.xml` へリソース参照を追加するには、次の手順に従います。

1. ナビゲーション ツリーで、Web アプリケーションの名前の下にある [サーブレット] ディレクトリを開き、[J2EE 参照] を選択します。
2. [J2EE 参照] 編集パネルで、[リソース参照] タブを選択して [追加] をクリックします。
3. 編集ダイアログで、以下を選択します。
   - [参照型]
   - [リソース共有]
   - [リソース認証]

   説明および参照名を追加して、[OK] をクリックします。
4. ナビゲーション ツリーで [Web Applications] の下の [WebLogic Settings] ノードを展開し、[J2EE Links] を選択して [追加] をクリックします。
5. 手順 3 で追加したリソース参照を選択し、WebLogic の JNDI 名を入力して [OK] をクリックします。

リスナ クラスの追加

Web アプリケーションにイベント リスナ クラスを追加するには、次の手順に従います。

1. ナビゲーション ツリーで、Web アプリケーションの名前の下にある [その他] を選択します。
2. [その他] パネルで、[リスナ] を選択して [追加] をクリックします。
3. イベントのクラス名を入力して、[OK] をクリックします。

フィルタ マッピングを指定したフィルタの追加

1. ナビゲーション ツリーの Web アプリケーションの名前の下で、[フィルタ] を選択します。
2. [フィルタ] 編集パネルで [追加] をクリックします。
3. 編集ダイアログで、フィルタの以下の表示設定を入力します。
   - [表示名]
   - [小さいアイコン] (Web アプリケーション内に配置される)
   - [説明] (省略可能)
   - [大きいアイコン] (Web アプリケーション内に配置される)
4. 同じダイアログで、[初期パラメータ] を選択して [追加] をクリックします。
5. フィルタのパラメータ名および値を入力して [OK] をクリックします。

マッチ マップ クラスの定義

次の手順に従って、Web アプリケーションの URL パターン マッチング用のクラスを指定するマッチ マップを定義します。この手順で記述された `url-match-map` 要素は、`weblogic.xml` に配置されます。url-match-map を参照してください。

1. ナビゲーション ツリーの Web アプリケーションの名前の下で、[その他] ノードを選択してから [コンテナ設定] タブを選択します。
2. [リダイレクトするコンテンツ] フィールド (リダイレクトに使用するユーザが読めるデータの値を指定) および [リダイレクトするコンテンツ タイプ] (サーブレット コンテナが内部リダイレクトの応答でタイプを設定するのに使用) に入力した値は保持されません。これらの値は、`weblogic.xml` のテキストで設定します。
3. チェックして、リダイレクトで絶対 URL を使用するかどうかを指定します。チェックを外すと、サーブレット コンテナはリダイレクトのロケーション ヘッダで相対 URL を絶対 URL に変換しません。
4. 転送で認証を確認するかどうかを指定します。チェックすると、リクエスト ディスパッチャは転送されたリクエストで認可を確認します。
5. この Web アプリケーションの URL マッチマップ クラスの名前を入力します。

ウェルカム ページおよびエラー ページの設定

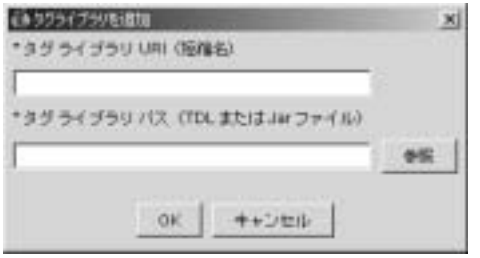
Web アプリケーションのウェルカム ページおよびエラー ページを設定するには、次の手順に従います。

1. ナビゲーション ツリーで、Web アプリケーションの名前を選択します。
2. 選択した Web アプリケーションの編集パネルで、[ウェルカム ファイル] を選択します。
3. [上に移動] および [下に移動] ボタンを使用して既存のウェルカム ファイルの順序を設定するか、またはファイル名を入力して [追加] をクリックし、新しいファイルを追加します。
4. [エラー ページ] タブを選択して [追加] をクリックします。
5. ファイル名を入力するか、ファイルを選択します。[HTTP エラーコード] または [例外タイプ] を設定して [OK] をクリックします。

タグ ライブラリの追加

タグ ライブラリを追加するには、次の手順に従います。

1. ナビゲーション ツリーで、Web アプリケーションの名前の下にある [タグ ライブラリ] を選択します。
2. [タグ ライブラリ] 編集パネルで、[追加] をクリックします。
3. タグ ライブラリの URI を入力します。
4. TLD または JAR ファイルの場所を入力するか、または参照して選択します。
5. [OK] をクリックします。



仮想ディレクトリの追加

Web アプリケーションに仮想ディレクトリを追加するには、次の手順に従います。

1. ナビゲーション ツリーの [Web Application] の下にある [WebLogic Settings] で、[Virtual Directory] を選択して [追加] をクリックします。
2. ローカル ディレクトリ パスを入力して設定します。その URL パターンを下部のテキスト フィールドに入力し、[追加] をクリックして追加します。次に [OK] をクリックします。

EJB の操作

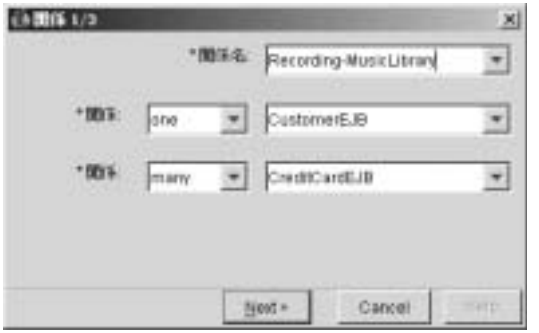
EJB に関する情報については、以下の節を参照してください。

- 2.0 CMP Bean 間の関係の作成
- エンティティ Bean への CMP フィールドの追加
- EJB へのファインダ メソッドの追加
- 2 つの Bean 間の ejb-reference の追加

2.0 CMP Bean 間の関係の作成

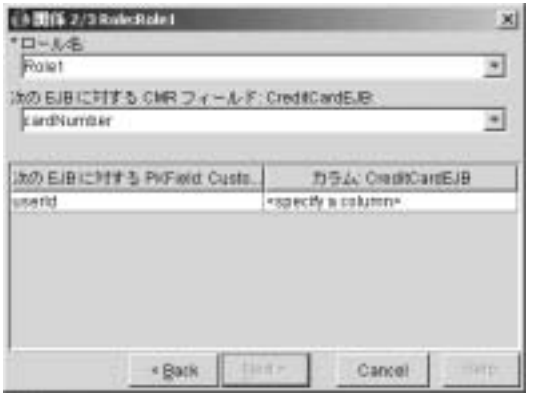
2 つの 2.0 CMP Bean 間の関係を作成するには、次の手順に従います。

1. ナビゲーション ツリーで、[EJB] ノードの下の [関係] ノードを右クリックして [関係を追加 ...] を選択します。モジュールに CMP Bean がない場合は、WebLogic Builder に [関係] ノードは表示されません。
2. [関係] ダイアログで、関係の名前を入力または選択します。



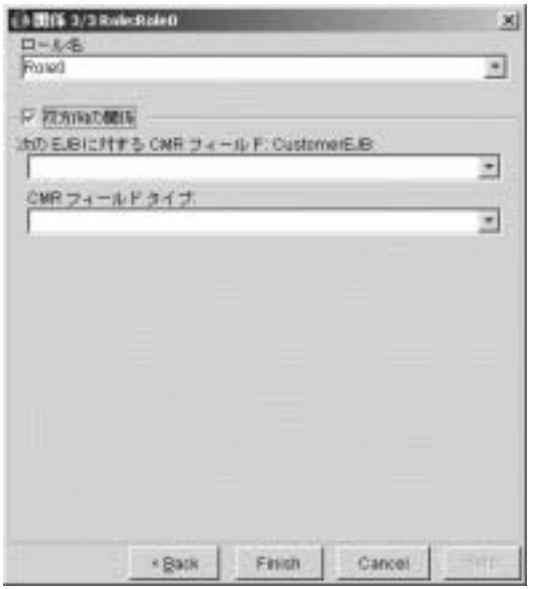
3. 1 つ目の Bean の [関係] オプションを [One] または [Many] に設定し、Bean を選択します。
4. 2 つ目の Bean の [関係] オプションを、1 つ目の Bean の [関係] オプションに合わせて設定し、2 つ目の Bean を選択して [次へ] をクリックします。

5. 2 番目の [関係] ダイアログで、ロール名、2 つ目の Bean の CMR (コンテナ管理による関係) フィールド、1 つ目の Bean の主キー フィールド、および 2 つ目の Bean のカラムを選択します。



6. [次へ] をクリックします。

7. 3 番目の [関係] ダイアログで、ロール名を選択します。また、必要に応じて、CMR フィールドとフィールド タイプを選択して双方向の関係を設定します。



8. [完了] をクリックします。

   WebLogic Builder によって、関係が `ejb-jar.xml` に記述され、関係のエントリが [関係] ノードに表示されます。

エンティティ Bean への CMP フィールドの追加

エンティティ Bean にコンテナ管理による永続性フィールドを追加するには、次の手順に従います。

1. ナビゲーション ツリーで、[EJB] ノードの下のエンティティ Bean ノードを展開し、[CMP フィールド] ノードを選択します。
2. [CMP フィールド] ダイアログで、Bean クラスのゲッターに対応するフィールド名を選択します。たとえば、Bean クラスに `getFirstName()` がある場合、CMP フィールドの名前は `firstName` となります。
3. 参照ボタンを使用して、テーブル名を選択します。サーバに接続していない場合は、参照ボタンをクリックすると [サーバに接続] ダイアログがアクティブになります。
4. カラム名を選択するには、参照ボタンをクリックして、テーブルを参照します。カラム名を選択したら、[OK] をクリックします。
5. カラム タイプを設定します。
6. [OK] をクリックします。

   ナビゲーション ツリーで、Bean の [CMP] ノードの下に、新しい CMP フィールドが表示されます。

EJB へのファインダ メソッドの追加

Bean にファインダ メソッドを追加するには、次の手順に従います。

1. ナビゲーション ツリーで、エンティティ Bean の名前の下にある Bean を展開して [ファインダ] を選択します。
2. Bean の [ファイダ] 編集パネルで、[追加] をクリックします。

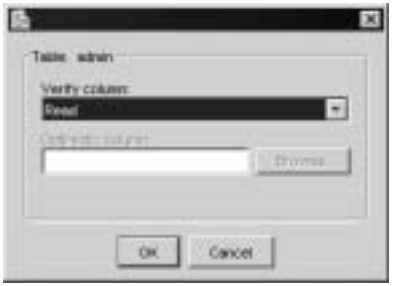
3. メソッド名を選択し、そのプロパティを入力して [OK] をクリックします。

Optimistic 同時方式の指定

平行トランザクションが衝突しそうにない場合、または応答時間の速度がトランザクションが衝突しないことより重要である場合は、CMP エンティティ Bean の Optimistic 同時方式を設定できます。WebLogic Builder のデフォルト設定は Pessimistic 同時方式です。Optimistic 同時方式を設定するには、次の手順に従います。

1. 左側のナビゲーション パネルでエンティティ Bean が選択されている状態で、[チューニング | キャッシュ] を選択します。[同時方式] 選択フィールドで [Optimistic] を選択します。

[同時実行性をコンフィグレーションします] をクリックし、[検証] カラムと [Optimistic] カラムを選択してエンティティ Bean をテーブルにマッピングします。



2. [検証] カラムで [バージョン] または [タイムスタンプ] を選択して [Optimistic] カラム フィールドを有効にします。
3. [参照] ボタンを使用すると、サーバに接続して、データベースを参照し、カラムを選択することができます。カラム名を直接入力することもできます。

2 つの Bean 間の ejb-reference の追加

1. ナビゲーション ツリーで、[EJB] を展開して [リソース] を選択します。
2. EJB の [リソース] 編集パネルで、[EJB 参照] タブまたは [EJB ローカル参照] タブを選択して [追加] をクリックします。

J2EE コンテナの操作

- モジュールの順序づけ
- EJB キャッシングの設定
- セキュリティ レルムの選択

モジュールの順序づけ

モジュールのデプロイ順を設定するには、次の手順に従います。

1. ナビゲーション ツリーで、モジュールを選択します。
2. モジュールの編集パネルで、[デプロイメント順] を選択します。
3. モジュールのコンポーネントをリストするフィールドでコンポーネントを選択し、[上へ移動] および [下へ移動] ボタンを使用してデプロイ順をリセットします。

EJB キャッシングの設定

EJB キャッシングを設定するには、次の手順に従います。

1. ナビゲーション ツリーの [EJB] ノードの下で、Bean を展開し、[チューニング] を選択します。
2. [チューニング] パネルで、キャッシングの条件を以下のように設定します。
   - 同時方式の名前を入力します。
   - トランザクション間のキャッシュを行うオプションをチェックするか、またはそのままにしておきます。
   - キャッシュ内の Bean の最大数、アイドル タイムアウト、および読み込みのタイムアウトを設定します。
   - キャッシュの参照用に、エンティティ キャッシュの名前を選択し、推定 Bean サイズを設定します。

セキュリティ レルムの選択

モジュールのセキュリティ レルムを設定するには、次の手順に従います。

1. ナビゲーション ツリーで、[WebLogic Application 設定] ノードを選択します。

2. [WebLogic Application 設定] 編集パネルで、[セキュリティ レルム] タブを選択します。
3. [セキュリティ レルム] タブで、レルム名を入力します。

WebLogic Builder のユーザ インタフェース

この節では、メニューのタスクについて説明し、WebLogic Builder インタフェースでのデプロイメント記述子要素の場所を示します。

20 ページの「メニューのタスク」

25 ページの「WebLogic Builder でのデプロイメント記述子の要素」

メニューのタスク

- アプリケーションを開く
- サーバに接続する
- デプロイメント
- コンパイラの選択
- アプリケーションを閉じる
- アプリケーションを保存する
- アプリケーションを検証する
- 記述子を生成する
- コンポーネントの記述子を削除する
- 新しい記述子の要素を追加する
- デプロイメント記述子の要素を削除する
- デプロイメント記述子 XML ファイルを表示する

アプリケーションを開く

アーカイブ形式または展開されたモジュールを開くには、[ファイル] メニューで [開く] を選択します。アーカイブ形式のモジュール、または展開されたモジュールを含むディレクトリに移動し、モジュールを選択して [開く] をクリックします。

トラブルシューティング

付属の MANIFEST.MF ファイルで参照されない、ネストされた JAR を持つ EAR のように、整形式でないモジュールをロードしようとすると、WebLogic Builder でモジュールを開くときに問題が起こる場合があります。

サーバに接続する

テスト用にモジュールをデプロイしたり、データ ソースにモジュールを接続したりするには、サーバに接続します。

[ツール] メニューで [サーバに接続] を選択します。

ダイアログに、接続に関する情報を入力して、[接続] をクリックします。

デプロイメント

[ツール] メニューから [モジュールのデプロイ] を選択します。サーバに接続していない場合は、[WebLogic Server に接続します] ダイアログが表示されます。

コンパイラの選択

1. [ツール] メニューで、[Options] を選択します。
2. [Options] ダイアログで、[EJBC コンパイラ] を選択します。
3. [参照] をクリックして、コンパイラを検索します。コンパイラを選択して、[開く] をクリックします。

アプリケーションを閉じる

[ファイル] メニューで、[閉じる] を選択します。

アプリケーションを保存する

[ファイル] メニューで、[保存] を選択します。

WebLogic Builder でデプロイメント記述子ファイルに対して行ったすべての変更がモジュールに保存されます。

アプリケーションを検証する

検証では、モジュールに対する新たな変更は保存されません。

[ツール] メニューで [記述子の検証] を選択して、モジュールを検証します。

記述子を生成する

新しいモジュールを開くと、WebLogic Builder から、開いたモジュールのデプロイメント記述子を生成するかどうかの確認を求められます。同意すると、WebLogic Builder では、新しい記述子を作成して、モジュール内の適切な場所に書き込みます。

コンポーネントの記述子を削除する

モジュールからコンポーネントを削除するには、WebLogic Builder を使用せずに、モジュールから関連する記述子の要素を削除します。

新しい記述子の要素を追加する

WebLogic Builder を使用せずに、モジュールに新しい記述子の要素を追加します。

デプロイメント記述子の要素を削除する

WebLogic Builder を使用せずに、モジュールからファイルを削除します。

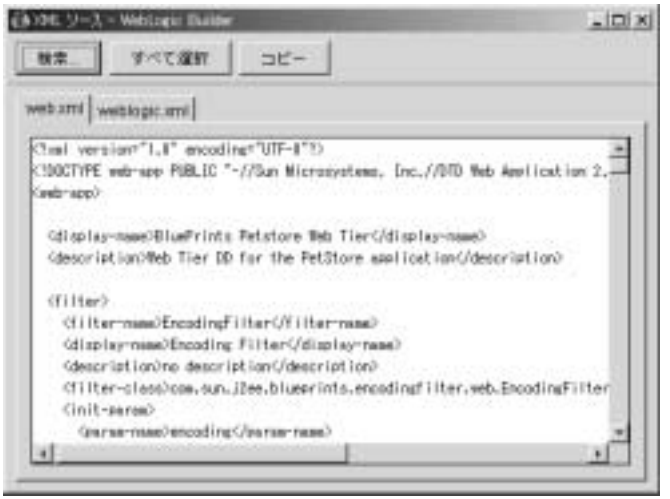
デプロイメント記述子 XML ファイルを表示する

選択しているコンポーネントの XML ファイルを表示するには、次の手順に従います。ここで表示する XML は読み込み専用であることに注意してください。

1. [表示] メニューで [XML ソース] を選択します。

   タブ付きの XML ビューアが表示されます。

2. タブを使用して、表示させる XML ファイルを選択します。

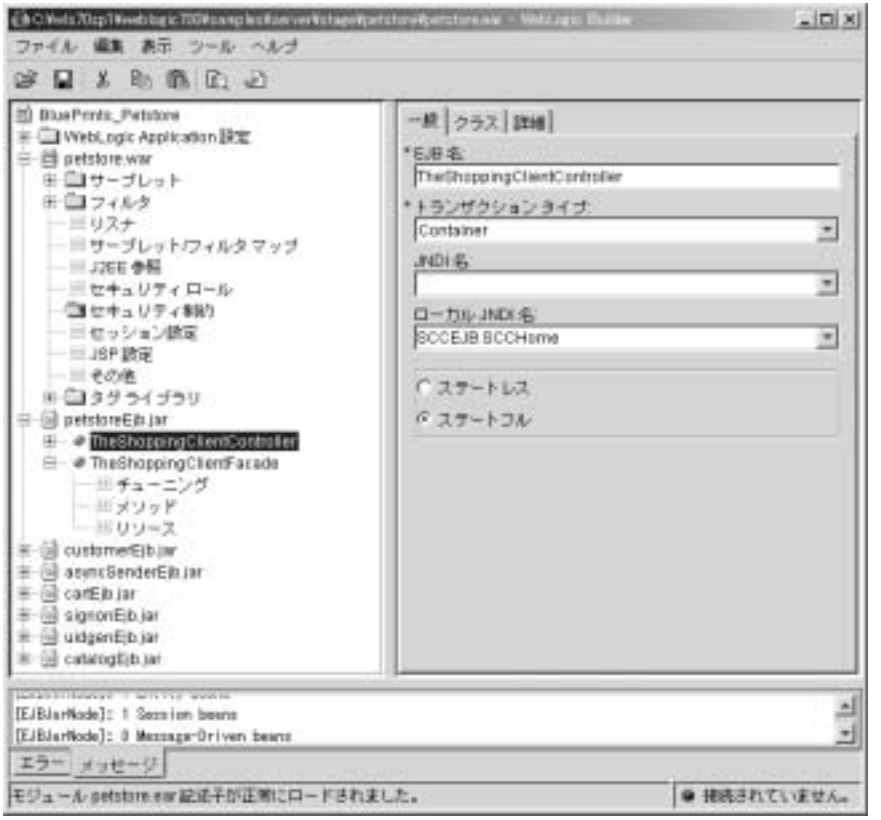

検索
すべて選択
コピー
web.xml
weblogic.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.
<web-app>
<display-name>BluePrints Petstore Web Tier</display-name>
<description>Web Tier DD for the PetStore application</description>
<filter>
<filter-name>EncodingFilter</filter-name>
<display-name>Encoding Filter</display-name>
<description>no description</description>
<filter-class>com.sun.j2ee.blueprints.encodingfilter.web.EncodingFilter
<init-param>
<param-name>encoding</param-name>

WebLogic Builder でのデプロイメント記述子の要素

以下の節では、WebLogic Builder におけるデプロイメント記述子の要素の場所を示します。

- WebLogic Builder での weblogic.xml の要素
- WebLogic Builder での web.xml の要素
- WebLogic Builder での weblogic-application.xml の要素
- WebLogic Builder での ejb-jar.xml の要素
- WebLogic Builder での weblogic-ejb-jar.xml の要素
- WebLogic Builder での Tag Lib の要素
- WebLogic Builder での weblogic-cmp20-rdbms-jar.xml の要素

左ペインのファイル ツリーに表示されているノードには、デプロイメント記述子のさまざまな要素が直感的にグループ化されています。これらのノードをクリックして、右側の複数の編集パネルをナビゲートすることにより、デプロイメント記述子の要素を変更できます。



WebLogic Builder での weblogic.xml の要素

次の表に、weblogic.xml の要素と、それらの WebLogic Builder インタフェースでの場所を示します。「weblogic.xml デプロイメント記述子の要素」を参照してください。

XML 要素	WebLogic Builder でのアクセス
description	[Web Application ｜ 表示]

XML 要素	WebLogic Builder でのアクセス
weblogic-version	
security-role-assignment : role-name、 principal-name	[Web Application ｜セキュリティ ロール]
reference-descriptor: resource-description、 res-ref-name、 ejb-reference-descriptio n、ejb-ref-name、 resource-env-descriptio n、res-env-ref-name	[Web Application ｜ J2EE 参照]

XML 要素	WebLogic Builder でのアクセス
session-descriptor: session-param、 param-name: (CacheSize、 ConsoleMainAttribute、 CookieComment、 CookieDomain、 CookieMaxAgeSecs、 CookieName、 CookiePath、 CookiesEnabled、 IDLength、 InvalidationIntervalSecs 、 JDBCConnectionTimeo utSecs、 PersistentStoreCookieN ame、 PersistentStoreDir、 PersistentStorePool、 PersistentStoreType、 SwapIntervalSecs、 TimeoutSecs、 TrackingEnabled、 URLRewritingEnabled)	[Web Application ｜セッション設定]

XML 要素	WebLogic Builder でのアクセス
jsp-descriptor: jsp-param、 param-name (compileCommand、 compileFlags、 compilerClass、 compilerSupportsEncoding、defaultFilename、 encoding、 keepgenerated、 noTryBlocks、 packagePrefix、 pageCheckSeconds、 precompile、verbose、 workingDir、debug)	[Web Application ｜ JSP 設定]
container-descriptor	[Web Application ｜その他｜コンテナ設定]
charset-params	Web Application ｜その他｜ IANA-Java 文字セット マップ]、および [Web Application ｜その他｜パス文字セット マップ]
virtual-directory-mapping: (local-path、url-pattern)	[Web Application ｜その他｜仮想ディレクトリ]
url-match-map	[Web Application ｜その他｜コンテナ設定]
security-permission	[Web Application ｜セキュリティ制約]

WebLogic Builder での web.xml の要素

次の表に、web.xml の要素と、それらの WebLogic Builder インタフェースでの場所を示します。「web.xml デプロイメント記述子の要素」を参照してください。

XML 要素および属性	WebLogic Builder でのアクセス
icon	[Web Application ｜表示]
display-name	[Web Application ｜表示]
description	[Web Application ｜表示]
distributable	サポートされていない。
context-param	[Web Application ｜コンテキスト パラメータ]
filter: icon、filter-name、 display-name、 description、 filter-class、init-param	[Web Application ｜フィルタ｜フィルタ]
filter-mapping	[Web Application ｜サーブレット / フィルタ マップ｜ フィルタ マッピング]
listener	[Web Application ｜リスナ｜リスナ クラス]
servlet: icon、servlet-name、 display-name、 description、 (servlet-class|jsp-file)、 init-param、 load-on-startup、 security-role-ref	[Web Application ｜サーブレット｜サーブレット]
servlet-mapping: servlet-name、 url-pattern	[Web Application ｜サーブレット / フィルタ マップ｜ サーブレット マッピング]
session-config: session-timeout	[Web Application ｜セッション設定｜一般]

XML 要素および属性	WebLogic Builder でのアクセス
mime-mapping: extension、mime-type	[Web Application ｜ MIME タイプ]
welcome-file-list	[Web Application ｜ウェルカム ファイル]
error-page: (error-code|exception-type)、location	[Web Application ｜エラー ページ]
taglib: taglib-uri、 taglib-location	[Web Application ｜タグ ライブラリ ｜ Tag Libraries]
resource-env-ref: description、 resource-env-ref-name、 resource-env-ref-type	[Web Application ｜ J2EE 参照｜リソース環境参照]
resource-ref: description、 res-ref-name、res-type、 res-auth、 res-sharing-scope	[Web Application ｜ J2EE 参照｜リソース参照]
security-constraint: display-name、 web-resource-collection 、auth-constraint、 user-data-constraint	[Web Application ｜セキュリティ制約｜リソース / ページ、ロール、SSL/ その他]
login-config: auth-method、 realm-name、 form-login-config	[Web Application ｜ログイン]
security-role: description、role-name	[&Web ApplicationWeb Application ｜セキュリティ ロール｜ロール名、説明、プリンシパル名]

XML 要素および属性	WebLogic Builder でのアクセス
security-role-ref: description、 role-name、role-link	[Web Application ｜ Servlets サーブレット ｜ Servlet ｜ Security Role Refs]
env-entry: description、 env-entry-name、 env-entry-value、 env-entry-type	[Web Application ｜ J2EE 参照 ｜ 環境エントリ]
ejb-ref description、 ejb-ref-name、 ejb-ref-type、home、 remote、ejb-link、 run-as	[Web Application ｜ J2EE 参照 ｜ EJB 参照]

WebLogic Builder での weblogic-application.xml の要素

次の表に、weblogic-application.xml の要素と、それらの WebLogic Builder インタフェースでの場所を示します。「weblogic-application.xml デプロイメント記述子の要素」を参照してください。

XML 要素および属性	WebLogic Builder でのアクセス
weblogic-application	[WebLogic Application 設定]
ejb: entity-cache (entity-cache-name、 (max-beans-in-cache | max-cache-size)、 caching-strategy、 start-mdbs-with-application	[WebLogic Application 設定 ｜ EJB 設定]

XML 要素および属性	WebLogic Builder でのアクセス
xml: parser-factory (saxparser-factory、 document-builder-factor y、 transformer-factory)、 entity-mapping (entity-mapping-name、 public-id、system-id、 entity-uri、 when-to-cache、 cache-timeout-interval)	[WebLogic Application 設定｜ XML パーサ ファクトリ、XML エンティティ マッピング]
security: realm-name	[WebLogic Application 設定｜セキュリティ レルム]

XML 要素および属性	WebLogic Builder でのアクセス
jdbc-connection-pool: data-source-name、connection-factory (factory-name、connection-properties)、pool-params (size-params、xa-params、login-delay-seconds、leak-profiling-enabled、connection-check-params)、driver-params (statement、prepared-statement、row-prefetch-enabled、row-prefetch-size、stream-chunk-size)、xa-params (debug-level、keep-conn-until-tx-complete-enabled、end-only-once-enabled、recover-only-once-enabled、tx-context-on-close-needed、new-conn-for-commit-enabled、prepared-statement-cache-size、keep-logical-conn-open-on-release、local-transaction-supported、resource-health-monitoring-enabled) acl-name	[WebLogic Application 設定｜ JDBC データ ソース｜一般、接続、プール、XA 設定、ドライバ]

WebLogic Builder での ejb-jar.xml の要素

次の表に、ejb-jar.xml の要素と、それらの WebLogic Builder インタフェースでの場所を示します。「WebLogic Server デプロイメント ファイル」を参照してください。

XML 要素および属性	WebLogic Builder でのアクセス
abstract-schema-name	[EJB ｜詳細]
acknowledge-mode	[Message Driven Bean ｜詳細]
security-role	[EJB ｜セキュリティ]
method-permission	[EJB ｜ ｜パーミッション]
container-transaction	[EJB ｜メソッド｜トランザクション]
cascade-delete	サポートされていない。
cmp-field: description、field-name	[EJB ｜ CMP フィールド]
cmp-version	[EJB ｜永続性]
cmr-field: description、 cmr-field-name、 cmr-field-type	[EJB ｜関係｜ (関係ノードを右クリックして) 関係を追加]
destination-type	[Message Driven Bean ｜一般]
ejb-class	[EJB ｜クラス]
ejb-client-jar	サポートされていない。
ejb-link	[EJB ｜リソース]

XML 要素および属性	WebLogic Builder でのアクセス
ejb-local-ref: description、 ejb-ref-name、 ejb-ref-type、 local-home、local、 ejb-link	[EJB ｜リソース｜ EJB ローカル参照]
ejb-name	[EJB ｜一般]
ejb-ql	[EJB Application ｜ファインダ]
ejb-ref: description、home、 remote、ejb-link	[EJB ｜リソース｜ EJB 参照]
ejb-relation: description、 ejb-relation-name、 ejb-relationship-role	[関係｜ (関係を右クリックして) 関係の追加]
ejb-relationship-role: description、 ejb-relationship-role-name、multiplicity、 relationship-role-source、cmr-field	[関係｜ (関係を右クリックして) 関係の追加]
ejb-relationship-role-name	[関係｜ (関係を右クリックして) 関係の追加]
ejb-class: home、remote、 local-home、local	[EJB ｜クラス]
primkey-field	[Entity Bean ｜永続性]

XML 要素および属性	WebLogic Builder でのアクセス
resource-env-ref: env-entry (description、 env-entry-name、 env-entry-type、 env-entry-value)	[EJB ｜リソース]
field-name	[Entity Bean ｜ CMP フィールド]
message-driven: ejb-name、ejb-class、 message-driven-destination	[Message Driven Bean ｜一般｜クラス]
message-selector: acknowledge-mode、 transaction-type	[Message Driven Bean ｜詳細]
subscription-durability	[Message Driven Bean ｜一般]
persistence-type	[Entity Bean ｜永続性]
prim-key-class	[Entity Bean ｜永続性]
primkey-field	[Entity Bean ｜永続性]
query: description、 query-method、 result-type-mapping、 ejb-ql	[EJB ｜ファインダ]
reentrant	[EJB ｜詳細]
relationships: description、 ejb-relation	[関係｜ (関係を右クリックして) 関係の追加]

XML 要素および属性	WebLogic Builder でのアクセス
resource-env-ref: description、 resource-env-ref-name、 resource-env-ref-type	[EJB ｜ リソース｜環境]
resource-ref: description、 res-ref-name、res-type、 res-auth、 res-sharing-scope	[EJB ｜リソース｜リソース参照]
role-name	[Enterprise Application ｜ Security]
session-type	[Session Bean ｜一般]
session: ejb-name、home, remote、local-home、 local、ejb-class、 session-type、 transaction-type	[Session Bean ｜一般、クラス]
session: env-entry、ejb-ref、 ejb-local-ref、 security-role-ref、 security-identity、 resource-ref、 resource-env-ref	[Session Bean ｜リソース｜環境、リソース参照、EJB 参照、EJB ローカル参照]
subscription-durability	[Message Driven Bean ｜一般]
transaction-type	[EJB ｜メソッド｜トランザクション]
trans-attribute	[EJB ｜メソッド｜トランザクション]

WebLogic Builder での weblogic-ejb-jar.xml の要素

次の表に、weblogic-ejb-jar.xml の要素と、それらの WebLogic Builder インタフェースでの場所を示します。「weblogic-ejb-jar.xml 文書型定義」を参照してください。

XML 要素および属性	WebLogic Builder でのアクセス
cache-between-transactions	[EJB ｜チューニング｜キャッシュ]
concurrency-strategy	[EJB ｜チューニング｜キャッシュ]
connection-factory-jndi-name	[Message Driven Bean ｜外部 JMS プロバイダ]
jms-polling-interval-seconds	[Message Driven Bean ｜詳細]
jms-client-id	[Message Driven Bean ｜詳細]
delay-updates-until-end-of-tx	[EJB ｜永続性]
destination-jndi-name	[Message Driven Bean ｜一般]
ejb-reference-description: ejb-ref-name、jndi-name	[EJB ｜リソース｜ EJB 参照]
ejb-local-reference-description: ejb-ref-name、jndi-name	[EJB ｜リソース｜ EJB ローカル参照]
enable-call-by-reference	セッション Bean の場合 :[EJB ｜詳細] エンティティ Bean の場合 :[EJB ｜永続性]
enable-dynamic-queries	サポートされていない。

XML 要素および属性	WebLogic Builder でのアクセス
entity-cache: max-beans-in-cache、idle-timeout-seconds、read-timeout-seconds、concurrency-strategy、cache-between-transactions	[EJB ｜チューニング｜キャッシュ] エンティティ Bean にはサポートされていない。
entity-cache-ref: entity-cache-name、concurrency-strategy、cache-between-transactions、estimated-bean-size	[EJB ｜チューニング｜プール]
entity-cache-name	サポートされていない。
estimated-bean-size	サポートされていない。
entity-clustering: home-is-clusterable、home-load-algorithm、home-call-router-class-name	[EJB ｜チューニング｜クラスタ]
enable-dynamic-queries	サポートされていない。
finders-load-bean	[EJB ｜詳細]
home-call-router-class-name	[Session and Entity Beans ｜チューニング｜クラスタ]
home-is-clusterable	[Session and Entity Beans ｜チューニング｜クラスタ]
home-load-algorithm	[Session and Entity Beans ｜チューニング｜クラスタ]
idempotent-methods	[EJB ｜チューニング｜クラスタ]

XML 要素および属性	WebLogic Builder でのアクセス
idle-timeout-seconds	[EJB ｜チューニング｜キャッシュ] ステートフル セッション Bean にはサポートされていない。
cache-type	サポートされていない。
initial-beans-in-free-pool	[EJB ｜チューニング｜クラスタ]
initial-context-factory	[Message Driven ｜外部 JMS プロバイダ]
is-modified-method-name	サポートされていない。
isolation-level	[｜メソッド｜トランザクション]
jndi-name	[Entity Bean ｜一般]
clients-on-same-server	サポートされていない。
local-jndi-name	[EJB ｜一般]
max-beans-in-cache	[EJB ｜チューニング｜キャッシュ]
max-beans-in-free-pool	[EJB ｜チューニング｜プール]
message-driven-descriptor	サポートされていない。
persistence-use	サポートされていない。
pool: max-beans-in-free-pool 、 initial-beans-in-free-pool	[EJB ｜チューニング｜プール]
read-timeout-seconds	[EJB ｜チューニング｜キャッシュ]
replication-type	[EJB ｜チューニング｜クラスタ]

XML 要素および属性	WebLogic Builder でのアクセス
security-role-assignment: role-name、principal-name	[EJB Application]
stateful-session-clustering: home-is-clusterable、home-load-algorithm、home-call-router-class-name、replication-type	[EJB ｜チューニング｜クラスタ]
stateful-session-cache: max-beans-in-cache、idle-timeout-seconds	[EJB ｜チューニング｜キャッシュ]
stateless-bean-call-router-class-name	サポートされていない。
stateless-bean-is-clusterable	サポートされていない。
stateless-bean-load-algorithm	サポートされていない。
stateless-bean-methods-are-idempotent	サポートされていない。

XML 要素および属性	WebLogic Builder でのアクセス
stateless-clustering: home-is-clusterable、 home-load-algorithm、 home-call-router-class-name、 stateless-bean-is-clusterable、 stateless-bean-load-algorithm、 stateless-bean-call-router-class-name、 stateless-bean-methods-are-idempotent	サポートされていない。
stateless-session-descriptor: pool、 stateless-clustering	サポートされていない。
transaction-isolation: isolation-level	サポートされていない。
trans-timeout-seconds	[Entity Bean ｜永続性]
type-identifier	サポートされていない。
provider-url	[Message Driven ｜外部 JMS プロバイダ]
invalidation-target: ejb-name	[Entity Bean ｜詳細]

WebLogic Builder での Tag Lib の要素

次の表に、タグ ライブラリの要素と、それらの WebLogic Builder インタフェースでの場所を示します。「タグ ライブラリ記述子の作成」を参照してください。

XML 要素および属性	WebLogic Builder でのアクセス
taglib: tlib-version、 jsp-version、 short-name、uri、 display-name、 small-icon、large-icon、 description、validator、 listener	[Web Application ｜タグ ライブラリ｜パス、URI]

WebLogic Builder での weblogic-cmp20-rdbms-jar.xml の要素

次の表に、weblogic-cmp20-rdbms-jar.xml の要素と、それらの WebLogic Builder インタフェースでの場所を示します。「WebLogic Server のコンテナ管理による永続性サービス」を参照してください。

XML 要素および属性	WebLogic Builder でのアクセス
create-default-dbms-table	[EJB ｜ Application]
delay-database-insert-until	[Entity Bean ｜詳細]
automatic-key-generation	[Entity Bean ｜自動キー生成]
field-group	サポートされていない。
table-map: table-name、field-map	[EJB Application ｜関係｜ (関係ノードを右クリックして) 関係の追加]
verify-columns、 optimistic-column	サポートされていない。

XML 要素および属性	WebLogic Builder でのアクセス
check-exists-on-method	[Entity Bean ｜詳細]
ejb-name	[EJB ｜一般]
data-source-name	[EJB ｜永続性]
table-name	[EJB Application ｜関係｜ (関係ノードを右クリックして) 関係の追加]
field-map: cmp-field、 dbms-column、 dbms-column-type	サポートされていない。
cmp-field	[EJB Application ｜関係｜ (関係ノードを右クリックして) 関係の追加]
dbms-column	[EJB Application ｜関係｜ (関係ノードを右クリックして) 関係の追加]
optimistic-column	サポートされていない。
dbms-column-type	[｜ CMP フィールド｜ CMP]
column-map: foreign-key-column、 key-column	[EJB Application ｜関係｜ (関係ノードを右クリックして) 関係の追加]
weblogic-rdbms-relation : relation-name、 table-name、 weblogic-relationship-role、 relationship-role-name	[EJB Application ｜関係｜ (関係ノードを右クリックして) 関係の追加]
relationship-role-map: foreign-key-table、 primary-key-table、 column-map	[EJB Application ｜関係｜ (関係ノードを右クリックして) 関係の追加]

XML 要素および属性	WebLogic Builder でのアクセス
group-name	[EJB Application ｜ファインダファインダクエリ設定]
cmr-field	[EJB Application ｜関係｜ (関係ノードを右クリックして) 関係の追加]
relationship-caching: caching-name、 caching-element	サポートされていない。
caching-name	サポートされていない。
caching-element: cmr-field、 group-name、 caching-element	サポートされていない。
weblogic-query: query-method、 weblogic-ql、 group-name、 max-elements、 include-updates	[EJB Application ｜ファインダファインダ]
sql-select-distinct	サポートされていない。
weblogic-ql	[EJB Application ｜ファインダファインダ]
method-name	[EJB Application ｜ファインダファインダ]
query-method	[EJB Application ｜ファインダファインダ]
max-elements	[EJB Application ｜ファインダファインダ]
include-updates	[EJB Application ｜ファインダファインダ]
sql-select-distinct	[EJB Application ｜ファインダファインダ]

XML 要素および属性	WebLogic Builder でのアクセス
automatic-key-generation: generator-type、 generator-name、 key-cache-size	[｜自動キー生成]
generator-type	[｜自動キー生成]
generator-name	[EJB ｜自動キー生成]
key-cache-size	[｜自動キー生成]
delay-database-insert-until	[EJB ｜詳細]
validate-db-schema-with	サポートされていない。
database-type	サポートされていない。

WebLogic Builder を使用した Smart Ticket の移植とデプロイ

この例では、Sun の BluePrint ワイヤレス アプリケーションである Smart Ticket を迅速にデプロイするシナリオを示します。WebLogic Builder を使用して、WebLogic Server 固有のデプロイメント記述子ファイルを生成および編集し、WebLogic Server にアプリケーションをデプロイします。

WebLogic Builder は、アプリケーションのデプロイメント記述子ファイルを生成および編集するためのビジュアル環境です。WebLogic Builder では記述子ファイルを表示させて視覚的に編集できるため、XML をテキストで編集する必要がありません。「WebLogic Builder」を参照してください。

この章の内容は以下のとおりです。

- アプリケーションおよび環境の設定
  - WebLogic Server 7.0 をダウンロードおよびインストールする
  - Smart Ticket をダウンロードおよびインストールする
  - Sun Wireless Toolkit をダウンロードおよびインストールする
  - 環境を設定する
  - Smart Ticket をビルドする
- WebLogic Builder を使用した変換とチューニング
  - WebLogic Builder を使用して weblogic.xml と weblogic-ejb-jar.xml を生成する
  - WebLogic Builder を使用して記述子を編集する
- 管理タスク
  - サーバ (WebLogic Server の Examples Server) を起動する
  - WebLogic Server Administration Console を使用してデータソースをコンフィグレーションする
  - PointBase サンプル データベースを使用するように populate.bat スクリプトと SQL クエリを変更する

- デプロイメントと実行
  - WebLogic Builder を使用して Smart Ticket をデプロイする
  - Smart Ticket を起動する
  - ユーザ アカウントを作成して映画のチケットを予約するために、データ ソースへの問い合わせと書き込みを行う

アプリケーションおよび環境の設定

この例を実行するには、次のものが必要です。

- WebLogic Server 7.0
- Smart Ticket
- Sun Wireless Toolkit (J2ME)
- `smarticketPointBase.sql` (付属)

ここでは、インストール、ビルド、および必要なパス設定を行います。

1. WebLogic Server 7.0 をダウンロードおよびインストールします。

   WebLogic Server 7.0 を http://www.bea.com からダウンロードし、*WL_HOME* と呼ぶ場所にインストールします。デフォルトでは、*WL_HOME* ÇÕ `c:\bea\weblogic700` です。

2. Smart Ticket 1.1 をダウンロードおよびインストールします。

   Smart Ticket デモ アプリケーションのソース コードを http://developer.java.sun.com/developer/releases/smarticket/ からダウンロードします。マシン上の新しいディレクトリにソース コードを展開します。このディレクトリを *SMARTICKET_HOME* と呼びます。

3. J2ME をダウンロードし、*J2MEWTK_HOME* と呼ぶ場所 (デフォルトは `C:\J2mewtk`) にインストールします。

   Sun Wireless Toolkit を http://java.sun.com/products/j2mewtoolkit/download.html からダウンロードします。ツールキットをインストールします。インストール中に JDK を選択

するよう要求されます。*BEA_HOME*/jdk131 ディレクトリに格納されている JDK を選択できます。

4. `J2MEWTK_HOME=C:\J2mewtk` を設定します。C:\J2mewtk は、J2ME のインストール ディレクトリとします。注意 : J2MEWTK_HOME を設定しないと、アプリケーションをビルドできなくなります。
5. *`WL_HOME`*`\samples\server\config\examples` にある、setExamplesEnv スクリプトを実行して環境を設定します。
6. *SMARTICKET_HOME*`\smarticket\localant.bat` の ANT_CLASSPATH 行の末尾に「%CLASSPATH%」を追加します。
7. *SMARTICKET_HOME*\smarticket ディレクトリに移動し、`localant.bat`を実行して Smart Ticket をビルドします。WebLogic Builder では、コンパイル済みの `.class` ファイルが必要です。`.java` ファイルは使用できません。

WebLogic Builder を使用した変換とチューニング

この節では、WebLogic Builder でデプロイメント記述子を生成し、その一部を編集します。その際、XML を参照する必要はありません。

記述子の生成

この手順では、WebLogic Builder で既存の記述子ファイルを読み込み、アプリケーションの `.class` ファイルを参照して、WebLogic Server でアプリケーションを実行できるようにするデプロイメント記述子ファイルを作成します。

既存のデプロイメント記述子ファイルが上書きされることはありません。

1. [スタート｜プログラム｜ BEA WebLogic Platform ｜ WebLogic Server 7.0 ｜ WebLogic Builder] を選択して、WebLogic Builder を起動します。

2. WebLogic Builder の [ファイル | 開く] メニューを選択して、*SMARTICKET_HOME* \build\server に移動し、[開く] をクリックします。[モジュールのデプロイメント記述子が見つかりません。記述子を新規作成しますか？] というダイアログが表示されます。[はい] をクリックすると、Smart Ticket のクラス ファイルが参照され、`weblogic.xml` および `weblogic-ejb-jar.xml` が生成されます。

3. [ファイル | 保存] を選択して、WebLogic Builder でアプリケーションを保存します。

4. WebLogic Builder で [ファイル | アーカイブを開く] を選択し、*SMARTICKET_HOME* `\bin\smarticket.ear` を指定して、`smarticket.ear` を作成します。

5. [スタート | プログラム | BEA WebLogic Platform | WebLogic Server 7.0 | Server Tour and Examples | Launch Examples Server] を選択して、Examples Server を起動します。[WebLogic Server Examples] ページが表示されます。

6. `http://localhost:7001/console` に移動して (または [WebLogic Server Examples] ページのリンクをたどって)、ユーザ名 `weblogic`、パスワード `weblogic` で署名して WebLogic Administration Console を開きます。

JNDI 名と <context-root> の指定

ここでは、WebLogic Server を使用して以下のことを行います。

- Web アプリケーションのコンテキスト パスを設定する
- Web アプリケーションの EJB 参照とリソース参照に JNDI 名を割り当てる
- EJB リソース参照に JNDI 名を割り当てる

1. WebLogic Builder で \web ノードの [コンテキスト パス] タブを選択します。[コンテキスト パス] フィールドに「SmarTicketApp」と入力します。これで `<context-root>` 要素が設定されました。

2. Builder の [J2EE 参照] ノードで [EJB 参照] パネルを選択して、EJB 参照に対する JNDI 名を以下のように設定します。

[参照名]	[EJB タイプ]	[JNDI 名]
ejb/MovieInfo	Session	MovieInfo
ejb/TicketSales	Entity	TicketSales
ejb/Customer	Entity	Customer
ejb/LocaleInfo	Session	LocaleInfo

3. Builder の [J2EE 参照] ノードで [リソース参照] パネルを選択して、[参照名]、[参照型]、および [JNDI 名] を以下の表のように設定します。[リソース認証] を [Container] に設定します。

[参照名]	[参照型]	[JNDI 名]
jdbc/MovieInfoDataSource	java.sql.DataSource	MovieInfoDataSource
jdbc/TicketSalesDataSource	java.sql.DataSource	TicketSalesDataSource
jdbc/CustomerDataSource	java.sql.DataSource	CustomerDataSource
jdbc/LocaleInfoDataSource	java.sql.DataSource	LocaleInfoDataSource

4. Builder の [EJB Resources] ノードで [リソース参照] パネルを選択して、EJB リソースに対する JNDI 名を以下のように設定します。

[リソース参照名]	[リソース参照型]	[JNDI 名]
jdbc/MovieInfoDataSource	java.sql.DataSource	MovieInfoDataSource
jdbc/TicketSalesDataSource	java.sql.DataSource	TicketSalesDataSource
jdbc/CustomerDataSource	java.sql.DataSource	CustomerDataSource
jdbc/LocaleInfoDataSource	java.sql.DataSource	LocaleInfoDataSource

5. Builder の [ファイル | 保存] を使用して、アーカイブへの変更を保存します。

管理タスク

この節では、以下のことを行います。

- WebLogic Server Administration Console を使用してデータ ソースをコンフィグレーションする
- PointBase サンプル RDBMS へ変換する

データ ソースのコンフィグレーション

これまでの作業により、WebLogic Server Administration Console を使用して、Smart Ticket アプリケーションで使用される 4 つの各 EJB のデータ ソースをコンフィグレーションできるようになりました。

1. まだ行っていない場合は、WebLogic Examples Server を起動し、ブラウザで `http://localhost:7001/console` を指定して Administration Console を開きます。
2. [JDBC] ノードを選択して [トランザクション データ ソース] をクリックします。
3. [新しい Tx Data Source のコンフィグレーション] を選択します。
4. データ ソースの名前を入力します。最初のフィールドに「MyCustomerDataSource」と入力します。[JNDI 名] フィールドに「CustomerDataSource」と入力します。[プール名] フィールドに「demoPool」(WebLogic サンプルで使用されるデフォルトの接続プール) と入力して、[作成] をクリックします。
5. [対象] タブをクリックします。[選択可] カラムで examplesServer を選択し、右矢印をクリックして対象に入れ、[適用] をクリックします。

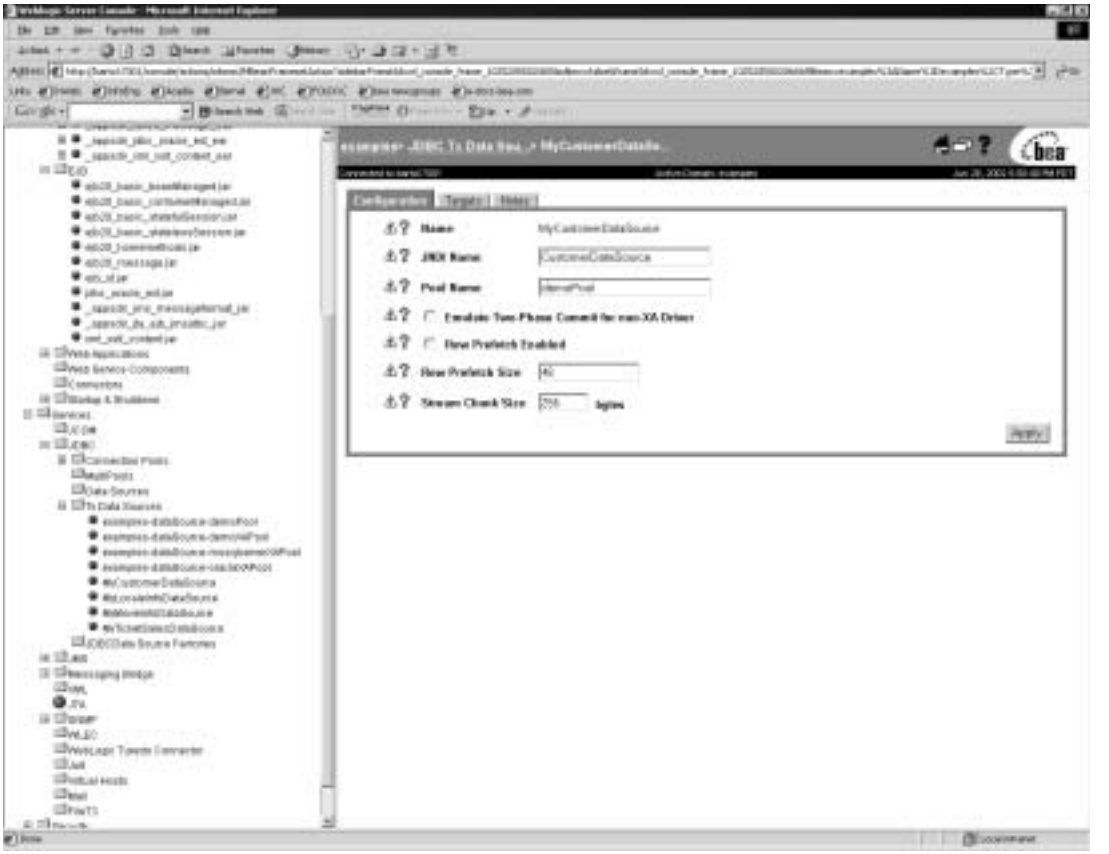


図 1　Administration Console での [トランザクション データ ソース] の設定

6. 他の 3 つのデータ ソース (MyMovieInfoDataSource、MyLocaleInfoDataSource、MyTicketSalesDataSource) にも、手順 4 と 5 を繰り返します。

7. Smart Ticket の Cloudscape データベースを、WebLogic Server 7.0 に付属の PointBase の評価版 RDBMS に置き換えるには、*SMARTICKET_HOME* `\smarticket\populate.bat` に次のコードを追加します。

```
set POINTBASEHOME=%SAMPLES_HOME%\server\eval\pointbase

java utils.Schema
jdbc:pointbase:server://localhost/demo,database.home=%POINTBASE
```

```
HOME% com.pointbase.jdbc.jdbcUniversalDriver -u examples -p
examples -verbose ./src/smarticketPointBase.sql
```

8. また、`smarticket.sql` を PointBase で扱いやすくするために、データ型「int」を「integer」に置換するスクリプトで smarticket.sql を置き換えます。作業としては、`smarticketPointBase.sql` を `SMARTICKET_HOME\smarticket\src` にコピーします。
9. データベースを設定するには、`populate.bat` を実行します。

デプロイメントと実行

以上の作業で、WebLogic Server 7.0 で Smart Ticket をデプロイおよび実行する準備が整いました。

1. WebLogic Builder の [ツール] メニューで [サーバに接続] を選択して、サーバに接続します。

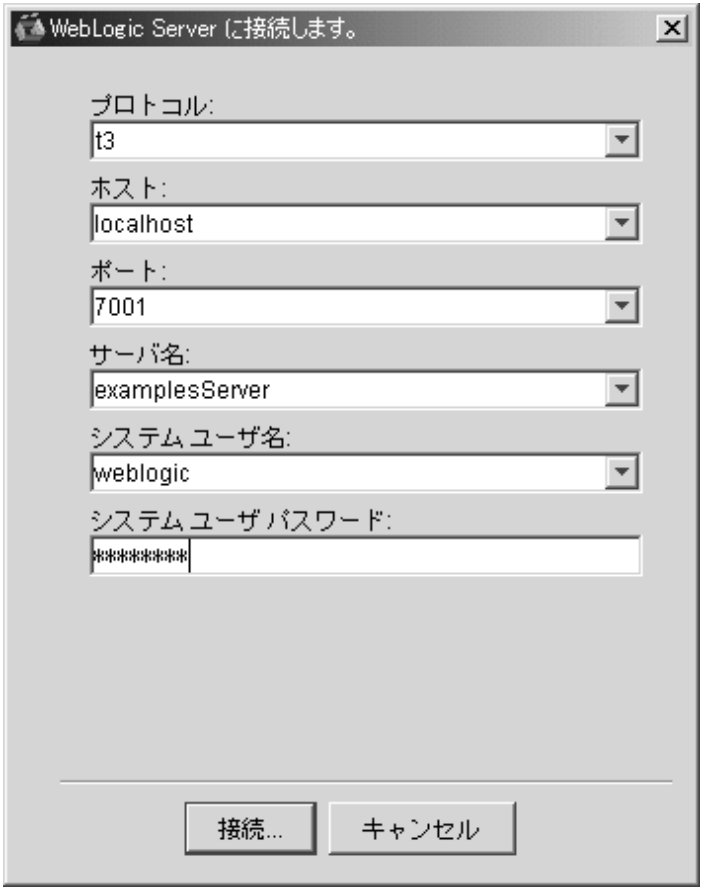
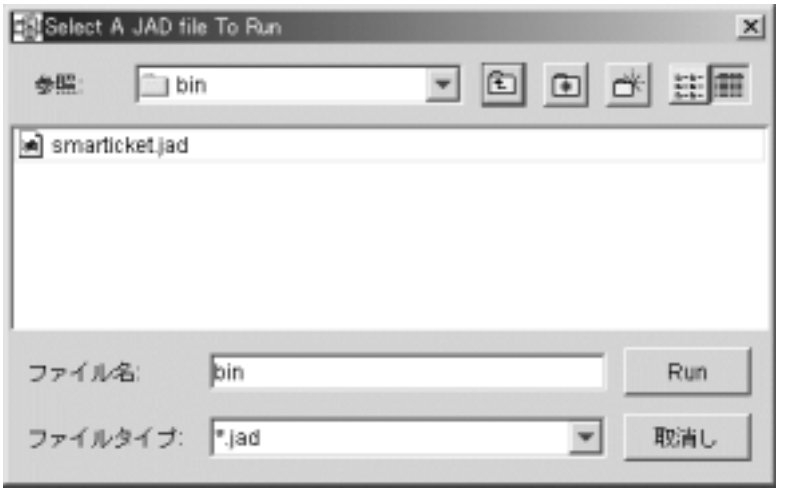


図 2　Examples Server への接続

2. WebLogic Builder の [ツール] メニューから [モジュールのデプロイ] を選択して、Smart Ticket をデプロイします。
3. 指定のポート番号 (8000) を Examples Server のポート番号 (デフォルトでは 7001) に置き換えて、*SMARTICKET_HOME*\bin\smarticket.jad のポートを設定します。
4. *SMARTICKET_HOME*\bin\smarticket.jad を起動するには、このファイルをダブルクリックするか、または [スタート | J2ME Wireless Application | Run MIDP Application] を選択してこのファイルを選択します。



5. 以上の作業により、Examples Server で Smart Ticket アプリケーションを実行できるようになりました。


DefaultGrayPhone
Locale
@locale@
en_GB
fr_FR
Back
Next
CLEAR
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PQRS
8 TUV
9 WXYZ
* –
0
+
MODE
SPACE

ワイヤレス アプリケーションの実行

Smart Ticket アプリケーションのユーザ アカウントを作成する際は、郵便番号に 95130 または 95054 を入力します。また、パスワードの長さは 6 文字でなければなりません。アカウントを作成したら、試しに [Poster] モードを選択してください。このモードでは、選択した映画の画像を電話またはエミュレータで表示できます。

エミュレータで利用できるオプションは複数あります。作成した J2ME Wireless Ticket からデフォルトのデバイスおよび任意のデバイスを操作してみてください。

以前にクライアント Smart Ticket アプリケーションを実行して、WebLogic でアプリケーションを再デプロイする場合、前のユーザ情報のデータベースをクリアする必要があります。これを行うには、Windows の [スタート] メニューから [J2ME Wireless Ticket Utility] を実行します。[Clean Database] をクリックすると、Smart Ticket Client が再起動されます。

まとめ

Sun のアプリケーションをダウンロードおよびビルドし、WebLogic Builder を使用して WebLogic Server での実行に必要な weblogic.xml ファイルと weblogic-ejb-jar.xml ファイルを生成しました。次に、WebLogic Builder でアプリケーションの要素を一部編集し、WebLogic Server Administration Console でアプリケーションのデータソースをコンフィグレーションしました。WebLogic Builder でアプリケーションをデプロイしてから、起動および実行しました。

関連情報

WebLogic Builder を使用せずに、Smart Ticket を WebLogic Server 7.0 に移植する手順については、
http://edocs.beasys.co.jp/e-docs/wls/docs70/quickstart/smarticket.html の「Java Smart Ticket Demo 1.1」を参照してください。

索引

F

G

H

I

J

K

L

M

T

U

V

W

て