



BEA WebLogic Server™

WebLogic JNDI プロ グラマーズ ガイド

著作権

Copyright © 2002, BEA Systems, Inc. All Rights Reserved.

限定的権利条項

本ソフトウェアおよびマニュアルは、BEA Systems, Inc. 又は日本ビー・イー・エー・システムズ株式会社（以下、「BEA」といいます）の使用許諾契約に基づいて提供され、その内容に同意する場合にのみ使用することができ、同契約の条項通りにのみ使用またはコピーすることができます。同契約で明示的に許可されている以外の方法で同ソフトウェアをコピーすることは法律に違反します。このマニュアルの一部または全部を、BEA からの書面による事前の同意なしに、複製、複製、翻訳、あるいはいかなる電子媒体または機械可読形式への変換も行うことはできません。

米国政府による使用、複製もしくは開示は、BEA の使用許諾契約、および FAR 52.227-19 の「Commercial Computer Software-Restricted Rights」条項のサブパラグラフ (c)(1)、DFARS 252.227-7013 の「Rights in Technical Data and Computer Software」条項のサブパラグラフ (c)(1)(ii)、NASA FAR 補遺 16-52.227-86 の「Commercial Computer Software--Licensing」条項のサブパラグラフ (d)、もしくはそれらと同等の条項で定める制限の対象となります。

このマニュアルに記載されている内容は予告なく変更されることがあり、また BEA による責務を意味するものではありません。本ソフトウェアおよびマニュアルは「現状のまま」提供され、商品性や特定用途への適合性を始めとする（ただし、これらには限定されない）いかなる種類の保証も与えません。さらに、BEA は、正当性、正確さ、信頼性などについて、本ソフトウェアまたはマニュアルの使用もしくは使用結果に関していかなる確約、保証、あるいは表明も行いません。

商標または登録商標

BEA, Jolt, Tuxedo, および WebLogic は BEA Systems, Inc. の登録商標です。BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop および How Business Becomes E-Business は、BEA Systems, Inc の商標です。

その他の商標はすべて、関係各社がその権利を有します。

WebLogic JNDI プログラマーズ ガイド

パート番号	マニュアルの改訂	ソフトウェアのバージョン
なし	2002年10月3日	WebLogic Server バージョン 7.0

目次

このマニュアルの内容

対象読者.....	v
e-docs Web サイト.....	vi
このマニュアルの印刷方法.....	vi
サポート情報.....	vi
表記規則.....	viii

1. WebLogic JNDI の概要

Java 仕様の実装.....	1-1
J2EE 仕様.....	1-1
JNDI 仕様.....	1-1
WebLogic Server の JNDI の概要.....	1-2

2. WebLogic JNDI を使用したプログラミング

Java クライアントからの WebLogic JNDI の使い方.....	2-1
InitialContext への JNDI 環境プロパティの設定.....	2-2
ハッシュ テーブルを使用したコンテキストの作成.....	2-4
WebLogic Environment オブジェクトを使用したコンテキストの作成.....	2-5
サーバサイド オブジェクトからのコンテキストの作成.....	2-6
JNDI コンテキストとスレッド.....	2-7
JNDI コンテキストの潜在的な問題を回避する方法.....	2-8
コンテキストを使用した名前付きオブジェクトのルックアップ.....	2-11
名前付きオブジェクトを使用したオブジェクト参照の取得.....	2-11
コンテキストのクローズ.....	2-12
クラスタ環境での WebLogic JNDI の使い方.....	2-13
J2EE サービスのクラスタ化.....	2-13
カスタム オブジェクトを WebLogic Server クラスタで使用できるように する方法.....	2-14
「データ キャッシュ」設計パターン.....	2-15
「サービスを各クラスタで一度だけ提供する」設計パターン.....	2-17
クラスタ環境でのクライアントからの WebLogic JNDI の使い方.....	2-17



このマニュアルの内容

このマニュアルでは、BEA WebLogic Server™ 製品で提供される JNDI 機能を使用したプログラミング方法について説明します。

このマニュアルの内容は以下のとおりです。

- 第 1 章「WebLogic JNDI の概要」では、WebLogic Server における JNDI 機能の概要について説明します。
- 第 2 章「WebLogic JNDI を使用したプログラミング」では、Java クライアント アプリケーションでの WebLogic JNDI 機能を使用したプログラミング方法について説明します。

対象読者

このマニュアルは、WebLogic Server でアプリケーションを開発し、JNDI 機能を使用するプログラマを対象としています。

このマニュアルは、Sun Microsystems の Java 2 Platform, Enterprise Edition (J2EE) を使用してアプリケーションを設計、開発、コンフィグレーション、および管理し、それぞれのエンタープライズで複数のネーミングおよびディレクトリ サービスに統一的なインタフェースを提供する JNDI API を使用するアプリケーション開発者を対象としています。JNDI 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 ホームページの [製品のドキュメント] をクリックするか、WebLogic Server 製品ドキュメント ページ (<http://edocs.beasys.co.jp/e-docs/wls70>) を直接表示してください。

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を一度に 1 章ずつ印刷できます。

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>) で無料で入手できます。

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@beasys.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の名前とバージョン、およびドキュメントのタイトルと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.beasys.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]	複数のキーを同時に押すことを示す。
<i>斜体</i>	強調または書籍のタイトルを示す。
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： <pre>import java.util.Enumeration; chmod u+w * config/examples/applications .java config.xml float</pre>
<i>斜体の等幅テキスト</i>	コード内の変数を示す。 例： <pre>String CustomerName;</pre>
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： <pre>LPT1 BEA_HOME OR</pre>
{ }	構文の中で複数の選択肢を示す。

表記法	適用
[]	<p>構文の中で任意指定の項目を示す。 例：</p> <pre>java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]</pre>
	<p>構文の中で相互に排他的な選択肢を区切る。 例：</p> <pre>java weblogic.deploy [list deploy undeploy update] password {application} {source}</pre>
...	<p>コマンドラインで以下のいずれかを示す。</p> <ul style="list-style-type: none"> ■ 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。
.	<p>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。 . . .</p>



1 WebLogic JNDI の概要

以下の節では、WebLogic Server の JNDI 実装の概要について説明します。

- Java 仕様の実装
- WebLogic Server の JNDI の概要

Java 仕様の実装

WebLogic Server は次の Java 仕様に準拠しています。

J2EE 仕様

WebLogic Server 7.0 は Sun Microsystems の J2EE 1.3 仕様に準拠しています。

JNDI 仕様

WebLogic Server 7.0 は JNDI 仕様バージョン 1.2.1 に完全に準拠し、プロダクション段階で使用できます。

WebLogic Server の JNDI の概要

企業内では、ネーミング サービスを使用することで、アプリケーションでネットワーク上のオブジェクトを見つけることができます。ネーミング サービスは、オブジェクトに名前を関連付けたり、指定した名前に基づいてオブジェクトを検索したりします。RMI レジストリは、ネーミング サービスの 1 つです。

Java アプリケーションにネーミング サービスを提供するアプリケーションプログラミング インタフェース (API)。Sun Microsystems の J2EE 技術の不可欠なコンポーネントです。

JNDI は、特定のネーミング サービスまたはディレクトリ サービスの実装とは無関係に定義されています。JNDI では、単一の方法で、さまざまな新しいサービスや既存のサービスにアクセスできます。このサポートでは、標準サービス プロバイダ インタフェース (SPI) 規約を使用して JNDI フレームワークに任意のサービスプロバイダ実装をプラグインできます。さらに、適切なサービスプロバイダ実装をプラグインすることで、WebLogic Server の Java アプリケーションから LDAP などの外部ディレクトリ サービスに標準化された方法でアクセスできるようになります。

WebLogic Server の JNDI 実装では、以下のようなメソッドが用意されています。

- クライアントに WebLogic ネーム サービスへのアクセスを提供するメソッド
- オブジェクトを WebLogic ネームスペースで使用可能にするメソッド
- オブジェクトを WebLogic ネームスペースから取り出すメソッド

各 WebLogic Server クラスタは、レプリケートされたクラスタワイドの JNDI ツリーによってサポートされています。このツリーから、レプリケートされたり、固定されたりしている RMI オブジェクトおよび EJB オブジェクトにアクセスできます。クラスタを表す JNDI ツリーは、クライアントからは単一のグローバルツリーのように見えますが、クラスタワイドのサービスを含むツリーは、実際には、クラスタ内の各 WebLogic Server 間でレプリケートされたものです。詳細については、2-13 ページの「クラスタ環境での WebLogic JNDI の使い方」を参照してください。

WebLogic Server JNDI によって提供される統合されたネーミング サービスは、他の多くの WebLogic サービスによって使用されます。たとえば、WebLogic RMI では、標準 RMI メソッドと JNDI メソッドの両方を使用して、リモートオブジェクトにバインドし、アクセスできます。

JNDI の標準 Java インタフェースに加えて、WebLogic Server には、標準 JNDI インタフェースを使用する独自の実装である

`weblogic.jndi.WLInitialContextFactory` もあります。

アプリケーション コードでは、このクラスを直接インスタンス化する必要はありません。代わりに、標準の `javax.naming.InitialContext` クラスを使用して、適切なハッシュ テーブル プロパティを設定できます (2-2 ページの「InitialContext への JNDI 環境プロパティの設定」を参照)。すべての対話は、`javax.naming.Context` インタフェースを経由して行われます (JNDI Javadoc を参照)。

クライアント接続用に WebLogic JNDI API を使用する手順については、「WebLogic JNDI を使用したプログラミング」を参照してください。

2 WebLogic JNDI を使用したプログラミング

この節では、WebLogic JNDI を使用するプログラミングについて説明します。内容は以下のとおりです。

- Java クライアントからの WebLogic JNDI の使い方
- InitialContext への JNDI 環境プロパティの設定
- コンテキストを使用した名前付きオブジェクトのルックアップ
- 名前付きオブジェクトを使用したオブジェクト参照の取得
- コンテキストのクローズ
- クラスタ環境での WebLogic JNDI の使い方

Java クライアントからの WebLogic JNDI の使い方

WebLogic Server JNDI サービス プロバイダインタフェース (SPI) には、リモート Java クライアントから WebLogic Server への接続を可能にする InitialContext 実装が用意されています。クライアントでは、特定の WebLogic Server デプロイメントを識別する標準の JNDI 環境プロパティと、WebLogic Server へのログインに関連する接続プロパティを指定できます。

WebLogic Server とセッションに参加するには、Java クライアントはリモートオブジェクトへのオブジェクト参照を取得し、そのオブジェクトで操作を呼び出す必要があります。この処理を行うには、クライアントアプリケーションコードで、以下の手順を実行します。

1. InitialContext に JNDI 環境プロパティを設定します。

2. WebLogic Server との InitialContext を作成します。
3. そのコンテキストを使用して、WebLogic Server ネームスペースで名前付きオブジェクトをルックアップします。
4. 名前付きオブジェクトを使用してリモート オブジェクトへの参照を取得し、そのリモート オブジェクトで操作を呼び出します。
5. セッションを終了します。

以降の節では、特定の WebLogic Server に接続するための JNDI クライアントの操作について説明します。WebLogic Server のクラスタでの JNDI の使い方については、「クラスタ環境でのクライアントからの WebLogic JNDI の使い方」を参照してください。

JNDI を使用して WebLogic Server 環境内のオブジェクトにアクセスする前に、WebLogic Server の JNDI ツリーにそのオブジェクトをロードしておく必要があります。JNDI ツリーにオブジェクトをロードする手順については、「JNDI の管理」を参照してください。

注意： WebLogic Server 6.1 のクラスを使用して構築したオブジェクトを WebLogic Server 7.0 以降にバインドしたり、デブatoiしたりすることはサポートされていないので、起動時にエラーになる場合があります。

InitialContext への JNDI 環境プロパティの設定

Java クライアントアプリケーションで、まず実行しなければならない作業は、環境プロパティの作成です。InitialContext ファクトリは、各種のプロパティを使用して、特定の環境に合った InitialContext をカスタマイズします。これらのプロパティは、ハッシュテーブル、または WebLogic Environment オブジェクトの set() メソッドを使用して設定できます。名前と値の組み合わせで指定される、これらのプロパティによって、WLInitialContextFactory ファクトリでのコンテキストの作成方法が決まります。

以下のプロパティが、InitialContext のカスタマイズに使用されます。

- `Context.PROVIDER_URL` - ネーム サービスを提供する **WebLogic Server** の URL を指定します。デフォルトは `t3://localhost:7001` です。

注意: T3 とデフォルト チャネルを使用した初期コンテキスト リクエストの場合、`Context.PROVIDER_URL` はロード バランサの IP アドレスにすることもできます。

- `Context.SECURITY_PRINCIPAL` - ユーザ (**WebLogic Server** セキュリティ レalm で定義されているユーザ) の認証用の ID を指定します。スレッドが既に **WebLogic Server** ユーザに関連付けられていない限り、プロパティのデフォルト値は `guest` になります。詳細については、「コンテキストを使用した名前付きオブジェクトのルックアップ」を参照してください。
- `Context.SECURITY_CREDENTIALS` - この `Context.SECURITY_CREDENTIALS` プロパティを使用して、`Context.SECURITY_PRINCIPAL` プロパティで定義されているユーザのパスワード、または `weblogic.security.acl.UserInfo` インタフェースを実装するオブジェクトのいずれかを指定します。このプロパティに `UserInfo` オブジェクトを渡すと、`Context.PROVIDER_URL` プロパティは無視されます。スレッドが既にユーザに関連付けられていない限り、プロパティのデフォルト値は `guest` になります。詳細については、「コンテキストを使用した名前付きオブジェクトのルックアップ」を参照してください。

クライアントまたはサーバのどちらでも、同じプロパティを使用できます。サーバサイドオブジェクトでプロパティを定義する場合は、ローカルコンテキストが使用されます。クライアントまたは別の **WebLogic Server** でプロパティを定義する場合は、`Context.PROVIDER_URL` プロパティで指定された **WebLogic Server** で実行中のリモート コンテキストに委託されます。

一部のプロパティは、コンテキストの作成後に変更することができません。これらのプロパティには、プロバイダ URL、ユーザ資格、およびファクトリがあります。AddToEnvironment を使用すると、その他のプロパティをコンテキストの作成後に変更できます。

コードリスト 2-1 は、`Context.INITIAL_CONTEXT_FACTORY` プロパティと `Context.PROVIDER_URL` プロパティを使用して、コンテキストを取得する方法を示しています。

コードリスト 2-1 コンテキストの取得

```
Context ctx = null;
Hashtable ht = new Hashtable();
ht.put(Context.INITIAL_CONTEXT_FACTORY,
        "weblogic.jndi.WLInitialContextFactory");
ht.put(Context.PROVIDER_URL,
        "t3://localhost:7001");

try {
    ctx = new InitialContext(ht);
    // プログラムではこのコンテキストを使用する
}
catch (NamingException e) {
// エラー発生
}
finally {
    try {ctx.close();}
    catch (Exception e) {
// エラー発生
    }
}
```

この他にも、セキュリティ パラメータをコンフィグレーションしたり、オブジェクトをクラスタワイドの JNDI ツリーにバインドする方法を指定したりするための WebLogic 固有のプロパティがあります。バインドは、クラスタ内にある各サーバの JNDI ツリー間でレプリケートされる場合とされない場合があります。このようなプロパティは、weblogic.jndi.WLContext クラス内の定数によって示されます。JNDI に関連するクラスタ化の問題の詳細については、「クラスタ環境でのクライアントからの WebLogic JNDI の使い方」を参照してください。

ハッシュ テーブルを使用したコンテキストの作成

ハッシュ テーブルを使用して、コンテキストを作成できます。ハッシュ テーブルには、2-2 ページの「InitialContext への JNDI 環境プロパティの設定」で説明したプロパティを指定しておきます。

まず、ハッシュ テーブルを InitialContext のコンストラクタに渡します。java.naming.factory.initial プロパティを使用して、InitialContext の作成方法を指定します。WebLogic JNDI を使用するには、

`java.naming.factory.initial` プロパティを常に `weblogic.jndi.WLInitialContextFactory` に設定する必要があります。この設定値は、コンテキストを実際に作成するファクトリを示します。

WebLogic Environment オブジェクトを使用したコンテキストの作成

`weblogic.jndi.environment` インタフェースによって実装される **WebLogic Environment** オブジェクトを使用して、コンテキストを作成することもできます。**Environment** オブジェクトは **WebLogic** 固有のものですが、以下のような利点があります。

- 一連のデフォルト値が用意されているので、記述するコードが少なく済みます。
- コンパイル タイプのタイプ保証を提供する便利な `set()` メソッドが用意されています。タイプ保証の `set()` メソッドを使用することで、コードの記述とデバッグの時間を短縮できます。

WebLogic Environment オブジェクトには、以下のデフォルト値が用意されています。

- `InitialContext` ファクトリを指定しない場合は、デフォルト値として `WLInitialContextFactory` が使用されます。
- `Context.SECURITY_PRINCIPAL` プロパティおよび `Context.CREDENTIALS` プロパティでユーザおよびパスワードを指定しない場合は、スレッドが既にユーザに関連付けられていない限り、ユーザとパスワードのデフォルト値として、`guest` が使用されます。
- `Context.PROVIDER_URL` プロパティを指定しない場合は、デフォルト値として `t3://localhost:7001` が使用されます。

これらのデフォルト値を使用して `InitialContext` を作成する場合のコードは、以下のようになります。

```
Environment env = new Environment();
Context ctx = env.getInitialContext();
```

たとえば、クライアントのクラスタ アクセスのために、1つの WebLogic Server だけにドメイン ネーム サービス (DNS) 名を設定する場合のコードは、以下のようになります。

```
Environment env = new Environment();
env.setProviderURL("t3://myweblogiccluster.com:7001");
Context ctx = env.getInitialContext();
```

注意: 新しい JNDI `Environment` オブジェクトを作成するたびに、新しいセキュリティ スコープが作成されます。このセキュリティ スコープは、`context.close()` メソッドで終了します。

`environment.getInitialContext()` メソッドは、IIOP プロトコルでは正しく動作しません。

コードリスト 2-2 は、JNDI `Environment` オブジェクトを使用してセキュリティ コンテキストを作成する方法を示しています。

コード リスト 2-2 JNDI `Environment` オブジェクトを使用したセキュリティ コンテキストの作成

```
weblogic.jndi.Environment environment = new
weblogic.jndi.Environment();
env.setInitialContextFactory(
    weblogic.jndi.Environment.DEFAULT_INITIAL_CONTEXT_FACTORY);
env.setProviderURL("t3://bross:4441");
env.setSecurityPrincipal("guest");
env.setSecurityCredentials("guest");
Hashtable props = env.getProperties();
InitialContext ctx = new InitialContext(props);
```

サーバサイド オブジェクトからのコンテキストの作成

エンタープライズ JavaBean (EJB) オブジェクトや Remote Method Invocation (RMI) オブジェクトのような、WebLogic Server の Java 仮想マシン (JVM) でインスタンス化されるオブジェクトからコンテキストを作成することが必要になる場合もあります。サーバサイド オブジェクトを使用する場合は、`Context.PROVIDER_URL` プロパティを指定する必要はありません。特定のユー

ザとして署名し、ログオンする場合にのみ、ユーザ名とパスワードが必要になります。サーバサイドのコンテキストは **WebLogic Server** のコンテキストで動作します。

サーバサイド オブジェクト内からコンテキストを作成するには、まず次のように、新しい **InitialContext** を作成する必要があります。

```
Context ctx = new InitialContext();
```

ファクトリやプロバイダの **URL** を指定する必要はありません。デフォルトで、コンテキストは **Context** として作成され、ローカルのネーミング サービスに接続されます。

JNDI コンテキストとスレッド

ユーザ名とパスワードを使用して **JNDI** コンテキストを作成すると、ユーザとスレッドが関連付けられます。コンテキストが作成されると、ユーザは、スレッドに関連付けられたコンテキスト スタックにプッシュされます。スレッドで新しいコンテキストを開始する前に、最初のコンテキストを閉じて、最初のユーザとスレッドの関連付けを断つ必要があります。そうしないと、新しいコンテキストが作成されるたびに、ユーザはスタックの下にプッシュされていきます。これはリソースの使い方として効率のよいものではなく、`ctx.lookup()` の呼び出しに対して正しくないユーザが返される場合があります。このシナリオについては、以下の手順を使って説明します。

1. `user1` に対して `ctx1` という名前のコンテキスト (ユーザ名と資格を指定して) を作成します。その過程で、`user1` はスレッドと関連付けられ、スレッドに関連付けられたスタックにプッシュされます。この時点では、現在のユーザは `user1` です。
2. `user2` に対して `ctx2` という名前の別のコンテキスト (ユーザ名と資格を指定して) を作成します。この時点で、スレッドにはユーザのスタックが関連付けられています。スタックの一番上には `user2` があり、`user1` はその下にあるので、現在のユーザとしては `user2` が使用されます。
3. `ctx1.lookup("abc")` の呼び出しを行うと、`user2` がスタックの一番上にあるので、**ID** としては `user1` ではなく `user2` が使用されます。意図した結果を得る、つまり `ctx1.lookup("abc")` の呼び出しが `user1` として行われるようにするためには、`ctx2.close()` を呼び出す必要があります。

`ctx2.close()` を呼び出すと、スレッドに関連付けられたスタックから `user2` が削除されて、`ctx1.lookup("abc")` の呼び出しは意図したとおりに `user1` を使用するようになります。

注意： `close()` の呼び出しでスタックから現在のユーザが削除されず、それが原因で JNDI コンテキストの問題が生じる状況が 2 つあります。JNDI コンテキストの問題を回避する方法については、2-8 ページの「JNDI コンテキストの潜在的な問題を回避する方法」を参照してください。

JNDI コンテキストの潜在的な問題を回避する方法

`close()` を呼び出すと通常は 2-7 ページの「JNDI コンテキストとスレッド」で説明されているとおりに動作しますが、予期される動作の例外が 2 つあります。

- 最初のログイン
- 最後の使用

最初のログイン

IIOP 以外のプロトコルを使用する場合、最初のユーザは他のユーザが存在しない場合にデフォルトのユーザになるという意味で「粘着性」があります。以下の手順では、このシナリオについて説明します。

1. `user1` に対して `ctx1` という名前のコンテキスト (ユーザ名と資格を指定して) を作成します。その過程で、`user1` はスレッドと関連付けられ、スタックに格納されます。つまり、現在の ID は `user1` に設定されます。
2. `ctx1.close()` を呼び出します。
3. `ctx1.lookup()` を呼び出します。現在の ID は `user1` です。
4. `user2` に対して `ctx2` という名前のコンテキスト (ユーザ名と資格を指定して) を作成します。その過程で、`user2` はスレッドと関連付けられ、スタックに格納されます。つまり、現在の ID は `user2` に設定されます。
5. `ctx2.close()` を呼び出します。
6. `ctx2.lookup()` を呼び出します。現在の ID は `user1` です。

ctx1 は最初のユーザなので、手順 4 の後も現在の ID は user1 に設定されたままになります。user1 はこのスレッドでの現在のユーザであるだけでなく、他の ID が定義されていないすべてのスレッドにおける現在のユーザでもあることに注意してください。つまり、他のユーザ ID が存在していない場合、user1 がデフォルト ユーザになります。その以降のログインでユーザ名と資格が示されないと、デフォルトとして user1 の ID が与えられてしまうので、これはよい方法ではありません。

この問題を回避するには、以下のいずれかの方法を実行します。

- **オプション 1:** クライアントが main() の制御を持つ場合、コードリスト 2-3 に示すラッパー コードをクライアント コードに実装します。

コード リスト 2-3 JNDI コンテキストとスレッドのラッパー コード

```
import java.security.PrivilegedAction;
import javax.security.auth.Subject;
import weblogic.security.Security;

public class client
{
    public static void main(String[] args)
    {
        Security.runAs(new Subject(),
            new PrivilegedAction() {
                public Object run() {
                    //
                    // クライアント コードに実装する場合、main() はここに記述する
                    //
                    return null;
                }
            });
    }
}
```

- **オプション 2:** クライアントが main() の制御を持たない場合、コードリスト 2-3 に示したラッパー コードを各スレッドの run() メソッドに実装します。
- **オプション 3:** 特権を持たないユーザ (どのグループのメンバーにもなっていないユーザ) としてログインするコンテキストを作成します。これがクライアントに最初にログオンするユーザになるようにします。直ちに ctx.close() を呼び出して、スレッドのユーザ スタックからそのユーザを

削除します。特権を持たないユーザが最初にログオンしたので、そのユーザがデフォルト ユーザとなります。以後、空のユーザ スタックを持つすべてのスレッドが特権を持たないユーザの ID を持ちます。

注意： オプション 3 を選択した場合、特権を持たないユーザと、WebLogic Server のこのリリースのセキュリティ レルムでコンフィグレーションする `users` および `everyone` グループとの関連に注意してください。`users` と `everyone` は、グローバル ロールとセキュリティ ポリシーを適用する場合に便利なグループです。デフォルトでは、特権を持たないユーザを含むすべての WebLogic Server ユーザは `everyone` グループのメンバーですが、特権を持たないユーザは `users` グループのメンバーではありません。

最後の使用

IIOP を使用している場合は、スタックに 1 つのコンテキストがあり、そのコンテキストが `close()` で削除されるときに、予期しない動作が生じます。スタックから削除される最後のコンテキストの ID で、ユーザの現在の ID が決まります。以下の手順では、このシナリオについて説明します。

1. `user1` に対して `ctx1` という名前のコンテキスト (ユーザ名と資格を指定して) を作成します。その過程で、`user1` はスレッドと関連付けられ、スタックに格納されます。つまり、現在の ID は `user1` に設定されます。
2. `ctx1.close()` を呼び出します。
3. `ctx1.lookup()` を呼び出します。現在の ID は `user1` です。
4. `user2` に対して `ctx2` という名前のコンテキスト (ユーザ名と資格を指定して) を作成します。その過程で、`user2` はスレッドと関連付けられ、スタックに格納されます。つまり、現在の ID は `user2` に設定されます。
5. `ctx2.close()` を呼び出します。
6. `ctx2.lookup()` を呼び出します。現在の ID は `user2` です。

コンテキストを使用した名前付きオブジェクトのルックアップ

すべての Java クライアントアプリケーションは、アプリケーションにサービスを提供する最初のオブジェクトを取得する必要があります。ただし、そのオブジェクトが WebLogic Server ネームスペースにロードされていない場合は、そのオブジェクトをルックアップできません。詳細については、『管理者ガイド』の「JNDI の管理」を参照してください。

コンテキストで `lookup()` メソッドを使用して、名前付きオブジェクトを取得します。`lookup()` メソッドに渡される引数は、目的のオブジェクト名を含む文字列です。コードリスト 2-4 は、`ServiceBean` という名前の EJB を取り出す方法を示しています。

コード リスト 2-4 名前付きオブジェクトのルックアップ

```
try {
    ServiceBean bean = (ServiceBean)ctx.lookup("ejb.serviceBean");
}
catch (NameNotFoundException e) {
    // バインドが存在しない
}
catch (NamingException e) {
    // エラー発生
}
```

名前付きオブジェクトを使用したオブジェクト参照の取得

EJB クライアントアプリケーションは、EJB ホームから EJB リモートオブジェクトへのオブジェクト参照を取得します。RMI クライアントアプリケーションは、最初の名前付きオブジェクトからその他の RMI オブジェクトへのオブジェクト参照を取得します。これらの最初の名前付きリモートオブジェクトは共に、

ファクトリとして **WebLogic Server** に認識されます。ファクトリとは、**WebLogic** ネームスペース内にある別のオブジェクトへの参照を返すことができるオブジェクトのことです。

クライアントアプリケーションは、ファクトリでメソッドを呼び出し、特定のクラスのリモート オブジェクトへの参照を取得します。その後、リモート オブジェクトでメソッドを呼び出し、必要な引数を渡します。

コードリスト 2-5 は、リモート オブジェクトを取得した後に、そのオブジェクトでメソッドを呼び出すコードを示しています。

コード リスト 2-5 名前付きオブジェクトを使用したオブジェクト参照の取得

```
ServiceBean bean = ServiceBean.Home.create("ejb.ServiceBean")
Servicebean.additem(66);
```

コンテキストのクローズ

コンテキストの操作が終了したら、クライアントはコンテキストを閉じてリソースを解放し、メモリ リークを回避してください。この処理は `finally{}` ブロックで行い、`close()` メソッドは `try{}` ブロックの中に入れることをお勧めします。エラーが原因で一度もインスタンス化されなかったコンテキストを閉じようとすると、**Java** クライアントアプリケーションでは例外が送出されます。

コードリスト 2-6 では、クライアントでコンテキストが閉じられ、リソースが開放されています。

コード リスト 2-6 コンテキストのクローズ

```
try {
    ctx.close();
} catch (Exception e) {
    // エラー発生
}
```

クラスタ環境での WebLogic JNDI の使い方

WebLogic JNDI の目的は、J2EE サービス、特に EJB、RMI、および Java Messaging Service (JMS) の各サービスに対してネーミング サービスを提供することです。そのため、クラスタ環境でオブジェクトを JNDI ツリーにバインドすることによる影響を十分理解しておく必要があります。

以降の節では、クラスタ環境での WebLogic JNDI の実装方法について説明し、独自のオブジェクト (カスタム オブジェクト) を JNDI クライアントで使用できるようにするための方法を紹介します。

J2EE サービスのクラスタ化

WebLogic RMI を使用すると、ある JVM のクライアントが別の JVM のクライアントから EJB サービスや JMS サービスにアクセスできるようになります。RMI スタブは、RMI オブジェクトに対するクライアントからの呼び出しを受信し、マーシャリングします。クライアントで J2EE サービスを利用できるようにするために、WebLogic は、特定のサービスの RMI スタブを、特定の名前で JNDI ツリーにバインドします。RMI オブジェクトの他のインスタンスがクラスタ内の他のサーバにデプロイされると、RMI スタブはその場所で更新されます。クラスタ内のサーバに障害が発生すると、他のサーバの JNDI ツリー内の RMI スタブが更新されてサーバの障害を反映します。

クライアントがクラスタに接続する場合、実際には、既にクラスタ内にある WebLogic Server の 1 つに接続されます。その WebLogic Server の JNDI ツリーには、そのサーバによって提供されるサービスだけでなく、クラスタ内の他のサーバによって提供されるすべてのサービスの RMI スタブが含まれているため、クライアントには、クラスタがクラスタワイドのサービスをすべて提供する 1 つの WebLogic Server のように表示されます。クラスタに新しい WebLogic Server が加わった場合、既にクラスタ内に存在する各 WebLogic Server は、それぞれの独自のサービスに関する情報を新しい WebLogic Server に渡して共有する必要があります。クラスタ内の他のすべてのサーバから収集された情報を使用して、新しいサーバはクラスタワイドの JNDI ツリーの独自のコピーを作成できます。

以下に示すように、RMI スタブは、クラスタ環境での WebLogic JNDI の実装方法に大きな影響を与えます。

- RMI スタブは比較的小さいものです。このため、WebLogic JNDI ではサーバ間通信のオーバーヘッドが少ない状態で、クラスタ内のすべての WebLogic Server にスタブをレプリケートできます。
- RMI スタブは、クラスタ全体をレプリケートするメカニズムとして機能します。RMI オブジェクトのインスタンスは 1 つの WebLogic Server にデプロイされますが、スタブはクラスタ全体にレプリケートされます。

カスタム オブジェクトを WebLogic Server クラスタで使用できるようにする方法

カスタム オブジェクト (非 RMI オブジェクト) を WebLogic Server のクラスタ内の JNDI ツリーにバインドすると、オブジェクトはクラスタ内の全サーバにレプリケートされます。ただし、ホストサーバがダウンした場合は、カスタム オブジェクトはクラスタの JNDI ツリーから削除されます。カスタム オブジェクトは、再度バインドされない限り、レプリケートされません。カスタム オブジェクトに加えられた変更を伝播するには、その都度カスタム オブジェクトをアンバインドし、リバインドする必要があります。そのため、WebLogic JNDI は、分散オブジェクトのキャッシュとして使用することはできません。分散キャッシュは、WebLogic Server でサードパーティ製ソリューションを使用して提供できます。

カスタム オブジェクトにアクセスする必要があるのは、1 つの WebLogic Server にデプロイされている EJB だけであると仮定します。この場合、明らかに、カスタム オブジェクトをクラスタ内のすべての WebLogic Server にレプリケートする必要はありません。実際には、サーバ間の不要な通信によるパフォーマンスの低下を回避するために、カスタム オブジェクトがレプリケートされないようにする必要があります。クラスタ内の WebLogic Server 間でレプリケートされないバインドを作成するには、カスタム オブジェクトをネームスペースにバインドするコンテキストの作成時に、`REPLICATE_BINDINGS` プロパティを指定します。コードリスト 2-7 は、`REPLICATE_BINDINGS` プロパティの使い方を示しています。

コード リスト 2-7 REPLICATE_BINDINGS プロパティの使い方

```

Hashtable ht = new Hashtable();
// バインドのレプリケーションを無効にする
ht.put(WLContext.REPLICATE_BINDINGS, "false");
try {
    Context ctx = new InitialContext(ht);
    // オブジェクトをバインドする
    ctx.bind("my_object", MyObect);
} catch (NamingException ne) {
    // エラー発生
}
    
```

この方法でカスタム オブジェクトを使用する必要がある場合は、**WebLogic Server** の `InitialContext` を明示的に取得する必要があります。クラスタ内の他の **WebLogic Server** に接続した場合は、バインドは **JNDI ツリー** に表示されません。

クラスタ内のどの **WebLogic Server** からカスタム オブジェクトにアクセスできるようにするには、**JNDI バインド** をレプリケートせず、カスタム オブジェクトをクラスタ内の各 **WebLogic Server** にデプロイします。

WebLogic JNDI を使用してバインドをレプリケートすると、バインドされているオブジェクトは、ホストになっている **WebLogic Server** によって所有されるように処理されます。ホストになっている **WebLogic Server** に障害が発生すると、クラスタ内のすべての **WebLogic Server** のすべての **JNDI ツリー** からカスタム オブジェクトが削除されます。この動作によって、カスタム オブジェクトの可用性が低下するおそれがあります。

注意： **JNDI** の `Context.bind(String Name)` では、/ または - は使用できません。バインド名の文字列に / または - が含まれる場合、`javax.naming.NameNotFoundException` が送出されます。

「データ キャッシュ」設計パターン

Web アプリケーション には、複数のオブジェクトによって使用されるデータを一定期間キャッシュし、データ計算や別のサービスへの接続に関連するオーバーヘッドを回避するという一般的なタスクがあります。

単一の WebLogic Server で正常に動作するカスタム データ キャッシュ オブジェクトを既に設計しており、WebLogic クラスタ内でこのオブジェクトの使用とします。いずれかの WebLogic Server の JNDI ツリーにそのデータ キャッシュ オブジェクトをバインドすると、デフォルトでは、WebLogic JNDI はクラスタ内のその他の各 WebLogic Server にそのオブジェクトをコピーします。このオブジェクトは RMI オブジェクトではないので、JNDI ツリーにバインドされ、他の WebLogic Server にコピーされているものはオブジェクト自身であり、いずれかの WebLogic Server をホストにするオブジェクトの単一のインスタンスを参照するスタブではない点に注意してください。WebLogic Server がサーバ間でオブジェクトをコピーしても、カスタム オブジェクトを分散キャッシュとして使用できるわけではありません。カスタム オブジェクトは、バインドされている WebLogic Server に障害が発生するとクラスタから削除されます。カスタム オブジェクトへの変更は、オブジェクトをアンバインドして JNDI ツリーにリバインドしない限り、クラスタ内に伝播されません。

パフォーマンスと可用性の理由から、クラスタ内のすべての WebLogic Server で高可用性が求められる場合は、WebLogic JNDI のバインドのレプリケーションを使用して大きなカスタム オブジェクトをコピーすることはお勧めできません。その代わりに、カスタム オブジェクトの別々のインスタンスを各 WebLogic Server にデプロイする方法があります。各 WebLogic Server の JNDI ツリーにオブジェクトをバインドするときは、「カスタム オブジェクトを WebLogic Server クラスタで使用できるようにする方法」で説明するように、バインドのレプリケーションを無効にしておく必要があります。この設計パターンでは、各 WebLogic Server はカスタム オブジェクトのコピーを保持しますが、サーバ間で大量のデータがコピーされることはありません。

どの手法を使用する場合でも、オブジェクトの各インスタンスは、クラスタ内の他のデータ キャッシュ オブジェクトとは無関係に自分のキャッシュを更新する必要があるときに備えて、独自のロジックを保持する必要があります。たとえば、ある WebLogic Server 上のデータ キャッシュにクライアントがアクセスするとします。そのキャッシュ オブジェクトがアクセスされるのはその時が初めてなので、オブジェクトは情報を計算または取得し、将来のリクエストに備えて情報のコピーを保存します。次に、別のクライアントがクラスタに接続し、最初のクライアントと同じタスクを実行するとします。ただし、今回は、クラスタ内の別の WebLogic Server に接続します。この特定のデータ キャッシュ オブジェクトにとってそれが最初に受け付けるアクセスである場合は、クラスタ内の他のデータ キャッシュ オブジェクトが既に情報をキャッシュしているかどうかに関係なく、アクセスされたオブジェクトは情報を計算する必要があります。もちろん、データ キャッシュ オブジェクトのこのインスタンスは、将来のリクエストに対しては保存してある情報を参照できます。

「サービスを各クラスタで一度だけ提供する」設計パターン

サービスをクラスタ内で一度だけ提供することが望ましい場合もあります。これを実現するには、1つのマシンだけにサービスをデプロイします。RMI オブジェクトの場合は、WebLogic JNDI のデフォルト動作を使用して、バインド (RMI スタブ) をレプリケートすることによって、クラスタ内のすべての WebLogic Server からオブジェクトの単一のインスタンスにアクセスできるようになります。このようなサービスは、固定サービスと呼ばれます。非 RMI オブジェクトの場合は、オブジェクトをネームスペースにバインドするときに、必ず `REPLICATE_BINDINGS` プロパティを使用してください。この場合、オブジェクトにアクセスするには、ホストになっている WebLogic Server に明示的に接続する必要があります。代わりに、RMI オブジェクトを作成し、ホストになっている同じ WebLogic Server にデプロイすることもできます。この RMI オブジェクトは、非 RMI オブジェクトのプロキシとして機能します。WebLogic JNDI のデフォルト動作を使用してプロキシのスタブをレプリケートすれば、クラスタ内の任意の WebLogic Server に接続したクライアントは、RMI プロキシを経由して非 RMI オブジェクトにアクセスできます。

このような、サービスを各クラスタで一度だけ提供する設計パターンでは、高可用性が求められるサービスの場合、新たな課題が発生します。WebLogic クラスタのフェイルオーバー機能は、クラスタ化されたサービスごとに複数のデプロイメントがあることに基づいているため、サービスを各クラスタで一度だけ提供の場合は、フェイルオーバー機能を使用できません。高可用性が求められるサービスの場合は、ホストになっている WebLogic Server にハードウェアによる高可用性 (HA:High-Availability) フレームワークを実装することをお勧めします。フレームワークにより、障害が発生した場合でも、サービスの可用性の低下を最小限に留めた状態で、WebLogic Server を再起動できます。

クラスタ環境でのクライアントからの WebLogic JNDI の使い方

オブジェクトの JNDI バインドは、クラスタ内の 1つの WebLogic Server の JNDI ツリーに表示することも、クラスタ内のすべての WebLogic Server にレプリケートすることもできます。目的のオブジェクトがただ 1つの WebLogic Server だけ

にバインドされている場合は、「Java クライアントからの WebLogic JNDI の使い方」で説明されているように、初期コンテキストの作成時に、`Context.PROVIDER_URL` プロパティをホストになっている **WebLogic Server** の URL に設定することで、ホストになっている **WebLogic Server** に明示的に接続する必要があります。

ただし、ほとんどの場合、目的のオブジェクトは、クラスタ化されたサービスか、または固定サービスです。つまり、クラスタ内の各 **WebLogic Server** の JNDI ツリーには、サービスのスタブが表示されます。この場合、クライアントは、ネーミング サービスを提供する特定の **WebLogic Server** の名前を指定する必要はありません。実際、**WebLogic** クラスタにネーミング サービスを提供するようリクエストするのが、クライアントにとって最善の方法です。その場合、**WebLogic Server** のコンテキスト ファクトリによって、クライアントに対して最も適切であると思われる **WebLogic Server** がクラスタの中から選択されます。現在、ネーミング サービス プロバイダは、DNS ラウンドロビン方式を使用して **WebLogic** のの中から選択されます。

一般に、クラスタ化されたサービスのクライアントに対して返されるコンテキストは、フェイルオーバー スタブとして実装されており、選択された **WebLogic Server** との間で通信障害などの障害が発生した場合は、ネーミング サービス プロバイダを透過的に変更できます。

コードリスト 2-8 は、クラスタのネーミング サービスをクライアントが使用する方法を示しています。

コードリスト 2-8 WebLogic クラスタでのネーミング サービスの使い方

```
Hashtable ht = new Hashtable();
ht.put(Context.INITIAL_CONTEXT_FACTORY,
        "weblogic.jndi.WLInitialContextFactory");
ht.put(Context.PROVIDER_URL, "t3://acmeCluster:7001");
try {
    Context ctx = new InitialContext(ht);
    // クライアントの作業を行う
}
catch (NamingException ne) {
    // エラー発生
}
finally {
    try {ctx.close();}
    catch (Exception e) {
// エラー発生
    }
}
```

プロバイダの URL の一部として指定されている `hostname` は、クラスタの DNS 名であり、`config.xml` の `Cluster` スタンザ内の `ClusterAddress` で定義できます。`ClusterAddress` は、このクラスタ内のネーミング サービスを提供するホストのリストにマップします。詳細については、『WebLogic Server クラスタの使用』の「クラスタのコンフィグレーションとアプリケーションのデプロイメント」を参照してください。

コードリスト 2-8 では、`acmeCluster` というクラスタ名を使用して、クラスタ内の任意の **WebLogic Server** に接続しています。結果として得られるコンテキストは、クラスタ内の任意の **WebLogic Server** に透過的にフェイルオーバーできるようにレプリケートされます。

WebLogic クラスタとの最初の接続ポイントを指定するもう 1 つの方法は、カンマで区切った DNS サーバ名または IP アドレスのリストを入力することです。

- 次の例では、同じポートを使用する **WebLogic Server** のリストが指定されています。

```
ht.put(Context.PROVIDER_URL, "t3://acme1,acme2,acme3:7001");
```

すべての **WebLogic Server** は URL の最後で指定されているポートでリスンする必要があります。

- 次の例では、異なるポートを使用する **WebLogic Server** のリストが指定されています。

```
ht.put(Context.PROVIDER_URL, "t3://node1:7001,node2:7002,node3:7003");
```

JNDI とクラスタの詳細については、「**WebLogic Server** のクラスタ化の概要」を参照してください。

