BEA WebLogic Server™

WebLogic J2EE
コネクタ アーキテクチャ

BEA WebLogic Server 6.1
マニュアルの日付：2002 年 6 月 24 日

著作権

限定的権利条項

商標または登録商標

WebLogic J2EE コネクタ アーキテクチャ

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 24 日	BEA WebLogic Server バージョン 6.1

目次

4. 接続管理

5. コンフィグレーション

6. J2EE コネクタ アーキテクチャに準拠したリソース アダ

プタの作成

7. リソース アダプタのデプロイメント

8. クライアントに関する考慮事項

A. weblogic-ra.xml デプロイメント記述子の要素

B. 一般的な BEA J2EE コネクタ アーキテクチャ例外の回避策

索引

このマニュアルの内容

このマニュアルでは、WebLogic J2EE コネクタ アーキテクチャを紹介し、リソース アダプタを WebLogic Server にコンフィグレーションおよびデプロイする方法について説明します。このマニュアルの構成は次のとおりです。

- 第 1 章「WebLogic J2EE コネクタ アーキテクチャの概要」では、WebLogic J2EE コネクタ アーキテクチャについて概説します。
- 第 2 章「セキュリティ」では、WebLogic J2EE コネクタ アーキテクチャのセキュリティに関する考慮事項について説明します。
- 第 3 章「トランザクション管理」では、WebLogic J2EE コネクタ アーキテクチャでサポートしているさまざまなタイプのトランザクション レベルについて紹介し、それらをリソース アダプタの `.rar` アーカイブで指定する方法を説明します。
- 第 4 章「接続管理」では、さまざまな接続の管理タスクについて説明します。
- 第 5 章「コンフィグレーション」では、リソース アダプタを WebLogic Server にデプロイするために実行するコンフィグレーション タスクについて概説します。
- 第 6 章「J2EE コネクタ アーキテクチャに準拠したリソース アダプタの作成」では、リソース アダプタ（`.rar`）を記述するための要件について説明します。
- 第 7 章「リソース アダプタのデプロイメント」では、リソース アダプタについて概説し、それらを WebLogic Server にコンフィグレーションおよびデプロイする方法を説明します。
- 第 8 章「クライアントに関する考慮事項」では、WebLogic J2EE コネクタ アーキテクチャのクライアントに関する考慮事項について説明します。
- 付録 A「weblogic-ra.xml デプロイメント記述子の要素」では、`weblogic-ra.xml` の DTD および デプロイメント記述子の要素を明記しています。

- 付録 B「一般的な BEA J2EE コネクタ アーキテクチャ例外の回避策」では、2 つの一般的なコネクタ例外のトラブルシューティングについて説明します。

対象読者

このマニュアルは、Sun Microsystems の Java 2 Platform, Enterprise Edition（J2EE）を使った e- コマース アプリケーションを構築するアプリケーション開発者を対象としています。Web テクノロジ、オブジェクト指向プログラミング手法、および Java プログラミング言語に読者が精通していることを前提として書かれています。

e-docs Web サイト

BEA 製品のドキュメントは、BEA の Web サイトで入手できます。BEA のホーム ページで [製品のドキュメント] をクリックします。

このマニュアルの印刷方法

Web ブラウザの [ファイル | 印刷] オプションを使用すると、Web ブラウザからこのマニュアルのメイン トピックを一度に 1 つずつ印刷できます。

このマニュアルの PDF 版は、Web サイトで入手できます。PDF を Adobe Acrobat Reader で開くと、マニュアルの全体（または一部分）を書籍の形式で印刷できます。PDF を表示するには、WebLogic Server ドキュメントのホーム ページを開き、[ドキュメントのダウンロード] をクリックして、印刷するマニュアルを選択します。

Adobe Acrobat Reader は、Adobe の Web サイト（http://www.adobe.co.jp）から無料で入手できます。

関連情報

BEA の Web サイトでは、WebLogic Server の全マニュアルを提供しています。特に、以下のドキュメントを参照してください。

- BEA WebLogic J2EE コネクタ アーキテクチャの Javadoc（製品の配布 CD を参照）
- WebLogic 固有のリソース アダプタの文書型定義（付録 A「weblogic-ra.xml デプロイメント記述子の要素」を参照）
- BEA WebLogic Application Integration（http://edocs.beasys.co.jp/e-docs/wlintegration/v2_1/devadapt/index.htm を参照）このマニュアルでは、リソース アダプタをビルドする方法について説明しています。

Sun Microsystems の以下のドキュメントも参照してください。

- J2EE コネクタ アーキテクチャ ー http://java.sun.com/j2ee/connector/index.html
- J2EE コネクタ仕様、バージョン 1.0、最終草案 2 ー http://java.sun.com/j2ee/download.html#connectorspec
- J2EE プラットフォーム仕様、バージョン 1.3、最終草案 3 ー http://java.sun.com/j2ee

サポート情報

BEA のドキュメントに関するユーザからのフィードバックは弊社にとって非常に重要です。質問や意見などがあれば、電子メールで docsupport-jp@bea.com までお送りください。寄せられた意見については、ドキュメントを作成および改訂する BEA の専門の担当者が直に目を通します。

電子メールのメッセージには、ご使用のソフトウェア名とバージョン名、およびマニュアルのタイトルと作成日付をお書き添えください。本バージョンの BEA WebLogic Server について不明な点がある場合、または BEA WebLogic Server のインストールおよび動作に問題がある場合は、BEA WebSUPPORT

（www.bea.com）を通じて BEA カスタマ サポートまでお問い合わせください。カスタマ サポートへの連絡方法については、製品パッケージに同梱されているカスタマ サポート カードにも記載されています。

カスタマ サポートでは以下の情報をお尋ねしますので、お問い合わせの際はあらかじめご用意ください。

- お名前、電子メール アドレス、電話番号、ファクス番号
- 会社の名前と住所
- お使いの機種とコード番号
- 製品の名前とバージョン
- 問題の状況と表示されるエラー メッセージの内容

表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

表記法	適用
〔Ctrl〕+〔Tab〕	同時に押すキーを示す。
斜体	強調または本のタイトルを示す。

等幅テキスト	コード サンプル、コマンドとそのオプション、Java クラス、データ型、ディレクトリ、およびファイル名とその拡張子を示す。等幅テキストはキーボードから入力するテキストも示す。
	例：
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

表記法	適用
斜体の等幅テキスト	コード内の変数を示す。 例 : `String *CustomerName*;`
すべて大文字のテキスト	デバイス名、環境変数、および論理演算子を示す。 例 : LPT1 BEA_HOME OR
{ }	構文内の複数の選択肢を示す。

[ ]	構文内の任意指定の項目を示す。
	例 :
	`java utils.MulticastTest -n *name* -a *address*`
	`[-p *portnumber*] [-t *timeout*] [-s *send*]`

|	構文の中で相互に排他的な選択肢を区切る。
	例 :
	`java weblogic.deploy [list|deploy|undeploy|update]`
	`password {application} {source}`

...	コマンドラインで以下のいずれかを示す。 ■ 引数を複数回繰り返すことができる。 ■ 任意指定の引数が省略されている。 ■ パラメータや値などの情報を追加入力できる。
. . .	コード サンプルまたは構文で項目が省略されていることを示す。

1 WebLogic J2EE コネクタ アーキテクチャの概要

以下の節では、BEA WebLogic J2EE コネクタ アーキテクチャの概要について説明します。

- WebLogic Server 6.1 と J2EE 1.2 および J2EE 1.3
- J2EE コネクタ アーキテクチャの用語
- WebLogic J2EE コネクタ アーキテクチャ実装の概要
- J2EE コネクタ アーキテクチャのコンポーネント
- Black Box サンプル

WebLogic Server 6.1 と J2EE 1.2 および J2EE 1.3

このマニュアルでは、最終的ではない仕様である J2EE 1.3 で定義されている機能について説明します。実行している WebLogic Server で、J2EE 1.3 機能が有効になっている必要があります。J2EE 1.2 機能しか備えていない WebLogic Server 6.1 配布キットでは、WebLogic Server コネクタ アーキテクチャはデプロイできません。

J2EE 1.2 機能のみを備えた WebLogic Server 6.1 の J2EE 1.2 認定の配布キット、および J2EE 1.3 機能が有効になっている WebLogic Server 6.1 配布キットは、どちらも http://commerce.bea.com/downloads/products.jsp から入手できます。また、製品 CD にも収録されています。

J2EE コネクタ アーキテクチャの用語

WebLogic J2EE コネクタ アーキテクチャのマニュアルで扱う主な用語と概念は以下のとおりです。

- Common Client Interface（CCI）― アプリケーション コンポーネント用の標準クライアント API を定義し、エンタープライズ アプリケーション統合（EAI : Enterprise Application Integration）フレームワークが共通のクライアント API を使用して異種 EIS 間での対話を実現します。J2EE コネクタ アーキテクチャは EIS アクセス用の CCI を定義します。
- コンテナ ― WebLogic Server などのアプリケーション サーバの一部で、アプリケーション コンポーネントをデプロイし、実行時サポートを提供します。コンテナを使用すると、サポートされているコンポーネントをモニタおよび管理するサービスだけでなく、それらのコンポーネントもモニタおよび管理できます。コンテナは以下のいずれかです。
  - リソース アダプタのホストとなるコネクタ コンテナ
  - JSP、サーブレット、および静的 HTML ページのホストとなる Web コンテナ

- EJB コンポーネントのホストとなる EJB コンテナ
- スタンドアロン アプリケーション クライアントのホストとなるアプリケーション クライアント コンテナ

  各標準コンテナの詳細については、エンタープライズ JavaBean（EJB）、JavaServer Page（JSP）、およびサーブレットの仕様を参照してください。

■ エンタープライズ情報システム（EIS : Enterprise Information System）リソース ― EIS 固有の機能をクライアントに提供します。以下のような機能があります。

- データベース システム内のレコードまたはレコード セット
- エンタープライズ リソース プランニング（ERP）システム内のビジネス オブジェクト
- トランザクション処理システム内のトランザクション プログラム

■ エンタープライズ情報システム（EIS）― 企業に情報インフラストラクチャを提供します。EIS は一連のサービスをクライアントに提供します。これらのサービスは、ローカルまたはリモートのインタフェースとしてクライアントにエクスポーズされます。EIS には以下の例があります。

- ERP システム
- メインフレーム トランザクション処理システム
- レガシー データベース システム

■ J2EE コネクタ ― リソース アダプタを参照してください。

■ J2EE コネクタ アーキテクチャ ― J2EE 準拠のアプリケーション サーバをエンタープライズ情報システム（EIS）と統合するためのアーキテクチャです。このアーキテクチャは、EIS ベンダのリソース アダプタと、リソース アダプタがプラグ インとして機能する WebLogic Server などのアプリケーション サーバという 2 つの部分で構成されます。このアーキテクチャでは、アプリケーション サーバのプラグインとして機能するためにリソース アダプタがサポートする必要があるトランザクション、セキュリティ、接続管理などの規約を定義します。J2EE コネクタ アーキテクチャは EIS アクセス用の Common Client Interface（CCI）も定義します。CCI は、異種 EIS と対話するためのクライアント API を定義します。

■ 管理対象の環境 ― EIS にアクセスする J2EE ベースの Web 対応多層アプリケーションの操作環境を定義します。アプリケーションは、コンテナにデプ

ロイされる 1 つまたは複数のアプリケーション コンポーネント（EJB、JSP、サーブレット）から構成されます。コンテナは以下のいずれかです。

- JSP、サーブレット、および静的 HTML ページのホストとなる Web コンテナ
- EJB コンポーネントのホストとなる EJB コンテナ
- スタンドアロン アプリケーション クライアントのホストとなるアプリケーション クライアント コンテナ

- 非管理対象の環境 ― 2 層アプリケーションの操作環境を定義します。アプリケーション クライアントは、リソース アダプタを直接使用して、2 層アプリケーションの第 2 層を定義する EIS にアクセスします。
- `.rar` ファイル ― リソース アダプタ アーカイブ。リソース アダプタを実行するのに必要なクラスやその他のファイルをロードするのに使われる圧縮（zip）ファイルです。
- `ra.xml` ファイル ― Sun Microsystems の標準 DTD を使用してリソース アダプタ関連の属性タイプとデプロイメント プロパティを記述します。
- リソース アダプタ ― システムレベルのソフトウェア ドライバ。WebLogic Server などのアプリケーション サーバが EIS に接続するために使用します。リソース アダプタは「J2EE コネクタ」として機能します。WebLogic J2EE コネクタ アーキテクチャは、エンタープライズ情報システム（EIS）ベンダおよびサードパーティ アプリケーション開発者が開発し、Sun Microsystems の J2EE プラットフォーム仕様バージョン 1.3 に準拠しているアプリケーション サーバにデプロイ可能なリソース アダプタをサポートしています。リソース アダプタには、Java コンポーネントに加えて、必要な場合には EIS との対話に必要なネイティブ コンポーネントが入っています。
- リソース マネージャ ― 共有 EIS リソースを管理する EIS の一部。リソース マネージャの例には、データベース システム、メインフレーム TP システム、ERP システムなどがあります。クライアントは、リソース マネージャが管理するリソースを使用するためにリソース マネージャへのアクセスを要求します。トランザクション対応リソース マネージャは、トランザクション マネージャが外部的に制御して調整するトランザクションに参加できます。J2EE コネクタ アーキテクチャのコンテキストでは、リソース マネージャのクライアントとして中間層サーバとクライアント層アプリケーションを含めることができます。リソース マネージャは通常、クライアントのアクセス元とは異なるアドレス空間または異なるマシン上にあります。

- サービス プロバイダ インタフェース（SPI）― EIS との接続を提供および管理したり、トランザクションの境界を設定したり、フレームワークでイベントのリスニングとリクエストの転送を行えるようにしたりするオブジェクトを含みます。J2EE コネクタ アーキテクチャ準拠のすべてのリソース アダプタは、これらのインタフェースの実装を `javax.resource.spi` パッケージで提供する必要があります。
- システム規約 ― エンティティ間で接続リクエストを渡すメカニズム。WebLogic Server と EIS など、アプリケーション サーバ間での標準のシステムレベル プラグイン可能性を実現するために、コネクタ アーキテクチャではアプリケーション サーバと EIS 間の標準のシステムレベル規約を定義しています。これらシステムレベル規約の EIS 側は、リソース アダプタで実装されます。
- `weblogic-ra.xml` ファイル ― WebLogic Server 固有の補足デプロイメント情報を `ra.xml` ファイルに追加します。

WebLogic J2EE コネクタ アーキテクチャ実装の概要

BEA WebLogic Server は、引き続き Sun Microsystems J2EE プラットフォーム仕様、バージョン 1.3 に基づいています。J2EE コネクタ アーキテクチャは、エンタープライズ情報システム（EIS）を簡単に J2EE プラットフォームに統合します。この目的は、コンポーネント モデル、トランザクション、およびセキュリティ インフラストラクチャを含む J2EE プラットフォームの長所を活かして、EIS の統合という困難な課題を解決することです。

J2EE コネクタ アーキテクチャは、数多くのアプリケーション サーバと EIS との間を接続するという問題を Java により解決します。コネクタ アーキテクチャを使用することで、EIS ベンダはアプリケーション サーバごとに自社製品をカスタマイズする必要がなくなります。J2EE コネクタ アーキテクチャに準拠することで、BEA WebLogic Server ではカスタム コードを追加しなくても、新しい EIS への接続機能を追加できるようになります。

コネクタ アーキテクチャを使用すると、EIS ベンダは自社 EIS 用の標準リソース アダプタを提供できます。このリソース アダプタは WebLogic Server のプラグインとなり、EIS と WebLogic Server 間の統合を実現する基本インフラストラクチャを提供します。

コネクタ アーキテクチャをサポートすることで、BEA WebLogic Server が複数の EIS に接続可能となります。同様に、EIS ベンダは、BEA WebLogic Server のプラグイン機能を持ち、コネクタ アーキテクチャに準拠した標準のリソース アダプタを提供するだけで済みます。

J2EE コネクタ アーキテクチャのコンポーネント

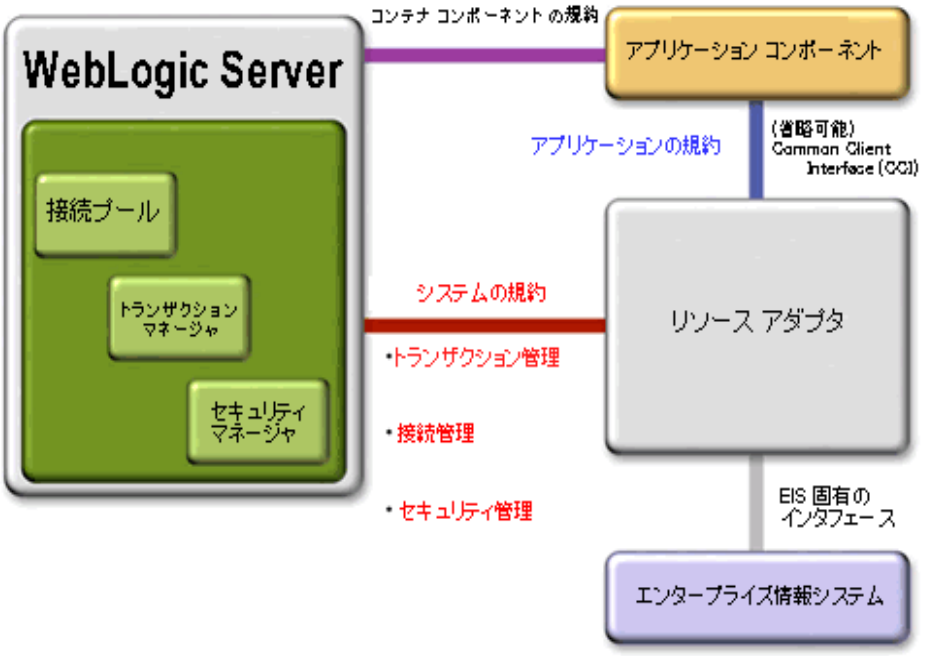
J2EE コネクタ アーキテクチャは、WebLogic Server や EIS 固有のリソース アダプタなどのアプリケーション サーバで実装されます。リソース アダプタとは EIS に固有のシステム ライブラリのことで、EIS への接続を提供するものです。リソース アダプタは JDBC ドライバと同様の機能を持っています。リソース アダプタと EIS 間のインタフェースは基底となる EIS に固有なので、ネイティブ インタフェースの場合もあります。

J2EE コネクタ アーキテクチャは主に 3 つのコンポーネントから構成されます。

- システムレベル規約 ― リソース アダプタとアプリケーション サーバ（WebLogic Server）の間
- Common Client Interface（CCI）― リソース アダプタにアクセスするための Java アプリケーションおよびデプロイメント ツールに対してクライアント API を提供します。
- パッケージ化とデプロイメント インタフェース ― さまざまなリソース アダプタが J2EE アプリケーション内でモジュール形式のプラグインとして機能できるようにします。

次の図は、J2EE コネクタ アーキテクチャの概要を示しています。

図 1-1　J2EE コネクタ アーキテクチャ



リソース アダプタは「J2EE コネクタ」として機能します。WebLogic J2EE コネクタ アーキテクチャは、エンタープライズ情報システム（EIS）ベンダおよびサードパーティ アプリケーション開発者が開発し、Sun Microsystems の J2EE プラットフォーム仕様、バージョン 1.3 に準拠しているアプリケーション サーバにデプロイ可能なリソース アダプタをサポートしています。リソース アダプタには、Java、および必要に応じて EIS との対話に必要なネイティブ コンポーネントが含まれます。

システムレベル規約

J2EE コネクタ アーキテクチャ仕様では、J2EE に準拠したアプリケーション サーバ（WebLogic Server）と EIS 固有のリソース アダプタとの間のシステムレベル規約を定義しています。WebLogic Server はこの仕様に従って、以下に関して定義されている標準規約を実装しています。

- 接続管理 — アプリケーション サーバに基底の EIS への接続プールを提供する規約。アプリケーション コンポーネントは、接続管理によって EIS に接続できます。これにより、EIS へのアクセスが必要な多数のクライアントをサポートするスケーラブルなアプリケーション環境が実現します。

注意： 接続管理の詳細については、第 4 章「接続管理」を参照してください。

- トランザクション管理 ― トランザクション マネージャと、EIS リソース マネージャへのトランザクション アクセスをサポートする EIS との間の規約。この規約により、アプリケーション サーバはトランザクション マネージャを使用して、複数のリソース マネージャ間にまたがるトランザクションを管理することができます。

注意： トランザクション管理の詳細については、第 3 章「トランザクション管理」を参照してください。

- セキュリティ管理 ― EIS へのセキュア アクセスと、セキュアなアプリケーション環境のサポートを提供する規約。これによって EIS に対する脅威が小さくなり、EIS が管理する情報リソースが保護されます。

注意： セキュリティ管理の詳細については、第 2 章「セキュリティ」を参照してください。

Common Client Interface（CCI）

Common Client Interface（CCI）では、アプリケーション コンポーネント用の標準クライアント API を定義しています。CCI により、アプリケーション コンポーネントとエンタープライズ アプリケーション統合（EAI）フレームワークが、共通のクライアント API を使用して異種 EIS 間で対話できます。

CCI は、エンタープライズ ツール ベンダと EAI ベンダを対象ユーザとしています。アプリケーション コンポーネント自体も API に書き込めますが、CCI は低レベルの API です。仕様で推奨されている CCI の利用方法は、ほとんどのアプリケーション開発者が使用するアプリケーションレベルのプログラミング インタフェースとしてではなく、ツール ベンダが提供するより多彩な機能を実現するための基盤として利用することです。

また、CCI では、EIS に対して関数を実行し、結果を取得することに重点を置いたリモート関数呼び出しインタフェースを定義しています。CCI は、EIS に固有のデータ型など、特定の EIS に依存していません。ただし、CCI はリポジトリの EIS 固有のメタデータで利用できます。

CCI により、WebLogic Server アプリケーションは EIS との接続を作成および管理したり、対話を処理したり、入力、出力、または戻り値のデータ レコードを管理したりすることができます。CCI の目的は、JavaBeans アーキテクチャおよび Java コレクション フレームワークを活用することです。

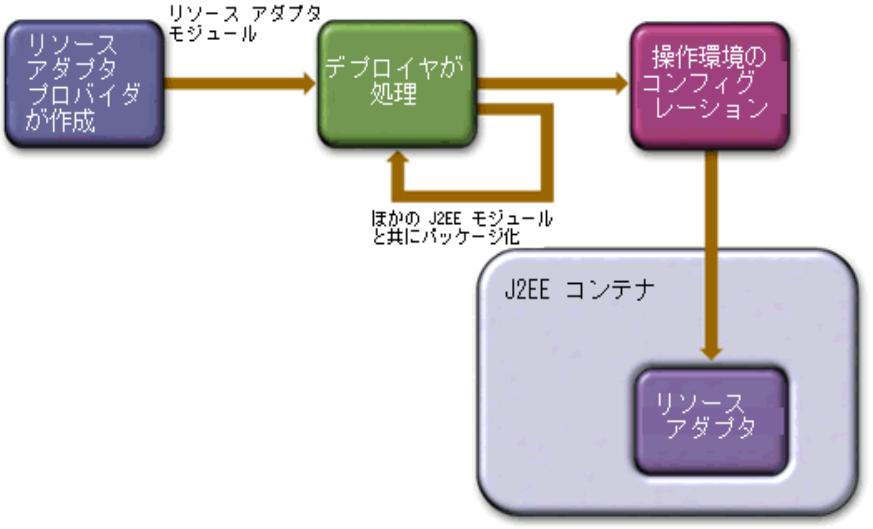
J2EE コネクタ アーキテクチャ バージョン 1.0 では、リソース アダプタが CCI をクライアント API としてサポートすることを推奨している一方で、リソース アダプタがシステム規約を実装する必要があることも規定しています。リソース アダプタでは、Java Database Connectivity（JDBC）API に基づいたクライアント API など、CCI 以外のクライアント API も使用できます。

注意： Common Client Interface の関連情報については、第 8 章「クライアントに関する考慮事項」を参照してください。

パッケージ化とデプロイメント

J2EE コネクタ アーキテクチャはパッケージ化およびデプロイメント インタフェースを提供するので、さまざまなリソース アダプタが WebLogic Server などの J2EE 準拠のアプリケーション サーバ内でモジュール形式のプラグインとして機能することができます。

図 1-2 パッケージ化とデプロイメント



リソース アダプタ プロバイダは、Java インタフェースおよびクラスをリソース アダプタの実装の一部として開発します。これらの Java クラスは J2EE コネクタ アーキテクチャ固有の規約と、リソース アダプタによって提供される EIS 固有の機能を実装します。リソース アダプタでは、基底の EIS に固有のネイティブ ライブラリの使用を必須にすることもできます。

Java インタフェースとクラスは、デプロイメント記述子を使用して（必須のネイティブ ライブラリ、ヘルプ ファイル、マニュアル、その他のリソースと一緒に）パッケージ化され、リソース アダプタ モジュールとなります。デプロイメント記述子では、リソース アダプタ プロバイダとリソース アダプタをデプロイするデプロイヤとの間の規約を定義します。

リソース アダプタ モジュールは、共有スタンドアロン モジュールとして、またはアプリケーションの一部としてパッケージ化して配布できます。デプロイメントでは、リソース アダプタ モジュールを WebLogic Server などのアプリケーション サーバにインストールしてから、対象の操作環境に合わせてコンフィグレーションします。リソース アダプタのコンフィグレーションは、リソース アダプタ モジュールの一部としてデプロイメント記述子に定義されているプロパティに基づいて行います。

注意： パッケージ化とデプロイメントの詳細については、第 6 章「J2EE コネクタ アーキテクチャに準拠したリソース アダプタの作成」、第 5 章「コンフィグレーション」および第 7 章「リソース アダプタのデプロイメント」を参照してください。

Black Box サンプル

このリリースでは、リソース アダプタの簡単なコード例が用意されています。このコード例は、JDBC 呼び出しによく似た Black Box リソース アダプタを使用します。EJB は Black Box のデータをモデル化するために使用され、Java クライアントは Black Box リソース アダプタにクエリを送り、結果を表示するために使用されます。サンプルでは、WebLogic Server 評価版に付属するオール Java の Cloudscape DBMS を使用します。詳細については、ダウンロード製品に付属の WebLogic J2EE コネクタ アーキテクチャのサンプル Javadoc を参照してください。

2 セキュリティ

以下の節では、WebLogic J2EE コネクタ アーキテクチャのセキュリティについて説明します。

- コンテナ管理およびアプリケーション管理によるサインオン
- セキュリティ プリンシパル マップ
- パスワード変換ツール

コンテナ管理およびアプリケーション管理によるサインオン

「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」で指定されているように、WebLogic J2EE コネクタ アーキテクチャ実装はコンテナ管理とアプリケーション管理の両方のサインオンをサポートしています。

実行時、Weblogic J2EE コネクタ アーキテクチャ実装は、呼び出し側クライアント コンポーネントのデプロイメント記述子に基づいて、指定されたサインオン メカニズムを判別します。Weblogic Server J2EE コネクタ アーキテクチャ実装が、リソース アダプタの接続ファクトリの JNDI ルックアップを正しく実行できないなどの理由で、サインオン メカニズムを判別できない場合、コネクタ アーキテクチャはコンテナ管理によるサインオンを試行します。

注意： ただしこの場合でも、クライアント コンポーネントが明示的なセキュリティ情報を指定していれば、その情報も接続を取得するための呼び出し時に提示されます。

詳細については、第 8 章「クライアントに関する考慮事項」の「ConnectionFactory（クライアントと JNDI 間の対話）の取得」を参照してください。

アプリケーション管理によるサインオン

アプリケーション管理によるサインオンの場合、クライアント コンポーネントは、エンタープライズ情報システム（EIS）に接続するための呼び出しを実行するときに、必要なセキュリティ情報（通常はユーザ名とパスワード）を提示します。この場合、アプリケーション サーバは、接続リクエストと一緒にこの情報を渡す以外にセキュリティ関連の処理を行いません。提供されるリソース アダプタは、セキュリティ情報を提示したクライアント コンポーネントを使用して、リソース アダプタ実装固有の方法で EIS サインオンを実行します。

コンテナ管理によるサインオン

コンテナ管理によるサインオンの場合、クライアント コンポーネントはセキュリティ情報を提示しないので、コンテナが必要なサインオン情報を判別し、接続を要求するための呼び出し時にその情報をリソース アダプタに提供しなければなりません。コンテナ管理によるサインオンでは、コンテナが適切なリソース プリンシパルを判別し、そのリソース プリンシパルの情報をリソース アダプタに Java Authentication and Authorization Service（JAAS）の Subject 形式で提供する必要があります。

セキュリティ プリンシパル マップ

「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」（http://java.sun.com/j2ee/download.html#connectorspec）の「EIS Sign-on」セクションでは、サインオンの実行を委任するリソース プリンシパルを定義するためのさまざまなオプションが示されています。Weblogic Server 実装では、この仕様のセキュリティ プリンシパル マップ オプションを実装しています。

このオプションの場合、リソース プリンシパルは、呼び出すコンポーネントの開始側 / 呼び出し側プリンシパル ID のマップによって判別されます。判別されたリソース プリンシパルはマップ元プリンシパルの ID またはセキュリティ属性を継承しませんが、定義されたマッピングに基づいて ID とセキュリティ属性（パスワード）を取得します。

したがって、コンテナ管理によるサインオンを有効にして使用するには、Weblogic Server が `initiating-principal` と `resource-principal` の関連付けを指定するメカニズムを用意する必要があります。このために WebLogic Server では、デプロイされるリソース アダプタごとに定義可能なセキュリティ プリンシパル マップを使用します。

コンテナ管理によるサインオンがクライアント コンポーネントに要求され、セキュリティ プリンシパル マップがデプロイされるリソース アダプタに合わせてコンフィグレーションされていない場合、接続の取得は試行されますが、提供される JAAS Subject は `NULL` となります。このシナリオが有効となるかどうかは、リソース アダプタの実装によって決まります。

セキュリティ プリンシパル マップをコンフィグレーションしていない状況でも、有効と見なされる場合があります。リソース アダプタが、すべての EIS 接続を、ハードコード化して事前にコンフィグレーションされているセキュリティ情報を使用して内部的に取得する（アプリケーション管理によるサインオンでもコンテナ管理によるサインオンでもない、別のシナリオと考えることもできます）ため、新しい接続の要求時に渡されるセキュリティ情報に依存しない場合がそれに該当します。

WebLogic Server と指定のリソース アダプタとの間でセキュリティ情報をやり取りする方法が定義済みの接続管理システム規約に定義されている場合、コンテナ管理によるサインオンとアプリケーション管理によるサインオンのどちらを使用するかは、接続を要求するクライアント アプリケーションに定義されているデプロイメント情報に基づいて決められます。接続管理システム規約の指定方法の詳細については、第 8 章「クライアントに関する考慮事項」を参照してください。

クライアント コンポーネントがサインオン メカニズムを指定する方法の詳細については、「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」（http://java.sun.com/j2ee/download.html#connectorspec）の「Connection Management」章の「Application Programming Model」節を参照してください。

J2EE コネクタ アーキテクチャ アプリケーションのセキュリティ モデルの詳細については、同「Application Security Model」を参照してください。

コンテナ管理によるサインオンの使い方

コンテナ管理によるサインオンを使用するには、WebLogic Server がリソース プリンシパルを識別してから、リソース プリンシパルに代わって接続を要求しなければなりません。WebLogic Server は、`weblogic-ra.xml` デプロイメント記述子ファイルの `security-principal-map` 要素で指定されているセキュリティ プリンシパル マッピングを探してリソース プリンシパルを識別します。

`security-principal-map` 要素は、`initiating-principal` と `resource-principal` の関係を定義します。

各 `security-principal-map` 要素は、リソース アダプタおよび EIS サインオン処理に合わせて適切なリソース プリンシパル値を定義するメカニズムを提供します。`security-principal-map` 要素では、管理対象の接続と接続ハンドルを割り当てる場合に使用する定義済みの開始プリンシパルと対応するリソース プリンシパルのユーザ名およびパスワードを指定します。

デフォルト リソース プリンシパル

デフォルト リソース プリンシパルは、security-principal-map 要素の接続ファクトリに合わせて定義できます。initiating-principal 値に「*」を指定し、対応する resource-principal 値を指定した場合、マップ内で現在の ID と一致するものがないときには必ず定義した resource-principal が利用されます。

ただし、この要素は省略できます。コンテナ管理によるサインオンがリソース アダプタにサポートされており、いずれかのクライアントに使用される場合は、何らかの形式で指定する必要があります。

また、デプロイ時に管理対象の接続を接続プールに取得する試みは、定義されている「デフォルト」リソース プリンシパル（指定されている場合）を使用して行われます。

J2EE コネクタ アーキテクチャの security-principal-map のコンフィグレーションとデプロイ済みの .rar（リソース アダプタ）との関連付けについては、第 5 章「コンフィグレーション」の「セキュリティ プリンシパル マップのコンフィグレーション」を参照してください。

パスワード変換ツール

BEA では、セキュリティ パスワード保護の重要性を考慮して、weblogic-ra.xml ファイルで提示されるすべてのパスワードを暗号化できる変換ツールを用意しています。

詳細については、第 5 章「コンフィグレーション」の「セキュリティ プリンシパル マップのコンフィグレーション」を参照してください。

3 トランザクション管理

以下の節では、WebLogic J2EE コネクタ アーキテクチャがサポートしているさまざまなトランザクション レベルと、リソース アダプタの .rar アーカイブにトランザクション レベルを指定する方法について説明します。

- サポートされているトランザクション レベル
- .rar コンフィグレーションでのトランザクション レベルの指定
- トランザクション管理規約

サポートされているトランザクション レベル

ビジネス アプリケーションでは、EIS へのトランザクション アクセスが重要な要件です。J2EE コネクタ アーキテクチャは、トランザクションの概念、つまりデータの一貫性および整合性を維持するために、データに関して一緒にコミットするか、または一切コミットしてはならない各種の処理をサポートしています。

BEA WebLogic Server J2EE コネクタ アーキテクチャ実装では、WebLogic Server の堅牢なトランザクション マネージャ実装を利用し、（「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」で説明されている）以下のトランザクション レベルに対応したリソース アダプタをサポートしています。

- XA トランザクション サポート ― リソース アダプタ以外（したがって EIS 以外）のトランザクション マネージャによってトランザクションを管理できるようにします。リソース アダプタには、`ra.xml` ファイルの transaction-support 要素を指定することでトランザクション サポートの種類（リソース アダプタがサポートできるのは 1 種類のみ）を定義します。アプリケーション コンポーネントが EIS 接続リクエストをトランザクションの一部として境界設定する場合、アプリケーション サーバがトランザクション マネージャで XA リソースを有効にする必要があります。アプリケーション コンポーネントがその接続を閉じると、アプリケーション サーバはトランザクション マネージャのリストから XA リソースを削除し、トランザクションが終了した時点で EIS 接続をクリーンアップします。
- ローカル トランザクション サポート ― アプリケーション サーバがリソース アダプタと同じローカル リソースを管理できるようにします。XA トランザクションと違い、2 フェーズ コミット プロトコル（2PC）に関わることができません。リソース アダプタには、`ra.xml` ファイルの transaction-support 要素を指定することでトランザクション サポートの種類（リソース アダプタがサポートできるのは 1 種類のみ）を定義します。アプリケーション コンポーネントが EIS 接続を要求すると、アプリケーション サーバは現在のトランザクション コンテキストに基づいてローカル トランザクションを開始します。アプリケーション コンポーネントがその接続を閉じると、アプリケーション サーバはローカル トランザクションをコミットし、トランザクションが終了した時点で EIS 接続をクリーンアップします。

注意： ra.xml の文書型定義の詳細については、以下の Sun Microsystems のドキュメントを参照してください。
http://java.sun.com/dtd/connector_1_0.dtd

- トランザクション非サポート ― 一般に、リソース アダプタが XA またはローカル トランザクションをサポートしていない（したがって非サポートを「サポート」している）場合、つまりアプリケーション コンポーネントがそのリソース アダプタを使用する必要がある場合、アプリケーション コンポーネントはリソース アダプタによって表される EIS との接続をトランザクションで利用してはなりません。ただし、アプリケーション コンポーネントがトランザクションで EIS 接続を必要とする場合、アプリケーション コンポーネントは XA またはローカル トランザクションをサポートするリソース アダプタと対話しなければなりません。

サポートされているトランザクション レベルの詳細については、「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」（http://java.sun.com/j2ee/download.html#connectorspec）の「Transaction Management」章を参照してください。

.rar コンフィグレーションでのトランザクション レベルの指定

リソース アダプタは、サポート対象のトランザクションの種類を Sun Microsystems から提供される ra.xml デプロイメント記述子ファイルで指定します。.rar でのトランザクション レベルの種類を指定する方法については、第 5 章「コンフィグレーション」の「トランザクション レベル タイプのコンフィグレーション」を参照してください。

注意： ra.xml の文書型定義の詳細については、以下の Sun Microsystems のドキュメントを参照してください。
http://java.sun.com/dtd/connector_1_0.dtd

トランザクション管理規約

多くの場合、トランザクション（ローカル トランザクションと呼ばれます）は 1 つの EIS システムに制限され、EIS リソース マネージャ自体がそうしたトランザクションを管理します。一方、XA トランザクション（またはグローバル トランザクション）は複数のリソース マネージャに及ぶことがあります。このトランザクションでは、通常はアプリケーション サーバに付属する外部トランザクション マネージャがトランザクションを調整する必要があります。トランザクション マネージャは 2 フェーズ コミット プロトコル（2PC）を使用して、複数のリソース マネージャ（EIS）にまたがるトランザクションを管理します。XA トランザクションに 1 つのリソース マネージャだけが関わる場合は、リソース マネージャは 1 フェーズ コミットを使用して最適化します。

J2EE コネクタ アーキテクチャでは、アプリケーション サーバとリソース アダプタ（と基底のリソース マネージャ）との間のトランザクション管理規約を定義します。トランザクション管理規約は、接続管理規約を拡張し、ローカル トランザクションと XA トランザクションの両方のサポートを提供します。トランザクション管理規約は、トランザクションの種類に応じて 2 つの部分から構成されます。

- トランザクション マネージャと EIS リソース マネージャとの間の JTA XAResource ベースの規約
- ローカル トランザクション管理規約

これらの規約により、WebLogic Server などのアプリケーション サーバはトランザクション管理用のインフラストラクチャと実行時環境を提供できます。アプリケーション コンポーネントはこのトランザクション インフラストラクチャを利用して、コンポーネントレベルのトランザクション モデルをサポートします。

EIS 実装は多様なので、トランザクションを柔軟にサポートする必要があります。J2EE コネクタ アーキテクチャでは、トランザクションを管理するために EIS に要求される条件はありません。EIS 内のトランザクションの実装に応じて、リソース アダプタは以下のようにサポートを提供します。

- トランザクション非サポート ― 従来のアプリケーションと多くのバックエンド システムでは一般的です。
- ローカル トランザクションだけをサポート

- ローカル トランザクションと XA トランザクションの両方をサポート

WebLogic Server は、トランザクションの 3 つのレベルすべてをサポートしているので、異なるトランザクション レベルの EIS もサポートします。

4 接続管理

以下の節では、BEA WebLogic J2EE 接続管理アーキテクチャ関連のさまざまな接続管理作業について説明します。

- エラー ロギングとトレース機能
- 接続プロパティのコンフィグレーション
- BEA WebLogic Server 拡張接続管理機能
- Console を使用した接続プールのモニタ

エラー ロギングとトレース機能

「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」では、アプリケーション サーバの要件の 1 つとして、ManagedConnectionFactory.set/getLogWriter を使用してエラー ロギングおよびトレース機能をリソース アダプタに提供することが記載されています。

BEA WebLogic Server では、この機能をコンフィグレーションするために、weblogic-ra.xml 記述子ファイルに 2 つの要素を用意しています。

- logging-enabled 要素はロギングがオンまたはオフのどちらであるかを示します。この要素のデフォルト値は、false です。
- log-filename 要素は、ロギング情報を書き込むファイル名を指定します。

詳細については、付録 A「weblogic-ra.xml デプロイメント記述子の要素」を参照してください。

接続プロパティのコンフィグレーション

ra.xml デプロイメント記述子ファイルには、ManagedConnectionFactory インスタンスごとに 1 つのコンフィグレーション設定を宣言するための config-property 要素が入っています。通常、リソース アダプタ プロバイダは、これらのコンフィグレーション プロパティを設定します。ただし、コンフィグレーション プロパティが設定されていない場合、リソース アダプタのデプロイ担当者がプロパティの値を指定する必要があります。

WebLogic Server では、weblogic-ra.xml デプロイメント記述子ファイルの map-config-property 要素を使用してコンフィグレーション プロパティを設定できます。リソースのアダプタの一連のコンフィグレーション プロパティをコンフィグレーションするには、宣言するコンフィグレーション プロパティごとに map-config-property-name と map-config-property-value の組み合わせを指定します。

また、map-config-property 要素を使用すると、ra.xml デプロイメント記述子ファイルで指定した値をオーバーライドできます。WebLogic Server は、起動時に map-config-property の値を ra.xml ファイルの config-property の値と比較します。コンフィグレーション プロパティ名が一致した場合、WebLogic Server は対応するコンフィグレーション プロパティ名の map-config-property-value を使用します。

BEA WebLogic Server 拡張接続管理機能

「J2EE コネクタ仕様 バージョン 1.0、最終草案 2」に記載されている接続管理要件に加えて、BEA WebLogic Server は、接続プールのサイズをコンフィグレーションして自動的に維持するオプション設定とサービスを提供します。

ManagedConnection 作成に関する実行時パフォーマンス コストの最小化

ManagedConnection の作成では、ManagedConnection が表すエンタープライズ情報システム（EIS）の複雑さに応じて大きなコストが発生します。そのため、WebLogic Server の起動時に接続プールに初期数の ManagedConnection を登録し、実行時には作成しないようにします。この設定は、weblogic-ra.xml 記述子ファイルの initial-capacity 要素を使用してコンフィグレーションします。この要素のデフォルト値は、1 ManagedConnection です。

「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」に記されているように、アプリケーション コンポーネントがリソース アダプタを使用して EIS との接続を要求すると、WebLogic Server はまず、接続プール内で利用可能な ManagedConnection の中で要求されている接続タイプと一致するものがないか探します。しかし、一致するものが見つからない場合には、新規の ManagedConnection を作成して接続リクエストに応じます。

WebLogic Server の設定を使用すると、一致するものがない場合に ManagedConnection を自動的に追加作成できます。この機能により、時間の経過と共に増加する接続プールのサイズとサイズが増加するたびに低下するサーバの

パフォーマンスを柔軟に制御できます。この設定は、weblogic-ra.xml 記述子ファイルの capacity-increment 要素を使用してコンフィグレーションします。デフォルト値は 1 ManagedConnection です。

WebLogic Server の起動時には開始セキュリティ プリンシパルまたはリクエスト コンテキスト情報が不明なので、initial-capacity でコンフィグレーションされている初期数の ManagedConnection は、デフォルト サブジェクトと null のクライアント リクエスト情報の入ったデフォルト セキュリティ コンテキストを使用して作成されます。capacity-increment でコンフィグレーションされている数の ManagedConnection が追加作成される場合、最初の ManagedConnection は接続リクエストの既知の開始プリンシパルおよびクライアント リクエスト情報を使用して作成されます。残りの ManagedConnection は、capacity-increment の制限に達するまで、最初の ManagedConnection を作成したときと同じデフォルト セキュリティ コンテキストを使用して作成されます。

デフォルト リソース プリンシパルのコンフィグレーションの詳細については、第 2 章「セキュリティ」を参照してください。

接続プールの増加数の制御

ManagedConnection の追加に伴い、それぞれの ManagedConnection が消費するメモリやディスク領域などのシステム リソースも増えていきます。エンタープライズ情報システム（EIS）によっては、この消費量がシステム全般のパフォーマンスに影響します。WebLogic Server では、ManagedConnection がシステム リソースに与える影響を制御するため、割り当て済み ManagedConnection の最大許容数の設定をコンフィグレーションできます。

この設定は、weblogic-ra.xml 記述子ファイルの maximum-capacity 要素を使用してコンフィグレーションします。接続リクエスト中に ManagedConnection を（capacity-increment が 2 以上の場合は複数）新規作成する必要がある場合、WebLogic Server が最大許容数を超えて ManagedConnection を作成することはありません。最大数に達すると、WebLogic Server は接続プールから ManagedConnection を再利用しようとします。ただし、再利用できる接続がない場合、再利用の試みが失敗したことと、接続リクエストが最大許容数の接続分のみ許可されることを示す警告がログに記録されます。maximum-capacity のデフォルト値は 10 ManagedConnection です。

システム リソースの使用量の制御

ManagedConnection の最大数を設定すると、処理能力を超えた数の ManagedConnection を割り当てることが原因でサーバが過負荷になることはありませんが、常に必要に応じてシステム リソースの量を効率的に制御するわけではありません。WebLogic Server では、リソース アダプタのデプロイメント中に接続プール内の ManagedConnection の動作状況をモニタするサービスを利用できます。使用量が減少し、そのレベルで一定期間とどまっている場合、接続プールのサイズは継続的な接続リクエストを十分満たせるだけの量にまで縮小されます。

このシステム リソース使用量サービスはデフォルトで有効です。ただし、`weblogic-ra.xml` 記述子ファイルの `shrinking-enabled` 要素を `false` に設定すると、このサービスを無効にできます。`weblogic-ra.xml` 記述子ファイルの `shrink-period-minutes` 要素を使用すると、WebLogic Server が接続プールのサイズを縮小する必要があるかどうかを計算し、縮小する必要がある場合に未使用の ManagedConnection をプールから削除する頻度を設定できます。この要素のデフォルト値は、15 分です。

接続リークの削除

「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」に記載されているように、EIS 接続を終了すると、アプリケーション コンポーネントは接続解除リクエストを送信します。この時点で、WebLogic Server は必要なクリーンアップを実行し、接続を将来の接続リクエストで使用できるようにする必要があります。ただし、アプリケーション コンポーネントが接続解除に失敗した場合、接続プールが利用可能な接続を使い果たすため、将来の接続リクエストが失敗することがあります。

WebLogic Server では、設定されている使用時間が経過した ManagedConnection を自動的に閉じることによって、こうした事態を避けるためのサービスが用意されています。使用時間は、`weblogic-ra.xml` 記述子ファイルの `connection-duration-time` 要素で設定します。また、`connection-cleanup-frequency` 要素を使用すると、WebLogic Server が使用中の ManagedConnection の使用時間を計算し、設定時間を超えたものを閉じる頻度を設定できます。

接続リーク検出サービスを無効にするには、connection-cleanup-frequency 要素を -1 に設定します。デフォルトでは、このサービスは有効です。これらの要素で使用される単位は秒です。

Console を使用した接続プールのモニタ

BEA WebLogic Server Administration Console を使用してコネクタ接続中のすべての接続プールをモニタするには、以下の手順に従います。

1. Console の左ペインでモニタするコネクタを選択します。
2. マウスを右クリックし、ポップアップ メニューから [すべての接続中のコネクタ接続プールのモニタ] を選択します。

   接続プールの接続情報が選択したコネクタに関して右ペインに表示されます。

5 コンフィグレーション

以下の節では、WebLogic J2EE コネクタ アーキテクチャ実装のコンフィグレーション要件について説明します。

- リソース アダプタの開発者向けツール
- リソース アダプタの開発者向けツール
- ra.xml ファイルのコンフィグレーション
- weblogic-ra.xml ファイルのコンフィグレーション
- セキュリティ プリンシパル マップのコンフィグレーション
- パスワード変換ツールの使い方
- トランザクション レベル タイプのコンフィグレーション

リソース アダプタの開発者向けツール

BEA では、リソース アダプタの作成とコンフィグレーションを支援するツールを提供しています。この節では、これらのツールについて説明します。

スケルトン デプロイメント記述子を作成する ANT タスク

スケルトン デプロイメント記述子を作成するときに、WebLogic ANT ユーティリティを利用できます。ANT ユーティリティは WebLogic Server 配布キットと共に出荷されている Java クラスです。ANT タスクによって、リソース アダプタを含むディレクトリが調べられ、そのリソース アダプタで検出されたファイルを基にデプロイメント記述子が作成されます。ANT ユーティリティは、個別のリソース アダプタに必要なコンフィグレーションやマッピングに関する情報をすべて備えているわけではないので、ANT ユーティリティによって作成されるスケルトン デプロイメント記述子は不完全なものです。ANT ユーティリティがスケルトン デプロイメント記述子を作成した後で、テキスト エディタ、XML エディタ、または Administration Console を使ってデプロイメント記述子を編集し、リソース アダプタのコンフィグレーションを完全なものにしてください。

ANT ユーティリティを使用してデプロイメント記述子を作成する方法の詳細については、「リソース アダプタのパッケージ化」を参照してください。

リソース アダプタのデプロイメント記述子エディタ

WebLogic Server の Administration Console には、統合されたデプロイメント記述子エディタがあります。この統合エディタを使用する前に、少なくともスケルトン ra.xml デプロイメント記述子を作成しておく必要があります。詳細については、付録 A「weblogic-ra.xml デプロイメント記述子の要素」を参照してください。

XML エディタ

BEA では、XML ファイルの作成と編集のために簡単で使いやすい Ensemble のツールを用意しました。このツールを使うと、指定した DTD または XML スキーマに従って XML コードの有効性を検証できます。この XML エディタは、Windows または Solaris のマシンで使用でき、BEA dev2dev からダウンロードできます。

リソース アダプタのコンフィグレーション

この節では、WebLogic Server にデプロイするためのリソース アダプタをコンフィグレーションする方法について説明します。

リソース アダプタの概要

WebLogic J2EE コネクタ アーキテクチャを使用すると、エンタープライズ情報システム（EIS）ベンダとサードパーティ アプリケーション開発者は、Sun Microsystems の J2EE プラットフォーム仕様、バージョン 1.3 に準拠しているアプリケーション サーバにデプロイ可能なリソース アダプタを開発できます。

リソース アダプタは WebLogic J2EE コネクタ アーキテクチャの中核をなすもので、クライアント コンポーネントと EIS との間の J2EE コネクタとして機能します。リソース アダプタを WebLogic Server 環境にデプロイすると、リモート EIS システムにアクセスする堅牢な J2EE プラットフォーム アプリケーションを開発できるようになります。リソース アダプタには、Java コンポーネントに加えて、必要な場合には EIS との対話に必要なネイティブ コンポーネントが入っています。

リソース アダプタの作成については、Sun Microsystems の J2EE コネクタ アーキテクチャのページと「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」を参照してください。これらの参照先は Sun Microsystems の Web サイトで公開されており、それぞれの URL は以下のとおりです。

http://java.sun.com/j2ee/connector/
http://java.sun.com/j2ee/download.html#connectorspec

リソース アダプタの作成と変更：主な手順

リソース アダプタを作成するには、個々のリソース アダプタ用のクラス（ConnectionFactory や Connection など）とコネクタ固有のデプロイメント記述子を作成してから、それらを WebLogic Server にデプロイする `jar` ファイルにすべてパッケージ化する必要があります。

新規リソース アダプタ（.rar）の作成

リソース アダプタ（`.rar`）を作成する主な手順を以下に説明します。

1. 「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」（http://java.sun.com/j2ee/download.html#connectorspec）に準拠して、リソース アダプタ（ConnectionFactory や Connection など）に必要な各種クラスの Java コードを記述します。

   リソース アダプタを実装する場合は、以下のように `ra.xml` ファイルでクラスを指定しなければなりません。

   - `<managedconnectionfactory-class>com.sun.connector.blackbox.LocalTxManagedConnectionFactory</managedconnectionfactory-class>`
   - `<connectionfactory-interface>javax.sql.DataSource</connectionfactory-interface>`
   - `<connectionfactory-impl-class>com.sun.connector.blackbox.JdbcDataSource</connectionfactory-impl-class>`
   - `<connection-interface>java.sql.Connection</connection-interface>`
   - `<connection-impl-class>com.sun.connector.blackbox.JdbcConnection</connection-impl-class>`

2. インタフェースと実装の Java コードをクラス ファイルにコンパイルします。

   コンパイルの詳細については、『WebLogic Server アプリケーションの開発』の「コンパイルの準備」を参照してください。

3. Java クラスを Java アーカイブ（.jar）ファイルにパッケージ化します。

4. リソース アダプタ固有のデプロイメント記述子を作成します。

   - ra.xml は、Sun Microsystems の標準 DTD を使用して、リソース アダプタ関連の属性タイプとそのデプロイメント プロパティを記述します。
   - weblogic-ra.xml ファイルは、WebLogic Server 固有のデプロイメント情報を追加します。

   詳細については、「ra.xml ファイルのコンフィグレーション」および 5-12 ページの「weblogic-ra.xml ファイルのコンフィグレーション」を参照してください。

   注意： リソース アダプタ .rar が weblogic-ra.xml ファイルを含んでいない場合、WebLogic Server はこのファイルを自動的に作成します。詳細については、「weblogic-ra.xml ファイルの自動生成」を参照してください。

5. リソース アダプタのアーカイブ ファイル（.rar ファイル）を作成します。

   a. 最初に、空のステージング ディレクトリを作成します。

   b. リソース アダプタの Java クラスが入った .rar ファイルをステージング ディレクトリに格納します。

   c. デプロイメント記述子を META-INF というサブディレクトリに格納します。

   d. 次に、ステージング ディレクトリで次のように jar コマンドを実行して、リソース アダプタのアーカイブを作成します。

      ```
      jar cvf myRAR.rar *
      ```

      リソース アダプタのアーカイブ ファイルの作成については、5-11 ページの「リソース アダプタ（.rar）のパッケージ化」を参照してください。

6. .rar リソース アダプタ ファイルを WebLogic Server にデプロイするか、またはエンタープライズ アプリケーションの一部としてデプロイするエンタープライズ アーカイブ（.ear）に含めます。

コンポーネントとアプリケーションのデプロイの詳細については、「WebLogic Server コンポーネントとアプリケーションのパッケージ化」の「アプリケーションとコンポーネントのデプロイ」を参照してください。

既存のリソース アダプタ（.rar）の変更

以下は、既存のリソース アダプタ（.rar）を、WebLogic Server にデプロイするために変更する方法の例です。この場合、デプロイメント記述子 weblogic-ra.xml を追加し、再パッケージ化する必要があります。

1. リソース アダプタをステージングするための一時ディレクトリを作成します。

   ```
   mkdir c:/stagedir
   ```

2. 一時ディレクトリにデプロイするリソース アダプタをコピーします。

   ```
   cp blackbox-notx.rar c:/stagedir
   ```

3. リソース アダプタ アーカイブの中身を展開します。

   ```
   cd c:/stagedir
   jar xf blackbox-notx.rar
   ```

   ステージング ディレクトリには、以下のものが格納されます。

   - リソース アダプタを実装する Java クラスが入った jar ファイル
   - Manifest.mf および ra.xml ファイルが入った META-INF ディレクトリ

   以下のコマンドを実行してこれらのファイルを確認します。

   ```
   c:/stagedir> ls
          blackbox-notx.jar
          META-INF
   c:/stagedir> ls META-INF
          Manifest.mf
          ra.xml
   ```

4. weblogic-ra.xml ファイルを作成します。このファイルは、リソース アダプタ用の WebLogic 固有のデプロイメント記述子です。このファイルには、接続ファクトリ、接続プール、およびセキュリティ マッピングのパラメータを指定します。

注意： リソース アダプタ .rar が weblogic-ra.xml ファイルを含んでいない場合、WebLogic Server はこのファイルを自動的に作成します。詳細については、「weblogic-ra.xml ファイルの自動生成」を参照してください。

weblogic-ra.xml ファイルの詳細については、5-13 ページの「weblogic-ra.xml ファイルのコンフィグレーション」および付録 A「weblogic-ra.xml デプロイメント記述子の要素」を参照してください。

5. weblogic-ra.xml ファイルを一時ディレクトリの META-INF サブディレクトリにコピーします。META-INF ディレクトリは、.rar ファイルを展開した一時ディレクトリ、またはリソース アダプタを展開ディレクトリ形式で格納しているディレクトリ内にあります。次のコマンドを使用します。

```
cp weblogic-ra.xml c:/stagedir/META-INF
c:/stagedir> ls META-INF
       Manifest.mf
       ra.xml
       weblogic-ra.xml
```

6. リソース アダプタ アーカイブを作成します。

```
jar cvf blackbox-notx.jar -C c:/stagedir
```

7. WebLogic Server にリソース アダプタをデプロイします。WebLogic Server へのリソース アダプタのデプロイについては、第 7 章「リソース アダプタのデプロイメント」を参照してください。

weblogic-ra.xml ファイルの自動生成

リソース アダプタ .rar が weblogic-ra.xml ファイルを含んでいない場合、WebLogic Server はこのファイルを自動的に作成します。この機能により、大幅な変更を加えなくてもサードパーティ製リソース アダプタを WebLogic Server にデプロイすることができます。変更の必要があるのは、WebLogic Server が weblogic-ra.xml ファイルに生成する 2 つのデフォルト属性の値、<connection-factory-name> と <jndi-name> だけです。

- WebLogic Server は、デフォルト値 __TMP_CFNAME_ の前に <connection-factory-name> を付加します。
- デフォルト値 __TMP_JNDINAME_ の前には <jndi-name> が付加されます。

これらのデフォルト値の変更方法については、付録 A「weblogic-ra.xml デプロイメント記述子の要素」の「Administration Console デプロイメント記述子エディタを使用したファイルの編集」を参照してください。

生成された weblogic-ra.xml ファイルは、デフォルト値を変更する前には以下のようになります。

コード リスト 5-1 weblogic-ra.xml のデフォルト値

```
<weblogic-connection-factory-dd>
<connection-factory-name>__TMP_CFNAME_.\config\mydomain\applicati
ons\whitebox-notx.rar</connection-factory-name>
<jndi-name>__TMP_JNDINAME_.\config\mydomain\applications\whitebox
-notx.rar</jndi-name>
       <pool-params>
              <initial-capacity>0</initial-capacity>
              <max-capacity>1</max-capacity>
              <capacity-increment>1</capacity-increment>
              <shrinking-enabled>false</shrinking-enabled>
              <shrink-period-minutes>200</shrink-period-minutes>
       </pool-params>
       <security-principal-map>
       </security-principal-map>
</weblogic-connection-factory-dd>
```

ra-link-ref 要素のコンフィグレーション

オプションの <ra-link-ref> 要素を使用すると、デプロイ済みの複数のリソース アダプタを 1 つのデプロイ済みリソース アダプタに関連付けることができます。つまり、属性のサブセットを変更するだけで、基本リソース アダプタでコンフィグレーションされているリソースを別のリソース アダプタにリンク（再利用）できます。<ra-link-ref> 要素により、可能な場合にはリソース（クラス、.jar ファイル、画像ファイルなど）の重複を避けることができます。デプ

ロイ済みの基本リソース アダプタで定義されている値はすべて、<ra-link-ref> 要素でそれ以外の値が指定されていない限り、リンク先のリソース アダプタが継承します。

オプションの <ra-link-ref> 要素を使用する場合は、<pool-params> 要素のすべての値を指定するか、まったく指定しないかのどちらかです。<pool-params> 要素は、基本リソース アダプタからリンク先のリソース アダプタに部分的には継承されません。

以下のいずれかを実行します。

- Administration Console のデプロイメント記述子エディタを使用して、<max-capacity> 要素に値 0（ゼロ）を割り当てます。これにより、リンク先のリソース アダプタは基本リソース アダプタから <pool-params> 要素の値を継承できます。
- <max-capacity> 要素に 0（ゼロ）以外の任意の値を割り当てます。リンク先リソース アダプタは、基本リソース アダプタから値を継承しなくなります。このオプションを選択する場合は、リンク先リソース アダプタの <pool-params> 要素のすべての値を指定する必要があります。

weblogic-ra.xml ファイルの編集の詳細については、付録 A「weblogic-ra.xml デプロイメント記述子の要素」の「Administration Console デプロイメント記述子エディタを使用したファイルの編集」を参照してください。

パッケージ化のガイドライン

リソース アダプタは .rar アーカイブ ファイルに入った WebLogic Server コンポーネントで、applications\ ディレクトリにあります。デプロイメント プロセスは、リソース アダプタ プロバイダによって作成されたコンパイル済みリソース アダプタ インタフェースと実装クラスを格納する .rar ファイルまたはデプロイメント ディレクトリで開始されます。.rar ファイルとデプロイメント ディレクトリは、どちらがコンパイル済みクラスを格納している場合でも、Java パッケージ構造と一致するサブディレクトリに入っている必要があります。

リソース アダプタは、共通のディレクトリ形式を使用します。この形式は、リソース アダプタを .rar ファイルとして展開ディレクトリ形式でパッケージ化するときにも使用されます。リソース アダプタの構造の例を示します。

コード リスト 5-2 リソース アダプタの構造

```
\META-INF\ra.xml
\META-INF\weblogic-ra.xml

\images\ra.jpg

\readme.html
\eis.jar
\utilities.jar
\windows.dll
unix.so
```

リソース アダプタ内のファイルについては以下の点に注意してください。

- デプロイメント記述子（ra.xml と weblogic-ra.xml）は、META-INF というサブディレクトリに入っていなければなりません。
- リソース アダプタには、リソース アダプタが使用する Java クラスおよびインタフェースを格納する複数の jar ファイル（たとえば eis.jar や utilities.jar）を含めることができます。
- リソース アダプタには、EIS との対話用にリソース アダプタが必要とするネイティブ ライブラリ（たとえば windows.dll や unix.so）を含めることができます。

リソース アダプタには、マニュアルやリソース アダプタが直接には使用しない関連ファイル（たとえば readme.html や \images\ra.jpg）を含めることができます。

リソース アダプタ（.rar）のパッケージ化

1 つまたは複数のリソース アダプタを 1 つのディレクトリにステージングして、それらを jar ファイルにパッケージ化できます。リソース アダプタをパッケージ化する前に、WebLogic Server がクラスをロードする方法について説明した『WebLogic Server コンポーネントとアプリケーションのパッケージ化』の「コンポーネント間のクラス参照の解決」を一読してください。

リソース アダプタをステージングおよびパッケージ化するには、次の手順に従います。

1. 一時的なステージング ディレクトリを作成します。
2. 対象となるリソース アダプタの Java クラスをステージング ディレクトリにコンパイルまたはコピーします。
3. リソース アダプタの Java クラスを入れる .jar ファイルを作成します。この .jar ファイルをステージング ディレクトリの最上位に追加します。
4. ステージング ディレクトリに META-INF サブディレクトリを作成します。
5. META-INF サブディレクトリに ra.xml デプロイメント記述子を作成して、そのリソース アダプタのエントリを追加します。

   注意： ra.xml 文書型定義の詳細については、Sun Microsystems のドキュメント（http://java.sun.com/dtd/connector_1_0.dtd）を参照してください。

6. META-INF サブディレクトリに weblogic-ra.xml デプロイメント記述子を作成して、そのリソース アダプタのエントリを追加します。

   注意： weblogic-ra.xml 文書型定義の詳細については、付録 A「weblogic-ra.xml デプロイメント記述子の要素」を参照してください。

7. すべてのリソース アダプタ クラスとデプロイメント記述子をステージング ディレクトリに配置すると、次のような jar コマンドを使用してリソース アダプタ JAR ファイルを作成できます。

   ```
   jar cvf jar-file.rar -C staging-dir.
   ```

   このコマンドによって作成された jar ファイルは、WebLogic Server にデプロイすることも、またはアプリケーション JAR ファイルにパッケージ化することもできます。

-C staging-dir オプションを指定すると、jar コマンドはディレクトリを staging-dir に変更します。これにより、JAR ファイルに記録されるディレクトリ パスがリソース アダプタのステージング ディレクトリを基準にした相対パスとなります。

リソース アダプタを作成する方法と、WebLogic Server へのデプロイ用に既存のリソース アダプタを変更する方法については、5-4 ページの「リソース アダプタの作成と変更 : 主な手順」を参照してください。

ra.xml ファイルのコンフィグレーション

ra.xml ファイルがない場合は、手動で作成するか、または既存のファイルを編集して、リソース アダプタに必要なデプロイメント プロパティを設定します。プロパティの編集には、テキスト エディタを使用します。ra.xml ファイルの作成の詳細については、「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」（http://java.sun.com/j2ee/download.html#connectorspec）を参照してください。

weblogic-ra.xml ファイルのコンフィグレーション

weblogic-ra.xml ファイルには、リソース アダプタを WebLogic Server にデプロイするために必要な情報が入ります。このファイルには特定の属性を指定します。このファイルは、WebLogic Server の EJB および Web アプリケーション用の拡張子 .xml のファイルと同じ機能を持つほか、WebLogic 固有のデプロイ記述子をデプロイ可能なアーカイブに追加するものです。

以下の節では、weblogic-ra.xml ファイルをコンフィグレーションして、WebLogic Server にデプロイするために WebLogic Server 固有の内容を定義する方法について説明します。

weblogic-ra.xml ファイルのコンフィグレーション

基本の .jar またはデプロイメント ディレクトリは、そのまま WebLogic Server にデプロイすることができません。最初に weblogic-ra.xml ファイルで、WebLogic Server 固有のデプロイメント プロパティを作成およびコンフィグレーションし、その XML ファイルをデプロイメントに追加する必要があります。

weblogic-ra.xml ファイルでは、接続ファクトリ、接続プール パラメータ、セキュリティ プリンシパル マッピングなどを定義します。このファイルで設定可能なプロパティの一覧については、付録 A「weblogic-ra.xml デプロイメント記述子の要素」を参照してください。

コンフィグレーション可能な weblogic-ra.xml のエンティティ

weblogic-ra.xml ファイルには、リソース アダプタを WebLogic Server にデプロイするために必要な情報が入ります。このファイルには以下の属性を指定します。

- 接続ファクトリの名前
- 接続ファクトリの説明用テキスト
- 接続ファクトリにバインドされる JNDI 名
- 現在のリソース アダプタと共有可能なリソース アダプタ コンポーネントを含み、別にデプロイされた接続ファクトリへの参照
- すべての共有ライブラリをコピーするディレクトリ
- 以下の動作を設定する接続プール パラメータ
  - WebLogic Server がデプロイ時に割り当てようとする管理対象接続の初期数
  - WebLogic Server が一度に割り当て可能な管理対象接続の最大数
  - WebLogic Server が新規接続のリクエストに応じるときに割り当てようとする管理対象接続の数

- システム リソースを節約するために WebLogic Server が未使用の管理対象接続を再利用しようとするかどうか
- 未使用の管理対象接続の再利用を試みるまで WebLogic Server が待機する時間
- 接続が設定されている使用時間を超えたことを検出し、接続を再利用する頻度
- 1 つの接続に許容する使用時間

■ J2EE リソース アダプタ デプロイメント記述子 `ra.xml` の `<config-entry>` 要素で定義するコンフィグレーション プロパティの値

■ リソース アダプタ /EIS サインオン処理用のセキュリティ プリンシパルのマッピング。このマッピングは、コンテナ管理によるセキュリティを使用するアプリケーション用の EIS 接続を要求する場合と、初期デプロイメントで要求される EIS 接続用に使用するリソース プリンシパルを識別します。

■ ManagedConnectionFactory または ManagedConnection に関するロギングが必要かどうかを示すフラグ

■ ManagedConnectionFactory または ManagedConnection に関するロギング情報を保存するファイル

注意： `weblogic-ra.xml` のパラメータの設定については、付録 A「weblogic-ra.xml デプロイメント記述子の要素」の `weblogic-ra.xml` DTD を参照してください。ダウンロード製品に付属する Simple Black Box リソース アダプタ例に含まれている `weblogic-ra.xml` ファイルを参照することもできます。

注意： リソース アダプタの接続プロパティのコンフィグレーションの詳細については、第 4 章「接続管理」を参照してください。

セキュリティ プリンシパル マップのコンフィグレーション

コンテナ管理によるサインオンを使用するには、WebLogic Server がリソース プリンシパルを識別してから、リソース プリンシパルに代わって EIS 接続を要求しなければなりません。WebLogic Server は、weblogic-ra.xml デプロイメント記述子ファイルの <security-principal-map> 要素で指定されているセキュリティ プリンシパル マップを探してリソース プリンシパルを識別します。

このマップによって、WebLogic Server 開始プリンシパル（WebLogic Server セキュリティ レルムで定義された ID を持つユーザ）がリソース プリンシパル（リソース アダプタ /EIS システムに登録済みユーザ）と関連付けられます。

また、<security-principal-map> を使用すると、実行時に識別された開始プリンシパルがマップ内に見つからない場合に適切なリソース プリンシパルにマップするデフォルトの開始プリンシパルを定義できます。以下のように、値が * の <initiating-principal> 要素を付けた <security-principal-map> でデフォルトの開始プリンシパルを設定します。

```
<initiating-principal>*</initiating-principal>
```

ユーザ名およびパスワードを指定する <security-principal-map> 要素には対応する <resource-principal> エントリも含める必要があります。

次の例は、WebLogic Server 開始プリンシパルとリソース プリンシパルとの関連付けを示します。

コード リスト 5-3 <initiating-principal> および <resource-principal> エントリの例

```
<security-principal-map>
   <map-entry>
       <initiating-principal>*</initiating-principal>
          <resource-principal>
              <resource-username>default</resource-username>
```

```
                <resource-password>try</resource-password>
            </resource-principal>
    </map-entry>
</security-principal-map>
```

WebLogic Server が接続を初期化するよう接続プール パラメータで設定されている場合、デフォルトの開始プリンシパルはデプロイ時にも使用されます。デフォルトの開始プリンシパルのエントリがない場合、または <security-principal-map> 要素がない場合、WebLogic Server はコンテナ管理によるセキュリティを使用して接続を作成しません。

パスワード変換ツールの使い方

WebLogic Server のリソース アダプタ用の現在のコンフィグレーションおよびパッケージ化の要件では、weblogic-ra.xml ファイルを手動で編集する必要があるので、security-principal-map エントリで指定する新規のパスワードはクリアテキストで指定されます。

BEA では、セキュリティ パスワード保護の重要性を考慮して、weblogic-ra.xml ファイルで提示されるすべてのパスワードを暗号化できる変換ツールを用意しています。変換ツールは、標準の weblogic.jar ファイルで提供されます。

リソース アダプタが WebLogic Server 環境にないクリアテキストのパスワードを持つ必要がある場合は、新規クリアテキスト パスワードが追加されるたびに変換ツールを使用して、追加後の weblogic-ra.xml ファイルを後処理しなければなりません。

付属のパスワード変換ツールを実行して、クリアテキストのリソース パスワード値すべてを暗号化されたパスワード値に変換する必要があります。この変換ツールは、クリアテキスト パスワードを含む既存の weblogic-ra.xml ファイルを解析し、暗号化されたパスワードを含む新しい weblogic-ra.xml ファイルを作成します。この新しいファイルは、WebLogic Server へのデプロイメント用に .rar ファイルにパッケージするファイルとなります。

実行方法

変換ツールを実行するには、DOS コマンド シェルで次の構文を使用します。

コード リスト 5-4　変換ツールの構文

```
java weblogic.Connector.ConnectorXMLEncrypt
 <input-weblogic-ra.xml> <output-weblogic-ra.xml>
 <domain-config-directory-location>
```

セキュリティのヒント

暗号化 / 解読処理に使用するドメイン固有のセキュリティのヒントでは、<domain config directory location> を含める必要があります。変換ツールにはこのドメイン固有のヒントを使用するよう指示しなければなりません。暗号化されたパスワードはこのドメインに固有のものです。したがって、暗号化されたパスワード値を持つ .rar は、そのドメインにのみデプロイ可能となります。

トランザクション レベル タイプのコンフィグレーション

リソース アダプタがサポートするトランザクション レベル タイプを ra.xml デプロイメント記述子ファイルに指定する必要があります。トランザクションのサポート レベルを指定するには次の手順に従います。

- トランザクション非サポートの場合、次のエントリを ra.xml デプロイメント記述子ファイルに追加します。
  <transaction-support>NoTransaction</transaction-support>
- XA トランザクションの場合、次のエントリを ra.xml デプロイメント記述子ファイルに追加します。
  <transaction-support>XATransaction</transaction-support>

- ローカル トランザクションの場合、次のエントリを ra.xml デプロイメント記述子ファイルに追加します。
  <transaction-support>LocalTransaction</transaction-support>

.xml ファイルの編集の詳細については、付録 A「weblogic-ra.xml デプロイメント記述子の要素」の「XML デプロイメント ファイルの手動による編集」および「Administration Console デプロイメント記述子エディタを使用したファイルの編集」を参照してください。

.rar コンフィグレーションでのトランザクション レベルの指定方法については、「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」（http://java.sun.com/j2ee/download.html#connectorspec）で「Packaging and Deployment」の「Resource Adapter XML DTD」を参照してください。

6 J2EE コネクタ アーキテクチャに準拠したリソース アダプタの作成

以下の節では、「J2EE プラットフォーム仕様、バージョン 1.3、最終草案 3」（http://java.sun.com/j2ee）に準拠したリソース アダプタの開発に必要な条件について説明します。以下の節は、この仕様で規定されているシステム規約の要件に対応しています。

- 接続管理
- セキュリティ管理
- トランザクション管理
- パッケージ化とデプロイメント

また、Weblogic 固有の制限事項についても取り上げます。

注意： リソース アダプタを構築する手順については、http://edocs.beasys.co.jp/e-docs/wlintegration/v2_1/devadapt/index.htm の BEA WebLogic Application Integration のマニュアルを参照してください。

接続管理

リソース アダプタに関する接続管理規約の要件は以下のとおりです。

- リソース アダプタは、以下のインタフェース実装を提供する必要があります。
  - `javax.resource.spi.ManagedConnectionFactory`
  - `javax.resource.spi.ManagedConnection`
  - `javax.resource.spi.ManagedConnectionMetaData`
- リソース アダプタで提供する `ManagedConnection` 実装は、以下のインタフェースとクラスを使用して、接続管理（および後述のトランザクション管理）用のサポートをアプリケーション サーバに提供する必要があります。
  - `javax.resource.spi.ConnectionEvent`
  - `javax.resource.spi.ConnectionEventListener`

  非管理対象の環境をサポートするには、リソース アダプタが上記 2 つのインタフェースを使用しなくても内部オブジェクトの対話を実行できます。
- リソース アダプタは、以下のメソッドを実装することで、基本的なエラーロギングおよび追跡機能をサポートする必要があります。
  - `ManagedConnectionFactory.set/getLogWriter`
  - `ManagedConnnection.set/getLogWriter`
- リソース アダプタは、`javax.resource.spi.ConnectionManager` インタフェースのデフォルト実装を提供する必要があります。この実装クラスは、リソース アダプタを非管理対象の 2 層アプリケーションで使用する場合に機能します。アプリケーション サーバの管理対象の環境では、リソース アダプタはデフォルト `ConnectionManager` 実装クラスを使用してはなりません。

  `ConnectionManager` のデフォルト実装を使用すると、リソース アダプタは固有のサービスを提供できます。これらのサービスには、接続プール、エラーのロギングおよび追跡、セキュリティ管理などがあります。デフォルトの `ConnectionManager` は、基底の EIS との物理接続の作成を `ManagedConnectionFactory` に委託します。
- 管理対象の環境では、リソース アダプタが独自の内部接続プールをサポートしてはなりません。この場合、アプリケーション サーバが接続プールを処理します。ただし、リソース アダプタは、単一の物理パイプ上でアプリケー

ション サーバおよびコンポーネントに対して透過的に接続を多重化（物理接続ごとに 1 つまたは複数の ConnectionManager インスタンスを作成）することができます。

非管理対象の 2 層アプリケーションの場合、リソース アダプタはリソース アダプタ内部用の接続プールをサポートできます。

セキュリティ管理

リソース アダプタに関するセキュリティ管理規約の要件は以下のとおりです。

- リソース アダプタは、メソッド ManagedConnectionFactory.createManagedConnection を実装することでセキュリティ規約をサポートする必要があります。
- リソース アダプタは、ManagedConnection.getConnection メソッド実装の一部として再認証をサポートする必要がありません。
- リソース アダプタは、デプロイメント記述子の一部としてセキュリティ規約のサポートを指定する必要があります。関連するデプロイメント記述子の要素は、authentication-mechanism、authentication-mechanism-type、reauthentication-support、および credential-interface です。「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」（http://java.sun.com/j2ee/download.html#connectorspec）の 10.6 節「Resource Adapter XML DTD」を参照してください。

トランザクション管理

この節では、リソース アダプタに関するトランザクション管理規約の要件について説明します。リソース アダプタは、トランザクション サポートのレベルに基づいて以下のように分けられます。

- レベル NoTransaction — リソース アダプタはリソース マネージャのローカル トランザクションも JTA トランザクションもサポートしません。リソース アダプタは、XAResource インタフェースも LocalTransaction インタフェースも実装しません。

- レベル `LocalTransaction` — リソース アダプタは、`LocalTransaction` インタフェースの実装によってリソース マネージャのローカル トランザクションをサポートします。ローカル トランザクション管理規約は、「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」（http://java.sun.com/j2ee/download.html#connectorspec）の 6.7 節で指定されています。
- レベル `XATransaction` — リソース アダプタは、`LocalTransaction` および `XAResource` インタフェースの実装によってリソース マネージャのローカル トランザクションと JTA トランザクションの両方をサポートします。XAResource ベースの規約の要件は、「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」（http://java.sun.com/j2ee/download.html#connectorspec）の 6.6 節で指定されています。

注意： その他のサポート レベル（基底のリソース マネージャがサポートするトランザクション最適化を含む）は、コネクタ アーキテクチャの適用範囲外です。

上記のレベルは、外部トランザクションの調整を可能にするためにリソース アダプタが必要とするトランザクション サポートの主なステップを反映しています。リソース アダプタのトランザクション機能と基底の EIS の要件に従って、リソース アダプタでは上記のトランザクション サポート レベルのいずれもサポート対象として選択できます。

パッケージ化とデプロイメント

以下の節では、リソース アダプタに関するパッケージ化とデプロイメントの要件について説明します。

制限

現時点では、WebLogic J2EE コネクタ アーキテクチャでは、シリアライズ可能な ConnectionFactory 実装のみがサポートされています（参照のみ可能な ConnectionFactory 実装はサポートされていません）。

また、WebLogic J2EE コネクタ アーキテクチャは、`javax.resource.spi.security.GenericCredential` credential-interface も Kerbv5 `authentication-mechanism-type` もサポートしていません。デプロイされているリソース アダプタの `ra.xml` ファイルで、`<authentication-mechanism>` にこれらの値のいずれかを指定すると、デプロイメントが失敗します。

パッケージ化

パッケージ化されたリソース アダプタ モジュールのファイル形式は、リソース アダプタ プロバイダとデプロイヤとの間の規約を定義します。パッケージ化されたリソース アダプタには以下の要素が含まれます。

- コネクタ アーキテクチャの規約とリソース アダプタの機能を実装するために必要な Java クラスおよびインタフェース
- リソース アダプタのユーティリティ Java クラス
- リソース アダプタが必要とするプラットフォーム依存ネイティブ ライブラリ
- ヘルプ ファイルとマニュアル
- 上記の要素をまとめた説明用のメタ情報

デプロイメント

パッケージ化の要件の詳細については、「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」（http://java.sun.com/j2ee/download.html#connectorspec）の 10.3.1 節「Resource Adapter Provider」および 10.5.1 節「Responsibilities」を参照してください。10.3.1 節ではデプロイメントの要件について、10.5.1 節では JNDI コンフィグレーションおよびルックアップのサポート方法について説明しています。

7 リソース アダプタのデプロイメント

以下の節では、コンフィグレーション済みのリソース アダプタのデプロイメントを WebLogic Server にデプロイ、アンデプロイ、および更新する方法について説明します。

- リソース アダプタのデプロイメントの概要
- Administration Console の使い方
- アプリケーション ディレクトリの使い方
- weblogic.deploy の使い方
- エンタープライズ アプリケーション（.ear ファイル）へのリソース アダプタの追加

リソース アダプタのデプロイメントの概要

リソース アダプタのデプロイメントは、Web アプリケーション、EJB、およびエンタープライズ アプリケーションのデプロイメントとほぼ同じです。これらのデプロイメント ユニットと同様、リソース アダプタも展開ディレクトリ形式でデプロイしたり、アーカイブ ファイルとしてデプロイしたりすることができます。

デプロイメント オプション

リソース アダプタは以下の方法でデプロイできます。

- コマンド ラインまたは Administration Console を使用して動的にデプロイする
- WebLogic Server の実行中に、アーカイブ ファイルまたは展開ディレクトリを WebLogic Server ドメインの `applications` ディレクトリにコピーして自動的にデプロイする
- `.ear` というアーカイブ ファイルとしてデプロイされるエンタープライズ アプリケーションの一部としてデプロイする

デプロイメント記述子

Web アプリケーション、EJB、およびエンタープライズ アプリケーションと同様、リソース アダプタは 2 つのデプロイメント記述子を使用して操作パラメータを定義します。デプロイメント記述子 `ra.xml` は、Sun Microsystems の「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」で定義されています。
`weblogic-ra.xml` デプロイメント記述子は、WebLogic Server に固有のもので、WebLogic Server に対してのみ有効な操作パラメータを定義します。
`weblogic-ra.xml` デプロイメント記述子の詳細については、付録 A 「weblogic-ra.xml デプロイメント記述子の要素」を参照してください。

リソース アダプタのデプロイメント名

リソース アダプタの .rar ファイルまたはデプロイメント ディレクトリをデプロイする場合は、myResourceAdapter のように、デプロイメント ユニットの名前を指定する必要があります。この名前を使用すると、後でリソース アダプタをアンデプロイしたり更新したりする場合に、リソース アダプタのデプロイメントを簡単に参照できます。

リソース アダプタをデプロイする場合は、WebLogic Server が、.rar ファイルまたはデプロイメント ディレクトリのパスおよびファイル名と一致するデプロイメント名を明示的に割り当てます。この名前を使用すると、サーバが起動した後にリソース アダプタをアンデプロイまたは更新できます。

リソース アダプタのデプロイメント名は、サーバが再起動されるまで、WebLogic Server 内でアクティブなままです。リソース アダプタをアンデプロイしても、関連付けられたデプロイメント名は削除されません。リソース アダプタをデプロイするために後でその名前を使う場合があるからです。

Administration Console の使い方

この節では、Administration Console を使用したリソース アダプタのデプロイメント作業について説明します。

Administration Console を使用したリソース アダプタのデプロイ

WebLogic Server Administration Console を使用してリソース アダプタをデプロイするには次の手順に従います。

1. WebLogic Server を起動します。
2. Administration Console を起動します。
3. 作業を行うドメインを開きます。

4. 左ペインで、[デプロイメント] の下の [コネクタ] を選択します。デプロイ済みのコネクタ（リソース アダプタ）が右ペインの [リソース コネクタ] テーブルに表示されます。
5. [新しい Connector Component のコンフィグレーション] を選択します。
6. 以下の情報を入力します。
   - [名前] ― 必要に応じてコネクタ コンポーネントのデフォルト名を変更します。
   - [Path] ― リソース アダプタ `.rar` ファイルへの絶対パス、または展開ディレクトリ形式のリソース アダプタが格納されているディレクトリを入力します。例：`c:\myaps\components\myResourceAdapter.rar`
   - [デプロイ] ― リソース アダプタの `.rar` ファイルを作成時にデプロイするかどうかを示します。
7. [作成] ボタンをクリックします。
8. 新しいリソース アダプタが右ペインの [リソース コネクタ] テーブルに表示されるようになりました。

Administration Console を使用したデプロイ済みリソース アダプタの参照

デプロイ済みのリソース アダプタを Administration Console で参照するには次の手順に従います。

1. Administration Console の左ペインで、[デプロイメント] の下の [コネクタ]（リソース アダプタ）を選択します。
2. 右ペインの [リソース コネクタ] テーブルでデプロイ済みのコネクタのリストを参照します。

Administration Console を使用したデプロイ済みリソース アダプタのアンデプロイ

WebLogic Server Administration Console からデプロイ済みのリソース アダプタをアンデプロイするには次の手順に従います。

1. Administration Console の左ペインで、[デプロイメント] の下の [コネクタ]（リソース アダプタ）を選択します。
2. [リソース コネクタ] テーブルでアンデプロイするコネクタを選択します。
3. [コンフィグレーション] タブで、[デプロイ] チェックボックスの選択を解除します。
4. [適用] をクリックします。

リソース アダプタをアンデプロイしても、リソース アダプタ名は WebLogic Server から削除されません。リソース アダプタは、Server セッションが終了するまでアンデプロイされた状態が続きます。ただし、アンデプロイ後にリソース アダプタを変更した場合を除きます。サーバを再起動するまで、deploy 引数でデプロイメント名を再利用することはできません。次の節で説明するように、デプロイメントの更新にそのデプロイメント名を再使用できます。

Administration Console を使用したデプロイ済みリソース アダプタの更新

WebLogic Server にデプロイ済みのリソース アダプタの `.rar` ファイルまたはデプロイメント ディレクトリの内容を更新した場合、更新内容は以下のいずれかを実行するまで WebLogic Server に反映されません。

- サーバを再起動します（`.rar` またはディレクトリを自動的にデプロイする場合）。

  または

- WebLogic Server Administration Console を使用してリソース アダプタのデプロイメントを更新します。

WebLogic Server の Administration Console を使用する場合

1. Administration Console の左ペインで、[デプロイメント] の下の [コネクタ]（リソース アダプタ）を選択します。
2. [リソース コネクタ] テーブルで更新するコネクタを選択します。
3. 必要に応じてコネクタ名とデプロイ ステータスを更新します。
4. [適用] をクリックします。

アプリケーション ディレクトリの使い方

WebLogic Server の実行中に、リソース アダプタを自動的にデプロイすることができます。`\applications` ディレクトリは WebLogic Server の実行中にモニタされるので、新しい `.rar` が追加された場合（デプロイメント）、または既存の `.rar` が削除された場合（アンデプロイメント）には検出されます。

`applications` ディレクトリを使用してリソース アダプタをデプロイするには次の手順に従います。

1. リソース アダプタの入った `.rar` アーカイブまたは展開ディレクトリをドメインの `applications` ディレクトリにコピーします。

   たとえば、`myResourceAdapter` というリソース アダプタを展開形式でコピーしたら、WebLogic Server のインストール ディレクトリは次のようになります（リソース アダプタの一部のファイルは含まれていません）。

コード リスト 7-1 myResourceAdapter

```
\---beaHome
       \---wlserver6.x
              \---config
                     \---mydomain
                            \---applications
                                   \---myResourceAdapter
                                          eis.jar
                                          readme.html
                                          unix.so
                                          utilities.jar
                                          windows.dll
                                          \---META-INF
                                                 ra.xml
                                                 weblogic-ra.xml
```

.rar ファイルをコピーしたら、WebLogic Server のインストール ディレクトリは次のようになります。

コード リスト 7-2 rar ファイル

```
       \---beaHome
              \---wlserver6.x
                     \---config
                            \---mydomain
                                   \---applications
                                          myResourceAdapter.rar
```

2. WebLogic Server を起動します。起動すると WebLogic Server は、指定したリソース アダプタの .rar ファイルまたはデプロイメント ディレクトリを自動的にデプロイしようとします。
3. Administration Console を起動します。
4. 右ペインで、[Resource Adapter Deployments] をクリックします。
5. リソース アダプタがリストに入っており、[デプロイ] ボックスが選択されていることを確認します。

weblogic.deploy の使い方

WebLogic Server にデプロイされていないリソース アダプタの .rar ファイルまたはデプロイメント ディレクトリをデプロイするには、次のコマンドを使用します。

```
% java weblogic.deploy -port port_number -host host_name
deploy password name source
```

各値の説明は次のとおりです。

- *name* はこのリソース アダプタのデプロイメント ユニットに割り当てる名前が入った文字列です。
- *password* は WebLogic Server のシステム アカウントのパスワードです。
- *source* は、デプロイするリソース アダプタの .rar ファイルの絶対パスおよびファイル名、またはリソース アダプタのデプロイメント ディレクトリの絶対パスです。

weblogic.deploy を使用したデプロイ済みリソース アダプタの参照

ローカルの WebLogic Server にデプロイされているリソース アダプタをコマンド ラインから参照するには、次のように入力します。

```
% java weblogic.deploy list password
```

password は WebLogic Server のシステム アカウントのパスワードです。

リモート サーバのデプロイ済みリソース アダプタをコマンド ラインからリストするには、次のように入力します。

```
% java weblogic.deploy -port port_number -host host_name
list password
```

weblogic.deploy を使用したデプロイ済みリソース アダプタのアンデプロイ

リソース アダプタをアンデプロイしても、リソース アダプタの .rar ファイルまたはデプロイメント ディレクトリに関連付けられたデプロイメント名は削除されません。デプロイメント名は、後でリソース アダプタを更新することができるようにサーバ内に残されます。

デプロイ済みのリソース アダプタをコマンド ラインからアンデプロイするには、次のように、割り当てられているデプロイメント ユニット名を参照するだけです。

```
%java weblogic.deploy -port 7001 -host localhost undeploy
password myResourceAdapter
```

リソース アダプタをアンデプロイしても、リソース アダプタ名は WebLogic Server から削除されません。リソース アダプタは、Server セッションが終了するまでアンデプロイされた状態が続きます。ただし、アンデプロイ後にリソース アダプタを変更した場合を除きます。サーバを再起動するまで、deploy 引数でデプロイメント名を再利用することはできません。7-10 ページの

「weblogic.deploy を使用したデプロイ済みリソース アダプタの更新」で説明するように、デプロイメントを更新する場合には、デプロイメント名を再利用できます。

weblogic.deploy を使用したデプロイ済みリソース アダプタの更新

WebLogic Server にデプロイ済みのリソース アダプタの .rar ファイルまたはデプロイメント ディレクトリの内容を更新した場合、更新内容は以下のいずれかを実行するまで WebLogic Server に反映されません。

- サーバを再起動します（.rar またはディレクトリを自動的にデプロイする場合）。
- WebLogic Server Administration Console を使用してリソース アダプタのデプロイメントを更新します。

コマンド ラインからリソース アダプタを更新または再デプロイするには、update 引数を使用して、アクティブなリソース アダプタのデプロイメント名を指定します。

```
%java weblogic.deploy -port 7001 -host localhost update
password myResourceAdapter
```

エンタープライズ アプリケーション（.ear ファイル）へのリソース アダプタの追加

「J2EE プラットフォーム仕様、バージョン 1.3、最終草案 3」に記載されているとおり、リソース アダプタ アーカイブ（.rar）ファイルをエンタープライズ アプリケーション アーカイブ（.ear）に含めてから、WebLogic Server にそのアプリケーションをデプロイできるようになりました。

リソース アダプタ アーカイブを含むエンタープライズ アプリケーションをデプロイするには次の手順に従います。

1. .war または .jar アーカイブと同じように、.rar ファイルを .ear アーカイブに入れます。
2. 有効な application.xml を作成し、.ear アーカイブの META-INF ディレクトリに格納します。

   application.xml を作成する際には以下の点に注意してください。

   アプリケーション デプロイメント記述子には、.ear アーカイブ内のリソース アダプタ アーカイブを識別するための新しい <connector> 要素を含める必要があります。次に例を示します。

   ```
   <connector>RevisedBlackBoxNoTx.rar</connector>
   ```

   <connector> は J2EE プラットフォーム仕様、バージョン 1.3 で新たに追加された要素なので、application.xml ファイルには、J2EE プラットフォーム仕様、バージョン 1.3 のデプロイメント記述子として識別するために次の DOCTYPE エントリを含める必要があります。

コード リスト 7-3　DOCTYPE エントリ

```
<!DOCTYPE application PUBLIC '-//Sun Microsystems, Inc.//DTD
J2EE Application 1.3//EN'
       'http://java.sun.com/dtd/application_1_3.dtd'>
```

   DOCTYPE エントリを含めなかった場合、リソース アダプタはデプロイされません。

   application.xml ファイルの例を以下に示します。

コード リスト 7-4　application.xml ファイル

```
<application>
       <display-name> ConnectorSampleearApp </display-name>
       <module>
              <connector>RevisedBlackBoxNoTx.rar</connector>
       </module>
```

```
        <module>
                <ejb>ejb_basic_beanManaged.jar</ejb>
        </module>
</application>
```

3. エンタープライズ アプリケーションを WebLogic Server にデプロイします。

   エンタープライズ アプリケーションのデプロイ作業の概要については、「WebLogic Server アプリケーションについて」の「エンタープライズ アプリケーション」を参照してください。

8　クライアントに関する考慮事項

以下の節では、WebLogic J2EE コネクタ アーキテクチャのクライアントに関する考慮事項について説明します。

- Common Client Interface（CCI）
- ConnectionFactory と接続
- ConnectionFactory（クライアントと JNDI 間の対話）の取得

Common Client Interface（CCI）

EIS にアクセスするためのアプリケーション コンポーネントが使用するクライアント API は、次のように定義されます。

- 「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」（http://java.sun.com/j2ee/download.html#connectorspec）の第 9 章「Common Client Interface」の標準 Common Client Interface（CCI）
- リソース アダプタおよび基底の EIS に固有のクライアント API。こうした EIS 固有のクライアント API の例として、リレーショナル データベース用の JDBC があります。

CCI は、EIS にアクセスするための共通クライアント API です。CCI は、エンタープライズ アプリケーション統合（EAI）およびエンタープライズ ツール ベンダを対象としています。

J2EE コネクタ アーキテクチャは EIS アクセス用の Common Client Interface（CCI）も定義します。CCI では、アプリケーション コンポーネント用の標準クライアント API を定義します。アプリケーション コンポーネントおよび EAI フレームワークは、CCI を使用することにより、異種 EIS 間の対話処理を制御できます。

ConnectionFactory と接続

接続ファクトリは EIS インスタンスに接続するためのパブリック インタフェースで、ConnectionFactory はリソース アダプタによって提供されるインタフェースです。アプリケーションは、JNDI ネームスペースで ConnectionFactory インスタンスをルックアップし、それを使って EIS 接続を取得します。

J2EE コネクタ アーキテクチャの目標の 1 つは、CCI と EIS 固有のクライアント API 間で一貫したアプリケーション プログラミング モデルをサポートすることです。このモデルを実現するために、ConnectionFactory インタフェースと Connection インタフェースの両方を対象としたインタフェース テンプレートとして指定されている設計パターンを使用します。

この設計パターンの詳細については、「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」（http://java.sun.com/j2ee/download.html#connectorspec）の 5.5.1 節「ConnectionFactory and Connection」を参照してください。

ConnectionFactory（クライアントと JNDI 間の対話）の取得

この節では、ConnectionFactory を使用して EIS との接続を取得する方法について説明します。詳細については、「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」（http://java.sun.com/j2ee/download.html#connectorspec）の 5.4.1 節「Managed Application Scenario」を参照してください。

管理対象アプリケーションでの接続の取得

管理対象アプリケーションが `res-type` 変数で指定したように ConnectionFactory から EIS インスタンスとの接続を取得する際には、以下のタスクを実行します。

1. アプリケーション アセンブラまたはコンポーネント プロバイダが、デプロイメント記述子のメカニズムを使用して、アプリケーション コンポーネントに関する接続ファクトリの要件を指定します。次に例を示します。
   - `res-ref-name: eis/myEIS`
   - `res-type: javax.resource.cci.ConnectionFactory`
   - `res-auth: Application` または `Container`
2. リソース アダプタのデプロイ担当者が、リソース アダプタのコンフィグレーション情報を設定します。
3. アプリケーション サーバが、コンフィグレーション済みのリソース アダプタを使用して、基底の EIS との物理接続を作成します。リソース アダプタのパッケージ化とデプロイメントの詳細については、「J2EE コネクタ仕様、バージョン 1.0、最終草案 2」（http://java.sun.com/j2ee/download.html#connectorspec）の 10 章を参照してください。

4. アプリケーション コンポーネントは、JNDI インタフェースを使用して、コンポーネントの環境内で接続ファクトリのインスタンスをルックアップします。

コード リスト 8-1 JNDI ルックアップ

```
// 初期 JNDI ネーミング コンテキストを取得する
Context initctc = new InitialContext();

// JNDI ルックアップを実行して、接続ファクトリを取得する
javax.resource.cci.ConnectionFactory cxf =
       (javax.resource.cci.ConnectionFactory)
              initctx.lookup("java:comp/env/eis/MyEIS");
```

メソッド NamingContext.lookup で渡される JNDI 名は、デプロイメント記述子の res-ref-name 要素で指定されているものと同じです。JNDI ルックアップによって、res-type 要素の指定のとおり、java.resource.cci.ConnectionFactory 型の接続ファクトリのインスタンスが取得されます。

5. アプリケーション コンポーネントは、接続ファクトリで getConnection メソッドを呼び出して EIS 接続を取得します。返された接続インスタンスは、基底の物理接続へのアプリケーション レベルのハンドルを表します。アプリケーション コンポーネントは、接続ファクトリで getConnection メソッドを複数回呼び出して、複数の接続を取得します。

```
javax.resource.cci.Connection cx = cxf.getConnection();
```

6. アプリケーション コンポーネントは、返された接続を使用して、基底の EIS にアクセスします。
7. 接続を使い終わると、コンポーネントは Connection インタフェースの close メソッドを使用して接続を閉じます。

```
cx.close();
```

8. アプリケーション コンポーネントが割り当てられた接続を使用後に閉じなかった場合、その接続は未使用の接続と見なされます。アプリケーション サーバは、未使用の接続のクリーンアップを管理します。コンテナがコンポーネント インスタンスを終了する場合、コンテナはそのコンポーネント インスタンスが使用していたすべての接続をクリーンアップします。

非管理対象アプリケーションでの接続の取得

非管理対象アプリケーションの場合、アプリケーション開発者は管理対象アプリケーションと同様のプログラミング モデルに従う必要があります。非管理対象アプリケーションでは、接続ファクトリ インスタンスのルックアップ、EIS 接続の取得、接続を使用した EIS アクセス、および接続の終了を行います。

非管理対象アプリケーションが ConnectionFactory から EIS インスタンスとの接続を取得する際には、以下のタスクを実行します。

1. アプリケーション クライアントは、（JNDI ルックアップで返される）`javax.resource.cci.ConnectionFactory` インスタンスのメソッドを呼び出して、基底の EIS インスタンスとの接続を取得します。

2. `ConnectionFactory` インスタンスは、アプリケーションから接続リクエストをデフォルトの `ConnectionManager` インスタンスに委ねます。リソース アダプタは、デフォルト `ConnectionManager` 実装を提供します。

3. `ConnectionManager` インスタンスは、`ManagedConnectionFactory.createManagedConnection` メソッドを呼び出して、基底の EIS インスタンスとの新しい物理接続を作成します。

4. `ManagedConnectionFactory` インスタンスは、`ManagedConnection` インスタンスで表される基底の EIS との新しい物理接続を作成することによって、createManagedConnection メソッドを処理します。`ManagedConnectionFactory` は、Subject インスタンスとして渡されたセキュリティ情報、すべての `ConnectionRequestInfo` とコンフィグレーション済みのプロパティ（ポート番号やサーバ名など）を使用して、新しい `ManagedConnection` インスタンスを作成します。

5. `ConnectionManager` は `ManagedConnection.getConnection` メソッドを呼び出して、アプリケーションレベルの接続ハンドルを取得します。`getConnection` メソッドを呼び出すたびに EIS インスタンスとの新しい物理接続が作成されるわけではありません。`getConnection` を呼び出すと、アプ

リケーションが基底の物理接続にアクセスするための一時ハンドルが作成されます。実際の基底の物理接続は、ManagedConnection インスタンスで表されます。

6. ConnectionManager インスタンスは ConnectionFactory インスタンスへの接続ハンドルを返し、このインスタンスは接続リクエストを初期化したアプリケーションへの接続を返します。

A weblogic-ra.xml デプロイメント記述子の要素

以下の節では、WebLogic Server リソース アダプタ アーカイブで使用する WebLogic Server 6.1 固有の全 XML デプロイメント プロパティをリファレンス形式で示し、XML デプロイメント プロパティを手動で編集する方法について説明します。リソース アダプタ用のデプロイメント記述子を参照する必要がある場合は、これらの節を参考にしてください。

注意： リソース アダプタ `.rar` が `weblogic-ra.xml` ファイルを含んでいない場合、WebLogic Server はこのファイルを自動的に作成します。詳細については、第 5 章「コンフィグレーション」の「weblogic-ra.xml ファイルの自動生成」を参照してください。

- XML デプロイメント ファイルの手動による編集
- Administration Console デプロイメント記述子エディタを使用したファイルの編集
- weblogic-ra.xml DTD
- weblogic-ra. xml の要素の階層図
- weblogic-ra.xml の要素の説明

XML デプロイメント ファイルの手動による編集

WebLogic Server のリソース アダプタ アーカイブで使用される XML デプロイメント記述子を定義または変更する場合は、weblogic-ra.xml ファイルで XML 要素を手動で定義または編集する必要があります。

基本規約

XML 要素を手動で編集する場合は、

- XML の形式の変更や、ファイルを無効にする可能性のある文字の挿入を行わない ASCII テキスト エディタを必ず使用します。
- 使用しているオペレーティング システムで大文字小文字が区別されない場合であっても、ファイル名やディレクトリ名の大文字小文字は正確に指定します。
- 省略可能な要素に対してデフォルト値を使用する場合は、要素の定義全体を省略するか、または次のように空白値を指定することができます。

```
<max-config-property></max-config-property>
```

DOCTYPE ヘッダ情報

XML デプロイメント ファイルの編集、作成時に、各デプロイメント ファイルに対して正しい DOCTYPE ヘッダを指定することが重要です。特に、DOCTYPE ヘッダ内部に不正な PUBLIC 要素を使用すると、原因究明が困難なパーサ エラーになることがあります。

このヘッダは、デプロイメント記述子の文書型定義（DTD）ファイルの場所およびバージョンを表します。このヘッダは外部 URL の java.sun.com を参照していますが、WebLogic Server には独自の DTD ファイルが用意されているので、ホスト サーバがインターネットにアクセスする必要はありません。ただし、こ

の要素の DTD のバージョンはデプロイメント記述子のバージョンの識別に使用されるので、<!DOCTYPE...> 要素を ra.xml ファイルに含めて、外部 URL を参照させる必要があります。

ra.xml および weblogic-ra.xml ファイルの DOCTYPE ヘッダ全体は以下のようになります。

XML ファイル	DOCTYPE ヘッダ
ra.xml	<!DOCTYPE connector PUBLIC '-//Sun Microsystems, Inc.//DTD Connector 1.0//EN' 'http://java.sun.com/dtd/connector_1_0.dtd'>
weblogic-ra.xml	<!DOCTYPE connector PUBLIC '-//BEA Systems, Inc.//DTD WebLogic 6.0.0 Connector//EN' 'http://www.bea.com/servers/wls600/dtd/weblogic600-ra .dtd'

XML の解析ユーティリティ（ejbc など）でヘッダ情報が不正な XML ファイルを解析すると、次のようなエラー メッセージが表示されることがあります。

```
SAXException: This document may not have the identifier
'identifier_name'
```

identifier_name には通常、PUBLIC 要素内の不正な文字列が表示されます。

検証用 DTD（Document Type Definitions : 文書型定義）

XML ファイルの内容および要素の配置は、使用する各ファイルの文書型定義（DTD）に準拠している必要があります。WebLogic Server ユーティリティでは、XML デプロイメント ファイルの DOCTYPE ヘッダ内に埋め込まれた DTD は無視され、代わりにサーバと共にインストールされた DTD の場所が使用されます。ただし、DOCTYPE ヘッダ情報には、パーサ エラーを避けるために有効な URL 構文を指定する必要があります。

以下のリンクでは、WebLogic Server で使用される XML デプロイメント ファイル用の DTD の場所が示されています。

- connector_1_0.dtd には、すべてのリソース アダプタに必要な標準 ra.xml デプロイメント ファイルの DTD が含まれています。この DTD は「J2EE コネクタ仕様、バージョン 1.0」の一部として保守されています。connector_1_0.dtd で使用される要素の詳細については、この仕様（http://java.sun.com/j2ee/download.html#connectorspec）を参照してください。
- weblogic-ra.dtd には、WebLogic Server にデプロイする際に使用されるリソース アダプタ プロパティを定義する weblogic-ra.xml を作成するための DTD が含まれています。このファイルは、http://www.bea.com/servers/wls600/dtd/weblogic600-ra.dtd にあります。

注意： ほとんどのブラウザでは、.dtd ファイルの内容は表示されません。DTD ファイルの内容をブラウザで見るには、リンクをテキスト ファイルとして保存し、テキスト エディタで開いて表示します。

Administration Console デプロイメント記述子エディタを使用したファイルの編集

この節では、Administration Console のデプロイメント記述子エディタを使用して以下のリソース アダプタのデプロイメント記述子を編集する手順を説明します。

- ra.xml
- weblogic-ra.xml

リソース アダプタ デプロイメント記述子の要素の詳細については、『WebLogic J2EE コネクタ アーキテクチャ』を参照してください。

リソース アダプタのデプロイメント記述子を編集するには、次の手順に従います。

1. ブラウザで次の URL を指定して、Administration Console を起動します。

   http://host:port/console

   host は WebLogic Server が実行されているコンピュータ名で、port はリスンしているポート番号を表します。

2. 左ペインの [デプロイメント] ノードをクリックして展開します。
3. [デプロイメント] ノードの [コネクタ] ノードをクリックして展開します。
4. 編集対象のデプロイメント記述子があるリソース アダプタの名前を右クリックし、ドロップダウン メニューから [コネクタ記述子の編集] を選択します。

   Administration Console ウィンドウが新しいブラウザに表示されます。左側のペインでは、2 つのリソース アダプタのデプロイメント記述子のすべての要素がツリー形式で表示され、右側のペインには、`ra.xml` ファイルの説明要素のためのフォームがあります。
5. リソース アダプタのデプロイメント記述子の要素を編集、削除、または追加するには、以下のリストで説明されているように、左側のペインで編集対象のデプロイメント記述子に対応するノードをクリックして展開します。
   - [RA] ノードには、`ra.xml` デプロイメント記述子の要素が含まれています。
   - [WebLogic RA] ノードには、`weblogic-ra.xml` デプロイメント記述子の要素が含まれています。
6. いずれかのリソース アダプタ デプロイメント記述子の既存の要素を編集するには、次の手順に従います。
   a. 左側のペインでツリーをナビゲートし、編集対象の要素が見つかるまで親要素をクリックします。
   b. 要素をクリックします。右側のペインに、属性または下位要素のどちらかをリストするフォームが表示されます。
   c. 右側のペインのフォームで、テキストを編集します。
   d. [適用] をクリックします。
7. いずれかのリソース アダプタ デプロイメント記述子の新しい要素を追加するには、次の手順に従います。
   a. 左側のペインでツリーをナビゲートし、作成対象の要素の名前が見つかるまで親要素をクリックします。
   b. 目的の要素を右クリックして、ドロップダウン メニューから [新しい（要素名）のコンフィグレーション] を選択します。
   c. 右側のペインに表示されるフォームで、要素情報を入力します。

d. [作成] をクリックします。

8. いずれかのリソース アダプタ デプロイメント記述子の既存の要素を削除するには、次の手順に従います。

   a. 左側のペインでツリーをナビゲートし、削除対象の要素の名前が見つかるまで親要素をクリックします。

   b. 目的の要素を右クリックして、ドロップダウン メニューから [（要素名）の削除] を選択します。

   c. [はい] をクリックすると、要素の削除が確定されます。

9. リソース アダプタ デプロイメント記述子への変更がすべて完了したら、左側のペインでツリーのルート要素をクリックします。ルート要素は、リソース アダプタの `*.rar` アーカイブ ファイルの名前またはリソース アダプタの表示名です。

10. リソース アダプタ デプロイメント記述子のエントリが有効かどうかを確認する場合は、[検証] をクリックします。

11. [永続化] をクリックして、デプロイメント記述子ファイルの編集を、WebLogic Server のメモリだけでなくディスクに書き込みます。

weblogic-ra.xml DTD

コード リスト A-1 weblogic-ra.xml DTD

```
weblogic-ra.xml の DTD

<!--

Weblogic 固有のリソース アダプタ デプロイメント記述子 1.0 用の XML DTD

-->

<!--

この DTD は、デプロイ済みリソース アダプタの接続ファクトリを定義するための
 Weblogic 固有のデプロイメント情報を定義します。この要素は、接続プール
パラメータを始めとしてコンフィグレーション可能なすべての接続ファクトリ
パラメータおよびリソース プリンシパル マップ用のセキュリティ パラメータを
指定し、ra.xml デプロイメント記述子のコンフィグレーション パラメータの値を
定義する機能を提供します。
```

```
-->

Copyright (c) 2001 by BEA Systems, Inc. All Rights Reserved.

<!--

weblogic-connection-factory-dd 要素は Weblogic 固有のデプロイメント
記述子のルート要素で、デプロイ済みリソース アダプタ用です。

-->

<!ELEMENT weblogic-connection-factory-dd (connection-factory-name,

 description?, jndi-name, ra-link-ref?, native-libdir?,

 pool-params?, logging-enabled?, log-filename?,
 map-config-property*, security-principal-map?)>

<!--

connection-factory-name 要素は、特定のリソース アダプタのデプロイメ


connection-factory-name の値は、ra-link-ref 要素を介して他のデプロイ済
みリソース アダプタで使用できます。これにより、複数のデプロイ済み接続ファクト
リ間で、指定されているコンフィグレーションを共有するだけでなく、共通の
デプロイ済みリソース アダプタを利用することもできます。

この要素は必須です。

-->

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

<!--

description 要素は、親要素を示すテキストの指定に使用します。description
要素には、デプロイヤがデプロイ済みファクトリについて説明するための情報を含め
ます。

この要素は省略できます。

-->

<!ELEMENT description (#PCDATA)>

<!--

jndi-name 要素は、接続ファクトリ オブジェクトを Weblogic JNDI ネーム
スペースにバインドするための名前を定義します。クライアント EJB および
サーブレットも、weblogic 固有のデプロイメント記述子で定義されている
 Reference Descriptor 要素でこの JNDI を使用します。

この要素は必須です。

-->

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

```
<!--

ra-link-ref では、複数のデプロイ済み接続ファクトリを 1 つのデプロイ済み
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。オプションの ra-link-ref
 要素に別のデプロイ済み接続ファクトリを示す値を指定すると、新しくデプロイされ
る接続ファクトリが、参照先の接続ファクトリと一緒にデプロイされたリソース
アダプタを共有します。

また、参照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すべての値は、
その他の値が指定されていない限り、新しくデプロイされるこの接続ファクトリが
継承します。

この要素は省略できます。

-->

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

<!--

native-libdir 要素は、このリソース アダプタ デプロイメントのすべての
ネイティブ ライブラリ用に使用するディレクトリの場所を示します。デプロイメント
処理の一部として、検出されたネイティブ ライブラリはすべて指定された場所に
コピーされます。

管理者は、Weblogic Server の実行中にライブラリが見つかるようにプラット
フォームのアクション を実行する必要があります。

この要素は、ネイティブ ライブラリが存在する場合には必須です。

-->

<!ELEMENT native-libdir (#PCDATA)>

<!--

pool-params 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
指定するための親要素です。

Weblogic は、管理対象の接続が保持するプールの動作を制御する際にこれらの指定
を使用します。

この要素は省略できます。この要素またはこの要素に固有の項目を指定しないと、
デフォルト値が割り当てられます。指定されているデフォルト値については、
それぞれの要素の説明を参照してください。

-->

<!ELEMENT pool-params (initial-capacity?, max-capacity?,
 capacity-increment?, shrinking-enabled?, shrink-period-minutes?,
 connection-cleanup-frequency?, connection-duration-time?)>

<!--

initial-capacity 要素は、管理対象の接続の初期数を示します。Weblogic
 Server はデプロイメント中にこの数の接続を取得しようとします。

この要素は省略できます。
```

この値を指定しないと、Weblogic は定義されているデフォルト値を使用します。

デフォルト値 : 1

-->

<!ELEMENT initial-capacity (#PCDATA)>

<!--

max-capacity 要素は、Weblogic Server が許容する管理対象の接続の最大数を示します。この制限を超えて管理対象の接続の割り当てを要求すると、呼び出し側に ResourceAllocationException が返されます。

この要素は省略できます。

この値を指定しないと、Weblogic は定義されているデフォルト値を使用します。

デフォルト値 : 10

-->

<!ELEMENT max-capacity (#PCDATA)>

<!--

capacity-increment 要素は、管理対象の接続の追加数を示します。Weblogic Server は、保持している接続プールのサイズを変更する際にこの数の接続を取得しようとします。

この要素は省略できます。

この値を指定しないと、Weblogic は定義されているデフォルト値を使用します。

デフォルト値 : 1

-->

<!ELEMENT capacity-increment (#PCDATA)>

<!--

shrinking-enabled 要素は、接続プールがシステム リソースの管理手段として未使用の管理対象接続を再利用するかどうかを示します。

この要素は省略できます。

この値を指定しないと、Weblogic は定義されているデフォルト値を使用します。

値の範囲 :true または false

デフォルト値 :true

-->

<!ELEMENT shrinking-enabled (#PCDATA)>

<!--

```
shrink-period-minutes 要素は、接続プール管理によって未使用の管理対象接続
を再利用しようとする間隔を示します。

この要素は省略できます。

この値を指定しないと、Weblogic は定義されているデフォルト値を使用します。

デフォルト値 : 15

-->

<!ELEMENT shrink-period-minutes (#PCDATA)>

<!--

connection-cleanup-frequency 要素は、接続プール管理によって設定されて
いる使用時間を超えた接続ハンドルを破棄しようとする間隔（秒単位）を示します。
この要素は connection-duration-time と連携して、アプリケーションが使用後
の接続を閉じなかった場合に接続リークを防ぎます。

この要素は省略できます。

この値を指定しないと、Weblogic は定義されているデフォルト値を使用します。

デフォルト値 :-1

-->

<!ELEMENT connection-cleanup-frequency (#PCDATA)>

<!--

connection-duration-time 要素は、接続ハンドルがアクティブな状態を続ける
時間（秒単位）を示します。この要素は connection-cleanup-frequency と
連携して、アプリケーションが使用後の接続を閉じなかった場合にリークを防ぎます。

この要素は省略できます。

この値を指定しないと、Weblogic は定義されているデフォルト値を使用します。

デフォルト値 :-1

-->

<!ELEMENT connection-duration-time (#PCDATA)>

<!--

logging-enabled 要素は、ManagedConnectionFactory または
 ManagedConnection に対してログ ライタが設定されているかどうかを示します。
この要素を true に設定すると、ManagedConnectionFactory または
 ManagedConnection から生成された出力は、log-filename 要素で指定した
ファイルに送られます。

この要素は省略できます。

この値を指定しないと、Weblogic は定義されているデフォルト値を使用します。
```

```
値の範囲 :true または false

デフォルト値 :false

-->

<!ELEMENT logging-enabled (#PCDATA)>

<!--

log-filename 要素は、ManagedConnectionFactory または
 ManagedConnection から生成された出力を送るログ ファイルの名前を指定します。

ファイル名は絶対アドレスで指定する必要があります。

この要素は省略できます。

-->

<!ELEMENT log-filename (#PCDATA)>

<!--

各 map-config-property 要素は、対応する config-property-name 名を持つ
 ra.xml の config-entry 要素に対応するコンフィグレーション プロパティの
名前および値を示します。


デプロイメント時には、map-config-property で指定されたすべての値が
 ManagedConnectionFactory で設定されます。

map-config-property を介して指定された値は、対応する ra.xml の
 config-entry 要素で指定されたデフォルト値に優先します。

この要素は省略できます。

-->

<!ELEMENT map-config-property (map-config-property-name,
 map-config-property-value)>

<!ELEMENT map-config-property-name (#PCDATA)>

<!ELEMENT map-config-property-value (#PCDATA)>

<!--

各 security-principal-map 要素は、Weblogic 実行時の既知の開始
プリンシパルに基づいて、リソース アダプタ /EIS の許可処理用のリソース
プリンシパル値を定義するためのメカニズムを提供します。

このマップにより、管理対象の接続と接続ハンドルを割り当てる際に使用される
開始プリンシパルと対応するリソース プリンシパルのユーザ名およびパスワードの
セットを指定できます。

デフォルトのリソース プリンシパルは、このマップに基づいて接続ファクトリ用に
定義できます。initiating-principal 値に「*」を指定し、対応する
```

resource-principal 値を指定した場合、マップ内で現在の ID と一致するものがないときには必ず定義した resource-principal が利用されます。

この要素は省略できますが、コンテナ管理によるサインオンがリソース アダプタでサポートされており、いずれかのクライアントで使用される場合は指定する必要があります。

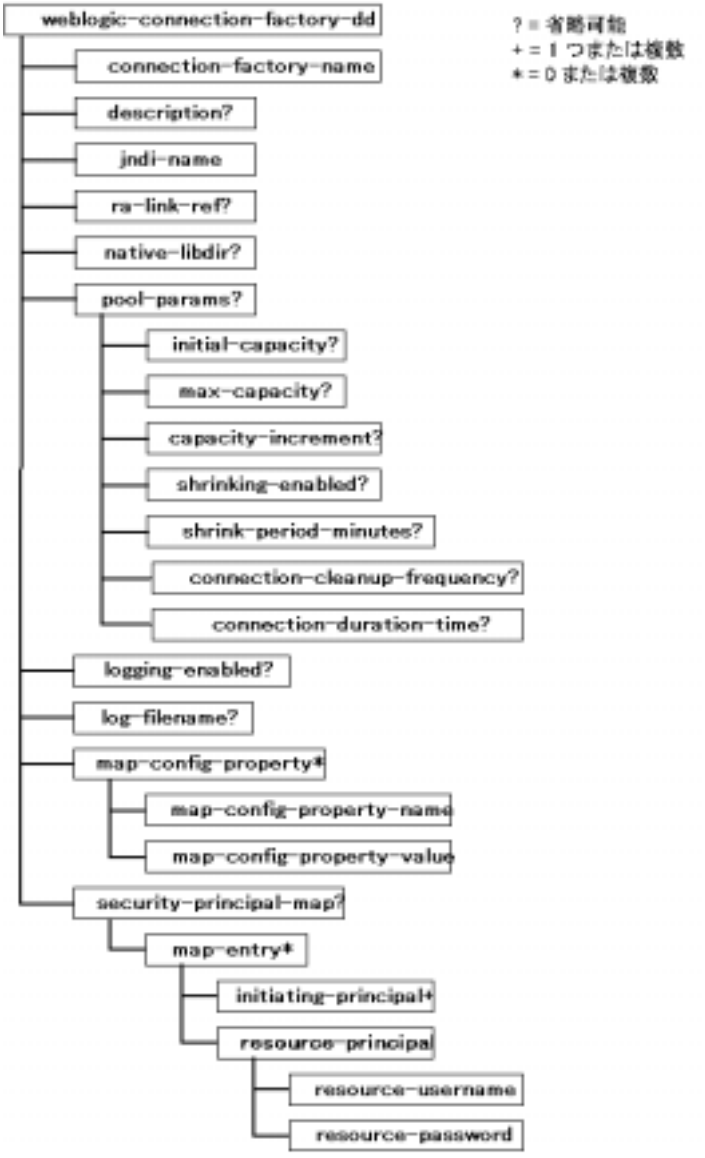
また、デプロイ時に管理対象の接続を接続プールに取得する試みは、定義されている「デフォルト」リソース プリンシパル（指定されている場合）を使用して行われます。

```
-->

<!ELEMENT security-principal-map (map-entry*)>

<!ELEMENT map-entry (initiating-principal+, resource-principal)>

<!ELEMENT initiating-principal (#PCDATA)>

<!ELEMENT resource-principal (resource-username,
 resource-password)>

<!ELEMENT resource-username (#PCDATA)>

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

weblogic-ra. xml の要素の階層図

以下の図は、weblogic-ra.xml デプロイメント記述子の構造を示しています。

図 A-1　weblogic-ra.xml の要素の階層



weblogic-ra.xml の要素の説明

以降の節では、weblogic-ra.xml ファイル内に表示される各要素について説明します。

- *weblogic-connection-factory-dd*（必須）— Weblogic 固有のデプロイメント記述子のルート要素で、デプロイ済みリソース アダプタ用です。
- *connection-factory-name*（必須）— 特定のリソース アダプタのデプロイメントおよび対応する接続ファクトリに関連付けられる論理名を定義します。この要素の値は、*ra-link-ref* 要素を介して他のデプロイ済みリソース アダプタで使用できます。これにより、複数のデプロイ済み接続ファクトリ間で、指定されているコンフィグレーションを共有するだけでなく、共通のデプロイ済みリソース アダプタを利用することもできます。
- *description*（省略可能）— 親要素について説明するテキストを提供します。この要素には、デプロイヤがデプロイ済みファクトリについて説明するための情報を含めます。
- *jndi-name*（必須）— 接続ファクトリ オブジェクトを Weblogic JNDI ネームスペースにバインドするための名前を定義します。クライアント EJB およびサーブレットも、Weblogic 固有のデプロイメント記述子で定義されている Reference Descriptor 要素でこの JNDI を使用します。
- *ra-link-ref*（省略可能）— 複数のデプロイ済み接続ファクトリを 1 つのデプロイ済みリソース アダプタに論理的に関連付けることができます。オプションの *ra-link-ref* 要素に別のデプロイ済み接続ファクトリを示す値を指定すると、新しくデプロイされる接続ファクトリが、参照先の接続ファクトリと一緒にデプロイされたリソース アダプタを共有します。また、参照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すべての値は、その他の値が指定されていない限り、新しくデプロイされるこの接続ファクトリが継承します。
- *native-libdir*（ネイティブ ライブラリが存在する場合は必須）— このリソース アダプタ デプロイメントのすべてのネイティブ ライブラリ用に使用するディレクトリの場所を示します。デプロイメント処理の一部として、検出されたネイティブ ライブラリはすべて指定された場所にコピーされます。管理者は、Weblogic Server の実行中にライブラリが見つかるようにプラットフォームのアクションを実行する必要があります。

- *pool-params*（省略可能）― この接続ファクトリの接続プール固有のパラメータを指定するための親要素です。WebLogic Server は、管理対象の接続が保持するプールの動作を制御する際にこれらの指定を使用します。

  この要素またはこの要素に固有の項目を指定しないと、デフォルト値が割り当てられます。指定されているデフォルト値については、それぞれの要素の説明を参照してください。

- *initial-capacity*（省略可能）― 管理対象の接続の初期数を示します。Weblogic Server はデプロイメント中にこの数の接続を取得しようとします。

  この値を指定しないと、WebLogic Server は定義されているデフォルト値を使用します。

  デフォルト値 : 1

- *max-capacity*（省略可能）― Weblogic Server が許容する管理対象の接続の最大数を示します。この制限を超えて管理対象の接続の割り当てを要求すると、呼び出し側に *ResourceAllocationException* が返されます。

  この値を指定しないと、WebLogic Server は定義されているデフォルト値を使用します。

  デフォルト値 : 10

- *capacity-increment*（省略可能）― 管理対象の接続の最大追加数を示します。Weblogic Server は、保持している接続プールのサイズを変更する際にこの数の接続を取得しようとします。

  この値を指定しないと、WebLogic Server は定義されているデフォルト値を使用します。

  デフォルト値 : 1

- *shrinking-enabled*（省略可能）— 接続プールがシステム リソースの管理手段として未使用の管理対象接続を再利用するかどうかを示します。

  この値を指定しないと、WebLogic Server は定義されているデフォルト値を使用します。

  値の範囲 :true または false

  デフォルト値 :true

- *shrink-period-minutes*（省略可能）― 接続プール マネージャが未使用の管理対象接続を再利用しようとする間隔を示します。

この値を指定しないと、WebLogic Server は定義されているデフォルト値を使用します。

デフォルト値 : 15

- *connection-cleanup-frequency* （省略可能）― 接続プール管理によって設定されている使用時間を超えた接続ハンドルを破棄しようとする間隔を示します。この要素は connection-duration-time と連携して、アプリケーションが使用後の接続を閉じなかった場合に接続リークを防ぎます。

  この値を指定しないと、Weblogic は定義されているデフォルト値を使用します。

  デフォルト値 : -1

- *connection-duration-time* （省略可能）― 接続がアクティブな状態を続ける時間を示します。この要素は connection-cleanup-frequency と連携して、アプリケーションが使用後の接続を閉じなかった場合にリークを防ぎます。

  この値を指定しないと、Weblogic は定義されているデフォルト値を使用します。

  デフォルト値 : -1

- *logging-enabled* （省略可能）― *ManagedConnectionFactory* または *ManagedConnection* に対してログ ライタが設定されているかどうかを示します。この要素を true に設定すると、*ManagedConnectionFactory* または *ManagedConnection* から生成された出力は、*log-filename* 要素で指定したファイルに送られます。

  この値を指定しないと、WebLogic Server は定義されているデフォルト値を使用します。

  値の範囲 :true または false

  デフォルト値 :false

- *log-filename* （省略可能）― *ManagedConnectionFactory* または *ManagedConnection* から生成された出力を送るログ ファイルの名前を指定します。

  ファイル名は絶対アドレスで指定する必要があります。

- *map-config-property* （省略可能、ゼロまたは 1 つ以上）― 対応する *config-property-name* 名を持つ ra.xml config-entry 要素に対応するコンフィグレーション プロパティの名前および値を示します。デプロイメント時には、map-config-property で指定されたすべての値が

ManagedConnectionFactory で設定されます。map-config-property を介して指定された値は、対応する ra.xml config-entry 要素で指定されたデフォルト値に優先します。

- *map-config-property-name*（省略可能）― 対応する config-property-name を持つ ra.xml config-entry に対応する名前を示します。
- *map-config-property-value*（省略可能）― 対応する config-property-name を持つ ra.xml config-entry に対応する値を示します。
- *security-principal-map*（省略可能）― Weblogic 実行時の既知の initiating-principal に基づいて、リソース アダプタおよび EIS の許可処理用の resource-principal 値を定義するためのメカニズムを提供します。このマップにより、管理対象の接続と接続ハンドルを割り当てる際に使用される開始プリンシパルと対応するリソース プリンシパルのユーザ名およびパスワードのセットを指定できます。

  デフォルトの resource-principal は、このマップに基づいて接続ファクトリ用に定義できます。initiating-principal 値に「*」を指定し、対応する resource-principal 値を指定した場合、マップ内で現在の ID と一致するものがないときには必ず定義した resource-principal が利用されます。

  この要素は省略できますが、コンテナ管理によるサインオンがリソース アダプタでサポートされており、いずれかのクライアントで使用される場合は指定する必要があります。

  また、定義済みの「デフォルト」リソース プリンシパル（指定されている場合）を使用して、デプロイメント時に管理対象の接続を接続プールに取得するよう試行されます。
- *map-entry* ― security-principal-map 内のエントリを示します。
- *initiating-principal*（省略可能、ゼロまたは 1 つ以上）
- *resource-principal*（省略可能）― security-principal-map を介して接続ファクトリ向けに定義できます。*initiating-principal* 値に「*」を指定し、対応する resource-principal 値を指定した場合、マップ内で現在の ID と一致するものがないときには必ず定義した resource-principal が利用されます。

- *resource-username*（省略可能）― resource-principal で示されるユーザ名です。管理対象の接続および接続ハンドルを割り当てるときに使用されます。
- *resource-password*（省略可能）― resource-principal で示されるパスワードです。管理対象の接続および接続ハンドルを割り当てるときに使用されます。

B 一般的な BEA J2EE コネクタ アーキテクチャ例外の回避策

ここでは、2 つの一般的な BEA J2EE コネクタ アーキテクチャ例外の原因とその回避策について説明します。

- 例外 1 : Problem Granting Connection Request to a ManagedConnectionFactory That Does Not Exist in Connection Pool. Check Your MFC's HashCode()
- 例外 2 : ClassCastException

例外 1 : Problem Granting Connection Request to a ManagedConnectionFactory That Does Not Exist in Connection Pool. Check Your MFC's HashCode()

この例外は、指定した ManagedConnectionFactory に関連付けられた ConnectionPool が見つからない場合に、ConnectionPoolManager の `getConnection(ManagedConnectionFactory mcf, ConnectionRequestInfo cxInfo)` メソッドによって WebLogic Server の内部に送出されます。

この例外の原因と回避策

この例外には 2 つの原因があり、関連する動作も 1 つあるため少々複雑です。

- 原因 1 : クライアントが変更した ManagedConnectionFactory が、その後のルックアップでも見つかるようにサーバ上でハッシュ化されていない
- 原因 2 : クライアントがリモート JVM からリソース アダプタを使用しようとしている
- 関連する動作 : クライアントサイドのミューテータがうまく機能しない

原因 1：クライアントが変更した ManagedConnectionFactory が、その後のルックアップでも見つかるようにサーバ上でハッシュ化されていない

この問題は、Java での Hashtable の基本動作と、JNDI での Serializable オブジェクトの動作方法の組み合わせにより発生します。

Hashtable を使用すると、キーと値の任意のペアをメモリに格納でき、後からキーを使ってすばやくキーと値のペアを見つけることができます。キーとそれに関連付けられたオブジェクトは Java Hashtable に書き込まれ、そのキーの `hashCode()` メソッドを呼び出すことで整数値が取得できます。この整数値はユニークとは限りませんが、すべての Hashtable キーに配布されることは保証されています。

このハッシュ化は、オブジェクトが Hashtable に書き込まれたときに 1 度だけ発生します。サーバでは、このオブジェクトがデータ構造に書き込まれます。これは、それと同じ派生整数値を、一致するキーの候補セットをすばやく見つけるためのメソッドとして繰り返し使用できるようにするためです。

その後サーバが Hashtable へのルックアップを実行すると、オーバーライドされた（または `java.lang.Object` で見つかった）`hashCode()` メソッドによって、要求されたキーがハッシュ化されます。デフォルトの `hashCode()` メソッドは単純に、`hashCode()` メソッドが呼び出されたオブジェクトのメモリ アドレスを返します。次に、ルックアップ キーのハッシュ化されたキー値と一致するハッシュ値を持つバッキング データ構造内のキーのセットが取得されます。つづいて、次のコードを実行することによって、実装ではこの候補リストを繰り返し、適切な一致があるかどうかが識別されます。

```
for (Entry e = tab[index] ; e != null ; e = e.next) {
    if ((e.hash == hash) && e.key.equals(key)) {
        return e.value;
    }
}
return null;
```

このコードでは、各候補キーで `equals()` メソッドを呼び出して、ルックアップ キーを各候補リストと比較しています。一致したものが見つかると値が返されます。一致したものが見つからなかった場合は null 値が返されます。

アプリケーション サーバで Serializable JNDI エントリのみがサポートされている (Referenceable エントリはサポートされていない）場合は、リソース アダプタのデプロイメントでのオブジェクトの対話と、リソース アダプタの ConnectionFactory 実装の JNDI ルックアップを考慮してください。リソース アダプタのデプロイメントにおいては、そのリソース アダプタに関連付けられた ManagedConnectionFactory のインスタンスがアプリケーション サーバによって作成されます。さらに、その ConnectionManager 実装が、作成したばかりの ManagedConnectionFactory に関連付けられます。

次に、ManagedConnectionFactory によって ConnectionFactory のインスタンスが作成され、この ConnectionFactory がその ManagedConnectionFactory（MCF）に関連付けられます。この時点で、不可欠な 3 つのリソースが設定できました。内部的には、このリンクが Hashtable のペアとして表現されます。ペアの 1 つは JNDI 名を MCF（`jndiToMCFMap`）にマップするもの、もう 1 つは MCF を接続プール（`poolTable`）の内部表現にマップするものです。ConnectionManager はアプリケーション サーバに対して暗黙的であるため、明示的なマッピングは必要ありません。

ConnectionFactory（リソース アダプタのクライアントが見るもの）は MCF を持ち、MCF は、上記の暗黙的なマッピングによってアプリケーション サーバから提供される ConnectionManager 実装を持ちます。最後に、ConnectionFactory（クライアントが見るオブジェクト）が JNDI にバインドされます。ここが、問題の発生する可能性がある部分です。

現在の WebLogic Server JNDI 実装では、`java.io.Serializable` オブジェクトはサポートされていますが、`javax.naming.Referenceable` オブジェクトはサポートされていません。Referenceable オブジェクトが JNDI にバインドされると、実際のオブジェクトへの参照（エンドポイントを含む）が実際に JNDI にバインドされます。このバインドでは、オブジェクトはネーミング ツリーには転送されません。サーバによって Serializable オブジェクトが JNDI ツリーにバインドされると、実質的にはオブジェクト（および参照されるすべての Serializable オブジェクト）がツリーにシリアライズされます。

クライアントが ConnectionFactory のネーミング コンテキストで `lookup()` を実行すると、同じシリアライズのプロセスが逆方向に発生します。ConnectionFactory 実装は、クライアントのアドレス空間にシリアライズされま

す。このシリアライゼーションは、クライアントが JNDI 実装（サーバ）と同じアドレス空間で実行されている場合でも行われます。これでアプリケーション コンポーネント（クライアント）は、JNDI ツリー内の実際のオブジェクト グラフのコピーを持つことになります。

MCF は、オブジェクト グラフの一部としてクライアントにシリアライズされます。リソース アダプタの MCF には、MCF のステートの管理に使用するフィールドが含まれているものとみなします。さらに、これらのフィールドは、MCF のオーバーライドされた `hashCode()` メソッドおよび `equals()` メソッドによって考慮されるものとみなします。たとえば、デバッグ フラグを考えてみます。

リソース アダプタがデプロイされると、エントリは `jndiToMCFMap` Hashtable になります。Hashtable が Java でどのように機能するかを思い出してください。ハッシュは、デバッグ フラグを含め、すべてのフィールドを使用して計算されます。これらのフィールドは `hashCode()` メソッドで使用されているためです。クライアントは、JNDI `lookup()` を実行すると、CF（およびその MCF）のコピーを取得できます。

次に、クライアントはデバッグ フラグを `true` に設定して CF の `getConnection()` を呼び出します。つづいて CF は、その MCF と ConnectionRequestInfo オブジェクトをパラメータにして CM の `allocateConnection()` を呼び出します。CM は、MCF をキーとして、その `poolTable` Hashtable 内の MCF をルックアップします。Java Hashtable 実装は、MCF（デバッグ フラグを含む）をハッシュ化し、`poolTable` Hashtable 内で一致するものの候補リストを取得します。MCF の内部ステートが、元々のデプロイ時の Hashtable への `put` からその次のルックアップまでの間に変更されたため、MCF が見つからず次のような例外がロギングされます。

```
Problem granting connection request to a ManagedConnectionFactory
which does not exist in connection pool. Check your MCF's
hashcode().
```

この例外の防止策

この例外を防止するもっとも簡単な方法は、WebLogic Server で、`javax.naming.Referenceable` インタフェースを実装する JNDI オブジェクトがサポートされるようにすることです。この場合、リソース アダプタ開発者としては何もする必要はありません。上記のサポートは、次期のメジャー リリースで計画されています。

それまでは、いくつかの手順を踏むことによってこの例外を回避できます。手順は、リソース アダプタの MCF ステートの要件によって異なります。もっとも簡単な解決策は、リソース アダプタの MCF 用に `hashCode()` メソッドおよび `equals()` メソッドを実装することです。もちろん、このマニュアルの記述に基づいて、リソース アダプタ固有の別の方法で問題を解決しても構いません。

`hashCode()` および `equals()` を実装する際のガイドラインとしては、これらのメソッドで考慮されるのが、サーバ内で稼動する各リソース アダプタ インスタンスを一意に識別できることが絶対に必要な内部ステートのみという点です。たとえば、リソース アダプタをメインフレーム TP インスタンスと対話するように構築した場合、その `hashCode()` メソッドおよび `equals()` メソッドにおけるホスト IP アドレス、ポート、および領域が考慮されます。また、同じコンフィグレーション データを使用することになった 2 種類のリソース アダプタを区別できるような何かをメソッドにもたせることも必要です。たとえば、名前、クラス タイプなどが考えられます。

原因 2 : クライアントがリモート JVM からリソース アダプタを使用しようとしている

この例外の 2 つめの原因はより単純な場合です。J2EE コネクタ アーキテクチャは、リモートでアクセスするためのモデルではありません。このモデルは当面は変更されません。定義されているインタフェースはいずれもリモートではなく、構築されたシステム規約でも MCF と ConnectionManager とのローカル関係が仮定されています。

それでもやはり、WebLogic Server J2EE コネクタ アーキテクチャの実装で、ConnectionFactory にリモートでアクセスしようとすると MFC の `hashCode()` 実装に問題があるとレポートされるのはバグであり、将来のリリースでは修正されます。エラー メッセージも改善される予定です。

関連する動作 : クライアントサイドのミューテータがうまく機能しない

Referenceable JNDI 実装の不足によるもう 1 つの副作用は、クライアントの対話の間に MCF のコピーが 2 つ作成されるために、クライアントサイドでの MCF への変更がサーバサイドに反映されないことです。

リソース アダプタのデプロイメント中は、WebLogic Server によってオブジェクト グラフが JNDI ツリーにバインドされることに注意してください。また、クライアントが `Naming.lookup()` を使ってこのグラフをルックアップすると、元のグラフではなく、シリアライズされたグラフのコピーが取得される点にも注意が必要です。クライアントが MCF の内部ステートを変更しても、それらの変更はサーバサイドには反映されません。デバッグ フラグを再び使用して、クライアントが `setDebug(true)` メソッドを実行する場合、MCF のステート（デバッグ）への変更はそのクライアントの MCF のコピーに対してローカルであり、サーバサイドのコピーはクライアントと同じステートを共有しません。

例外 2 : ClassCastException

現在、WebLogic J2EE コネクタ アーキテクチャ コンテナではカスタム クラスローダが使用されています。これは、クラスのロードおよび検索に関連する拡張機能をリソース アダプタで使用できるようにするためのものです。現在の実装には、Web（サーブレット /JSP）コンテナ、EJB コンテナなどの WebLogic Server コンテナからリソース アダプタ クラスを参照する動作があまり直観的ではないという問題があります。タイプの比較とキャストを実行する場合、そのクラスとクラス タイプをロードしたクラスローダが JVM によって先頭に追加されます（つまり、`com.acb.ra.MyCF` が内部的には `RARClassloader@abcdef:com.acb.ra.MyCF` として表現されます）。詳細については、Java 言語仕様を参照してください。

Web コンテナおよび EJB コンテナでは、WebLogic Server クラスローダから派生したクラスローダを使用します。したがって、他の WebLogic Server リソースとの相性も良く、ホット デプロイ などの機能も利用できます。J2EE コネクタ アーキテクチャで使用される RARClassLoader は、Java クラスローダから派生したもので、WebLogic 階層の中にはありません。この点は、できる限り早く修正される予定です。

Web コンテナまたは EJB コンテナで実行されているアプリケーション コンポーネントが JNDI 内の ConnectionFactory をルックアップしたときに返されるオブジェクトは、RARClassloader（`RARClassloader@abcdef.com.acb.ra.MyCF`）によって作成されたオブジェクトのインスタンスです。アプリケーション コンポーネントが実際に想定するオブジェクトは `WebLogicClassloader@fedcba:com.acb.ra.MyCF` です。変数またはキャストを割り当てようとすると、`ClassCastException` が送出されます。

この例外の防止策

このエラーを防止する方法として最も信頼性が高いのは、WebLogic Server の起動スクリプトの CLASSPATH 設定にリソース アダプタへのパスを追加することにより、リソース アダプタのクラスを WebLogic Server クラスパスに配置する方法です。これにより、WebLogic Server が提供するホット デプロイ機能および再デプロイ機能が無効になります。ただし、これも望ましい方法ではなく、早急に修正される予定です。

索引

記号

A

B

C

D

I

J

L

X

あ

い

え

か

く

こ

さ

し

せ

て

と

ろ