Borland AppServer™ 6.7
DTDs

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

BAS67SCHEMA
2006 年 12 月

Borland®

目次

第 1 章

Borland AppServer の概要

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

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

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

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

AppServer は次のコンポーネントを備えています。

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

AppServer の機能

AppServer では次の機能が提供されます：

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

Borland AppServer のドキュメント

AppServer 関連のドキュメントには次のものがあります：

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

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

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

AppServer オンライン ヘルプ トピックへのアクセス

オンライン ヘルプにアクセスするには（次のいずれかの方法を利用）:

Windows の場合

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

UNIX の場合

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

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

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

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

ドキュメント表記規則

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

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

プラットフォームの表記規則

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

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

Borland サポートへのお問い合わせ

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

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

ボーランド社のサポートへの連絡にあたっては、次の情報をご用意ください。

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

オンライン リソース

ネットワーク上の次のサイトから情報を得ることができます。

ワールド ワイド ウェブ： http://www.borland.com

オンライン サポート： http://support.borland.com （ユーザー ID が必要）

ワールド ワイド ウェブ

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

特に、次の URL を確認されることをお勧めします：

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

Borland ニュースグループ

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

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

第 2 章

application-client-borland.xml

DTD

```
<!ELEMENT application-client (ejb-ref*, resource-ref*,
        resource-env-ref*, property*)>
<!ELEMENT ejb-ref (ejb-ref-name, jndi-name?)>
<!ELEMENT resource-ref (res-ref-name, jndi-name)>
<!ELEMENT resource-env-ref (resource-env-ref-name, jndi-name)>
<!ELEMENT property (prop-name, prop-type, prop-value)>
<!ELEMENT prop-name (#PCDATA)>
<!ELEMENT prop-type (#PCDATA)>
<!ELEMENT prop-value (#PCDATA)>
<!ELEMENT ejb-ref-name (#PCDATA)>
<!ELEMENT jndi-name (#PCDATA)>
<!ELEMENT res-ref-name (#PCDATA)>
<!ELEMENT resource-env-ref-name (#PCDATA)>
```

<application-client>

```
<!ELEMENT application-client (ejb-ref*, resource-ref*,
   resource-env-ref*, property*)>
```

この要素は `application-client-borland.xml` のルート ノードで、VisiClient コンテナで実行されるアプリケーション クライアントに必要な、ランタイム情報を定義します。このノードには、サブ要素 `ejb-ref`、`resource-ref`、`resource-env-ref`、`property` がそれぞれ 1 つ以上含まれています。

サンプル

```
<!DOCTYPE application-client PUBLIC "-//Borland Corporation//DTD J2EE
Application Client 1.3//EN"
 "http://www.borland.com/devsupport/appserver/dtds/application-client_1_3-
borland.dtd">
<application-client>
        <ejb-ref>
          <ejb-ref-name>ejb/Sort</ejb-ref-name>
          <jndi-name>sort</jndi-name>
        </ejb-ref>
        <resource-ref>
```

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

関連要素

- <ejb-ref>
- <resource-ref>
- <resource-env-ref>
- <property>

<ejb-ref>

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

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

サンプル

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

関連要素

- <ejb-ref-name>
- <jndi-name>

<ejb-ref-name>

```
<!ELEMENT ejb-ref-name (#PCDATA)>
```

この要素は、クライアント アプリケーションによってリソース リファレンスとして使用される EJB の名前を提供します。

サンプル

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

関連要素

- <ejb-ref>

<jndi-name>

```
<!ELEMENT jndi-name (#PCDATA)>
```

この要素は、クライアント アプリケーションによって参照されるリソースに対して、JNDI サービス ルックアップ機能を提供するためのものです。

サンプル

```
<jndi-name>sort</jndi-name>
```

関連要素

- <ejb-ref>
- <resource-ref>
- <resource-env-ref>

<property>

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

この要素は、実行時にクライアントに必要なプロパティ値を指定するために使用されます。各 `property` エントリは、対応するサブ要素を使用して、プロパティの名前、型、および値を指定します。

サンプル

```
<property>
   <prop-name>vbroker.security.disable</prop-name>
   <prop-type>security</prop-type>
   <prop-value>false</prop-value>
</property>
```

関連要素

- <prop-name>
- <prop-type>
- <prop-value>

<prop-name>

```
<!ELEMENT prop-name (#PCDATA)>
```

設定するプロパティの名前を指定します。

サンプル

```
<prop-name>vbroker.security.disable</prop-name>
```

関連要素

- <property>

<prop-type>

```
<!ELEMENT prop-type (#PCDATA)>
```

設定するプロパティの型を指定します。

サンプル

```
<prop-type>security</prop-type>
```

関連要素

- <property>

<prop-value>

```
<!ELEMENT prop-value (#PCDATA)>
```

設定するプロパティの値を指定します。

サンプル

```
<prop-value>false</prop-value>
```

関連要素

- <property>

<resource-env-ref-name>

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

この要素は、クライアント アプリケーションがリソース環境リファレンスにアクセスするために使用する名前を提供します。

サンプル

```
<res-env-ref-name>jdbc/CheckingDataSource</res-ref-name>
```

関連要素

- <resource-env-ref>

<res-ref-name>

```
<!ELEMENT res-ref-name (#PCDATA)>
```

この要素は、クライアント アプリケーションがリソース リファレンスにアクセスするために使用する名前を提供します。

サンプル

```
<res-ref-name>jdbc/CheckingDataSource</res-ref-name>
```

関連要素

- <resource-ref>

<resource-env-ref>

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

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

サンプル

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

関連要素

- <resource-env-ref-name>
- <jndi-name>

<resource-ref>

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

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

サンプル

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

関連要素

- <res-ref-name>
- <jndi-name>

第 3 章

ejb-borland.xml

DTD

```
<!ELEMENT ejb-jar (enterprise-beans, datasource-definitions?,
table-properties*, relationships?, authorization-domain?,
property*, assembly-descriptor?)>

<!ELEMENT enterprise-beans (session | entity | message-driven)+>

<!ELEMENT session (ejb-name, bean-home-name?, bean-local-home-name?,
timeout?, ejb-ref*, ejb-local-ref*, resource-ref*,resource-env-ref*, property*)>

<!ELEMENT entity (ejb-name, bean-home-name?, bean-local-home-name?,
ejb-ref*, ejb-local-ref*, resource-ref*, resource-env-ref*,
(cmp-info | cmp2-info)?, property*)>

<!ELEMENT message-driven (ejb-name, message-driven-destination-name,
connection-factory-name, pool?, ejb-ref*, ejb-local-ref*,
resource-ref*, resource-env-ref*, property*)>

<!ELEMENT pool (max-size?, init-size?, wait-timeout?)>
<!ELEMENT ejb-ref (ejb-ref-name, jndi-name?)>
<!ELEMENT ejb-local-ref (ejb-ref-name, jndi-name?)>
<!ELEMENT resource-ref (res-ref-name, jndi-name, cmp-resource?)>
<!ELEMENT resource-env-ref (resource-env-ref-name, jndi-name)>
<!ELEMENT datasource-definitions (datasource*)>
<!ELEMENT datasource (jndi-name, url, username?, password?,
 isolation-level?, driver-class-name?, jdbc-property*, property*)>
<!ELEMENT jdbc-property (prop-name, prop-value)>
<!ELEMENT property (prop-name, prop-type, prop-value)>
<!ELEMENT cmp-info (description?, database-map?, finder*)>
<!ELEMENT database-map (table?, column-map*)>
<!ELEMENT finder (method-signature, where-clause, load-state?)>
<!ELEMENT column-map (field-name, column-name?, column-type?, ejb-ref-
name?)>
<!ELEMENT connection-factory-name (#PCDATA)>
<!ELEMENT message-driven-destination-name (#PCDATA)>
<!ELEMENT max-size (#PCDATA)>
<!ELEMENT init-size (#PCDATA)>
<!ELEMENT wait-timeout (#PCDATA)>
<!ELEMENT cmp-resource (#PCDATA)>
```

```
<!ELEMENT method-signature (#PCDATA)>
<!ELEMENT where-clause (#PCDATA)>
<!ELEMENT load-state (#PCDATA)>
<!ELEMENT prop-name (#PCDATA)>
<!ELEMENT prop-type (#PCDATA)>
<!ELEMENT prop-value (#PCDATA)>
<!ELEMENT field-name (#PCDATA)>
<!ELEMENT column-name (#PCDATA)>
<!ELEMENT column-type (#PCDATA)>
<!ELEMENT table (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT ejb-name (#PCDATA)>
<!ELEMENT bean-home-name (#PCDATA)>
<!ELEMENT bean-local-home-name (#PCDATA)>
<!ELEMENT timeout (#PCDATA)>
<!ELEMENT ejb-ref-name (#PCDATA)>
<!ELEMENT jndi-name (#PCDATA)>
<!ELEMENT res-ref-name (#PCDATA)>
<!ELEMENT resource-env-ref-name (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT username (#PCDATA)>
<!ELEMENT password (#PCDATA)>
<!ELEMENT isolation-level (#PCDATA)>
<!ELEMENT driver-class-name (#PCDATA)>
<!ELEMENT authorization-domain (#PCDATA)>
<!ELEMENT table-properties (table-name, column-properties*, property*)>
<!ELEMENT column-properties (column-name, property*)>
<!ELEMENT table-name (#PCDATA)>
<!ELEMENT cmp2-info (cmp-field*, table-name, table-ref*)>
<!ELEMENT relationships (ejb-relation+)>
<!ELEMENT ejb-relation (ejb-relationship-role, ejb-relationship-role)>
<!ELEMENT ejb-relationship-role (relationship-role-source, cmr-field?)>
<!ELEMENT relationship-role-source (ejb-name)>
<!ELEMENT cmr-field (cmr-field-name, table-ref, property*)>
<!ELEMENT cmr-field-name (#PCDATA)>
<!ELEMENT table-ref (left-table, cross-table*, right-table)>
<!ELEMENT left-table (table-name, column-list)>
<!ELEMENT right-table (table-name, column-list)>
<!ELEMENT cross-table (table-name, column-list, column-list)>
<!ELEMENT column-list (column-name+)>
<!ELEMENT cmp-field (field-name, (cmp-field-map* | column-name),property*)>
<!ELEMENT cmp-field-map (field-name, column-name)>
<!ELEMENT assembly-descriptor (security-role*)>
<!ELEMENT security-role (role-name, deployment-role?)>
<!ELEMENT role-name (#PCDATA)>
<!ELEMENT deployment-role (#PCDATA)>
```

<assembly-descriptor>

```
<!ELEMENT assembly-descriptor (security-role*)>
```

この要素は、`ejb-jar.xml` 内の同じ要素の上に作成されます。その子ノードを使用して、そのアーカイブ内のモジュールが属している 1 つ以上のセキュリティ ロールに関する情報を提供します。

サンプル

```
<assembly-descriptor>
  <security-role>
    <role-name>administrator</role-name>
```

```
        <deployment-role>administrator</deployment-role>
    </security-role>
</assembly-descriptor>
```

関連要素

- <ejb-jar>
- <security-role>

<authorization-domain>

```
<!ELEMENT authorization-domain (#PCDATA)>
```

アーカイブが属する認証ドメインの名前。

サンプル

```
<authorization-domain>GroupJ</authorization-domain>
```

関連要素

- <ejb-jar>

<bean-home-name>

```
<!ELEMENT bean-home-name (#PCDATA)>
```

Bean のホーム インターフェイスを参照するために使用される名前を指定します。

サンプル

```
<bean-home-name>insurance/remote/clerk</bean-home-name>
```

関連要素

- <session>
- <entity>
- <message-driven>

<bean-local-home-name>

```
<!ELEMENT bean-local-home-name (#PCDATA)>
```

Bean のローカル ホーム インターフェイスを参照するために使用される名前を指定します。

サンプル

```
<bean-local-home-name>insurance/remote/clerk</bean-local-home-name>
```

関連要素

- <session>

- <entity>
- <message-driven>

<cascade-delete>

<!ELEMENT cascade-delete />

<cascade-delete> エンティティ Bean オブジェクトを削除する場合に使用します。オブジェクトに対してカスケード削除を指定すると、コンテナは、そのオブジェクトの従属オブジェクトをすべて自動的に削除します。

サンプル

```
<ejb-relation>
 <ejb-relation-name>Customer-Account</ejb-relation-name>
 <ejb-relationship-role>
  <ejb-relationship-role-name>Account-Has-Customer
  </ejb-relationship-role-name>
  <multiplicity>one</multiplicity>
  <cascade-delete/>
 </ejb-relationship-role>
</ejb-relation>
```

関連要素

- <ejb-relationship-role>

<cmp-field>

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

<cmp-field> 要素を使用して、基本的なフィールド マッピングが行われます。この要素の子ノードでは、フィールド名とマップ先の対応列を指定します。一般には粗粒度のエンティティ Bean で Java クラスを実装して細粒度のデータを表しています。3 番目の子ノード <cmp-field-map> は、細粒度クラスとその規定のデータベース表現とのフィールド マップを定義し、<column-name> 要素のかわりに使用できます。

サンプル

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

関連要素

- <field-name>
- <column-name>
- <cmp-field-map>

<cmp-field-map>

<!ELEMENT cmp-field-map (field-name, column-name)>

<cmp-field-map> 要素は、細粒度 Java クラスとその基底のデータベース表現とのフィールド マップを定義します。このようなクラスでは、java.io.Serializable を実装するものとし、そのすべてのデータ メンバーはパブリックであるものとします。

サンプル

```
<cmp-field>
  <field-name>Address</field-name>
  <cmp-field-map>
   <field-name>Address.AddressLine</field-name>
   <column-name>STREET</column-name>
  </cmp-field-map>
  <cmp-field-map>
  ...
 </cmp-field>
```

関連要素

- <cmp-field>
- <field-name>
- <column-name>

<cmp-info>

```
<!ELEMENT cmp-info (description?, database-map?, finder*)>
```

<cmp-info> 要素を使用して、CMP 1.x エンティティ Bean に関する情報を提供します (CMP 2.x を使用している場合には、<cmp2-info> を使用します)。この要素には 2 つの子ノード、<database-map> および <finder> があり、Bean の背後にあるデータベースへアクセスし、適切なクエリを使用するために必要なデータを指定します。

サンプル

```
<cmp-info>
        <description/>
        <database-map>
           <table>Courses</table>
           <column-map>
              <field-name>students</field-name>
              <ejb-ref-name>ejb/Student.findByCourse</ejb-ref-name>
           </column-map>
        </database-map>
        <finder>
           <method-signature>findByStudent(Student s)</method-signature>
           <where-clause>SELECT course_dept, course_number FROM
 Enrollment WHERE student  = :s[ejb/Student]</where-clause>
           <load-state>False</load-state>
        </finder>
</cmp-info>
```

関連要素

- <database-map>
- <finder>

<cmp-resource>

<!ELEMENT cmp-resource (#PCDATA)>

参照先のリソースが container-managed-persistence を使用する場合は、このフラグを `True` に設定します。この要素は CMP 1.x リソースでのみ有効です。このフラグが設定されている場合、BES は、標準の `ejb-jar.xml` デプロイメント ディスクリプタ内にある対応するリソース リファレンスを無視します。

サンプル

<cmp-resource>True**</cmp-resource>**

関連要素

- <resource-ref>

<cmp2-info>

<!ELEMENT cmp2-info (cmp-field*, table-name, table-ref*)>

CMP 2.0 を使用している場合は、`<cmp2-info>` 要素を使用して、エンティティ Bean を管理するコンテナに情報を提供します。この要素とその子ノードには、CMP フィールドをデータベース列にマップするためのすべての情報が含まれています。

サンプル

```
<cmp2-info>
 <cmp-field>
  <field-name>orderNumber</field-name>
  <column-name>ORDER_NUMBER</column-name>
 </cmp-field>
 <cmp-field>
  <field-name>line</field-name>
  <column-name>LINE</column-name>
 </cmp-field>
</cmp2-info>
```

関連要素

- <entity>
- <cmp-field>
- <table-name>
- <table-ref>

<cmr-field>

<!ELEMENT cmr-field (cmr-field-name, table-ref, property*)>

エンティティが別のエンティティへのマップに使用するフィールドと、それに付随する基底のテーブル マッピングを定義します。

サンプル

```
<cmr-field>
   <cmr-field-name>specialInformation</cmr-field-name>
   <table-ref>
    <left-table>
     <table-name>CUSTOMER</table-name>
     <column-list>CUSTOMER_NO</column-list>
    </left-table>
    <right-table>
     <table-name>SPECIAL_INFO</table-name>
     <column-list>CUSTOMER_NO</column-list>
    </right-table>
   </table-ref>
</cmr-field>
```

関連要素

- <ejb-relationship-role>
- <cmr-field-name>
- <table-ref>
- <property>

<cmr-field-name>

```
<!ELEMENT cmr-field-name (#PCDATA)>
```

エンティティが、関係を保持する別のエンティティへのマップに使用するフィールド。

サンプル

```
<cmr-field-name>specialInformation</cmr-field-name>
```

関連要素

- <cmr-field>

<column-list>

```
<!ELEMENT column-list (column-name+)>
```

別のテーブルの列にマップされる、テーブルの列を指定します。各列は、子ノード `<column-name>` で表されます。

サンプル

```
<column-list>
   <column-name>EMP_NO</column-name>
   <column-name>LAST_NAME</column-name>
   <column-name>PROJ_ID</column-name>
</column-list>
```

関連要素

- <table-ref>

- <left-table>
- <cross-table>
- <table-name>
- <column-name>

<column-map>

<!ELEMENT column-map (field-name, column-name?, column-type?, ejb-ref-name?)>

この要素は、エンティティ Bean が使用するデータベース列に関する情報を CMP 1.x エンジンに提供するために使用されます。エンティティ Bean のフィールド名のほか、そのフィールドに対応する列の情報またはその列を表す EJB リファレンスの名前のどちらかを提供します。

サンプル

```
<column-map>
    <field-name>students</field-name>
    <ejb-ref-name>ejb/Student.findByCourse</ejb-ref-name>
</column-map>
```

関連要素

- <table>
- <database-map>
- <field-name>
- <column-name>
- <column-type>
- <ejb-ref-name>

<column-name>

<!ELEMENT column-name (#PCDATA)>

エンティティ マッピングまたはプロパティ設定のためのデータベース列の名前を指定します。

サンプル

```
<column-name>course_dept</column-name>
```

関連要素

- <database-map>
- <field-name>
- <column-type>
- <ejb-ref-name>
- <cmp-field-map>
- <cmp-field>
- <column-properties>

<column-properties>

<!ELEMENT column-properties (column-name, property*)>

この要素を使用して、データベース テーブル内の列に固有のプロパティを指定します。この要素の子ノードで、列名や関連するプロパティを指定します。

関連要素

- <table-properties>
- <column-name>
- <property>

<column-type>

<!ELEMENT column-type (#PCDATA)>

エンティティ Bean の CMP クエリの一部としてデータベース列に格納されているデータの型を指定します。

サンプル

```
<column-type>integer</column-type>
```

関連要素

- <database-map>
- <field-name>
- <column-name>
- <column-type>
- <ejb-ref-name>

<connection-factory-name>

<!ELEMENT connection-factory-name (#PCDATA)>

Bean が JMS サービスに接続するために使用する JMS トピックまたはキューの接続ファクトリの JNDI 名。

サンプル

```
<connection-factory-name>serial://jms/xaqcf</connection-factory-name>
```

関連要素

- <message-driven>
- <message-driven-destination-name>

<cross-table>

<!ELEMENT cross-table (table-name, column-list, column-list)>

多対多の関係を定義する場合、CMP エンジンに、左右テーブルの関係をモデル化する、クロステーブルを作成させなければなりません。これを行う際に、`<cross-table>` 要

素を使用します。クロステーブルには、子ノードの <table-name> 要素で、好きな名前をつけることができます。2 つの子要素 <column-list> は、関係モデルを作成する左右テーブルの列に相当します。

サンプル

```
<table-ref>
 <left-table>
  <table-name>EMPLOYEE</table-name>
  <column-list>
   <column-name>EMP_NO</column-name>
          <column-name>LAST_NAME</column-name>
          <column-name>PROJ_ID</column-name>
  </column-list>
 </left-table>
 <cross-table>
  <table-name>EMPLOYEE_PROJECTS</table-name>
  <column-list>
   <column-name>EMP_NAME</column-name>
       <column-name>PROJ_ID</column-name>
  </column-list>
  <column-list>
   <column-name>PROJ_ID</column-name>
         <column-name>PROJ_NAME</column-name>
  </column-list>
 </cross-table>
 <right-table>
  <table-name>PROJECT</table-name>
  <column-list>
   <column-name>PROJ_ID</column-name>
          <column-name>PROJ_NAME</column-name>
           <column-name>EMP_NO</column-name>
  </column-list>
 </right-table>
</table-ref>
```

関連要素

- <table-ref>
- <left-table>
- <right-table>
- <table-name>
- <column-list>
- <column-name>

<database-map>

```
<!ELEMENT database-map (table?, column-map*)>
```

<database-map> 要素は、エンティティ Bean のフィールドにデータを格納したり、検索メソッドを実行するために CMP 1.x エンジンが必要とするデータソース情報を提供するために使用されます。その子ノードで、使用するデータベース テーブルを指定します。また、エンティティ Bean が自分のフィールドにデータを格納するために使用するフィールドと列も指定します。

サンプル

```
<database-map>
    <table>Courses</table>
    <column-map>
      <field-name>students</field-name>
      <ejb-ref-name>ejb/Student.findByCourse</ejb-ref-name>
    </column-map>
</database-map>
```

関連要素

- <table>
- <column-map>

<datasource-definitions>

```
<!ELEMENT datasource-definitions (datasource*)>
```

古い形式の JDBC 1.x データソースを使用している場合に、この要素は、アーカイブ内の Bean が使用するデータソースに関する情報を提供するために使用されます。各データソースは、個別の `<datasource>` 要素内で定義されます。これは、`<datasource-definitions>` の子ノードです。通常は、新しい形式の `jndi-definitions.xml` ファイルを使ってデータソースを定義します。この要素は JDBC 1.x のユーザー専用です。

関連要素

- <datasource>

<datasource>

```
<!ELEMENT datasource (jndi-name, url, username?, password?,
  isolation-level?, driver-class-name?, jdbc-property*, property*)>
```

この要素は、JDBC 1.x データソースを記述するために使用されます。通常は、新しい形式の `jndi-definitions.xml` ファイルを使ってデータソースを定義します。この要素とその子ノードは JDBC 1.x にだけ適用されます。

関連要素

- <datasource-definitions>
- <jndi-name>
- <url>
- <username>
- <password>
- <isolation-level>
- <driver-class-name>
- <jdbc-property>
- <property>

<deployment-role>

```
<!ELEMENT deployment-role (#PCDATA)>
```

アーカイブ内のモジュールによって使用されるデプロイメント ロールのロール名。

サンプル

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

関連要素

- <security-role>

<description>

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

このオプションの要素を使用して、親ノードの説明を記述します。

サンプル

```
<description>sorting bean</description>
```

<driver-class-name>

```
<!ELEMENT driver-class-name (#PCDATA)>
```

JDBC 1.x 専用。定義されるデータソースにアクセスするために使用されるドライバのクラス名。

関連要素

- <datasource>

<ejb-jar>

```
<!ELEMENT ejb-jar (enterprise-beans, datasource-definitions?,
 table-properties*, relationships?, authorization-domain?,
 property*, assembly-descriptor?)>
```

`<ejb-jar>` 要素は、`ejb-borland.xml` のルート ノードです。その子ノードは、JAR によってデプロイされるエンタープライズ Bean を定義し、Bean 間の関係に関する情報を提供します。また、Bean が使用するデータソース（データベース、JMS プロバイダなど）に関する情報のほか、アーカイブ プロパティやセキュリティ関連情報を指定することもできます。

サンプル

```
<ejb-jar>
   <enterprise-beans>
      <session>
         <ejb-name>AsyncSenderEJB</ejb-name>
         <bean-local-home-name>ejb/local/petstore/asyncsender/AsyncSender</
bean-local-home-name>
         <timeout>0</timeout>
         <resource-ref>
            <res-ref-name>jms/queue/QueueConnectionFactory</res-ref-name>
            <jndi-name>serial://jms/xaqcf</jndi-name>
         </resource-ref>
         <resource-env-ref>
```

```
            <resource-env-ref-name>jms/queue/AsyncSenderQueue</resource-env-
ref-name>
            <jndi-name>serial://jms/queue/opc/OrderQueue</jndi-name>
          </resource-env-ref>
        </session>
    </enterprise-beans>
    <assembly-descriptor/>
</ejb-jar>
```

関連要素

- <enterprise-beans>
- <datasource-definitions>
- <table-properties>
- <relationships>
- <authorization-domain>
- <property>
- <assembly-descriptor>

<ejb-name>

```
<!ELEMENT ejb-name (#PCDATA)>
```

<ejb-name> 要素を使用して、定義するエンタープライズ JavaBeans の名前を指定します。この要素は、ejb-jar.xml 内の同じ要素と同様に、リモートからの Bean のルックアップに使用する名前を指定します。

サンプル

```
<ejb-name>clerk</ejb-name>
```

関連要素

- <session>
- <entity>
- <message-driven>
- <relationship-role-source>

<ejb-ref>

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

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

サンプル

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

関連要素

- <ejb-ref-name>
- <jndi-name>

<ejb-ref-name>

<!ELEMENT ejb-ref-name (#PCDATA)>

この要素は、Bean によってリソース リファレンスとして使用される EJB の名前を提供します。

サンプル

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

関連要素

- <ejb-ref>
- <column-map>

<ejb-relation>

<!ELEMENT ejb-relation (ejb-relationship-role, ejb-relationship-role)>

この要素は、2 つのエンティティの関係を表します。各エンティティ Bean からもう一方への Bean への関係は、それぞれ個別に子ノード `<ejb-relationship-role>` を使用して定義されます。これは一方向の関係の場合でも該当します。

サンプル

これらの関係は、以下の例のように、さまざまな形式になります。

- 単一方向の 1 対 1 の関係
- 双方向の 1 対多の関係

単一方向の 1 対 1 の関係

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

```
 </ejb-relationship-role>
 <ejb-relationship-role>
  <relationship-role-source>
   <ejb-name>SpecialInfo</ejb-name>
  </relationship-role-source>
 </ejb-relationship-role>
</ejb-relation>
</relationships>
```

この関係は単一方向なので、`SpecialInfo` Bean に対してテーブル情報を指定する必要はありません。

双方向の 1 対多の関係

```
<relationships>
 <ejb-relation>
  <ejb-relationship-role>
   <relationship-role-source>
    <ejb-name>Customer</ejb-name>
   </relationship-role-source>
   <cmr-field>
    <cmr-field-name>orders</cmr-field-name>
       <table-ref>
          <left-table>
             <table-name>CUSTOMER</table-name>
             <column-list>
                <column-name>CUSTOMER_NO</column-name>
             </column-list>
          </left-table>
          <right-table>
             <table-name>ORDER</table-name>
             <column-list>
                <column-name>CUSTOMER_NO</column-name>
             </column-list>
          </right-table>
       </table-ref>
   </cmr-field>
  </ejb-relationship-role>
  <ejb-relationship-role>
   <relationship-role-source>
    <ejb-name>Customer</ejb-name>
   </relationship-role-source>
   <cmr-field>
    <cmr-field-name>customers</cmr-field-name>
       <table-ref>
          <left-table>
             <table-name>ORDER</table-name>
             <column-list>
                <column-name>CUSTOMER_NO</column-name>
             </column-list>
          </left-table>
          <right-table>
             <table-name>CUSTOMER</table-name>
             <column-list>
                <column-name>CUSTOMER_NO</column-name>
             </column-list>
          </right-table>
       </table-ref>
   </cmr-field>
  </ejb-relationship-role>
 </ejb-relation>
 .
```

.
</relationships>

このテーブルは両方向にリンクされるため、各方向にテーブル データが提供されます。

関連要素

- <relationships>
- <ejb-relationship-role>

<ejb-relationship-role>

<!ELEMENT ejb-relationship-role (relationship-role-source, cmr-field?)>

1 つのエンティティを定義し、別のエンティティとの関係も定義します。エンティティ自体は子ノード `<relationship-role-source>` を使って提供され、関係のもう一方のエンティティと共有するフィールドは、子ノード `<cmr-field>` で定義されます。

サンプル

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

関連要素

- <ejb-relation>
- <relationship-role-source>
- <cmr-field>

<enterprise-beans>

<!ELEMENT enterprise-beans (session | entity | message-driven)+>

`<enterprise-beans>` 要素を使用して、アーカイブ内の Java Bean を定義します。3 種類のエンタープライズ Java Bean（セッション Bean、エンティティ Bean、メッセージ駆動型 Bean）のそれぞれに対応する子ノードがあり、ここでこれらの Bean に関する情報を提供します。各 Bean のタイプに基づいて、アーカイブ内の Bean ごとにエントリを作成します。

サンプル

```
<enterprise-beans>
    <session>
      <ejb-name>UniqueIdGeneratorEJB</ejb-name>
      ...
    </session>
    <entity>
      <ejb-name>CounterEJB</ejb-name>
      ...
    </entity>
</enterprise-beans>
```

関連要素

- <ejb-jar>
- <session>
- <entity>
- <message-driven>

<entity>

```
<!ELEMENT entity (ejb-name, bean-home-name?, bean-local-home-name?,
 ejb-ref*, ejb-local-ref*, resource-ref*, resource-env-ref*,
 (cmp-info | cmp2-info)?, property*)>
```

<entity> 要素は、アーカイブに含まれるエンティティ Bean に関する情報を提供します。この要素の子ノードを使用して、Bean のさまざまなインターフェイスとリファレンスを指定できます。コンテナ管理の永続性情報のほか、各エンティティ Bean に固有の汎用プロパティも提供できます。

サンプル

```
<entity>
    <ejb-name>claim</ejb-name>
    <bean-local-home-name>Claim</bean-local-home-name>
    <cmp2-info>
      <cmp-field>
        <field-name>claimId</field-name>
        <column-name>CLAIM_ID</column-name>
      </cmp-field>
      <cmp-field>
        <field-name>policyHolderNumber</field-name>
        <column-name>POLICYHOLDER_NUMBER</column-name>
      ...
      <table-name>CLAIMS</table-name>
    </cmp2-info>
</entity>
```

関連要素

- <enterprise-beans>
- <ejb-name>
- <bean-home-name>
- <bean-local-home-name>
- <ejb-ref>
- <resource-ref>
- <resource-env-ref>

- <cmp-info>
- <cmp2-info>
- <property>

<field-name>

<!ELEMENT field-name (#PCDATA)>

基底のデータソース内の列にマップされるエンティティ Bean フィールドの名前。

サンプル

```
<field-name>students</field-name>
```

関連要素

- <database-map>
- <column-name>
- <column-type>
- <ejb-ref-name>
- <cmp-field-map>
- <cmp-field>

<finder>

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

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

サンプル

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

関連要素

- <cmp-info>
- <method-signature>
- <where-clause>
- <load-state>

<init-size>

<!ELEMENT init-size (#PCDATA)>

プールが最初に作成されたときに、BES がプールを満たすために使用する接続の数です。これは `ejb.mdb.init-size` プロパティと同じです。

サンプル

<pool> <max-size>0</max-size> <init-size>0</init-size> <wait-timeout>0</wait-timeout> </pool>

関連要素

- <message-driven>
- <pool>
- <max-size>
- <wait-timeout>

<isolation-level>

<!ELEMENT isolation-level (#PCDATA)>

JDBC 1.x 専用。定義されるデータソースの分離レベル。次の値の 1 つを指定できます。

- TRANSACTION_NONE
- TRANSACTION_READ_COMMITTED
- TRANSACTION_READ_UNCOMMITTED
- TRANSACTION_REPREATABLE_READ
- TRANSACTION_SERIALIZEABLE

関連要素

- <datasource>

<jdbc-property>

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

JDBC 1.x 専用。定義されるデータソースとともに使用される JDBC プロパティを指定します。子ノード `<prop-name>` と `<prop-value>` に対して、それぞれプロパティ名とその値を指定します。

関連要素

- <datasource>
- <prop-name>
- <prop-value>

<jndi-name>

<!ELEMENT jndi-name (#PCDATA)>

この要素は、Bean によって参照されるリソースの、JNDI サービス ルックアップ名を提供します。

サンプル

<jndi-name>sort</jndi-name>

関連要素

- <ejb-ref>
- <resource-ref>
- <resource-env-ref>

<left-table>

<!ELEMENT left-table (table-name, column-list)>

`<cmp2-info>` を指定する場合、この要素は、1 つの列を共有する 2 つのテーブルの一方を定義します。一方がもう一方の外部キーになります。

この要素を使って `<relationships>` を記述する場合、`<left-table>` が関係の元、`<right-table>` が関係の先となります。つまり、関係の方向は左から右です。双方向の関係を定義する場合は、各方向ごとに 1 つ、つまり 1 つの関係に対して 2 つの `<table-ref>` 要素を作成する必要があります。多対多の関係では、`<left-table>` は、共通部分を定義する `<cross-table>` を作成するために使用される 2 つのテーブルの一方を作成します。

サンプル

```
<left-table>
    <table-name>CUSTOMER</table-name>
    <column-list>CUSTOMER_NO</column-list>
</left-table>
```

関連要素

- <table-ref>
- <right-table>
- <cross-table>
- <table-name>
- <column-list>

<load-state>

<!ELEMENT load-state (#PCDATA)>

`ejbCreate()` メソッドの呼び出し時に、コンテナにエンティティ Bean の現在の状態をロードする場合は、このフラグを `true` に設定します。

サンプル

```
<load-state>false</load-state>
```

関連要素

- <finder>
- <method-signature>
- <where-clause>

<max-size>

<!ELEMENT max-size (#PCDATA)>

これは、接続プールで許容される最大接続数です。これは `ejb.mdb.max-size` プロパティと同じです。

サンプル

<pool> <max-size>0</max-size> <init-size>0</init-size> <wait-timeout>0</wait-timeout> </pool>

関連要素

- <message-driven>
- <pool>
- <init-size>
- <wait-timeout>

<message-driven-destination-name>

```
<!ELEMENT message-driven-destination-name (#PCDATA)>
```

Bean がサブスクライブする個別のキューまたはトピックの JNDI 名を指定します。

サンプル

```
<message-driven-destination-name>serial://resources/tcf</message-driven-
destination-name>
```

関連要素

- <message-driven>
- <connection-factory-name>

<message-driven>

```
<!ELEMENT message-driven (ejb-name, message-driven-destination-name,
 connection-factory-name, pool?, ejb-ref*, ejb-local-ref*,
 resource-ref*, resource-env-ref*, property*)>
```

アーカイブにデプロイされるメッセージ駆動型 Bean を記述します。この要素の子ノードを使用して、Bean のさまざまなインターフェイスとリファレンスを指定できます。Bean の接続先のキューやトピック、および接続に使用する接続ファクトリに関するデータを提供することもできます。プールでの Bean の動作を指定することもできます。

サンプル

```
<message-driven>
   <ejb-name>MsgBean</ejb-name>
   <message-driven-destination-name>serial://jms/queue/supplier/
PurchaseOrderQueue</message-driven-destination-name>
   <connection-factory-name>serial://jms/xaqcf</connection-factory-name>
   <pool>
        <max-size>0</max-size>
        <init-size>0</init-size>
        <wait-timeout>0</wait-timeout>
   </pool>
</message-driven>
```

関連要素

- <ejb-name>
- <message-driven-destination-name>
- <connection-factory-name>
- <pool>
- <ejb-ref>
- <resource-ref>
- <resource-env-ref>
- <property>

<method-signature>

```
<!ELEMENT method-signature (#PCDATA)>
```

データベース クエリの実行に使用されるメソッド。エンティティ Bean に示されるとおりです。

サンプル

```
<method-signature>findByStudent(Student s)</method-signature>
```

関連要素

- <finder>
- <where-clause>
- <load-state>

<password>

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

JDBC 1.x 専用。定義されるデータソースにアクセスするためのパスワード。

関連要素

- <datasource>

<pool>

```
<!ELEMENT pool (max-size?, init-size?, wait-timeout?)>
```

MDB のためのリソース プール プロパティを指定する際に使用する子ノードを保持します。

サンプル

```
<pool>
    <max-size>0</max-size>
    <init-size>0</init-size>
    <wait-timeout>0</wait-timeout>
</pool>
```

関連要素

- <message-driven>
- <max-size>
- <init-size>
- <wait-timeout>

<prop-name>

```
<!ELEMENT prop-name (#PCDATA)>
```

設定するプロパティの名前を指定します。

サンプル

```
<prop-name>vbroker.security.disable</prop-name>
```

関連要素

- <property>

<prop-type>

```
<!ELEMENT prop-type (#PCDATA)>
```

設定するプロパティの型を指定します。

サンプル

```
<prop-type>security</prop-type>
```

関連要素

- <property>

<prop-value>

```
<!ELEMENT prop-value (#PCDATA)>
```

設定するプロパティの値を指定します。

サンプル

```
<prop-value>false</prop-value>
```

関連要素

- <property>

<property>

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

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

サンプル

```
<property>
   <prop-name>vbroker.security.disable</prop-name>
   <prop-type>security</prop-type>
   <prop-value>false</prop-value>
</property>
```

関連要素

- <prop-name>
- <prop-type>
- <prop-value>

<relationship-role-source>

<!ELEMENT relationship-role-source (ejb-name)>

別のエンティティ Bean とコンテナ管理の関係にあるエンティティ Bean の名前を指定します。

サンプル

```
<relationship-role-source>
   <ejb-name>Customer</ejb-name>
</relationship-role-source>
```

関連要素

- <ejb-relationship-role>
- <ejb-name>

<relationships>

<!ELEMENT relationships (ejb-relation+)>

テーブル関係を指定するには、`<relationships>` 要素を使用します。`<relationships>` 要素内で、ロールのソース（エンティティ Bean）を保持する `<ejb-relationship-role>` 要素と、関係を保持する `<cmr-field>` 要素を定義します。その後、ディスクリプタは <table-ref> 要素を使用して 2 つのテーブル `<left-table>` と `<right-table>` の間の関係を指定します。その際、次の原則を守る必要があります：

- 一方向につき `<ejb-relationship-role>` 要素 1 つを定義する必要があります。双方向関係の場合、お互いに参照している Bean それぞれに対して、`<ejb-relationship-role>` を定義する必要があります。
- 関係 1 つにつき、使用できる `<table-ref>` 要素は 1 つだけです。

多対多の関係を定義する場合、CMP エンジンに、左右テーブルの関係をモデル化する、クロステーブルを作成させなければなりません。これを行う際に、<cross-table> 要素を使用します。

サンプル

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

関連要素

- <ejb-jar>
- <ejb-relation>

<resource-env-ref-name>

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

この要素は、Bean がリソース環境リファレンスにアクセスするために使用する名前を提供します。

サンプル

```
<res-env-ref-name>jdbc/CheckingDataSource</res-ref-name>
```

関連要素

- <resource-env-ref>

<res-ref-name>

```
<!ELEMENT res-ref-name (#PCDATA)>
```

この要素は、Bean がリソース リファレンスにアクセスするために使用する名前を提供します。

サンプル

```
<res-ref-name>jdbc/CheckingDataSource</res-ref-name>
```

関連要素

- <resource-ref>

<resource-env-ref>

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

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

サンプル

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

関連要素

- <resource-env-ref-name>
- <jndi-name>

<resource-ref>

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

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

サンプル

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

関連要素

- <res-ref-name>

- <jndi-name>

<right-table>

<!ELEMENT right-table (table-name, column-list)>

<cmp2-info> を指定する場合、この要素は、1 つの列を共有する 2 つのテーブルの一方を定義します。一方がもう一方の外部キーになります。

この要素を使って <relationships> を記述する場合、<left-table> が関係の元、<right-table> が関係の先となります。つまり、関係の方向は左から右です。双方向の関係を定義する場合は、各方向ごとに 1 つ、つまり 1 つの関係に対して 2 つの <table-ref> 要素を作成する必要があります。多対多の関係では、<right-table> は、共通部分を定義する <cross-table> を作成するために使用される 2 つのテーブルの一方を作成します。

サンプル

```
<right-table>
    <table-name>CUSTOMER</table-name>
    <column-list>CUSTOMER_NO</column-list>
</right-table>
```

関連要素

- <table-ref>
- <left-table>
- <cross-table>
- <table-name>
- <column-list>

<role-name>

<!ELEMENT role-name (#PCDATA)>

アーカイブ内のモジュールによって使用されるセキュリティ ロールのロール名。

サンプル

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

関連要素

- <security-role>

<security-role>

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

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

サンプル

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

関連要素

- <assembly-descriptor>
- <role-name>
- <deployment-role>

<session>

```
<!ELEMENT session (ejb-name, bean-home-name?, bean-local-home-name?,
 timeout?, ejb-ref*, ejb-local-ref*, resource-ref*,resource-env-ref*, property*)>
```

`<session>` 要素は、アーカイブに含まれるセッション Bean に関する情報を提供します。この要素の子ノードを使用して、Bean のさまざまなインターフェイスとリファレンスを指定できます。セッション Bean に固有の属性（`timeout` など）のほか、個別のセッション Bean に固有の汎用プロパティも提供できます。

サンプル

```
<session>
      <ejb-name>UniqueIdGeneratorEJB</ejb-name>
      <bean-local-home-name>ejb/local/petstore/uidgen/UniqueIdGenerator</bean-
local-home-name>
      <timeout>0</timeout>
      <ejb-local-ref>
         <ejb-ref-name>ejb/local/Counter</ejb-ref-name>
      </ejb-local-ref>
</session>
```

関連要素

- <enterprise-beans>
- <ejb-name>
- <bean-home-name>
- <bean-local-home-name>
- <timeout>
- <ejb-ref>
- <resource-ref>
- <resource-env-ref>
- <property>

<table>

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

CMP 1.x エンティティ Bean が自分のフィールドにデータを格納するために使用するデータベース テーブルの名前を指定します。

サンプル

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

関連要素

- <database-map>

<table-name>

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

エンティティ マッピングまたはプロパティ設定のためのテーブルの名前を指定します。

サンプル

```
<table-name>CUSTOMER</table-name>
```

関連要素

- <cmp2-info>
- <table-properties>

<table-properties>

```
<!ELEMENT table-properties (table-name, column-properties*, property*)>
```

この要素を使用して、エンティティのデータベース リソースに関する詳細情報を提供します。その子ノードを使用して、テーブルの名前と、テーブルに関連付けられたプロパティを指定します（`create_tables` など）。`<column-properties>` 子ノードを使用して、データベース列に固有のプロパティを指定することもできます。

サンプル

```
<table-properties>
 <table-name>CUSTOMER</table-name>
  <property>
   <prop-name>create-tables</prop-name>
   <prop-value>True</prop-value>
  </property>
</table-properties>
```

関連要素

- <ejb-jar>
- <table-name>
- <column-properties>
- <property>

<table-ref>

```
<!ELEMENT table-ref (left-table, cross-table*, right-table)>
```

<cmp2-info> を指定する場合は、複数のテーブルに永続化される情報を含むエンティティを設定できます。これらのテーブルは、最低でも 1 つ、リンクされたテーブルでの外部キーを表す列でリンクする必要があります。この関係は、<table-ref> 要素を使用して記述できます。その子ノード <left-table> や <right-table> を使用して、関係を持つテーブル群と、それらが共有する列（複数可）を指定してください。

この要素は、<relationships> 要素を使用して、テーブル間の関係を記述するためにも使用されます。この場合、ディスクリプタでは、<table-ref> を使用して、<left-table> と <right-table> の間の関係を指定します。その際、次の原則を守る必要があります：

- 一方向につき <ejb-relationship-role> 要素 1 つを定義する必要があります。双方向関係の場合、お互いに参照している Bean それぞれに対して、<ejb-relationship-role> を定義する必要があります。
- 関係 1 つにつき、使用できる <table-ref> 要素は 1 つだけです。

多対多の関係を定義する場合、CMP エンジンに、左右テーブルの関係をモデル化する、クロステーブルを作成させなければなりません。これを行う際に、<cross-table> 要素を使用します。

サンプル

```
<table-ref>
  <left-table>
   <table-name>LINE_ITEM</table-name>
   <column-list>
    <column-name>LINE</column-name>
   </column-list>
  </left-table>
  <right-table>
   <table-name>QUANTITY</table-name>
   <column-list>
    <column-name>LINE</column-name>
   </column-list>
  </right-table>
 </table-ref>
```

関連要素

- <cmp2-info>
- <left-table>
- <right-table>
- <cross-table>
- <ejb-relationship-role>

<timeout>

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

セッション Bean が呼び出しを待機してタイムアウトになるまでの時間を秒単位で指定します。デフォルトは 0 秒です。

サンプル

```
<timeout>10</timeout>
```

関連要素

- <session>

<url>

<!ELEMENT url (#PCDATA)>

JDBC 1.x 専用。定義されるデータソースの URL。

関連要素

- <datasource>

<username>

<!ELEMENT username (#PCDATA)>

JDBC 1.x 専用。定義されるデータソースにアクセスするためのユーザー名。

関連要素

- <datasource>

<wait-timeout>

<!ELEMENT wait-timeout (#PCDATA)>

`maxPoolSize` の数の接続がすでにオープンになっている際、利用できる接続が提供されるまで待機する秒数。`maxPoolSize` プロパティを使用しており、プールがこれ以上接続を提供できない場合、待ち時間が無制限に設定されている（0 秒に設定）と、接続を検索するスレッドは、使用できる接続が見つかるまで待機してしまいます。ユーザーは、必要に応じて、この wait-timeout でそれを調整することができます。これは `ejb.mdb.wait_timeout` プロパティと同じです。

サンプル

<pool> <max-size>0</max-size> <init-size>0</init-size> <wait-timeout>0</wait-timeout> </pool>

関連要素

- <message-driven>
- <pool>
- <max-size>
- <init-size>

<where-clause>

<!ELEMENT where-clause (#PCDATA)>

where 節は、取得するレコードの範囲を限定する場合に必要な select 文の一部です。where 節の構文はかなり複雑になることがあり、EJB コンテナがこの節を正しく生成できるように、XML デプロイメント ディスクリプタ ファイルでは、一定の規則にしたがう必要があります。

まず、必ずしも <where-clause> で「where」リテラルを使用する必要はありません。このリテラルのない where 節を生成し、where 節記入はコンテナに任せることができます。ただし、コンテナは、<where-clause> 内が空文字列でない場合にのみ、これを行います。空文字列は空のままになります。たとえば、次のどちらの方法でも同じ where 節を定義できます。

```
<where-clause> where a = b </where-clause>
```

または

```
<where-clause> a = b </where-clause>
```

コンテナは、`a = b` を同義の where 節、`where a = b` に変換します。しかし、`<where-clause> "" </where-clause>` と定義されている空文字列の場合には、変換は行われません。

パラメータ置換は、where 節の重要な部分です。Borland EJB コンテナは、標準の SQL 置換接頭辞であるコロン（:）を見つけると、パラメータ置換を行います。置換される各パラメータは、XML デプロイメント ディスクリプタにある検索メソッド定義中のパラメータの名前に対応します。

コンテナは、複合パラメータもサポートします。複合パラメータは、テーブル名の後に、そのテーブル内の列が付きます。複合パラメータには、標準のドット（.）構文を使用します。この構文では、テーブル名と列名をドットで区切ります。複合パラメータでも、前にコロンを付けます。

エンティティ Bean は、検索メソッドのパラメータとしても使用できます。エンティティ Bean は複合型として使用できます。その場合、SQL クエリに渡すエンティティ Bean リファレンスのどのフィールドを使用するのかを CMP エンジンに指示する必要があります。複合型としてエンティティ Bean を使用しなければ、コンテナは、where 節にその Bean の主キーを代入します。

サンプル

```
<where-clause>SELECT course_dept, course_number FROM
  Enrollment WHERE student  = :s[ejb/Student]</where-clause>
```

関連要素

- <finder>
- <method-signature>
- <load-state>

第 4 章

jndi-definitions.xml

DTD

```
<!ELEMENT jndi-definitions (visitransact-datasource*, driver-datasource*, jndi-object*)>

  <!ELEMENT visitransact-datasource (jndi-name, driver-datasource-jndiname, property*)>

  <!ELEMENT driver-datasource (jndi-name, datasource-class-name, log-writer?, property* )>

  <!ELEMENT jndi-object (jndi-name, class-name, property* )>

   <!ELEMENT property (prop-name, prop-type, prop-value)>
   <!ELEMENT prop-name (#PCDATA)>
   <!ELEMENT prop-type (#PCDATA)>
   <!ELEMENT prop-value (#PCDATA)>
   <!ELEMENT jndi-name (#PCDATA)>
   <!ELEMENT driver-datasource-jndiname (#PCDATA)>
   <!ELEMENT datasource-class-name (#PCDATA)>
   <!ELEMENT log-writer (#PCDATA)>
   <!ELEMENT class-name (#PCDATA)>
```

<jndi-definitions>

```
<!ELEMENT jndi-definitions (visitransact-datasource*, driver-datasource*, jndi-object*)>
```

J2EE リソース接続ファクトリ オブジェクトを JNDI 定義モジュールの一部としてデプロイすると、J2EE リソース接続ファクトリ オブジェクトが JNDI に接続されます。この定義モジュールは、他の J2EE 標準 Java アーカイブ型に似ており、拡張子 .dar で終わります。そのため、このモジュールのことを DAR ともいいます。このモジュールは、JAR、WAR、RAR などの標準 J2EE モジュール タイプに追加されます。このモジュールはまた、EAR の一部としてパッケージする場合と、スタンドアロンでデプロイする場合があります。

提供する必要がある DAR の内容は、jndi-definitions.xml という名前の XML ディスクリプタ ファイルだけです。このファイルには、JNDI 名前空間に連結されるすべてのデータソース定義が含まれます。

`<jndi-definitions>` 要素は、スキーマのルート ノードです。`<visitransact-datasource>` および `<driver-datasource>` 子ノードを使って JDBC 接続ファク

トリ オブジェクトを定義し、<jndi-object> 子ノードを使って JMS リソース接続ファクトリ オブジェクトを定義します。

サンプル

```
<jndi-definitions>
   <visitransact-datasource>
      <jndi-name>serial://datasources/Oracle</jndi-name
      <driver-datasource-jndiname>serial://datasources/OracleDriver</driver-
datasource-jndiname>
      <property>
         <prop-name>connectionType</prop-name>
         <prop-type>Enumerated</prop-type>
         <prop-value>Direct</prop-value>
      </property>
   </visitransact-datasource>
   <driver-datasource>
      <jndi-name>serial://datasources/OracleDriver</jndi-name>
      <datasource-class-
name>oracle.jdbc.pool.OracleConnectionPoolDataSource</datasource-class-
name>
      <property>
         <prop-name>user</prop-name>
         <prop-type>String</prop-type>
         <prop-value>MisterKittles</prop-value>
      </property>
   </driver-datasource>
</jndi-definitions>
```

関連要素

- <visitransact-datasource>
- <driver-datasource>
- <jndi-object>

<class-name>

```
<!ELEMENT class-name (#PCDATA)>
```

JMS サービス プロバイダから提供され、ライブラリとして BES パーティションにデプロイされる接続ファクトリ クラスの名前。

サンプル

```
<class-name>progress.message.jclient.QueueConnectionFactory</class-name>
```

関連要素

- <jndi-object>

<datasource-class-name>

```
<!ELEMENT datasource-class-name (#PCDATA)>
```

リソース ベンダーから提供される接続ファクトリ クラスの名前を指定します。クラス自体は、ライブラリとして BES パーティションにデプロイされます。

サンプル

```
<datasource-class-name>oracle.jdbc.pool.OracleConnectionPoolDataSource</
datasource-class-name>
```

関連要素

- <driver-datasource>

<driver-datasource>

```
<!ELEMENT driver-datasource (jndi-name, datasource-class-name, log-writer?,
property* )>
```

この要素は、`<visitransact-datasource>` で開始されたデータソース定義の残り半分として、ドライバに関する情報を指定します。ここではドライバの JNDI 名を指定します。これは、`defined in <visitransact-datasource>` で定義されたデータソースの `<driver-datasource-jndiname>` と同じである必要があります。データソースドライバのクラス名、ログ動作（該当する場合）、および JDBC リソースに固有のプロパティ（ユーザー名、パスワードなど）も指定します。

サンプル

```
<driver-datasource>
      <jndi-name>serial://datasources/OracleDriver</jndi-name>
      <datasource-class-
name>oracle.jdbc.pool.OracleConnectionPoolDataSource</datasource-class-
name>
      <property>
         <prop-name>user</prop-name>
         <prop-type>String</prop-type>
         <prop-value>MisterKittles</prop-value>
      </property>
</driver-datasource>
```

関連要素

- <jndi-definitions>
- <visitransact-datasource>
- <jndi-name>
- <datasource-class-name>
- <log-writer>
- <property>

<driver-datasource-jndiname>

```
<!ELEMENT driver-datasource-jndiname (#PCDATA)>
```

ユーザーがライブラリとして BES パーティションへデプロイするドライバ クラスの JNDI 名は、データベース ベンダーまたは JMS ベンダーより指定されます。また、JDBC リソース定義の残りの部分を構成する `<driver-datasource>` 要素の子である `<jndi-name>` によって参照される名前でもあります。

サンプル

```
<driver-datasource-jndiname>serial://datasources/OracleDriver</driver-datasource-jndiname>
```

関連要素

- <visitransact-datasource>

<jndi-name>

```
<!ELEMENT jndi-name (#PCDATA)>
```

JNDI が参照するデータソースの名前。エンタープライズ Bean のリソース リファレンスにもある名前です。

サンプル

```
<jndi-name>serial://datasources/Oracle</jndi-name>
```

関連要素

- <visitransact-datasource>

<jndi-object>

```
<!ELEMENT jndi-object (jndi-name, class-name, property* )>
```

You define the `<jndi-object>` 要素を定義して、JNDI に JMS 接続ファクトリを登録します。その子ノードを使用して、JMS 接続を確立するための JNDI 検索、接続ファクトリ クラス、およびそのクラスに渡す必要がある JMS プロバイダに固有のプロパティを提供します。

サンプル

```
<jndi-object>
   <jndi-name>serial://jms/message</jndi-name>
   <class-name>progress.message.jclient.QueueConnectionFactory</class-name>
     <property>
        <prop-name>connectionURLS</prop-name>
        <prop-type>String</prop-type>
        <prop-value>localhost:2506</prop-value>
     </property>
     <property>
        <prop-name>sequential</prop-name>
        <prop-type>Boolean</prop-type>
        <prop-value>false</prop-value>
     </property>
     <property>
        <prop-name>loadBalancing</prop-name>
        <prop-type>Boolean</prop-type>
        <prop-value>true</prop-value>
     </property>
  </jndi-object>
```

関連要素

- <jndi-definitions>
- <jndi-name>
- <class-name>
- <property>

<log-writer>

<!ELEMENT log-writer (#PCDATA)>

この要素を使用して、一部のベンダー接続ファクトリ クラスの詳細モードを有効にできます。このプロパティの使用方法については、リソースのマニュアルを参照してください。

サンプル

```
<log-writer>True</log-writer>
```

関連要素

- <driver-datasource>

<prop-name>

<!ELEMENT prop-name (#PCDATA)>

設定するプロパティの名前を指定します。

サンプル

```
<prop-name>vbroker.security.disable</prop-name>
```

関連要素

- <property>

<prop-type>

<!ELEMENT prop-type (#PCDATA)>

設定するプロパティの型を指定します。

サンプル

```
<prop-type>security</prop-type>
```

関連要素

- <property>

<prop-value>

```
<!ELEMENT prop-value (#PCDATA)>
```

設定するプロパティの値を指定します。

サンプル

```
<prop-value>false</prop-value>
```

関連要素

- <property>

<property>

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

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

サンプル

```
<property>
   <prop-name>vbroker.security.disable</prop-name>
   <prop-type>security</prop-type>
   <prop-value>false</prop-value>
</property>
```

関連要素

- <prop-name>
- <prop-type>
- <prop-value>

<visitransact-datasource>

```
<!ELEMENT visitransact-datasource (jndi-name, driver-datasource-jndiname,
property*)>
```

アプリケーション コードが検索するデータソースを定義します。データソースの JNDI 名、そのドライバ、およびそのドライバに渡す必要があるプロパティを提供します。これを定義したら、`<driver-datasource>` 兄弟要素で、そのドライバに関する情報を提供する必要があります。

サンプル

```
<visitransact-datasource>
  <jndi-name>serial://datasources/Oracle</jndi-name
 <driver-datasource-jndiname>serial://datasources/OracleDriver</driver-
datasource-jndiname>
  <property>
   <prop-name>connectionType</prop-name>
   <prop-type>Enumerated</prop-type>
```

```
    <prop-value>Direct</prop-value>
  </property>
</visitransact-datasource>
```

関連要素

- <jndi-definitions>
- <driver-datasource>
- <jndi-name>
- <driver-datasource-jndiname>
- <property>

第 5 章

ra-borland.xml

DTD

```
<!ELEMENT connector (connection-factory)>
<!ELEMENT connection-factory (factory-name, factory-description?, jndi-name, ra-link-ref?, ra-libraries?, pool-parameters?, (logging-enabled, log-file-name)?, property*, security-map*, authorization-domain?)>
<!ELEMENT factory-name (#PCDATA)>
<!ELEMENT factory-description (#PCDATA)>
<!ELEMENT jndi-name (#PCDATA)>
<!ELEMENT ra-link-ref (#PCDATA)>
<!ELEMENT ra-libraries (#PCDATA)>
<!ELEMENT pool-parameters (initial-capacity?, maximum-capacity?, capacity-delta?, cleanup-enabled?, cleanup-delta?, wait-timeout?, busy-timeout?, idle-timeout?)>
<!ELEMENT initial-capacity (#PCDATA)>
<!ELEMENT maximum-capacity (#PCDATA)>
<!ELEMENT capacity-delta (#PCDATA)>
<!ELEMENT cleanup-enabled (#PCDATA)>
<!ELEMENT cleanup-delta (#PCDATA)>
<!ELEMENT logging-enabled (#PCDATA)>
<!ELEMENT log-file-name (#PCDATA)>
<!ELEMENT property (prop-name, prop-type, prop-value)>
<!ELEMENT prop-name (#PCDATA)>
<!ELEMENT prop-type (#PCDATA)>
<!ELEMENT prop-value (#PCDATA)>
<!ELEMENT security-map (description?, user-role+, (use-caller-identity|run-as))>
<!ELEMENT user-role (#PCDATA)>
<!ELEMENT use-caller-identity EMPTY>
<!ELEMENT run-as (description?, role-name)>
<!ELEMENT role-name (#PCDATA)>
<!ELEMENT authorization-domain (#PCDATA)>
<!ELEMENT description (#PCDATA)>
```

<authorization-domain>

```
<!ELEMENT authorization-domain (#PCDATA)>
```

authorization-domain 要素は、有効なユーザー ロールの定義セットを判断するための認証ドメインを指定します。

関連要素

- <connection-factory>

<busy-timeout>

<!ELEMENT busy-timeout (#PCDATA)>

ビジー接続が解放されるまで待つ時間を秒単位で指定します。指定しない場合は、デフォルト値の `600` が使用されます。

関連要素

- <pool-parameters>

<capacity-delta>

<!ELEMENT capacity-delta (#PCDATA)>

`capacity-delta` 要素は、管理されている接続プールのサイズを変更する際に、VisiConnect が新たに取得を試みる管理接続の数を指定します。指定しない場合は、デフォルト値の `1` が使用されます。

関連要素

- <pool-parameters>

<cleanup-delta>

<!ELEMENT cleanup-delta (#PCDATA)>

The `cleanup-delta` 要素は、接続プール管理が、使用されていない管理接続を回収する間隔を指定します。指定しない場合は、デフォルト値の `1` が使用されます。

関連要素

- <pool-parameters>

<cleanup-enabled>

<!ELEMENT cleanup-enabled (#PCDATA)>

`cleanup-enabled` 要素は、システム リソース管理の一環として、接続プールが使用されていない管理接続を回収するかどうかを指定します。使用されていない接続を回収するには、デフォルト値の `true` を使用してください。

関連要素

- <pool-parameters>

<connection-factory>

<!ELEMENT connection-factory (factory-name, factory-description?, jndi-name, ra-link-ref?, ra-libraries?, pool-parameters?, (logging-enabled, log-file-name)?, property*, security-map*, authorization-domain?)>

connection-factory 要素は、デプロイされるリソース アダプタに相当する、Borland 固有のデプロイメント ディスクリプタのルート要素です。

関連要素

- <connection-factory>
- <factory-description>
- <jndi-name>
- <ra-link-ref>
- <ra-libraries>
- <pool-parameters>
- <logging-enabled>
- <log-file-name>
- <property>
- <security-map>
- <authorization-domain>

<connector>

<!ELEMENT connector (connection-factory)>

`ra-borland.xml` スキーマのルート ノードです。各コネクタは、JCA リソースに接続するために使用する `connection-factory` を定義します。

関連要素

<connection-factory>

<description>

<!ELEMENT description (#PCDATA)>

セキュリティ マップまたは run-as ロールの説明を指定します。

関連要素

- <security-map>

<factory-description>

<!ELEMENT factory-description (#PCDATA)>

`factory-description` 要素は、親要素を記述するテキストを提供する際に使用されます。The `factory-description` 要素には、デプロイヤがデプロイする接続ファクトリについて記述したい情報を自由に含めることができます。

関連要素

- <connection-factory>

<factory-name>

<!ELEMENT factory-name (#PCDATA)>

`factory-name` 要素では、リソース アダプタのこの特定のデプロイメントとそれに相当する接続ファクトリに関連づけられる、論理名を定義します。

`factory-name` の値は、`ra-link-ref` 要素を使用することで、別のデプロイされたリソース アダプタで使用することができます。これにより、複数のデプロイ済みの接続ファクトリが、共通のデプロイ済みのリソース アダプタを利用するだけでなく、その設定使用も共有することができるようになります。

関連要素

- <connection-factory>

<idle-timeout>

<!ELEMENT idle-timeout (#PCDATA)>

プールされた接続がアイドル状態のまま閉じられるまでの時間を秒単位で指定します。指定しない場合は、デフォルト値の `600` が使用されます。アイドル状態の接続は、状態が変化していないかどうかが 60 秒ごとにチェックされます。

関連要素

- <pool-parameters>

<initial-capacity>

<!ELEMENT initial-capacity (#PCDATA)>

initial-capaciry 要素は、デプロイメントの際に、VisiConnect が取得を試みる管理接続の初期数を指定します。指定しない場合は、デフォルトの値 `1` が使用されます。

関連要素

- <pool-parameters>

<jndi-name>

<!ELEMENT jndi-name (#PCDATA)>

`jndi-name` 要素は、接続ファクトリ オブジェクトを JNDI 名前空間にバインドするために使用する名前を定義します。クライアントである EJB やサーブレットは、それぞれ定義される Borland 固有のデプロイメント ディスクリプタの参照要素で、同じ JNDI を使用します。

関連要素

- <connection-factory>

<log-file-name>

<!ELEMENT log-file-name (#PCDATA)>

`log-file-name` 要素は、`ManagedConnectionFactory` または `ManagedConnection` から生成された出力が送られるログ ファイルを指定します。ファイル名の完全なアドレスが必要になります。

関連要素

- <connection-factory>

<logging-enabled>

<!ELEMENT logging-enabled (#PCDATA)>

logging-enabled 要素では、`ManagedConnectionFactory` または `ManagedConnection` に対して、ログ ライタが設定されるかどうかを指定します。この要素が `true` に設定された場合、`ManagedConnectionFactory` または `ManagedConnection` で生成された出力は、`log-filename` 要素で指定されたファイルへ送られます。指定しない場合には、デフォルト値の `false` が使用されます。

関連要素

- <connection-factory>

<maximum-capacity>

<!ELEMENT maximum-capacity (#PCDATA)>

`maximum-capacity` 要素は、VisiConnect が許容する管理接続の最大数を指定します。この制限を越えて、管理接続の割り当てが要求されると、呼び出し元に、`ResourceAllocationException` が返されます。指定しない場合は、デフォルト値の `10` が使用されます。

関連要素

- <pool-parameters>

<pool-parameters>

<!ELEMENT pool-parameters (initial-capacity?, maximum-capacity?, capacity-delta?, cleanup-enabled?, cleanup-delta?, wait-timeout?, busy-timeout?, idle-timeout?)>

pool-parameters 要素は、この接続ファクトリの接続プール固有のパラメータを提供するためのルート要素です。

VisiConnect は、これらの指定を使用して、管理接続が保持されているプールの動作を制御します。

この要素は省略可能です。この要素またはこの要素固有の項目の指定に失敗している場合には、デフォルト値が割り当てられます。あらかじめ設定されているデフォルト値については、各要素の説明を参照してください。

関連要素

- <connection-factory>
- <initial-capacity>
- <maximum-capacity>
- <capacity-delta>
- <cleanup-enabled>
- <cleanup-delta>
- <wait-timeout>
- <busy-timeout>
- <idle-timeout>

<prop-name>

<!ELEMENT prop-name (#PCDATA)>

設定するプロパティの名前を指定します。

サンプル

<prop-name>vbroker.security.disable</prop-name>

関連要素

- <property>

<prop-type>

<!ELEMENT prop-type (#PCDATA)>

設定するプロパティの型を指定します。

サンプル

<prop-type>security</prop-type>

関連要素

- <property>

<prop-value>

<!ELEMENT prop-value (#PCDATA)>

設定するプロパティの値を指定します。

サンプル

<prop-value>false</prop-value>

関連要素

- <property>

<property>

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

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

サンプル

```
<property>
   <prop-name>vbroker.security.disable</prop-name>
   <prop-type>security</prop-type>
   <prop-value>false</prop-value>
</property>
```

関連要素

- <prop-name>
- <prop-type>
- <prop-value>

<ra-libraries>

<!ELEMENT ra-libraries (#PCDATA)>

ra-libraries 要素は、このリソース アダプタ デプロイメント内に存在するネイティブ ライブラリのために使用されるディレクトリの場所を指定します。デプロイメント プロセスの一部として、検知されたネイティブ ライブラリは、すべて指定された場所にコピーされます。

必要なプラットフォーム アクションを実行して、実行時にこれらのライブラリが検出されるようにするのは、管理者の役割です。

関連要素

- <connection-factory>

<ra-link-ref>

<!ELEMENT ra-link-ref (#PCDATA)

ra-link-ref 要素を使用すると、デプロイ済み複数の接続ファクトリを、デプロイ済みの 1 つのリソース アダプタに論理的に関連付けることができます。このオプションの `ra-link-ref` 要素を、デプロイ済みの個々の接続ファクトリを識別できる値と共に指定すると、その新たにデプロイされる接続ファクトリは、参照されている接続ファクトリと共にすでにデプロイされているリソース アダプタを、共有するようになります。

また、リファレンス先の接続ファクトリのデプロイメントで定義されている値は、特に指定されない限り、この新しくデプロイされた接続ファクトリによって継承されます。

関連要素

- <connection-factory>

<role-name>

<!ELEMENT role-name (#PCDATA)>

role-name 要素には、セキュリティ ロールの名前を指定します。この名前は、NMTOKEN の語彙規則に従う必要があります。

関連要素

- <run-as>

<run-as>

<!ELEMENT run-as (description?, role-name)>

run-as 要素では、エンタープライズ Bean を実行するために使用する run-as ID を指定します。この要素には、オプションの説明とセキュリティ ロールの名前を含めます。

関連要素

- <security-map>
- <description>
- <role-name>

<security-map>

<!ELEMENT security-map (description?, user-role+, (use-caller-identity|run-as))>

security-map 要素は、エンタープライズ Bean のメソッドを実行する際に呼び出し元のセキュリティ ID を使用するかどうか、または特定の run-as ID を使用するかどうかを指定します。この要素には、使用するセキュリティ ID のオプションの説明と指定を指定します。

各 security-map 要素は、run-as 要素を使用した、リソース アダプタ /EIS 認証処理のための適切なリソース ロールを定義するメカニズムを提供します。

この要素では、管理接続と接続ハンドルを割り当てる際に使用する定義済みのユーザー ロールのセットと、それに対応する run-as ロール（EIS ID を表す）を指定できます。

マップを使用して、デフォルトのリソース run-as ロールを接続ファクトリに対して定義できます。user-role 値 * と、それに対応する run-as ロールを指定すると、ロールがマップ内の他の場所で一致しない場合は、定義済みの run-as が使用されます。

この要素は省略可能ですが、コンテナ管理のサインオンがリソース アダプタによってサポートされ、いずれかのクライアントで使用されている場合は、なんらかの形で指定する必要があります。また、管理接続を持つ接続プールは、定義済みのデフォルトの run-as ロールが指定されている場合、そのデフォルトを使ってデプロイメント時に生成が試みられます。

関連要素

- <connection-factory>
- <description>
- <user-role>

- <use-caller-identity>
- <run-as>

<use-caller-identity>

<ELEMENT use-caller-identity EMPTY>

`use-caller-identity` 要素は、リソース アダプタのメソッドを実行するためのセキュリティ ID として使用する呼び出し元のセキュリティ ID を指定します。これは空の要素です。使用しない場合は、代わりに `run-as` 要素を指定する必要があります。

関連要素

- <security-map>

<user-role>

<!ELEMENT user-role (#PCDATA)>

`user-role` 要素には、リソースとの通信のために使用される、1 つ以上のロール名を指定します。これは、セキュリティ ID としてそのまま使用されるために定義されるか、もしくは、適切なリソース ロール run-as ID にマップされます。

関連要素

- <security-map>

<wait-timeout>

<!ELEMENT wait-timeout (#PCDATA)>

最大数の接続がすでにオープンになっている際、利用できる接続が提供されるまで待機する秒数を指定します。待ち時間を無制限に設定するには、この要素に値 `0` を設定します。指定しない場合は、デフォルト値の `30` が使用されます。

関連要素

- <pool-parameters>

第 6 章

web-borland.xml

DTD

```
<!ELEMENT web-app (context-root?, resource-env-ref*, resource-ref*, ejb-ref*,
  property*, web-deploy-path*, authorization-domain?, security-role*)>

  <!ELEMENT ejb-ref (ejb-ref-name, jndi-name?)>
  <!ELEMENT resource-ref (res-ref-name, jndi-name)>
  <!ELEMENT resource-env-ref (resource-env-ref-name, jndi-name)>
  <!ELEMENT property (prop-name, prop-type, prop-value)>
  <!ELEMENT web-deploy-path (service, engine, host)>
  <!ELEMENT context-root (#PCDATA)>
  <!ELEMENT prop-name (#PCDATA)>
  <!ELEMENT prop-type (#PCDATA)>
  <!ELEMENT prop-value (#PCDATA)>
  <!ELEMENT ejb-ref-name (#PCDATA)>
  <!ELEMENT jndi-name (#PCDATA)>
  <!ELEMENT res-ref-name (#PCDATA)>
  <!ELEMENT resource-env-ref-name (#PCDATA)>
  <!ELEMENT service (#PCDATA)>
  <!ELEMENT engine (#PCDATA)>
  <!ELEMENT host (#PCDATA)>
  <!ELEMENT authorization-domain (#PCDATA)>
  <!ELEMENT security-role (role-name, deployment-role?)>
  <!ELEMENT role-name (#PCDATA)>
  <!ELEMENT deployment-role (#PCDATA)>
```

<web-app>

```
<!ELEMENT web-app (context-root?, resource-env-ref*, resource-ref*, ejb-ref*,
  property*, web-deploy-path*, authorization-domain?, security-role*)>
```

Web アプリケーションのデプロイメント ディスクリプタのルート ノード。このディスクリプタは、`web.xml` 標準デプロイメント ディスクリプタを拡張して、追加のプロパティを提供したり、Web アプリケーションがホストされる場所を正確に指定したり、アプリケーションのセキュリティ情報を提供できるようにします。

サンプルサンプル

```
<web-app>
   <authorization-domain>default</authorization-domain>
</web-app>
```

関連要素

- <context-root>
- <resource-env-ref>
- <resource-ref>
- <ejb-ref>
- <property>
- <web-deploy-path>
- <authorization-domain>
- <security-role>

<authorization-domain>

```
<!ELEMENT authorization-domain (#PCDATA)>
```

アプリケーションが属する認証ドメインの名前。

サンプル

```
<authorization-domain>GroupJ</authorization-domain>
```

関連要素

- <web-app>

<context-root>

```
<!ELEMENT context-root (#PCDATA)>
```

通常、Web アプリケーションの名前は WAR の名前と同じです（`.war` 拡張子は除く）。`<context-root>` 構造を使用して、アプリケーションの名前を任意の名前に変更できます。

サンプル

```
<context-root>alienWare</context-root>
```

関連要素

- <web-app>

<deployment-role>

```
<!ELEMENT deployment-role (#PCDATA)>
```

Web アプリケーションを実行する BES ロールのロール名。

サンプル

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

関連要素

- <security-role>

<ejb-name>

```
<!ELEMENT ejb-name (#PCDATA)>
```

<ejb-name> 要素を使用して、定義するエンタープライズ JavaBeans の名前を指定します。この要素は、`ejb-jar.xml` 内の同じ要素と同様に、リモートからの Bean のルックアップに使用する名前を指定します。

サンプル

```
<ejb-name>clerk</ejb-name>
```

関連要素

- <ejb-ref>

<ejb-ref>

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

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

サンプル

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

関連要素

- <web-app>
- <ejb-ref-name>
- <jndi-name>

<ejb-ref-name>

```
<!ELEMENT ejb-ref-name (#PCDATA)>
```

この要素は、Web アプリケーションによってリソース リファレンスとして使用される EJB の名前を提供します。

サンプル

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

関連要素

- <ejb-ref>

<engine>

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

エンジン名を設定します。これは、Tomcat の `server.xml` ファイルで定義されたエンジンに対応している必要があります。

サンプル

```
<engine>cyrpi</engine>
```

関連要素

- <web-app>
- <service>
- <host>

<host>

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

ホスト名を設定します。これは、Tomcat の `server.xml` ファイルで定義されたホストに対応している必要があります。

サンプル

```
<host>it3</host>
```

関連要素

- <web-app>
- <service>
- <engine>

<jndi-name>

```
<!ELEMENT jndi-name (#PCDATA)>
```

この要素は、Web アプリケーションによって参照されるリソースの、JNDI サービス ルックアップ名を提供します。

サンプル

```
<jndi-name>sort</jndi-name>
```

関連要素

- <ejb-ref>
- <resource-ref>
- <resource-env-ref>

<prop-name>

<!ELEMENT prop-name (#PCDATA)>

設定するプロパティの名前を指定します。

サンプル

```
<prop-name>vbroker.security.disable</prop-name>
```

関連要素

- <property>

<prop-type>

<!ELEMENT prop-type (#PCDATA)>

設定するプロパティの型を指定します。

サンプル

```
<prop-type>security</prop-type>
```

関連要素

- <property>

<prop-value>

<!ELEMENT prop-value (#PCDATA)>

設定するプロパティの値を指定します。

サンプル

```
<prop-value>false</prop-value>
```

関連要素

- <property>

<property>

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

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

サンプル

```
<property>
   <prop-name>vbroker.security.disable</prop-name>
   <prop-type>security</prop-type>
   <prop-value>false</prop-value>
</property>
```

関連要素

- <prop-name>
- <prop-type>
- <prop-value>

<resource-env-ref-name>

```
<!ELEMENT resource-env-ref-name (#PCDATA)>
```

この要素は、Web アプリケーションがリソース環境リファレンスにアクセスするために使用する名前を提供します。

サンプル

```
<res-env-ref-name>jdbc/CheckingDataSource</res-ref-name>
```

関連要素

- <web-app>
- <resource-env-ref>

<res-ref-name>

```
<!ELEMENT res-ref-name (#PCDATA)>
```

この要素は、Web アプリケーションがリソース リファレンスにアクセスするために使用する名前を提供します。

サンプル

```
<res-ref-name>jdbc/CheckingDataSource</res-ref-name>
```

関連要素

- <resource-ref>

<resource-env-ref>

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

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

サンプル

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

関連要素

- <web-app>
- <resource-env-ref-name>
- <jndi-name>

<resource-ref>

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

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

サンプル

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

関連要素

- <web-app>
- <res-ref-name>
- <jndi-name>

<role-name>

```
<!ELEMENT role-name (#PCDATA)>
```

Web アプリケーションで使用される security-role のロール名。これは、BES のデプロイ環境のロールにマップされます。

サンプル

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

関連要素

- <security-role>

<security-role>

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

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

サンプル

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

関連要素

- <web-app>
- <role-name>
- <deployment-role>

<service>

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

サービス名を設定します。これは、Tomcat の `server.xml` ファイルで定義されたサービスに対応している必要があります。

サンプル

```
<service>tomcatX</service>
```

関連要素

- <web-app>
- <engine>
- <host>

<web-deploy-path>

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

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

サンプル

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

関連要素

- <web-app>
- <service>
- <engine>
- <host>

索引

記号

A

B

D

E

J

R

W

エ

カ

キ

コ

サ

セ

ソ

テ

ト

メ

リ

レ