BEA WebLogic Server™

BEA WebLogic Express™

管理者ガイド

著作権

限定的権利条項

商標または登録商標

WebLogic Server 管理者ガイド

マニュアルの日付	ソフトウェアのバージョン
2001 年 9 月 19 日	WebLogic Server バージョン 6.1

目次

このマニュアルの内容

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

1. WebLogic Sever 管理の概要

ドメイン、管理サーバ、管理対象サーバ 1-2
Administration Console の起動 1-4
実行時オブジェクトとコンフィグレーション オブジェクト 1-5
アクセス ログ メッセージの一元管理 1-7
新しいドメインの作成 1-8

2. WebLogic Server の起動と停止

WebLogic 管理サーバと WebLogic 管理対象サーバ 2-1
起動メッセージ 2-2
WebLogic 管理サーバの起動 2-2
WebLogic Server 起動時のパスワードの使用 2-3
[スタート] メニューを使用した WebLogic 管理サーバの起動 2-3
Windows サービスとしての WebLogic Server の起動と停止 2-4
コマンドラインからの WebLogic 管理サーバの起動 2-4
クラスパス オプションの設定 2-8
スクリプトを使用した管理サーバの起動 2-9
管理対象サーバの動作中における管理サーバの再起動 2-10
同じマシンでの管理サーバの再起動 2-10
別のマシンでの管理サーバの再起動 2-11
WebLogic 管理対象サーバのドメインへの追加 2-12
WebLogic 管理対象サーバの起動 2-12
スクリプトを使用した WebLogic 管理対象サーバの起動 2-14
Administration Console からの WebLogic Server の停止 2-15

コマンドラインからのサーバの停止 2-16
管理対象サーバのサスペンドと再開 2-17
WebLogic Server の Windows サービスとしての設定 2-17
Windows サービスとしての WebLogic Server の削除 2-18
Windows サービスとしてインストールされた WebLogic Server のパスワードの変更 2-19
WebLogic Server Windows サービス プログラム（beasvc.exe） 2-20
スタートアップ クラスとシャットダウン クラスの登録 2-21

3. ノード マネージャ

ノード マネージャの概要 3-1
ノード マネージャのログ 3-2
ノード マネージャの設定 3-4
セキュア ソケット レイヤ プロトコル向けのノード マネージャの設定 3-6
手順 1: デジタル証明書とプライベート キーの取得 3-7
手順 2 : WebLogic 形式のプライベート キーの変換 3-7
手順 3 : 証明書の証明書ファイルへの結合 3-8
ノード マネージャを使用するように管理サーバを設定 3-8
手順 1: マシンのコンフィグレーション エントリの作成 3-9
手順 2: 各マシンでのノード マネージャのコンフィグレーション 3-9
手順 3: 管理対象サーバの起動情報のコンフィグレーション 3-10
ノード マネージャのプラットフォーム サポート 3-11
コマンドラインからのノード マネージャの起動 3-12
環境の設定 3-12
Windows での環境変数の設定 3-13
Unix での環境変数の設定 3-13
クラスパスの設定 3-14
ノード マネージャの起動 3-14
コマンドライン引数 3-14
クラスパス オプション 3-16
起動スクリプトを使用したノード マネージャの起動 3-17
管理対象サーバのリモートでの起動と強制停止 3-17
管理対象サーバの停止と強制停止の区別 3-18
ドメインおよびクラスタの起動と強制停止 3-18

ノード マネージャの Windows サービスとしての設定 3-19
Windows サービスとしてのノード マネージャの削除 3-21

4. WebLogic Server とクラスタのコンフィグレーション

サーバとクラスタのコンフィグレーションの概要 4-2
管理サーバの役割 4-2
Administration Console の起動 4-4
動的コンフィグレーションの仕組み 4-5
クラスタ コンフィグレーションのプランニング 4-6
サーバ コンフィグレーションの作業 4-7
クラスタ コンフィグレーションの作業 4-11

5. WebLogic Server ドメインのモニタ

モニタの概要 5-1
サーバのモニタ 5-2
パフォーマンス 5-3
サーバのセキュリティ 5-3
JMS 5-3
JT 5-4
JDBC 接続プール のモニタ 5-4

6. ログ メッセージを使用した WebLogic Server の管理

ロギング サブシステムの概要 6-1
ローカル サーバのログ ファイル 6-4
クライアントのロギング 6-5
ログ ファイル フォーマット 6-5
メッセージの属性 6-6
メッセージ カタログ 6-7
メッセージの重要度 6-8
デバッグ メッセージ 6-9
ログ ファイルの参照 6-9
ログの表示 6-10
ドメイン ログ フィルタの作成 6-10

7. アプリケーションのデプロイメント

デプロイメントのサポート形式 7-1

Administration Console を使用したアプリケーションのデプロイ 7-2
手順 1: アプリケーションのコンフィグレーションとデプロイ.....7-2
手順 2: アプリケーション コンポーネントのデプロイメント 7-3
Web アプリケーション コンポーネントのデプロイメント 7-3
EJB コンポーネントのデプロイメント.....7-4
リソース アダプタ コンポーネントのデプロイメント.....7-5
デプロイ順.....7-6
自動デプロイメント.....7-7
自動デプロイメントの有効化または無効化 7-7
展開ディレクトリ形式によるアプリケーションの自動デプロイメント 7-8
自動デプロイメント アプリケーションのアンデプロイメントと再デプロイメント 7-8
展開形式で自動デプロイされたアプリケーションの再デプロイメント.....7-9

8. WebLogic Server Web コンポーネントのコンフィグレーション

概要 8-2
HTTP パラメータ 8-2
リスンポートのコンフィグレーション 8-4
Web アプリケーション 8-4
Web アプリケーションとクラスタ化.....8-5
デフォルト Web アプリケーションの指定 8-5
仮想ホスティングのコンフィグレーション.....8-7
仮想ホスティングとデフォルト Web アプリケーション 8-8
仮想ホストの設定 8-8
WebLogic Server による HTTP リクエストの解決方法 8-11
HTTP アクセス ログの設定.....8-14
ログ ローテーション.....8-14
Administration Console を使用した HTTP アクセス ログの設定 8-15
共通ログ フォーマット.....8-16
拡張ログ フォーマットを使用した HTTP アクセス ログの設定 8-17
Fields ディレクティブの作成.....8-18
サポートされるフィールド識別子 8-18
カスタム フィールド識別子の作成 8-20

POST サービス拒否攻撃の防止 8-25
HTTP トンネリングのための WebLogic Server の設定 8-26
HTTP トンネリング接続の設定 8-26
クライアントからの WebLogic Server への接続 8-27
静的ファイルを提供するネイティブ I/O の使用（Windows のみ） 8-28

9. 別の HTTP サーバへのリクエストのプロキシ

概要 9-1
セカンダリ HTTP サーバへのプロキシの設定 9-2
プロキシ サーブレットのデプロイメント記述子のサンプル 9-3

10. WebLogic クラスタ へのリクエストのプロキシ

概要 10-1
HttpClusterServlet の設定 10-2
HttpClusterServlet 用デプロイメント記述子のサンプル 10-5

11. Apache HTTP Server プラグインのインストールとコンフィグレーション

概要 11-2
Apache バージョン 1.3.x のキープアライブ接続 11-2
Apache バージョン 2.x のキープアライブ接続 11-2
リクエストのプロキシ 11-3
プラットフォーム サポート 11-3
Apache HTTP Server プラグインのインストール 11-4
動的共有オブジェクトとしてのインストール 11-4
静的リンク モジュールとしてのインストール 11-7
Apache HTTP Server プラグイン のコンフィグレーション 11-9
httpd.conf ファイルの編集 11-9
httpd.conf ファイルの編集に関する注意事項 11-11
Apache プラグインでの SSL の使用 11-12
Apache HTTP Server プラグインと WebLogic Server の間の SSL のコンフィグレーション 11-13
SSL-Apache コンフィグレーションに関する問題 11-14
接続エラーとクラスタのフェイルオーバ 11-15
接続の失敗 11-15

クラスタ化されていない単一 WebLogic Server での
フェイルオーバ 11-15
動的サーバ リスト 11-16
フェイルオーバ、クッキー、および HTTP セッション 11-16
httpd.conf ファイルのテンプレート 11-18
コンフィグレーション ファイルのサンプル 11-18
WebLogic クラスタを使用した例 11-19
複数の WebLogic Cluster を使用した例 11-19
WebLogic クラスタを使用しない例 11-19
IP ベースの仮想ホスティングのコンフィグレーション例 11-20
単一 IP アドレスによる名前ベースの仮想ホスティングの
コンフィグレーション例 11-20

12. Microsoft Internet Information Server（ISAPI）プラグインのインストールとコンフィグレーション

Microsoft Internet Information Server プラグインの概要 12-2
接続プールとキープアライブ 12-2
リクエストのプロキシ 12-2
プラットフォーム サポート 12-3
Microsoft Internet Information Server プラグインのインストール 12-3
IIS の複数の仮想 Web サイトのプロキシ 12-7
IIS を介した ACL の作成 12-8
iisproxy.ini ファイルのサンプル 12-9
Microsoft Internet Information Server プラグインでの SSL の使用 12-10
IIS から WebLogic Server へのサーブレットのプロキシ 12-11
インストールのテスト 12-12
接続エラーとクラスタのフェイルオーバ 12-13
接続の失敗 12-13
クラスタ化されていない単一 WebLogic Server での
フェイルオーバ 12-13
動的サーバ リスト 12-14
フェイルオーバ、クッキー、および HTTP セッション 12-14

13. Netscape Enterprise Server プラグイン（NSAPI）のインストールとコンフィグレーション

Netscape Enterprise Server プラグインの概要 13-2

接続プールとキープアライブ 13-3
リクエストのプロキシ 13-3
Netscape Enterprise Server プラグインのインストールとコンフィグレーション 13-3
obj.conf ファイルの修正 13-5
NSAPI プラグインでの SSL の使用 13-10
接続エラーとクラスタのフェイルオーバ 13-11
接続の失敗 13-11
クラスタ化されていない単一 WebLogic Server でのフェイルオーバ 13-11
動的サーバ リスト 13-12
フェイルオーバ、クッキー、および HTTP セッション 13-12
ファイアウォールとロード ディレクタを使用する場合のフェイルオーバの動作 13-14
obj.conf ファイルのサンプル（WebLogic クラスタを使用しない場合） 13-15
obj.conf ファイルのサンプル（WebLogic クラスタを使用する場合） 13-17

14. セキュリティの管理

セキュリティのコンフィグレーション手順 14-2
システム パスワードの変更 14-3
セキュリティ レルムの指定 14-5
ファイル レルムのコンフィグレーション 14-5
キャッシング レルムのコンフィグレーション 14-7
LDAP セキュリティ レルムのコンフィグレーション 14-13
LDAP セキュリティ レルム使用時の制限 14-15
LDAP レルム V1 のコンフィグレーション 14-16
LDAP レルム V2 のコンフィグレーション 14-20
Windows NT セキュリティ レルムのコンフィグレーション 14-22
UNIX セキュリティ レルムのコンフィグレーション 14-26
RDBMS セキュリティ レルムのコンフィグレーション 14-29
カスタム セキュリティ レルムのインストール 14-33
セキュリティ レルムの移行 14-35
ユーザの定義 14-36
グループの定義 14-38
ACL の定義 14-39
SSL プロトコルのコンフィグレーション 14-42

プライベート キーとデジタル証明書の取得 14-42
プライベート キーとデジタル証明書の保存 14-47
信頼された認証局の定義 14-48
SSL プロトコル用の属性の定義 14-48
SSL セッション キャッシングのパラメータの変更 14-54
相互認証のコンフィグレーション 14-55
SSL を使用した RMI over IIOP のコンフィグレーション 14-55
パスワードの保護 14-56
監査プロバイダのインストール 14-59
接続フィルタのインストール 14-60
Java セキュリティ マネージャの設定 14-61
レコーディング セキュリティ マネージャ ユーティリティの使い方 14-63
セキュリティ コンテキストの伝播のコンフィグレーション 14-63

15. トランザクションの管理

トランザクション管理の概要 15-1
トランザクションのコンフィグレーション 15-2
トランザクションのモニタとログ 15-4
別のマシンへのサーバの移動 15-5

16. JDBC 接続の管理

JDBC 管理の概要 16-1
Administration Console について 16-2
コマンドライン インタフェースについて 16-2
JDBC API について 16-2
関連情報 16-2
管理 16-3
JDBC と WebLogic jDrivers 16-3
トランザクション（JTA） 16-4
JDBC コンポーネント（接続プール、データ ソース、および
マルチプール） 16-4
接続プール 16-4
マルチプール 16-5
データ ソース 16-5
接続プール、マルチプール、およびデータソースの JDBC コンフィグ
レーションガイドライン 16-6

JDBC コンフィグレーションの概要 16-6
ローカル トランザクションをサポートするドライバ....16-8
分散トランザクションをサポートするドライバ 16-8
JDBC ドライバのコンフィグレーション....16-8
ローカル トランザクション用の JDBC ドライバのコンフィグレーション 16-9
分散トランザクション用の XA 対応 JDBC ドライバのコンフィグレーション 16-13
WebLogic jDriver for Oracle/XA のデータ ソース プロパティ ...16-16
分散トランザクション用の XA 非対応 JDBC ドライバのコンフィグレーション 16-19
Administration Console による JDBC 接続プール、マルチプール、およびデータソースのコンフィグレーションと管理 16-21
JDBC コンフィグレーション....16-21
JDBC オブジェクトの作成 16-21
JDBC オブジェクトの割り当て....16-22
Administration Console を使用した JDBC 接続のコンフィグレーション 16-23
コマンドライン インタフェースを使用した JDBC コンフィグレーション タスク 16-25
接続の管理とモニタ 16-26
Administration Console を使用した JDBC の管理....16-26
コマンドライン インタフェースを使用した JDBC の管理....16-29

17. JMS の管理

JMS と WebLogic Server 17-1
JMS のコンフィグレーション....17-2
JMS サーバのコンフィグレーション....17-3
接続ファクトリのコンフィグレーション....17-4
送り先のコンフィグレーション 17-6
JMS テンプレートのコンフィグレーション....17-7
送り先キーのコンフィグレーション....17-8
ストアのコンフィグレーション 17-9
JDBC ストア 17-10
JMS ストア....17-10
JMS ストア向けの JDBC 接続プールの推奨設定 17-11

セッション プールのコンフィグレーション 17-12
接続コンシューマのコンフィグレーション 17-13
JMS のモニタ 17-13
JMS オブジェクトのモニタ 17-14
恒久サブスクライバのモニタ 17-14
JMS のチューニング 17-15
永続ストレージ 17-15
ファイル ストアへの同期書き込みの無効化 17-15
永続ストレージへのメッセージのページング 17-16
ページングのコンフィグレーション 17-16
JMS のページング属性 17-23
WebLogic Server の障害からの回復 17-30
WebLogic Server の再起動または交換 17-30
プログラミングの考慮事項 17-32

18. JNDI の管理

JNDI 管理の概要 18-1
JNDI およびネーミング サービスの機能 18-1
JNDI ツリーの表示 18-2
JNDI ツリーへのオブジェクトのロード 18-2

19. WebLogic J2EE コネクタ アーキテクチャの管理

WebLogic J2EE コネクタ アーキテクチャの概要 19-2
新しいリソース アダプタのインストール 19-3
新しいコネクタのコンフィグレーションとデプロイメント 19-3
リソース アダプタのコンフィグレーションとデプロイメント 19-4
デプロイされたリソース アダプタの表示 19-5
デプロイされたリソース アダプタのアンデプロイメント 19-5
デプロイされたリソース アダプタの更新 19-6
モニタ 19-6
コネクタの削除 19-7
リソース アダプタのデプロイメント記述子の編集 19-7

20. WebLogic Server ライセンスの管理

WebLogic Server ライセンスのインストール 20-1
ライセンスの更新 20-2

A. WebLogic Java ユーティリティの使い方

AppletArchiver A-3
構文 A-3
Conversion A-4
der2pem A-5
構文 A-5
例 A-6
dbping A-7
構文 A-7
deploy A-9
構文 A-9
アクション（以下のいずれかを選択） A-9
他の必須引数 A-10
オプション A-11
例 A-13
getProperty A-16
構文 A-16
例 A-16
logToZip A-17
構文 A-17
例 A-18
MulticastTest A-19
構文 A-19
例 A-20
myip A-22
構文 A-22
例 A-22
pem2der A-23
構文 A-23
例 A-23
Schema A-24
構文 A-24
例 A-24
showLicenses A-26
構文 A-26
system A-27

構文 A-27
例 A-27
t3dbping A-28
構文 A-28
verboseToZip A-29
構文 A-29
UNIX の例 A-29
NT の例 A-29
version A-30
構文 A-30
例 A-30
writeLicense A-31
構文 A-31
例 A-31

B. WebLogic Server コマンドライン インタフェース リファレンス

コマンドライン インタフェースについて B-1
始める前に B-2
WebLogic Server のコマンドの使い方 B-2
構文 B-3
引数 B-3
WebLogic Server 管理コマンドのリファレンス B-4
CANCEL_SHUTDOWN B-6
構文 B-6
例 B-6
CONNECT B-7
構文 B-7
例 B-7
HELP B-8
構文 B-8
例 B-8
LICENSES B-9
構文 B-9
例 B-9
LIST B-10

構文 B-10
例 B-10
LOCK B-11
構文 B-11
例 B-11
PING B-12
構文 B-12
例 B-12
SERVERLOG B-13
構文 B-13
例 B-13
SHUTDOWN B-14
構文 B-14
例 B-14
THREAD_DUMP B-15
構文 B-15
UNLOCK B-16
構文 B-16
例 B-16
VERSION B-17
構文 B-17
例 B-17
WebLogic Server 接続プール管理コマンド リファレンス B-18
CREATE_POOL B-20
構文 B-20
例 B-22
DESTROY_POOL B-23
構文 B-23
例 B-23
DISABLE_POOL B-24
構文 B-24
例 B-24
ENABLE_POOL B-25
構文 B-25
例 B-25
EXISTS_POOL B-26

構文 B-26
例 B-26
RESET_POOL B-27
構文 B-27
例 B-27
Mbean 管理コマンド リファレンス B-28
CREATE B-29
構文 B-29
例 B-30
DELETE B-31
構文 B-31
例 B-31
GET B-32
構文 B-32
例 B-33
INVOKE B-34
構文 B-34
例 B-34
SET B-35
構文 B-35

C. Web サーバ プラグインのパラメータ

概要 C-1
Web サーバ プラグインの一般的なパラメータ C-2
Web サーバ プラグインの SSL パラメータ C-13

索引

このマニュアルの内容

このマニュアルでは、WebLogic Server の実装をコンフィグレーションおよびモニタするための管理サブシステムについて説明します。構成は次のとおりです。

- 第 1 章「WebLogic Sever 管理の概要」では、WebLogic Server 管理サブシステムのアーキテクチャについて説明します。
- 第 2 章「WebLogic Server の起動と停止」では、WebLogic Server の起動と停止の手順について説明します。
- 第 3 章「ノード マネージャ」では、ノード マネージャの設定および使用方法について説明します。ノード マネージャは、WebLogic Server のリモートでの起動および停止に使用できます。
- 第 4 章「WebLogic Server とクラスタのコンフィグレーション」では、WebLogic Server ドメインのリソースをコンフィグレーションするための機能について説明します。
- 第 5 章「WebLogic Server ドメインのモニタ」では、WebLogic Server ドメインを構成するリソースをモニタするための WebLogic の機能について説明します。
- 第 6 章「ログ メッセージを使用した WebLogic Server の管理」では、WebLogic Server ドメインを管理するためのローカル ログおよびドメイン全体のログの使い方について説明します。
- 第 7 章「アプリケーションのデプロイメント」では、WebLogic Sever でのアプリケーションのインストールとアプリケーション コンポーネントのデプロイメントについて説明します。
- 第 8 章「WebLogic Server Web コンポーネントのコンフィグレーション」では、WebLogic Server を Web サーバとして使用する方法について説明します。
- 第 9 章「別の HTTP サーバへのリクエストのプロキシ」では、HTTP リクエストを他の Web サーバに転送するプロキシとして、WebLogic Server を機能させる方法について説明します。

- 第 10 章「WebLogic クラスタ へのリクエストのプロキシ」では、WebLogic Server のクラスタに HTTP リクエストをプロキシする方法について説明します。
- 第 11 章「Apache HTTP Server プラグインのインストールとコンフィグレーション」では、WebLogic Server Apache プラグインをインストールおよびコンフィグレーションする方法について説明します。
- 第 12 章「Microsoft Internet Information Server（ISAPI）プラグインのインストールとコンフィグレーション」では、Microsoft Internet Information Server 用の WebLogic Server プラグインをインストールおよびコンフィグレーションする方法について説明します。
- 第 13 章「Netscape Enterprise Server プラグイン（NSAPI）のインストールとコンフィグレーション」では、Netscape Enterprise Server（プロキシ）プラグインをインストールおよびコンフィグレーションする方法について説明します。
- 第 14 章「セキュリティの管理」では、WebLogic Server のセキュリティ リソースとその管理について説明します。
- 第 15 章「トランザクションの管理」では、WebLogic Server ドメイン内で Java トランザクション サブシステムを管理する方法について説明します。
- 第 16 章「JDBC 接続の管理」では、WebLogic Server ドメインにおける Java Database Connectivity（JDBC）リソースの管理について説明します。
- 第 17 章「JMS の管理」では、WebLogic Server ドメインにおける Java Message Service の管理について説明します。
- 第 18 章「JNDI の管理」では、JNDI ネーミング ツリーでのオブジェクトの表示および編集や、JNDI ツリーへのオブジェクトのバインドなど、WebLogic JNDI ネーミング ツリーの使い方について説明します。
- 第 19 章「WebLogic J2EE コネクタ アーキテクチャの管理」では、他のエンタープライズ情報システムへの接続が可能になる WebLogic J2EE プラットフォームの拡張機能を管理する方法について説明します。
- 第 20 章「WebLogic Server ライセンスの管理」では、BEA ライセンスの更新方法について説明します。
- 付録 A「WebLogic Java ユーティリティの使い方」では、開発者およびシステム管理者に提供されている多数のユーティリティについて説明します。

- 付録 B「WebLogic Server コマンドライン インタフェース リファレンス」では、WebLogic Server ドメイン管理用のコマンドライン インタフェースの構文および用法について説明します。
- 付録 C「Web サーバ プラグインのパラメータ」では、Web サーバ プラグインのパラメータについて説明します。

対象読者

このマニュアルは、WebLogic Server プラットフォームとその各種サブシステムを管理するシステム管理者を主な対象としています。

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。

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`

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。

表記法	適用
[ ]	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 Sever 管理の概要

以下の節では、WebLogic Server の管理に利用できるツールについて説明します。

- ドメイン、管理サーバ、管理対象サーバ
- Administration Console の起動
- 実行時オブジェクトとコンフィグレーション オブジェクト
- アクセス ログ メッセージの一元管理
- 新しいドメインの作成

BEA WebLogic Server™ ソフトウェアの実装では、相互に関連する複数のリソースがユーザに提供されます。それらのリソースを管理する作業には、サーバの起動と停止、サーバまたは接続プールでのロード バランシング、リソース コンフィグレーションの選択とモニタ、問題の検出と修正、システム パフォーマンスのモニタと評価、Web アプリケーションやエンタープライズ JavaBean（EJB）といったリソースのデプロイメントなどがあります。

これらの作業を行うためのメイン ツールは、Web ベースの Administration Console です。Administration Console は、WebLogic Administration Service への入り口となるウィンドウです。Sun の Java Management Extension（JMX）規格の実装である Administration Service は、WebLogic リソースを管理するための機能を備えています。

Administration Console では、リソース属性のコンフィグレーション、アプリケーションやコンポーネントのデプロイメント、リソース使用状況（サーバの負荷、Java 仮想マシンのメモリ使用率、データベース接続プールの負荷など）のモニタ、ログ メッセージの表示、サーバの起動と停止といった管理作業を行えます。

ドメイン、管理サーバ、管理対象サーバ

1 単位として管理される WebLogic Server リソースの集合は、ドメインと呼ばれます。ドメインには、1 つまたは複数の WebLogic Server が含まれ、WebLogic Server クラスタが含まれる場合もあります。

ドメインのコンフィグレーションは、Extensible Markup Language （XML）で定義します。ドメインのコンフィグレーションの永続ストレージは、*install_dir*\config*domain_name*\config.xml（*install_dir* は WebLogic Server ソフトウェアがインストールされているディレクトリ）という 1 つの XML コンフィグレーション ファイルで実現します。

ドメインは、独立した管理単位です。アプリケーションがあるドメインにデプロイされた場合、そのアプリケーションのコンポーネントは、そのドメインに含まれないサーバにはデプロイできません。ドメインでクラスタがコンフィグレーションされている場合、クラスタのすべてのサーバもそのドメインの一部となります。ドメインには、複数のクラスタが存在できます。

J2EE アプリケーションとは、デプロイメント ユニット（EAR、WAR、または JAR ファイルなど）にグループ化されるコンポーネントの集合です。アプリケーションで必要とされるさまざまな WebLogic リソース（EJB または Web アプリケーション、サーバまたはクラスタ、JDBC 接続プールなど）は、1 つのドメイン コンフィグレーションの中で定義します。1 つの独立したドメインにそれらのリソースをグループ化すると、それらの相互に関連したリソースを管理するための一元的な観点、およびアクセス ポイントが提供されます。

Administration Service が動作している WebLogic Server は、管理サーバと呼ばれます。Administration Service では、ドメイン全体を一元的にコンフィグレーションおよびモニタできます。ドメインを管理するためには、管理サーバが動作していなければなりません。

1 つのサーバだけが管理サーバとなります。残りのサーバは、管理対象サーバと呼ばれます。各 WebLogic 管理対象サーバのコンフィグレーションは、起動時に管理サーバから取得されます。

同じクラス、`weblogic.Server` を、ドメインの管理サーバまたは WebLogic 管理対象サーバとして起動できます。管理対象サーバとして起動されていない WebLogic Server が管理サーバになります。

プロダクション システムの典型的なコンフィグレーションでは、ビジネス ロジックを備えるアプリケーションとコンポーネントが複数の管理対象サーバにデプロイされ、管理サーバは管理対象サーバをコンフィグレーションおよびモニタする役割を担います。管理サーバがダウンした場合でも、管理対象サーバにデプロイされたアプリケーションは影響を受けず、クライアントの要求は継続して処理されます。そのような場合、管理サーバはそれが再起動されたときにアクティブなドメインの管理を回復することができます。その仕組みについては、「管理対象サーバの動作中における管理サーバの再起動」を参照してください。

複数の管理対象サーバにわたってアプリケーションとそのコンポーネントを分散させることには、多くの利点があります。処理を実行する EJB などのコンポーネントを分散させることで、メイン アプリケーションのエントリ ポイントの可用性を確保できます。データベース アクセスやアカウント トランザクションといった異なる機能を実行するコンポーネントが別々の管理対象サーバに分散していると、パフォーマンスが向上する可能性があります。さまざまな機能またはアプリケーションのリソースである EJB などのコンポーネントを隔離できるので、その可用性が他のコンポーネントの状態とは無関係になります。アプリケーションは、1 つのドメインで複数をデプロイできます。

ドメインは、管理サーバがそのドメインのコンフィグレーションを使用して起動されている場合にアクティブになります。ドメインがアクティブである間は、管理サーバだけでコンフィグレーション ファイルを変更できます。Administration Console とコマンドライン管理ユーティリティで、ドメインのコンフィグレーションを変更することができます。ドメインがアクティブになった後は、Administration Console を使用してドメイン全体のリソースをモニタおよびコンフィグレーションできます。

ドメイン コンフィグレーションはコンフィグレーション リポジトリに配置でき、Administration Console を使用して編集できます。コンフィグレーション リポジトリは、`\config` ディレクトリ内の少なくとも 1 つのサブディレクトリで構成されます。各ドメインは、ドメインと同じ名前を持つサブディレクトリに配置された別々の `config.xml` ファイルで定義されます。アクティブではないドメイン コンフィグレーションにアクセスするには、Console を起動したときに表示される [BEA WebLogic Server へようこそ] ページの [ドメイン コンフィグレーション] リンクをたどります。

Administration Console の起動

Administration Console は、JavaServer Pages （JSP）を使用して管理サーバの管理リソースにアクセスする Web アプリケーションです。

管理サーバを起動した後（「WebLogic Server の起動と停止」を参照）、ブラウザで次の URL を指定して Administration Console を起動できます。

`http://hostname:port/console`

hostname では管理サーバの DNS 名または IP アドレスを指定し、*port* では管理サーバで要求がリスンされるポートのアドレス（デフォルトでは 7001）を指定します。セキュア ソケット レイヤ（SSL）を使用して管理サーバが起動されている場合は、次のように `http` の後に `s` を付ける必要があります。

`https://hostname:port/console`

ブラウザが HTTP リクエストをプロキシ サーバに送信するようコンフィグレーションしてある場合、管理サーバの HTTP リクエストをプロキシに送信しないようコンフィグレーションする必要があります。管理サーバがブラウザと同じサーバ上にある場合、`localhost` または `127.0.0.1` に送信されるリクエストがプロキシに送信されないようにする必要があります。

Administration Console の左ペインには、データ テーブル、コンフィグレーション ページ、およびモニタ ページに移動したり、ログにアクセスしたりするための階層ツリー（ドメイン ツリー）があります。ドメイン ツリーの項目を選択する（マウスの左ボタンでクリックする）ことで、特定の種類のリソース（WebLogic Server など）のデータ テーブルや、選択したリソースのコンフィグレーション ページおよびモニタ ページを表示できます。ドメイン ツリーの最上位ノードはコンテナです。コンテナに子ノードがある場合は、コンテナ左側の正符号をクリックしてツリーを展開し、子ノードにアクセスできます。

エンティティ テーブル（特定の種類のリソースのデータ テーブル）は、属性値を表示するカラムを追加または削除してカスタマイズできます。テーブルをカスタマイズするには、テーブルの上にある [このビューをカスタマイズ] リンクをクリックします。テーブルの各カラムは、テーブルに追加するように選択されている属性に対応しています。

Administration Console を起動するときには、パスワードの入力が要求されます。Administration Console の初めての起動では、管理サーバを起動したときのユーザ名とパスワードを使用できます。Administration Console を使用すると、管理

者グループにユーザを追加できます。ユーザ（またはユーザのグループ）が管理者グループに追加されると、それらのユーザも Administration Console を使用して管理作業を実行できます。管理者グループのデフォルト メンバーは `system` です。

管理サーバで管理できるのは 1 つのアクティブ ドメインだけなので、Administration Console を使用してアクセスできるのは一度に 1 つのアクティブ ドメインだけです。複数の管理サーバがそれぞれ独自のアクティブ ドメインで動作している場合は、アクセスする必要のある管理サーバ上の Administration Console を起動するだけで管理対象ドメインを切り替えることができます。

実行時オブジェクトとコンフィグレーション オブジェクト

管理サーバでは、Management Bean （MBean）と呼ばれる JavaBean に似たオブジェクトが使用されます。MBean は、Sun の Java Management Extension（JMX）規格に基づいています。このオブジェクトを使用することで、ドメインのリソースに管理を目的としてアクセスできます。

管理サーバには、コンフィグレーション MBean と実行時 MBean があります。コンフィグレーション MBean では、コンフィグレーション属性への SET （書き込み）アクセスと GET （読み込み）アクセスができます。

実行時 MBean では、現在の HTTP セッションや JDBC 接続プールの負荷などのドメイン リソースに関する特定の時点での情報が提供されます。ドメインの特定のリソース（Web アプリケーションなど）がインスタンス化されると、そのリソースについての情報を収集する MBean のインスタンスが作成されます。

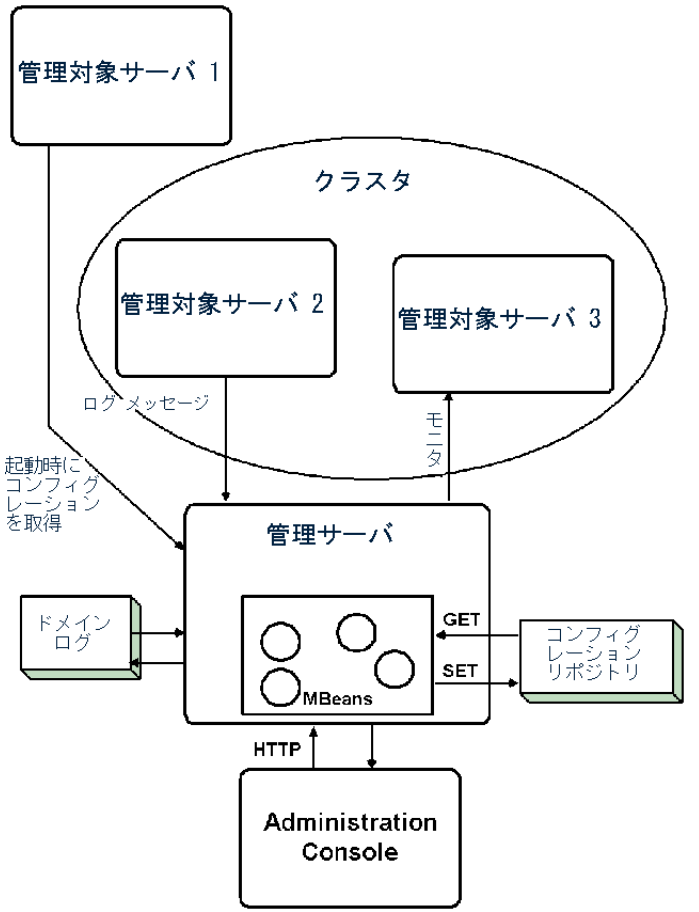
Administration Console で特定のリソースのモニタ ページにアクセスすると、管理サーバでは現在の属性値を取り出すための GET 処理が実行されます。

Administration Service を使用すると、ドメイン リソースのコンフィグレーション属性を動的に（WebLogic Server の動作中に）変更できます。多くの属性では、変更を有効にするためにサーバを再起動する必要がありません。その場合、コンフィグレーションの変更は、属性の現在の実行時値とコンフィグレーション

ファイルに永続的に格納されている値の両方で反映されます。WebLogic Server のコンフィグレーションの詳細については、「WebLogic Server とクラスタのコンフィグレーション」を参照してください。

Web ベースの Administration Console だけでなく、コマンドライン ユーティリティを使用しても、ドメイン リソースのコンフィグレーションにアクセスしたり、属性をモニタしたりできます。このツールは、システム管理を自動化するスクリプトを作成する場合に使用します。詳細については、この章の「WebLogic Server コマンドライン インタフェース リファレンス」を参照してください。

図 1-1　WebLogic Server の管理サブシステム



アクセス ログ メッセージの一元管理

管理サーバでは、ドメイン ログを通じて、すべてのサーバからの重要なシステム メッセージに一元的にアクセスできます。JMX には、特定のメッセージをサブスクライブしているエンティティにメッセージを転送する機能があります。サブスクライバ エンティティでは、目的のメッセージを選択するフィルタを用意

することで、どのメッセージを転送するのかを指定します。ローカル WebLogic Server の判断に基づいて他のネットワーク エンティティに転送されるメッセージは、通知と呼ばれます。JMX 通知は、ドメイン内のすべての WebLogic Server から選択したログ メッセージを管理サーバに転送するために使用します。

WebLogic 管理対象サーバが起動すると、管理サーバでは重要なログ メッセージを受信するための登録が行われます。これらのメッセージは、ドメイン ログに格納されます。WebLogic Server には、転送するメッセージを選択するために、管理サーバによって単一のドメイン ログ フィルタが登録されます。ドメイン ログ フィルタの変更、ドメイン ログの表示、およびローカル サーバ ログの表示は、Administration Console で行います。詳細については、「ログ メッセージを使用した WebLogic Server の管理」を参照してください。

新しいドメインの作成

この節では、新しいドメインの作成方法について説明します。WebLogic 管理ドメインのすべてのコンフィグレーション情報は、`\config` ディレクトリにあるコンフィグレーション リポジトリに格納されています。`\config` ディレクトリの下には各ドメインの個別のサブディレクトリがあります。ドメインのサブディレクトリの名前は、ドメインの名前と同じにしなければなりません。

WebLogic Server ソフトウェアを初めてインストールする場合、デフォルト *`install_dir`*`\config\`*`mydomain`* コンフィグレーション ディレクトリのコピーを収めた zip ファイルを作成することをお勧めします（*`install_dir`* は WebLogic Server ソフトウェアをインストールしたルート ディレクトリを表し、*`mydomain`* はインストール時に指定したデフォルト ドメインを表します）。この zip ファイルのコピーを、新しいドメインの作成に使用できるバックアップとして保存しておきます。このサブディレクトリには、`fileRealm.properties` ファイルやコンフィグレーション ファイルなど、使用中のコンフィグレーションに必要なコンポーネントが含まれます。

以下に説明する手順では、`mydomain` がインストール時にデフォルト コンフィグレーション ディレクトリ名として選択されたものと仮定しています。デフォルト コンフィグレーション ディレクトリに `mydomain` 以外の名前を指定した場合、`mydomain` をすべてその名前に置き換えます。

新しいドメインを作成するには、次の手順に従います。

1. mydomain などの既存のドメインで、管理サーバを起動します。
2. ブラウザで次のように指定して Administration Console を起動します。

   http://*hostname*:*port*/console

   hostname は管理サーバを起動したマシンの名前、*port* は管理サーバのリスン ポートです（デフォルトは 7001）。
3. [mydomain ｜他のドメインの作成または編集] を選択します。

   ドメイン テーブルが表示されます。
4. [デフォルト｜新しいドメインを作成] を選択します。

   新しいドメインの名前を入力して、[作成] クリックします。
5. 左側のドメインのリストから新しいドメインを選択して、現在のドメインにします。
6. 新しいドメインの管理サーバ エントリを作成する必要があります。

   a. [サーバ｜新しい Server のコンフィグレーション] を選択します。

   b. 新しい管理サーバの名前を入力して、[作成] をクリックします。
7. Administration Console では、ドメインの名前を持つ新しいサブディレクトリと、その下に config.xml というコンフィグレーション ファイルを作成します。そのドメイン ディレクトリの中に \applications サブディレクトリを作成する必要があります。\applications サブディレクトリは、コマンド シェルまたはエクスプローラ（Windows の場合）で作成できます。
8. デフォルトの mydomain ディレクトリには、WebLogic Server を起動するための起動スクリプトが含まれています。Windows 上のインストールの場合は、startWebLogic.cmd と startManagedWebLogic.cmd です。UNIX 上のインストールの場合は、startWebLogic.sh と startManagedWebLogic.sh です。これらの起動スクリプトを新しいドメイン ディレクトリにコピーします。
9. テキスト エディタで起動スクリプトを編集する必要があります。デフォルトでは、ドメインの名前は次のように設定されています。

   -Dweblogic.Domain=mydomain

   mydomain を新しいドメインの名前と置き換えます。

   デフォルトでは、管理サーバの名前は次のように設定されています。

```
-Dweblogic.Name=MyServer
```

MyServer を新しい管理サーバの名前と置き換えます。

10. 起動スクリプトの最後に、次のような cd コマンドがあります。

```
cd config\mydomain
```

mydomain を新しいドメインのサブディレクトリ名と置き換えます。起動スクリプトには次のような行もあります。

```
echo startWebLogic.cmd must be run from the config\mydomain
directory.
```

mydomain を新しいドメインの名前と置き換えます。

11. SerializedSystemIni.dat ファイルと fileRealm.properties ファイルを、デフォルトの mydomain ディレクトリから新しいドメイン ディレクトリにコピーします。これらのファイルをコピーするまでは新しい管理サーバを起動しないようにしてください。

この手順を完了したら、新しいドメインの管理サーバを起動できます。

2 WebLogic Server の起動と停止

以下の節では、管理サーバと管理対象サーバを起動および停止する手順について説明します。

- WebLogic 管理サーバと WebLogic 管理対象サーバ
- WebLogic 管理サーバの起動
- WebLogic 管理対象サーバのドメインへの追加
- WebLogic 管理対象サーバの起動
- Administration Console からの WebLogic Server の停止
- 管理対象サーバのサスペンドと再開
- WebLogic Server の Windows サービスとしての設定
- スタートアップ クラスとシャットダウン クラスの登録

WebLogic 管理サーバと WebLogic 管理対象サーバ

WebLogic Server **ドメイン**は、1 つまたは複数の WebLogic Server で構成されます。WebLogic Server は、管理サーバまたは管理対象サーバのいずれかとして起動できます。ドメイン内の WebLogic Server の中で 1 つだけが、そのドメインの管理サーバになります。他の WebLogic Server は、**管理対象**サーバです。WebLogic Server が管理サーバなのか、それとも管理対象サーバなのかは、サーバを起動するときに使用するコマンドライン オプションによって決まります。

WebLogic Server のデフォルトのロールは管理サーバです。したがって、ドメインに 1 つの WebLogic Server しかない場合は、そのサーバが管理サーバになります。マルチサーバ ドメインの WebLogic Server は、起動時に動作中の管理サーバからコンフィグレーションを取得するように指定されている場合だけ管理対象サーバになります。

管理サーバでは、WebLogic Server ドメインのコンフィグレーションへのアクセスが管理されるほか、モニタやログ メッセージの参照といった他の管理サービスも提供されます。管理サーバには、管理サービスにユーザがアクセスできるようにする Administration Console があります。

WebLogic 管理対象サーバは、起動時に管理サーバからコンフィグレーションを取得します。このため、マルチサーバの WebLogic Server ドメインを起動する手順は 2 段階になります。まず、管理サーバを起動し、その後で管理対象サーバを起動します。

注意： 管理対象サーバの WebLogic Server バージョンは、管理サーバと同じでなければなりません。

起動メッセージ

WebLogic Server の起動時には、通常のロギング サブシステムはまだ利用できません。したがって、起動時に発生したエラーは `stdout` に記録されます。ノード マネージャ を使用して、Administration Console からリモートの管理対象サーバを起動する場合、それらのメッセージは Administration Console の右ペインにも表示されます。

WebLogic 管理サーバの起動

WebLogic 管理サーバを起動するには、以下の複数の方法があります。

- コマンド ラインの使用

  WebLogic Server を起動するコマンドは、コマンド シェルに手動で入力するか、またはサーバを起動するたびにコマンドを入力しなくて済むようにスク

リプトに配置できます。サンプル スクリプトについては、「スクリプトを使用した WebLogic 管理対象サーバの起動」を参照してください。

- [スタート] メニューの使用（Windows の場合のみ）
- Windows サービスとしてインストールされた WebLogic Server は、コンピュータが再起動すると自動的に起動する。

WebLogic Server 起動時のパスワードの使用

インストール時には、サーバの起動時に必要になるパスワードを指定する必要があります。起動スクリプトを使用して管理サーバまたは管理対象サーバを起動する場合は、パスワードをコマンドライン引数として指定できます（「コマンドラインからの WebLogic 管理サーバの起動」を参照)。パスワードがコマンドライン引数として指定されていないスクリプトを使用してサーバを起動すると、パスワードを入力するように要求されます。パスワードがコマンドライン引数として指定されている場合は入力の要求はされませんが、そのパスワードはスクリプトファイルにクリアテキストで格納されます。

[スタート] メニューを使用した WebLogic 管理サーバの起動

BEA インストール プログラムを使用して Windows に WebLogic Server がインストールされている場合は、Windows の [スタート] メニューにある WebLogic Server のショートカットを使用して WebLogic 管理サーバを起動できます。次のように選択してください。

[スタート｜プログラム｜ BEA WebLogic E-Business Platform ｜ WebLogic Server バージョン｜ Start Default Server]

バージョンは、WebLogic Server ソフトウェアのバージョン番号です。

[スタート] メニューから WebLogic Server を起動すると、起動スクリプトの `startWeblogic.cmd` が実行されます。このスクリプトは、*`install_dir`*`\config\`*`domain_name`* に配置されています（*`domain_name`* はドメイン名。*`install_dir`* は WebLogic Server ソフトウェアをインストールしたディレクトリ名）。パスワードを入力することが要求されます。

Windows サービスとしての WebLogic Server の起動と停止

Windows サービスとしてインストールされている WebLogic Server は、Windows コンピュータが起動すると自動的に起動します。WebLogic Server は、`startWeblogic.cmd` などの起動スクリプトを実行することによって起動します。`startWebLogic.cmd` を使用して起動した WebLogic Server は、管理サーバとして起動します。「コマンドラインからの WebLogic 管理サーバの起動」を参照してください。

WebLogic Server を Windows サービスとして実行するには、そのことを前提として WebLogic Server をインストールする必要があります。Windows サービスとしての WebLogic Server をインストールおよび削除する方法については、「WebLogic Server の Windows サービスとしての設定」を参照してください。

WebLogic Server は、コントロール パネルの [サービス] を使用しても簡単に停止および起動できます。

1. [スタート | 設定 | コントロール パネル] を選択します。
2. コントロール パネルで [サービス] をダブルクリックします。
3. [サービス] ダイアログ ボックスで、スクロールしながら `WebLogic Server` を見つけます。WebLogic Server が「開始」状態の場合は、WebLogic Server を選択することで [停止] ボタンを使用できるようになります。[停止] ボタンをクリックすると、WebLogic Server が停止します。WebLogic Server が停止状態の場合は、[開始] ボタンを使用できます。

Windows サービスは、[スタートアップ] ボタンをクリックしてモードを選択することで、自動、手動、または無効として設定できます。

コマンドラインからの WebLogic 管理サーバの起動

WebLogic Server は Java クラス ファイルであり、他のどの Java アプリケーションとも同じように、`java` コマンドを使用して起動できます。コマンドラインから WebLogic Server を起動するために必要な引数は、かなり長くなる場合があ

り、起動のたびに入力するのは面倒です。起動コマンドの入力ミスを防ぐために、WebLogic Server を起動するときに使用できるスクリプトにコマンドを組み込むことをお勧めします。

java コマンドラインから WebLogic 管理サーバを起動するときには、以下の引数が必要です。

- Java ヒープ メモリの最小値と最大値を指定します。

  たとえば、デフォルトの 64 MB の Java ヒープ メモリを WebLogic Server に割り当ててサーバを起動するとします。そのためには、`java -ms64m` および `-mx64m` オプションを使用してサーバを起動できます。

  最高のパフォーマンスを得るには、JVM がヒープのサイズを変更しないように最小値と最大値を同じにしてください。

  パラメータに割り当てられたこれらの値は、WebLogic Server のパフォーマンスに大きく影響する可能性があり、ここでは一般的なデフォルト値としてのみ紹介しています。プロダクション環境では、実際のアプリケーションや環境に合った適切なメモリ ヒープ サイズを慎重に判断する必要があります。

- `java -classpath` オプションを設定します。

  このオプションで指定する最低限の内容は、「クラスパス オプションの設定」で説明されています。

- サーバの名前を指定します。

  ドメインのコンフィグレーションでは、サーバ名に基づいてコンフィグレーションが指定されます。コマンドラインでサーバの名前を指定するには、次の引数を使用します。

  `-Dweblogic.Name=`*`servername`*

  デフォルト値は *`myserver`* です。

- サーバのリスン アドレスを指定します。

  管理対象サーバを同じドメイン内の別のマシン上で実行する（または管理サーバと管理対象サーバをマルチホーム マシン上で実行する）場合、またはノード マネージャを使用してリモートで管理対象サーバを起動する場合、管理サーバのリスン アドレスを設定する必要があります。リスン アドレスを設定するには、次の引数を使用します。

  `-Dweblogic.ListenAddress=`*`host`*

  `host` は管理サーバの DNS 名または IP アドレスです。

- ユーザ パスワードを提供します。

  デフォルト ユーザは system であり、必要なパスワードはインストール時に指定されたパスワードです。パスワードを入力するには、次の引数を使用します。

  ```
  -Dweblogic.management.password=password
  ```

- WebLogic Server ルート ディレクトリから WebLogic Server を起動しない場合はルート ディレクトリを指定します。

  WebLogic Server ルート ディレクトリには、ドメインのセキュリティ リソースとコンフィグレーション リポジトリ（\config という名前のディレクトリ）が格納されます。ルート ディレクトリの位置は、コマンドラインで次の引数を使用して指定できます。

  ```
  -Dweblogic.RootDirectory=path
  ```

  path は、ルート ディレクトリのパスです。コマンドラインでこの属性を指定しない場合は、カレント ディレクトリがシステム ホーム ディレクトリに設定されます。

  WebLogic Server ソフトウェアがアップグレードされてもドメイン コンフィグレーションとアプリケーションを簡単に維持できるようにするには、ルート ディレクトリを WebLogic Server ソフトウェアのインストール ディレクトリとは別のディレクトリにします。RootDirectory 属性を使用すると、インストール ディレクトリ内にはない場合にドメイン コンフィグレーションを見つけることができます。

- bea.home ディレクトリを次のように指定します。

  ```
  -Dbea.home=root_install_dir
  ```

  root_install_dir は BEA WebLogic Server ソフトウェアをインストールしたディレクトリです。

- セキュア ソケット レイヤ（SSL）プロコトルを使用してサーバを起動するには、サーバで SSL プライベート キー ファイルを解読できるように、起動時にプライベート キーのパスワードを渡す必要があります。SSL プライベート キーのパスワードを起動時にサーバに渡すには、コマンドラインで次の引数を使用します。

  ```
  -Dweblogic.pkpassword=pkpassword
  ```

  pkpassword は、SLL プライベート キーのパスワードです。

- SSL を使用する場合、ホスト名検証をオフにしてもかまいません。WebLogic Server のホスト名検証では、デフォルトでデジタル証明書の SubjectDN と SSL 接続を開始したサーバのホスト名を比較します。SubjectDN とホスト名が一致しない場合、SSL 接続は中断されます。WebLogic Server に付属のデモ用デジタル証明書を使用する場合など、ホスト名検証をオフにする場合、コマンド ラインで次の引数を使用します。

  ```
  -Dweblogic.security.SSL.ignoreHostnameVerification=true
  ```

  ただし、プロダクション デプロイメントでデモ用デジタル証明書を使用したり、ホスト名検証をオフにしたりすることはお勧めしません。

- WebLogic Server でカスタム ホスト名検証を使用するには、コマンドラインで次の引数を使用します。

  ```
  -Dweblogic.security.SSL.HostnameVerifier=hostnameverifierimplmentation
  ```

  hostnameverifierimplmentation は、`weblogic.security.SSL.HostnameVerifier` インタフェースを実装するクラスの名前です。

- SSL セッション キャッシングはデフォルトで有効です。サーバセッション キャッシュのデフォルト サイズおよび存続期間を変更するには、コマンドラインで次の引数を使用します。

  ```
  -Dweblogic.security.SSL.sessionCache.size=sessionCacheSize
  -Dweblogic.security.SSL.sessionCache.ttl=sessionCacheTimeToLive
  ```

  sessionCacheSize はセッション キャッシュのサイズを表し、*sessionCacheTimeToLive* はセッション キャッシュの存続期間を表します。2 つのパラメータの最小値、最大値、およびデフォルト値は次のとおりです。

  `sessionCache.size`: 最小値 1、最大値 65537、デフォルト値 211
  `sessionCache.ttl`: 最小値 1、最大値 `Integer.MAX_VALUE`、デフォルト値 600

- コマンドラインで次の引数を使用すれば、管理サーバの起動時にドメイン コンフィグレーションの名前を指定できます。

  ```
  -Dweblogic.Domain=domain_name
  ```

  domain_name は、ドメインの名前です。この名前は、ドメインの起動に使用されるコンフィグレーション ファイルを格納するサブディレクトリの名前にもなります。

コンフィグレーション リポジトリは、\config ディレクトリ内のドメインで構成されます。コンフィグレーション リポジトリには、使用する可能性のある多様なドメイン コンフィグレーションを格納できます。それらの各ドメインは、ドメインと同じ名前を持つ個別のサブディレクトリに配置されます。したがって、*domain_name* を指定するときには、このサブディレクトリ名を指定することになります。指定されたサブディレクトリには、XML コンフィグレーション ファイル（config.xml）とドメインのセキュリティ リソースが格納されます。ファイル config.xml では、ドメインのコンフィグレーションが指定されます。

管理サーバの起動に使用されたドメイン コンフィグレーションが、アクティブなドメインになります。アクティブなドメインは 1 つだけです。

- WebLogic Server コンフィグレーション属性値もコマンドラインで指定できます。それらの値は属性の実行時値になり、永続的なコンフィグレーションに格納されている値は無視されます。コマンドラインで WebLogic Server 属性の実行時値を設定する書式は次のとおりです。

  -Dweblogic.*attribute*=*value*

- デフォルトで有効になる自動デプロイメント機能は、アクティブ ドメインの \applications ディレクトリを調査して、デプロイされているアプリケーションの変更を検出します。変更がないかアプリケーション ディレクトリをポーリングする AppManager スレッドは管理サーバ上でのみ作成されるので、この機能は管理サーバ上でのみ機能します。この機能は、プロダクション環境での使用はお勧めしません。自動デプロイメント機能が無効な状態で管理サーバが起動するようにするには、コマンドラインで次の引数を使用します。

  -Dweblogic.ProductionModeEnabled=true

クラスパス オプションの設定

java コマンドラインでは、-classpath オプションの値として以下のように指定する必要があります。

- /*weblogic*/lib/weblogic_sp.jar
- /*weblogic*/lib/weblogic.jar
- WebLogic Server には、Java だけで作られている Cloudscape というデータベース管理システム（DBMS）の試用版が付属しています。この DBMS を

使用する場合は、クラスパスで次のように指定する必要があります。
/weblogic/samples/eval/cloudscape/lib/cloudscape.jar

- WebLogic Enterprise Connectivity を使用する場合は、次のように指定する必要があります。
  /weblogic/lib/poolorb.jar

weblogic は、WebLogic Server がインストールされているディレクトリです。

スクリプトを使用した管理サーバの起動

WebLogic Server 配布キットには、WebLogic Server の起動に使用できるサンプル スクリプトが付属しています。それらのスクリプトは、実際の環境やアプリケーションに合わせて修正する必要があります。管理サーバの起動用と管理対象サーバの起動用に、別々のサンプル スクリプトが用意されています。管理サーバを起動するためのスクリプトは、`startWebLogic.sh`（UNIX）と `startWeblogic.cmd`（Windows）です。これらのスクリプトは、ドメインのコンフィグレーション サブディレクトリに配置されています。

サンプル スクリプトを使用するには、次の作業を行います。

- クラスパスの設定とディレクトリ名に注意します。
- 変数 `JAVA_HOME` の値を JDK の位置に変更します。
- UNIX ユーザは、ファイルを実行可能にするためにサンプル UNIX スクリプトのパーミッションを変更する必要があります。次に例を示します。

  ```
  chmod +x startWebLogic.sh
  ```

- 管理対象サーバを同じドメイン内の別のマシン（または管理サーバと同じマルチホーム マシン）上で実行する場合、ノード マネージャを使用して管理対象サーバを起動および強制停止したい場合、WebLogic Server 起動コマンドを編集して、管理サーバのリスン アドレスを設定する引数を追加する必要があります。

  ```
  -Dweblogic.ListenAddress=host
  ```

  host は管理サーバの DNS 名または IP アドレスです。

管理対象サーバの動作中における管理サーバの再起動

典型的なプロダクション システムの場合は、重要なビジネス ロジックの格納されているアプリケーションを管理対象サーバにデプロイすることをお勧めします。その場合の管理サーバの役割は、管理対象サーバをコンフィグレーションおよびモニタすることです。このようなコンフィグレーションの場合は、管理サーバが利用できなくなっても、管理対象サーバで動作しているアプリケーションではクライアントの要求を処理し続けることができます。

管理サーバが起動すると、アクティブなドメインの起動に使用されたコンフィグレーション ファイルのコピーが作成されます。これは、次のファイルに保存されます。

install_dir\config*domain_name*\config.xml.booted

install_dir は WebLogic Server ソフトウェアをインストールしたディレクトリで、*domain_name* はドメインの名前です。管理サーバは、起動シーケンスを正常に完了し、要求を処理できる準備ができてから、config.xml.booted ファイルを作成します。

Administration Console から行ったアクティブ コンフィグレーションに対する変更を元に戻す必要がある場合、使用中のコンフィグレーション ファイルに復帰できるよう、このファイルをコピーしておく必要があります。

管理対象サーバが動作を続けている状況で管理サーバがダウンした場合、ドメインの管理を回復するために、すでに動作している管理対象サーバを再起動する必要はありません。アクティブなドメインの管理を回復する手順は、管理サーバが起動したときと同じマシンで管理サーバを再起動できるかどうかによって異なります。

同じマシンでの管理サーバの再起動

管理対象サーバが動作を続けている状況で WebLogic 管理サーバを再起動する場合、管理サーバでは動作している管理対象サーバの存在を検出できます。ただし、検出を行うように指定することが必要です。管理サーバで管理対象サーバを検出するように指定するには、管理サーバを起動するときにコマンドラインで次の引数を入力します。

```
-Dweblogic.management.discover=true
```

この属性のデフォルト値は true です。ドメインのコンフィグレーション ディレクトリには、running-managed-servers.xml というファイルが含まれています。このファイルは、管理サーバが認識している管理対象サーバのリストです。管理サーバが起動時に検出を行うよう設定されている場合、管理サーバはこのリストを使用して動作している管理対象サーバの存在をチェックできます。

管理サーバの再起動では、静的にのみコンフィグレーションできる属性の変更を反映して管理対象サーバの実行時コンフィグレーションが更新されることはありません。静的なコンフィグレーション属性の変更を反映するためには、WebLogic Server を再起動する必要があります。管理対象サーバを検出すると、管理サーバでは管理対象サーバをモニタしたり、動的にコンフィグレーションできる属性の値を実行時に変更したりできます。

別のマシンでの管理サーバの再起動

マシンのクラッシュにより、同じマシンで管理サーバを再起動できない場合は、次のようにして動作している管理対象サーバの管理を回復できます。

1. 前の管理サーバ マシンと同じホスト名を別のコンピュータに割り当てます。
2. 新しい管理マシンで WebLogic Server ソフトウェアをインストールします（インストールされていない場合）。
3. 前のマシンで管理サーバを起動するために使用されていた \config ディレクトリ（コンフィグレーション リポジトリ）を、新しいマシンで利用できるようにします。\config ディレクトリは、たとえばバックアップ メディアからコピーするか、NFX マウントを通じて利用可能にできます。このディレクトリには、アクティブなドメイン、およびそのドメインの \applications ディレクトリにインストールされているアプリケーションやコンポーネントを起動するために使用するコンフィグレーション ファイル（config.xml）が格納されています。
4. コマンドラインで次の引数を指定して、新しいマシンで管理サーバを再起動します。

   ```
   -Dweblogic.management.discover=true
   ```

   この引数を指定すると、管理サーバでは動作している管理対象サーバの存在が検出されます。

WebLogic 管理対象サーバのドメインへの追加

WebLogic Server を管理対象サーバとして実行するためには、まず、ドメインのコンフィグレーションでそのサーバのエントリを作成する必要があります。そのためには、次の操作を行います。

1. ドメインの管理サーバを起動します。
2. ブラウザで `http://hostname:port/console` を指定して Administration Console を起動します。*hostname* は管理サーバが動作しているマシンの名前、*port* は管理サーバでコンフィグレーションされているリスン ポート番号（デフォルトは 7001）です。
3. 管理サーバ マシンと異なる場合は、サーバ マシンのエントリを作成します（[マシン | 新しい Machine のコンフィグレーション]）。
4. 新しいサーバのエントリを作成します（[サーバ | 新しい Server のコンフィグレーション]）。この管理対象サーバのマシンを、たった今エントリを作成したばかりのマシンに設定します。

サーバのコンフィグレーションの詳細については、「WebLogic Server とクラスタのコンフィグレーション」を参照してください。

WebLogic 管理対象サーバの起動

WebLogic 管理対象サーバは、以下のいずれかの方法で起動できます。

- 管理対象サーバを起動する必要のある対象マシン上のノード マネージャを使用し、Administration Console からリモートで起動できます。
- コマンド シェルの java コマンドラインでサーバを起動することで、ローカルで起動できます。

この節では、WebLogic 管理対象サーバをローカルで起動する方法について説明します。ノード マネージャを設定および使用して管理対象サーバをリモートで起動する方法については、「ノード マネージャ」を参照してください。

注意： Administration Console の左ペインでサーバの名前を右クリックした場合、表示されるオプションの 1 つは **[このサーバを開始 ...]** です。このオプションは、管理対象サーバのあるマシン上でノード マネージャが動作している状態で管理対象サーバを起動する場合にのみ使用できます。詳細については、「ノード マネージャ」を参照してください。

コンフィグレーションに WebLogic 管理対象サーバを追加したら（「WebLogic 管理対象サーバのドメインへの追加」を参照）、`java` コマンドラインから管理対象サーバを起動できます。WebLogic Server を起動するコマンドは、コマンド シェルに手動で入力するか、またはサーバを起動するたびにコマンドを入力しなくて済むようにスクリプトに配置できます。サンプル スクリプトについては、「スクリプトを使用した WebLogic 管理対象サーバの起動」を参照してください。

管理対象サーバの起動パラメータが管理サーバの場合とおもに違う点は、管理対象サーバがコンフィグレーションを要求する管理サーバの位置を引数として指定しなければならないことです。このパラメータなしで起動した WebLogic Server は、管理サーバとして実行されます。

WebLogic 管理対象サーバを起動するときには、管理サーバを起動するときに指定するパラメータ（「コマンドラインからの WebLogic 管理サーバの起動」を参照）を指定するとともに、以下の事項を指定する必要があります。

- サーバの名前を指定します。

  管理対象サーバが管理サーバにコンフィグレーション情報を要求するとき、管理サーバでは管理対象サーバがサーバ名で識別されます。このサーバ名による識別により、管理サーバでは適切なコンフィグレーションで応答することができます。このため、管理対象サーバを起動するときにはサーバ名も設定する必要があります。サーバ名を指定するには、WebLogic 管理対象サーバを起動するときにコマンドラインに次の引数を追加します。

  ```
  -Dweblogic.Name=servername
  ```

- WebLogic 管理サーバのホスト名とリスン ポートを指定します。

  管理対象サーバを起動するときには、管理対象サーバがコンフィグレーションを要求する管理サーバのホスト名とリスン ポートを指定する必要があります。ホスト名とリスン ポートを指定するには、管理対象サーバを起動するときにコマンドラインに次の引数を追加します。

```
-Dweblogic.management.server=host:port
```

または

```
-Dweblogic.management.server=http://host:port
```

host は管理サーバが動作しているマシンの名前または IP アドレス、*port* は管理サーバのリスン ポートです。管理サーバのデフォルトのリスン ポートは 7001 です。

管理サーバとの通信にセキュア ソケット レイヤ（SSL）を使用する場合は、管理サーバを次のように指定する必要があります。

```
-Dweblogic.management.server=https://host:port
```

管理対象サーバと管理サーバの通信で SSL プロコトルを使用するには、管理サーバで SSL を有効にする必要があります。SSL 設定の詳細については、「セキュリティの管理」を参照してください。

注意： 管理サーバの位置を指定しないで起動した WebLogic Server は、管理サーバとして起動します。

注意： 管理対象サーバは管理サーバからコンフィグレーションを受信するので、指定する管理サーバは管理対象サーバと同じドメインになければなりません。

スクリプトを使用した WebLogic 管理対象サーバの起動

WebLogic Server 配布キットには、WebLogic Server の起動に使用できるサンプル スクリプトが付属しています。それらのスクリプトは、実際の環境やアプリケーションに合わせて修正する必要があります。管理サーバの起動用と管理対象サーバの起動用に、別々のスクリプトが用意されています。管理対象サーバを起動するためのサンプル スクリプトは、`startManagedWebLogic.sh`（UNIX）と `startManagedWebLogic.cmd`（Windows）です。これらのスクリプトは、ドメインのコンフィグレーション サブディレクトリに配置されています。これらのスクリプトは、修正して独自の起動スクリプトを作成するために使用できるテンプレートです。

サンプル スクリプトを使用するには、次の作業を行います。

- クラスパスの設定とディレクトリ名に注意します。

- 変数 JAVA_HOME の値を JDK の位置に変更します。
- UNIX ユーザは、ファイルを実行可能にするためにサンプル UNIX スクリプトのパーミッションを変更する必要があります。次に例を示します。

```
chmod +x startManagedWebLogic.sh
```

スクリプトを使用して管理対象サーバを起動するには、以下の 2 通りの方法があります。

- 環境変数 SERVER_NAME および ADMIN_URL の値を設定する場合は、起動スクリプトを実行するときにそれらを引数として指定する必要はありません。SERVER_NAME には、起動する WebLogic 管理対象サーバの名前を設定します。ADMIN_URL は、管理サーバのホスト（ホスト名または IP アドレス）とリスン ポート番号（デフォルトは 7001）を示すように設定します。次に例を示します。

```
set SERVER_NAME=bigguy
set ADMIN_URL=peach:7001
startManagedWebLogic
```

- 起動スクリプトを実行し、次のようにコマンドラインで管理対象サーバの名前と管理サーバの URL を渡すことができます。

```
startManagedWebLogic server_name admin:url
```

server_name は起動する管理対象サーバの名前、*admin_url* は http://*host*:*port* または https://*host*:*port*（*host* は管理サーバのホスト名または IP アドレス、*port* は管理サーバのポート番号）です。

Administration Console からの WebLogic Server の停止

Administration Console の左ペインでサーバを右クリックすると、**[このサーバを強制停止 ...]** と **[このサーバを停止 ...]** という 2 つのオプションが表示されます。**[このサーバを強制停止 ...]** オプションを選択すると、管理対象サーバが動作しているマシン上のノード マネージャに管理サーバが要求を送信します。ノード マネージャは、対象の WebLogic Server プロセスを強制停止します。**[このサーバを強制停止 ...]** オプションは、管理サーバを停止するためには使用できませ

ん。**[このサーバを強制停止 ...]** オプションを使用するには、目的の管理対象サーバのあるマシンでノード マネージャが動作している必要があります。ノード マネージャの設定と起動については、「ノード マネージャ」を参照してください。

[このサーバを停止 ...] オプションを選択すると、選択されたサーバに管理サーバが停止要求を送信します。ノード マネージャはこの場合には使用されません。**[このサーバを強制停止 ...]** オプションとは違って、**[このサーバを停止 ...]** オプションは管理サーバを停止するために使用できます。

[このサーバを停止 ...] オプションでは管理対象サーバの管理機能を使用して停止を開始するので、サーバがアクティブで、管理要求に応答している場合にのみ使用できます。**[このサーバを強制停止 ...]** オプションは、通常は、目的の管理対象サーバがハングしているか、管理サーバからの管理要求に応答していない場合に使用します。

コマンドラインからのサーバの停止

WebLogic Server は、次のコマンドを使用してコマンドラインからでも停止できます。

```
java weblogic.Admin -url host:port SHUTDOWN -username adminname
-password password
```

各値の説明は次のとおりです。

- *host* は、WebLogic Server が動作しているマシンの名前または IP アドレスです。
- *port* は、WebLogic Server のリスン ポート（デフォルトは 7001）です。
- *adminname* では、対象 WebLogic Server のコンソール アクセス制御リスト（ACL）のメンバー（またはコンソール ACL のメンバーであるグループのメンバー）であるユーザを指定します。コンソール ACL のデフォルト メンバーは system です。
- *password* は、*adminname* のパスワードです。

管理対象サーバのサスペンドと再開

Administration Console では、WebLogic 管理対象サーバをサスペンドできます。サスペンドしている管理対象サーバでは、管理サーバからの要求だけが受け入れられます。この機能は通常、WebLogic Server が別のサーバの「ホット」バックアップとして動作している状況で使用します。バックアップ サーバは、要求処理の開始を指示するまでサスペンド状態にあります。

注意： WebLogic Server のサスペンドでは、HTTP リクエストに対するサーバの応答だけをサスペンドします。Java アプリケーションまたは RMI の呼び出しはサスペンドされません。

WebLogic 管理対象サーバを個別にサスペンドするには、次の操作を行います。

- Administration Console のドメイン ツリー（左ペイン）で、サスペンドするサーバを右クリックします。
- **[このサーバをサスペンド]** オプションを選択します。

クライアント要求の処理を再開するように管理対象サーバに指示するには、次の操作を行います。

- Administration Console のドメイン ツリー（左ペイン）で、クライアント要求の処理を再開するサーバを右クリックします。
- **[このサーバを再開]** オプションを選択します。

WebLogic Server の Windows サービスとしての設定

WebLogic Server は、Windows サービスとして実行できます。Windows サービスとしてインストールされている WebLogic Server は、Windows コンピュータが起動すると自動的に起動します。WebLogic Server は、起動スクリプト `startWeblogic.cmd` を実行することでこのように起動します。WebLogic Server が管理サーバとして起動するのか、それとも管理対象サーバとして起動するのか

は、WebLogic Server を起動する java コマンドのパラメータで決まります。WebLogic 管理対象サーバの起動とコマンドラインからの WebLogic 管理サーバの起動を参照してください。

Windows サービスとして動作するように WebLogic Server を設定したり、Windows サービスとしてではなく動作するようにコンフィグレーションし直したりするには、管理者レベルの特権が必要です。WebLogic Server を Windows サービスとしてインストールするには、次の操作を行います。

1. *weblogic*\config*mydomain* ディレクトリに移動します。*weblogic* は WebLogic Server がインストールされたディレクトリ、*mydomain* はドメインのコンフィグレーションが格納されているサブディレクトリです。
2. スクリプト installNTService.cmd を実行します。

Windows サービスとしての WebLogic Server の削除

Windows サービスとしての WebLogic Server を削除するには、次の操作を行います。

1. *weblogic*\config*mydomain* ディレクトリに移動します。*weblogic* は WebLogic Server がインストールされたディレクトリ、*mydomain* はドメインのコンフィグレーションが格納されているサブディレクトリです。
2. スクリプト uninstallNTService.cmd を実行します。

Windows サービスとしての WebLogic Server は、Windows の [スタート] メニューを使用してもアンインストールできます。

Windows サービスとしてインストールされた WebLogic Server のパスワードの変更

デフォルト サーバを Windows サービスとしてインストールした場合、サービスを作成するときには、WebLogic Server ソフトウェアのインストール時に入力したシステム パスワードが使用されます。このパスワードが後で変更された場合、次の手順を行います。

1. uninstallNTService.cmd スクリプト（*install_dir*\config*domain_name* ディレクトリに格納されている。*install_dir* は WebLogic Server をインストールしたディレクトリ）を使用して、Windows サービスとしての WebLogic Server をアンインストールします。

2. installNTservice.cmd スクリプトには、次のコマンドが記述されています。

   ```
   rem *** Install the service
   "C:\bea\wlserver6.0\bin\beasvc" -install -svcname:myserver
   -javahome:"C:\bea\jdk130" -execdir:"C:\bea\wlserver6.0"
   -extrapath:"C\bea\wlserver6.0\bin" -cmdline:
   %CMDLINE%
   ```

   次の文字列をコマンドに追加します。

   -password:"*your_password*"

   your_password は新しいパスワードです。

3. 修正された installNTservice.cmd スクリプトを実行します。これで、パスワードが更新された新しいサービスが作成されます。

WebLogic Server Windows サービス プログラム（beasvc.exe）

Windows サービスとして WebLogic Server をインストールおよび削除するためのスクリプトは、WebLogic Server Windows サービス プログラム `beasvc.exe` を起動します。`beasvc.exe` では、WebLogic Server の複数のインスタンスを Windows サービスとしてインストールまたは削除できます。beasvc.exe プログラムは、ノード マネージャを Windows サービスとしてインストールまたは削除するためにも使用できます。Windows サービスとしてノード マネージャをインストールおよび削除する方法については、「ノード マネージャ」を参照してください。

複数のサービスのすべてのコンフィグレーションは、別々のサービス名を使用して Windows レジストリに、および次の場所にあるサーバ固有のハイブに格納されます。

```
HKEY_LOCAL_MACHINE\SYSTEM\Current\ControlSet\Services
```

サービスを起動すると、Windows レジストリのエントリが選択され、JVM が初期化および開始されます。インストールされる各サービスは互いに独立しているので、各サービスにユニークな名前が割り当てられていれば、WebLogic Server の複数のインスタンスをインストールして Windows サービスとして実行できます。

`beasvc.exe` では以下のオプションを利用できます。

`-install`
: 指定されたサービスをインストールします。

`-remove`
: 指定されたサービスを削除します。

`-svcname:` *`service_name`*
: インストールまたは削除されるサービスのユーザ指定の名前です。

`-cmdline:` *`java_cmdline_parameters`*
: WebLogic Server を Windows サービスとして起動するときに使用する `java` コマンドライン パラメータです。

`-javahome:` *`java_directory`*
: Java がインストールされるルート ディレクトリです。起動コマンドは、`\bin\java` を *`java_directory`* に追加して作成します。

`-execdir: base_dir`
この起動コマンドが実行されるディレクトリです。

`-extrapath: additional_env_settings`
このコマンドの実行に適用できるパスの先頭に付けられる追加のパス設定です。

`-help`
`beasvc.exe` コマンドの使い方を出力します。

Win32 システムには、コマンドラインの長さについて 2K の制限があります。Windows サービス起動のためのクラスパス設定が非常に長い場合は、2K の制限を超えることがあります。バージョン 1.2 以降の Sun Microsystems JVM を使用する場合は、@ オプションを使用してクラスパスが格納されているファイルを指定できます。このオプションは、beasvc.exe で次のように使用できます。

```
beasvc -install -svcname:myservice -classpath:@C:\temp\myclasspath.txt
```

スタートアップ クラスとシャットダウン クラスの登録

WebLogic Server には、WebLogic Server が起動するときまたは正常に停止するときに処理を実行するメカニズムがあります。スタートアップ クラスは、WebLogic Server が起動または再起動するときに自動的にロードされて実行される Java プログラムです。スタートアップ クラスは、他のすべてのサーバ初期化処理が完了した後にロードされて実行されます。

シャットダウン クラスは、スタートアップ クラスと同じように機能します。シャットダウン クラスは、Administration Console または `weblogic.admin shutdown` コマンドを使用して WebLogic Server が停止されるときに自動的にロードされて実行されます。

WebLogic Server でスタートアップ クラスまたはシャットダウン クラスを使用するには、それらのクラスを登録する必要があります。スタートアップ クラスとシャットダウン クラスは、Administration Console で登録できます。

スタートアップ クラスまたはシャットダウン クラスを登録するには、次の操作を行います。

1. Administration Console のドメイン ツリー（左ペイン）から [起動と停止] テーブルにアクセスします。このテーブルには、ドメイン コンフィグレーションでシャットダウン クラスまたはスタートアップ クラスのエントリを作成するためのオプションがあります。
2. 追加するスタートアップ クラスまたはシャットダウン クラスの [コンフィグレーション] タブ ページでクラス名と必要な引数を指定します。

以下の機能の詳細については、Administration Console のオンライン ヘルプを参照してください。

- スタートアップ クラス
- シャットダウン クラス

3　ノード マネージャ

以下の節では、ノード マネージャの使用方法について説明します。

- ノード マネージャの概要
- ノード マネージャの設定
- ノード マネージャのプラットフォーム サポート
- コマンドラインからのノード マネージャの起動
- 起動スクリプトを使用したノード マネージャの起動
- 管理対象サーバのリモートでの起動と強制停止
- ノード マネージャの Windows サービスとしての設定

ノード マネージャの概要

ノード マネージャは、Administration Console からリモートの WebLogic 管理対象サーバを起動および強制停止できるようにする Java プログラムです。ノード マネージャは、WebLogic Server ソフトウェアに付属する独立した Java プログラムです。

ノード マネージャは、Administration Console で提供される管理対象サーバを停止する機能の代わりとして、リモートの管理対象サーバを強制停止するために使用できます。リモート サーバ プロセスの強制停止は、サーバがハングしているか、応答していない状況で行います。

管理対象サーバのリモートでの起動を可能にするには、管理対象サーバが動作する各マシンで 1 つのノード マネージャをコンフィグレーションおよび実行する必要があります。マシン上の 1 つのノード マネージャ プロセスで、そのマシン上のすべての管理対象サーバのリモートでの起動と強制停止を処理できます。ノード マネージャを利用できるようにするには、ノード マネージャを Unix マシン上ではデーモン、Windows NT マシンでは Windows NT サービスとしてコン

フィグレーションする必要があります。そのようにコンフィグレーションすることで、ノード マネージャはそのマシン上の管理対象サーバを起動するために利用できるようになります。

ノード マネージャが動作している場合、そのノード マネージャは管理サーバの要求に基づいてそのマシンでインストールおよびコンフィグレーションされているすべての管理対象サーバを起動または強制停止することができます。ノード マネージャと管理サーバの間のすべての通信では、セキュア ソケット レイヤ プロトコルが使用されます。

ノード マネージャのログ

WebLogic Server を起動するときには、さまざまな起動またはエラー メッセージが STDOUT または STDERROR に出力される可能性があります。それらのメッセージは、サーバの起動時に Administration Console の右ペインにも表示されます。それらのファイルは、Administration Console の左ペインでサーバを右クリックし、**[このサーバの StdOut を取得]** または **[このサーバの StdErr を取得]** を選択することで取得できます。

ノード マネージャは、それらのメッセージをノード マネージャ ログ ファイル ディレクトリのファイルに保存します。デフォルトでは、このディレクトリは NodeManagerLogs という名前で、ノード マネージャを起動するディレクトリに作成されます。ディレクトリの名前を変更する必要がある場合は、ノード マネージャを起動するときにコマンドラインで行うことができます。詳細については、コマンドライン引数を参照してください。

ノード マネージャによって起動される管理対象サーバごとに別々のログ ファイル サブディレクトリが作成されます。以下のログが、このディレクトリに格納されます。

servername.pid
: *servername* という名前の管理対象サーバのプロセス ID を保存します。管理サーバによって要求されときにサーバ プロセスを強制停止するためにノード マネージャが使用します。

config
: 管理対象サーバを起動するときに管理サーバからノード マネージャに渡される起動コンフィグレーション情報を保存します。

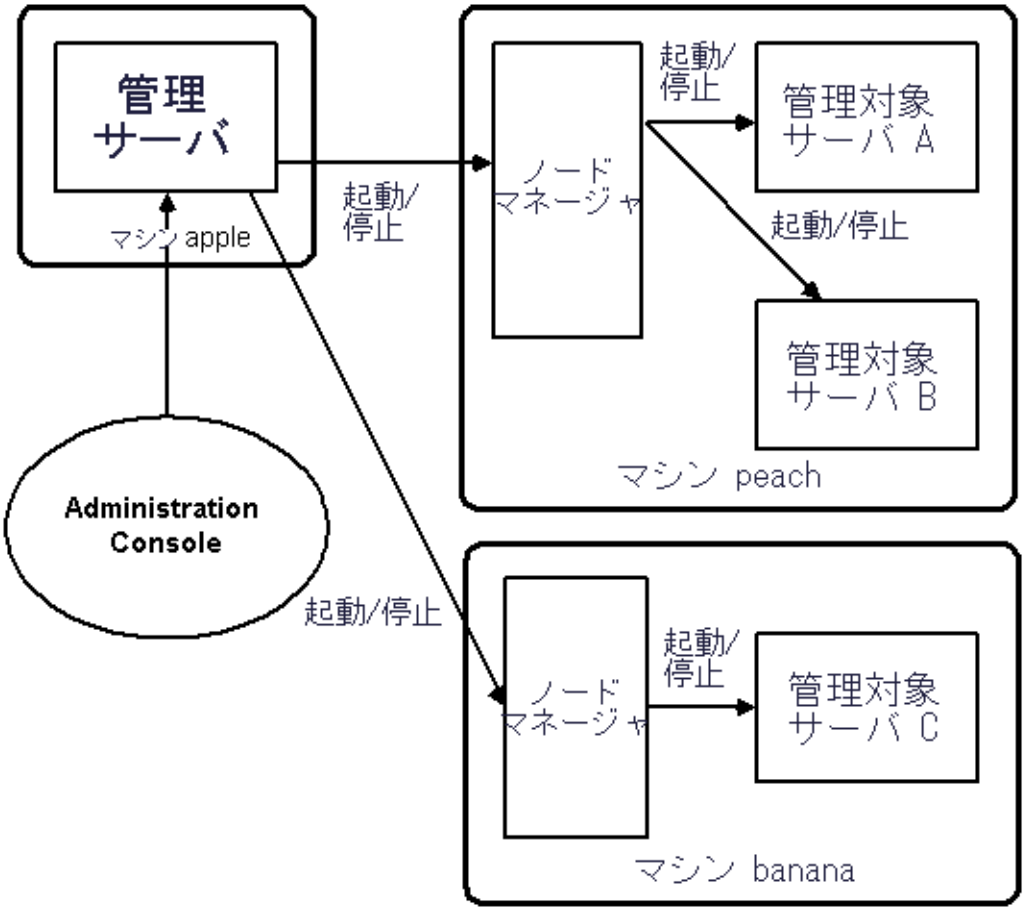
servername-output.log
: *servername* という名前の管理対象サーバの起動がノード マネージャによって試行されたときに StdOut に出力される情報を保存します。サーバの起動が新たに試行されると、このファイルは _PREV を付け加えることによって名前変更されます。

servername-error.log
: *servername* という名前の管理対象サーバの起動がノード マネージャによって試行されたときに StdErr に出力される情報を保存します。サーバの起動が新たに試行されると、このファイルは _PREV を付け加えることによって名前変更されます。

ノード マネージャのログは、管理サーバ マシンの \config\NodeManagerClientLogs というディレクトリ内の一時ファイルにも管理サーバによって格納されます。ノード マネージャを通じて起動が試行された管理対象サーバごとにサブディレクトリが存在します。それらのサブディレクトリの各ログは、サーバの起動や強制停止といったアクションを実行する試行に対応します。ログ ファイルの名前には、アクションが試行された時刻を示すタイムスタンプが含まれます。過去のアクションから蓄積されたクライアント ログは、ノード マネージャを使用して定期的に削除するようにしてください。

ノード マネージャのアーキテクチャは、図 3-1 で示されています。

図 3-1　ノード マネージャのアーキテクチャ



ノード マネージャの設定

ノード マネージャと管理サーバの間のすべての通信では、セキュア ソケット レイヤ プロトコルが使用されます。セキュア ソケット レイヤ プロトコルでは、認証と暗号化を利用できます。管理サーバとノード マネージャの間のすべての通信で相互認証が使用されるように、クライアント認証が実施されます。また、さ

らにセキュリティを高めるために、ノード マネージャでは信頼性のあるホストのリストも使用します。それらのホストのいずれかにある管理サーバからのコマンドだけが受け入れられることになります。ノード マネージャをコンフィグレーションするには、信頼性のあるホスト ファイルを編集して、このノード マネージャにコマンドを送信できる管理サーバがあるマシンごとに 1 行を追加する必要があります。デフォルトでは、信頼性のあるホスト ファイルは nodemanager.hosts という名前で、\config ディレクトリにインストールされます。デフォルトでは、このファイルには以下の 2 つのエントリが格納されます。

```
localhost
127.0.0.1
```

ノード マネージャが信頼性のあるホストのリストを検索するファイルの名前はコマンドラインで変更できます。詳細については、「コマンドライン引数」を参照してください。

各信頼性のあるホストの IP アドレスまたは DNS 名のいずれかを使用できます。ただし、DNS 名を使用する場合は、ノード マネージャを起動するときに DNS の逆引き参照を有効にする必要があります。そのためには、次のコマンドライン引数を使用します。

```
-Dweblogic.nodemanager.reverseDNSenabled=true
```

デフォルトでは、DNS の逆引き参照は無効です。

通常のプロダクション環境では、ノード マネージャは管理サーバと同じマシン上では動作しません。したがって、同じマシン上にある管理対象サーバを起動または強制停止する管理サーバを実行するマシンだけがリストされるように信頼性のあるホスト ファイルを編集する必要があります。信頼性のあるホスト ファイルの各エントリは、管理サーバ マシンの DNS ホスト名または IP アドレスのいずれかである 1 行で構成されます。

注意： ノード マネージャが管理対象サーバを起動するときに管理サーバと通信することを可能にするには、管理サーバの起動時に管理サーバのリスン アドレスに DNS 名または IP アドレスを設定しておく必要があります。

セキュア ソケット レイヤ プロトコル向けのノード マネージャの設定

ノード マネージャでは、管理サーバとの通信でセキュア ソケット レイヤ（SSL）プロトコルを使用します。ノード マネージャと管理サーバの通信でセキュリティを確保するために、2 方向の SSL 認証を使用します。

認証では、公開鍵インフラストラクチャを使用する必要があります。これには、証明書だけでなくプライベート キーが含まれます。証明書は、通常はユーザの公開鍵を格納し、ユーザ名とその公開鍵の結びつきを証明するために証明書の発行者によって署名されています。

ノード マネージャでは、X509 形式の証明書を使用します。ノード マネージャで使用するプライベート キーは、PKCS（Private Key Cryptography Standards）#5 および #8 に準拠しています。PKCS #5 はパスワード ベースの暗号化規格であり、パスワードを使用してプライベート キーを暗号化する方法を規定します。PKCS #8 はプライベート キー構文規格であり、プライベート キーの特性を規定します。

ノード マネージャで使用する公開鍵インフラストラクチャのさまざまな要素は、WebLogic Server デジタル証明書（もっと古い規格に準拠している）で使用される形式とは異なります。主な違いは次のとおりです。

- ノード マネージャでは、ユーザの公開 ID を格納する証明書だけでなくプライベート キーを格納する 1 つの証明書ファイルを使用します。
- ノード マネージャで使用するプライベート キーは、PKCS #5/#8 規格に従ってパスワードで保護されていなければなりません。

WebLogic ソフトウェアでは、ノード マネージャで使用するデモ用の証明書が用意されています。この証明書は、`\config\demo.crt` にあります。プロダクション環境用に新しい証明書を取得することをお勧めします。

ノード マネージャで使用するデジタル証明書を設定する手順は次のとおりです。

手順 1: デジタル証明書とプライベート キーの取得

ノード マネージャで使用するデジタル証明書は、以下の 2 通りの方法で取得できます。

- プライベート キーと X509 形式のデジタル証明書は認証局から取得できます。デジタル証明書を取得する方法については、「セキュリティの管理」を参照してください。プライベート キーが PKCS #5/#8 形式でない場合は、手順 2 で説明されているように WebLogic Server の変換ツールを使用してそれを変換する必要があります。認証局から PKCS #5/#8 形式のプライベート キーを取得した場合は、「手順 3 : 証明書の証明書ファイルへの結合」に進んでください。
- WebLogic Server の証明書ジェネレータから派生した証明書を使用する場合には、ノード マネージャで使用するようにそれらを変換できます。

手順 2 : WebLogic 形式のプライベート キーの変換

ノード マネージャで WebLogic 形式の証明書を使用する場合には、まずそのプライベート キーを新しい PKCS #5/#8 形式に変換する必要があります。WebLogic ソフトウェアでは、そのためのツールが用意されています。

WebLogic 形式の証明書をノード マネージャで使用するように変換するツールは `wlkeytool` という名前で次の場所に配置されています。

- Windows システムの場合は、WebLogic をインストールしたルート ディレクトリの `\bin` ディレクトリにあります。
- Unix システムの場合は、WebLogic をインストールしたルート ディレクトリの `/lib` ディレクトリにあります。

`wlkeytool` を使用するための構文は次のとおりです。

```
wlkeytool old_key new_key
```

古いキーのロックを解除するためにプライベート キーのパスワードを入力することが要求されます。パスワードがない場合は〔Enter〕を押します。〔Enter〕を押した後は、新しいキーを暗号化するために使用するパスワードの入力が要求されます。ノード マネージャではパスワードが必須です。

次に例を示します。

```
wlkeytool demokey.pem demokey_new
```

手順 3：証明書の証明書ファイルへの結合

WebLogic Server では、プライベート キー、公開鍵、および認証局に対して独立した証明書ファイル（拡張子 .pem）を使用します。プライベート キーがパスワードで保護された PKSC #5/#8 形式でなければならないという必要条件に加えて、ノード マネージャでは証明書のそれらの要素を 1 つの証明書ファイル（拡張子 .crt）に結合します。

注意： ユーザ SSL ID の要素が 1 つのファイルに結合されますが、プライベート キーの情報はサーバ間で転送されません。

それらの 3 つの要素は、拡張子が .crt である 1 つのファイルにそのまま連結されます。次に例を示します。

```
cat demokey_new democert.pem ca.pem > demo.crt
```

この例では、ca.pem は WebLogic 認証局ファイルであり、内容の点ではデフォルトの trustedCerts ファイル trusted.crt と同じです。democert.pem は公開鍵ファイルです。ファイル demokey_new は、demokey.pem で wlkeytool を実行した結果です（「手順 2 : WebLogic 形式のプライベート キーの変換」を参照）。

デジタル証明書とセキュア ソケット レイヤの詳細については、「セキュリティの管理」を参照してください。

ノード マネージャを使用するように管理サーバを設定

ノード マネージャを使用して WebLogic 管理対象サーバを起動および停止するように管理サーバをコンフィグレーションする場合は、実行する必要のあるいくつかの手順があります。それらの作業は、WebLogic Administration Console を使用して行います。

手順 1: マシンのコンフィグレーション エントリの作成

管理対象サーバをインストールしたマシンごとにドメイン コンフィグレーションでエントリを作成する必要があります。そのためには、次の操作を行います。

1. 管理サーバが動作している状態で、Administration Console を起動します（まだ動作していない場合）。
2. 左ペインで [マシン] テーブルを表示します。
3. テーブルの上部にある [新しい Machine のコンフィグレーション]（または [新しい Unix Machine のコンフィグレーション]）リンクを選択します。
4. マシンの情報を入力して [作成] をクリックし、新しいマシン エントリを作成します。

手順 2: 各マシンでのノード マネージャのコンフィグレーション

ノード マネージャを使用する各マシンについて、次のようにコンフィグレーション エントリを修正します。

1. Administration Console において、[マシン | *machine_name* | ノード マネージャ] を選択します。*machine_name* は、ノード マネージャが実行されるマシンの名前です。
2. [ノード マネージャ] タブのフィールドに情報を入力します。
   - リスン アドレスは、ノード マネージャが管理サーバからの要求を待ち受けるホスト名または IP アドレスです。これは、「ノード マネージャの起動」時に指定するリスン アドレスです。
   - リスン ポート番号は、そのマシンでノード マネージャを起動するときに使用するポート番号とも一致していなければなりません。
   - このノード マネージャと通信するために管理サーバが使用する証明書。デフォルトの証明書は `config\demo.crt` です。プロダクション環境用に新しい証明書を取得することをお勧めします。証明書取得の方法については、「セキュリティの管理」を参照してください。
   - 証明書のパスワードは暗号化されているので表示されません。ノード マネージャによって使用される証明書を変更する場合は、新しいデジタル

証明書のプライベート キーを暗号化するために使用されたパスワードに合わせてパスワードを変更する必要があります。

- trustedCerts ファイルには、認識される認証局のリストが格納されます。デフォルトは `config\trusted.crt` です。使用するデジタル証明書で示される認証局は、このファイルでリストされていなければなりません。

3. [適用] をクリックします。

手順 3: 管理対象サーバの起動情報のコンフィグレーション

ノード マネージャが WebLogic 管理対象サーバを起動するためには、その管理対象サーバを起動するときに使用する起動パラメータとオプションが必要です。その設定は次のように行います。

1. Administration Console を起動します（まだ動作していない場合）。
2. Administration Console において、[*server_name* | コンフィグレーション | リモート スタート] を選択します。*server_name* は、管理対象サーバの名前です。

   ここには、目的の管理対象サーバを起動するときに管理サーバが使用するコンフィグレーション情報を入力できる 5 つのフィールドがあります。

   注意： これらのフィールドで値を指定せずに、Administration Console から目的のサーバを起動しようとすると、ノード マネージャはノード マネージャを起動するときに使用したそれらの属性の値でサーバを起動しようとします。ノード マネージャを起動するときにコマンドラインで必要な値が指定されている場合であれば、ノード マネージャはそのようなケースでも管理対象サーバを起動できます。

   - [BEA Home]

     BEA ホーム ディレクトリを指定できます。これは、目的の管理対象サーバについてすべての BEA 製品とライセンスがインストールされたルート ディレクトリです。

   - [ルート ディレクトリ]

     これは、WebLogic ソフトウェアがインストールされたルート ディレクトリです。

   - [クラスパス]

     管理対象サーバを起動するためのクラスパスです。

最低でも、クラスパス オプションの以下の値を指定する必要があります。

`/weblogic/lib/weblogic_sp.jar`

`/weblogic/lib/weblogic.jar`

管理対象サーバを起動するときに使用する JDK をインストールしたルート ディレクトリのパスを指定することが必要な場合もあります。クラスパスの設定の詳細については、「WebLogic Server の起動と停止」を参照してください。

- 引数

  [引数] フィールドでは、起動コマンドに渡す他の引数を入力します。

  たとえば、Java ヒープ メモリの最大値と最小値を設定する場合が考えられます。`-ms64m` オプションと `-mx64m` オプションを使用すると、WebLogic Server にデフォルトの 64 MB の Java ヒープ メモリが割り当てられます。

注意： サーバ名、ユーザ名、またはパスワードを指定しないでください。また、管理サーバのアドレスとポートも指定しないでください。

- [セキュリティポリシー ファイル]

  JVM のセキュリティ ポリシー ファイルがデフォルトで使用されます。`weblogic\lib\weblogic.policy` にある WebLogic セキュリティ ポリシー ファイルも利用できます。

3. [適用] をクリックします。

ノード マネージャのプラットフォーム サポート

ノード マネージャは、Windows および Unix のプラットフォームのみで利用可能です。ネイティブ ライブラリが、Windows、Solaris、および HP UX オペレーティング システムでノード マネージャを実行するために用意されています。Solaris および HP UX 以外の Unix オペレーティング システムの場合は、ノード マネージャを起動するときに `java` コマンドラインで次の引数を使用する必要があります。

```
-Dweblogic.nodemanager.nativeVersionEnabled=false
```

注意： Solaris または HP UX 以外の UNIX オペレーティング システム上でノード マネージャを起動する場合、java コマンド ラインに渡すパラメータでスペースを使用することはできません。たとえば、次のパラメータを指定するとします。

```
-Dweblogic.Name=big iron
```

big iron にスペースが含まれているので、このパラメータは無効です。

コマンドラインからのノード マネージャの起動

ノード マネージャは、2 通りの方法で起動できます。ノード マネージャは、java コマンドラインまたは起動スクリプトを使用して起動できます。スクリプトの使い方については、「起動スクリプトを使用したノード マネージャの起動」を参照してください。ノード マネージャは、Windows サービスとして設定することもできます。Windows サービスとして設定されているノード マネージャは、Windows の再起動時に自動的に再起動されます。ノード マネージャを Windows サービスとして設定する方法については、「ノード マネージャの Windows サービスとしての設定」を参照してください。

環境の設定

ノード マネージャを起動する前には、多くの環境変数を設定する必要があります。環境変数を設定する 1 つの方法は、WebLogic Server ソフトウェアで用意されているスクリプトを実行することです。このスクリプトは、Unix では setEnv.sh、Windows では setEnv.cmd という名前になっています。このスクリプトは、*install_dir*\config*domain_name* というディレクトリに配置されています。*install_dir* は WebLogic をインストールしたディレクトリで、*domain_name* はドメインの名前です。

注意： ノード マネージャ起動スクリプト（Windows では startNodeManager.cmd、Unix では startNodeManager.sh）を使用してノード マネージャを起動する場合、環境変数はノード マネージャ起動スクリプトによって設定されるので独自に設定する必要はありません。詳細については、「起動スクリプトを使用したノード マネージャの起動」を参照してください。

Windows での環境変数の設定

JAVA_HOME 環境変数では、必ず、ノード マネージャで使用する JDK をインストールしたルート ディレクトリを示すようにします。次に例を示します。

```
set JAVA_HOME=D:\bea\jdk131
```

ノード マネージャには、WebLogic Server と同じ JDK のバージョンに関する必要条件があります。

また、WL_HOME 環境変数を設定することも必要です。次に例を示します。

```
set WL_HOME=D:\bea\wlserver6.1
```

さらに、ノード マネージャのクラスおよび java 実行ファイルにアクセスするための PATH 環境変数を設定する必要があります。次に例を示します。

```
set PATH=%WL_HOME%\bin;%JAVA_HOME%\bin;%PATH%
```

Unix での環境変数の設定

WL_HOME 環境変数が WebLogic をインストールしたディレクトリに設定されていると仮定して、WebLogic および JDK ソフトウェアを示すように PATH を設定する例を次に示します。

```
PATH=$WL_HOME/bin;$JAVA_HOME/jre/bin:$JAVA_HOME/bin:$PATH
```

上の例では、JAVA_HOME 変数が JDK をインストールしたルート ディレクトリを示すものと仮定しています。

ノード マネージャで使用されるネイティブ Unix ライブラリのパスを設定することも必要です。次に、Solaris での例を示します。

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$WL_HOME/lib/solaris:$WL_HOME/lib/solaris/oci8
16_8
```

次に、HP UX での例を示します。

```
SHLIB_PATH=$SHLIB_PATH:$WL_HOME/lib/hpux11:$WL_HOME/lib/hpux11/oci816_8
```

クラスパスの設定

クラスパスは、java コマンドラインのオプションとして、または環境変数として設定できます。次に、環境変数としてクラスパスを設定する例を示します（Windows NT）。

```
set CLASSPATH=.;.\lib\weblogic_sp.jar;.\lib\weblogic.jar
```

ノード マネージャの起動

起動スクリプトを使用しないでノード マネージャを起動する場合は、WebLogic Server ソフトウェアをインストールしたルート ディレクトリでノード マネージャを起動するようにしてください。これは、\config サブディレクトリのあるディレクトリです。

ノード マネージャを起動するコマンドは次のとおりです。

```
java weblogic.nodemanager.NodeManager
```

コマンドライン引数

ノード マネージャのリスン アドレスは、起動時に設定する必要があります。次のパラメータを使用すると、管理サーバからの要求をノード マネージャがリスンするアドレスを指定できます。

```
-Dweblogic.nodemanager.listenAddress=host
```

host は、ノード マネージャを起動するマシンの DNS 名または IP アドレスです。

ノード マネージャが管理サーバからの要求をリスンするデフォルトのポートは 5555 です。これは、次の起動パラメータで変更できます。

```
-Dweblogic.nodemanager.listenPort=port
```

ノード マネージャでは、その責任範囲にある各管理対象サーバのログを作成します。デフォルトでは、それらは NodeManagerLogs ディレクトリ内のサブディレクトリです。このディレクトリの位置は、次の起動パラメータで変更できます。

`-Dweblogic.nodemanager.savedLogsDirectory=`*`path`*

ノード マネージャでは、管理サーバとの通信でセキュア ソケット レイヤを使用します。このため、ノード マネージャの起動時にはデジタル証明書を指定することが必要となります。証明書の位置は、次の起動パラメータで指定できます。

`-Dweblogic.nodemanager.certificateFile=`*`path_to_cert`*

ノード マネージャで使用するデフォルトの証明書の種類は RSA です。DSA などの別の種類の証明書を指定する場合には、コマンドラインで次の引数を使用します。

`-Dweblogic.nodemanager.certificateType=`*`type`*

`type` は、`RSA` または `DSA` のいずれかです。

暗号化されたプライベート キーにアクセスするために使用するパスワードを渡すには、コマンドラインで次の引数を使用します。

`-Dweblogic.nodemanager.certificatePassword=`*`pkpassword`*

`pkpassword` は、プライベート キーのパスワードです。

ユーザの ID を証明するために使用される認証局（または認証局のチェーン）は、信頼性のある認証局ファイルに格納されます。デフォルトでは `config\demo.crt` です。別の信頼性のある認証局ファイルを指定するには、コマンドラインで次の引数を使用します。

`-Dweblogic.nodemanager.trustedCerts=`*`path`*

`path` は、信頼性のある認証局ファイルの位置です。

BEA ホーム ディレクトリ（すべての BEA 製品とライセンスがインストールされるルート ディレクトリ）の位置を指定する必要もあります。BEA ホーム ディレクトリは、次のコマンドライン引数で指定できます。

`-Dbea.home=`*`directory`*

信頼性のあるホスト ファイルで IP アドレスではなく DNS ホスト名を使用した場合には、次の起動パラメータも必要です。

`-Dweblogic.nodemanager.reverseDnsEnabled=true`

デフォルトでは、DNS の逆引き参照は無効です。

次の起動パラメータを使用すると、信頼性のあるホストのリストが格納されるファイルの名前を指定できます。

```
-Dweblogic.nodemanager.trustedHosts=path
```

path は、信頼性のあるホスト ファイルの位置です。デフォルトでは、このファイルは \config ディレクトリに配置されます。

WebLogic セキュリティ ポリシー ファイルのデフォルトの位置は、weblogic\lib\weblogic.policy です。別の位置を指定するには、コマンドラインで次の引数を使用します。

```
-Djava.security.policy==policy_file
```

policy_file では、WebLogic ポリシー ファイルの位置を指定します。

デフォルトでは、ノード マネージャでは SSL ホスト名が確認されません。ホスト名を確認するには、コマンドラインで次の引数を使用します。

```
-Dweblogic.nodemanager.sslHostNameVerificationEnabled=true
```

クラスパス オプション

ノード マネージャでは、WebLogic Server で使用される同じ Java クラスのいくつかも必要となります。ノード マネージャを起動するときには、java コマンドラインの -classpath オプションで次の値を含める必要があります。

- */weblogic*/lib/weblogic_sp.jar
- */weblogic*/lib/weblogic.jar

起動スクリプトを使用したノード マネージャの起動

ノード マネージャの起動に使用できるサンプルの起動スクリプトが用意されています。それらのスクリプトは、WebLogic Server ソフトウェアをインストールした \config ディレクトリに配置されています。Windows 用の起動スクリプトは、startNodeManager.cmd という名前です。Unix 用の起動スクリプトは、startNodeManager.sh という名前です。これらのスクリプトは、ノード マネージャの起動時に必要な環境値も設定します。

起動コマンドがノード マネージャのリスン アドレスを設定するように、ノード マネージャの起動スクリプトを編集する必要があります。リスン アドレスを設定するには、起動コマンドに次の引数を含めます。

```
-Dweblogic.nodemanager.listenAddress=host
```

host は、ノード マネージャを実行するマシンの DNS 名または IP アドレスです。

管理対象サーバのリモートでの起動と強制停止

管理対象サーバのコンフィグレーションされているマシン上でノード マネージャが動作している場合は、次のようにして管理対象サーバを起動できます。

1. Administration Console を起動します（まだ動作していない場合）。
2. ナビゲーション ツリー（左ペイン）でサーバの名前を右クリックします。
3. **[このサーバを開始 ...]** を選択します。

管理対象サーバを起動するときには、通常は WebLogic Server の起動時に STDOUT または STDERROR に出力されるメッセージが Administration Console の右ペインに表示されます。それらのメッセージは、そのサーバのノード マネージャ ログ ファイルにも書き込まれます。

管理対象サーバは、同じようにして停止できます。

1. 左ペインで管理対象サーバの名前を右クリックします。
2. **[このサーバを強制停止 ...]** を選択します。

[このサーバを強制停止 ...] オプションは、目的の管理対象サーバが動作しているマシン上のノード マネージャに WebLogic Server プロセスを強制停止するように指示します。

注意： **[このサーバを強制停止 ...]** オプションは、管理サーバを停止するためには使用できません。

管理対象サーバの停止と強制停止の区別

Administration Console の左ペインでサーバの名前を右クリックした場合、表示されるオプションの 1 つは **[このサーバを停止 ...]** です。このオプションでは、ノード マネージャを使用しないで選択されたサーバを停止します。**[このサーバを停止 ...]** オプションを選択すると、選択されたサーバに管理サーバが停止要求を送信します。ノード マネージャはこの場合には使用されません。**[このサーバを強制停止 ...]** オプションとは違って、**[このサーバを停止 ...]** オプションは管理サーバを停止するために使用できます。

[このサーバを停止 ...] オプションでは管理対象サーバの管理機能を使用して停止を開始するので、サーバがアクティブで、管理要求に応答している場合にのみ使用できます。**[このサーバを強制停止 ...]** オプションは、通常は、目的のサーバがハングしているか、管理サーバからの管理要求に応答していない場合に使用します。

同じポップアップ メニューで、管理対象サーバで生成される `StdOut` および `StdErr` 出力にもアクセスできます。**[このサーバの StdOut を取得]** オプションを選択すると StdOut 出力が表示され、**[このサーバの StdErr を取得]** を選択すると StdErr 出力が表示されます。

ドメインおよびクラスタの起動と強制停止

アクティブなドメインのすべての管理対象サーバを起動または強制停止することもできます。

1. 左ペインでアクティブなドメインの名前を右クリックします。
2. **[このサーバを強制停止 ...]** または **[このサーバを停止 ...]** を選択します。

Administration Console からドメイン全体を起動した場合、右ペインに表示される結果はそのドメインにコンフィグレーションされた各管理対象サーバの結果への一連のリンクになります。

選択したクラスタのすべての管理対象サーバの起動と強制停止も、同じような方法で 1 アクションで実行できます。

1. 左ペインでクラスタの名前を右クリックします。
2. **[このクラスタを強制停止 ...]** または **[このクラスタを開始 ...]** を選択します。

注意： ノード マネージャを使用して管理サーバを起動または強制停止することはできません。

ノード マネージャの Windows サービスとしての設定

ディレクトリ *install_dir*\config*mydomain*（*install_dir* は WebLogic Server をインストールしたルート ディレクトリ、*mydomain* はインストール時に指定されるデフォルトのコンフィグレーション ディレクトリ名）には、WebLogic Server を Windows サービスとしてインストールまたはアンインストールするためのスクリプトがあります。スクリプト `installNtService.cmd` は WebLogic Server を Windows サービスとしてインストールするために使用し、スクリプト `uninstallNtService.cmd` は Windows サービスとしての WebLogic Server をアンインストールするために使用します。これらのスクリプトをコピーして修正すると、ノード マネージャを Windows サービスとしてインストールまたは削除できます。

次の手順では、`mydomain` がインストール時に指定されたデフォルト コンフィグレーション ディレクトリであることを前提にしています。デフォルト コンフィグレーション ディレクトリにそれ以外の名前を指定した場合、`mydomain` をすべてその名前に置き換えます。

ノード マネージャを Windows サービスとしてインストールするには、次の操作を行います。

1. *install_dir*\config\mydomain ディレクトリ（*install_dir* は WebLogic ソフトウェアをインストールしたルート ディレクトリ）にあるスクリプト installNtService.cmd のコピーを作成し、その名前を installNMNtService.cmd に変更します。
2. *install_dir*\config\mydomain ディレクトリ（*install_dir* は WebLogic ソフトウェアをインストールしたルート ディレクトリ）にあるスクリプト uninstallNtService.cmd のコピーを作成し、その名前を uninstallNMNtService.cmd に変更します。
3. スクリプト installNMNtService.cmd を修正して、ノード マネージャの起動で使用するようコマンドラインに指定します。必ず、起動コマンドを修正して、目的の起動クラスを weblogic.Server から weblogic.nodemanager.NodeManager に変更します。コマンドライン オプションについては、「コマンドラインからのノード マネージャの起動」を参照してください。
4. サービスの名前を適切な名前（nodemanager など）に変更します。
5. スクリプト uninstallNMNtService.cmd を修正して、目的のサービスの名前を、ノード マネージャを Windows サービスとして起動する際に installNMNtService.cmd スクリプトで使用する名前に変更します。
6. installNMNtService.cmd スクリプトは、c:\bea\wlserver6.1 のように、必ず WebLogic ソフトウェアをインストールしたルート ディレクトリで起動されるようにします。
7. スクリプト installNMNtService.cmd を実行して、ノード マネージャを Windows サービスとしてインストールします。
8. 次のように選択して、ノード マネージャを Windows サービスとして起動します。
   [スタート｜設定｜コントロール パネル｜管理ツール｜サービス]

Windows サービスとしてのノード マネージャの削除

Windows サービスとしてのノード マネージャをアンインストールするには、スクリプト `uninstallNMNtService.cmd` を実行します。

4 WebLogic Server とクラスタのコンフィグレーション

以下の節では、WebLogic Server および WebLogic Server クラスタの設定方法について説明します。

- サーバとクラスタのコンフィグレーションの概要
- 管理サーバの役割
- Administration Console の起動
- 動的コンフィグレーションの仕組み
- クラスタ コンフィグレーションのプランニング
- サーバ コンフィグレーションの作業
- クラスタ コンフィグレーションの作業

サーバとクラスタのコンフィグレーションの概要

WebLogic Server およびクラスタから成るドメインの永続的なコンフィグレーションは、XML コンフィグレーション ファイルに格納されます。このファイルは 3 通りの方法で変更できます。

- ドメイン コンフィグレーションを管理およびモニタするための BEA のグラフィカル ユーザ インタフェース (GUI) である Administration Console を使用します。Administration Console は、ドメイン コンフィグレーションを修正またはモニタするための第一の手段です。
- WebLogic Server に付属のアプリケーション プログラミング インタフェース（API）に基づき、プログラムを記述してコンフィグレーション属性を変更します。
- ドメイン リソースのコンフィグレーション属性にアクセスするための WebLogic Server コマンドライン ユーティリティを実行します。このユーティリティは、ドメイン管理を自動化するスクリプトを作成する場合に使用します。

管理サーバの役割

いずれの方法を選択する場合でも、ドメイン コンフィグレーションを変更するときに管理サーバが動作している必要があります。

管理サーバとは、Administration Service が動作している WebLogic Server のことです。Administration Service は、WebLogic Server のための機能を提供し、ドメイン全体のコンフィグレーションを管理します。

デフォルトでは、WebLogic Server のインスタンスは管理サーバとして扱われます。管理サーバが起動すると、コンフィグレーション ファイルがロードされます。コンフィグレーション ファイルは、デフォルトでは、*WEBLOGIC_HOME* ディレクトリの `config` というディレクトリに格納されます。`config` ディレクトリには、管理サーバで利用できる各ドメイン用のサブディレクトリがあります。実

際のコンフィグレーション ファイルは、そのドメイン専用のディレクトリに配置され、config.xml と呼ばれます。デフォルトでは、管理サーバが起動すると、WebLogic Server ソフトウェアのインストール時に指定されたデフォルトのドメイン ディレクトリでコンフィグレーション ファイル（config.xml）が検索されます。

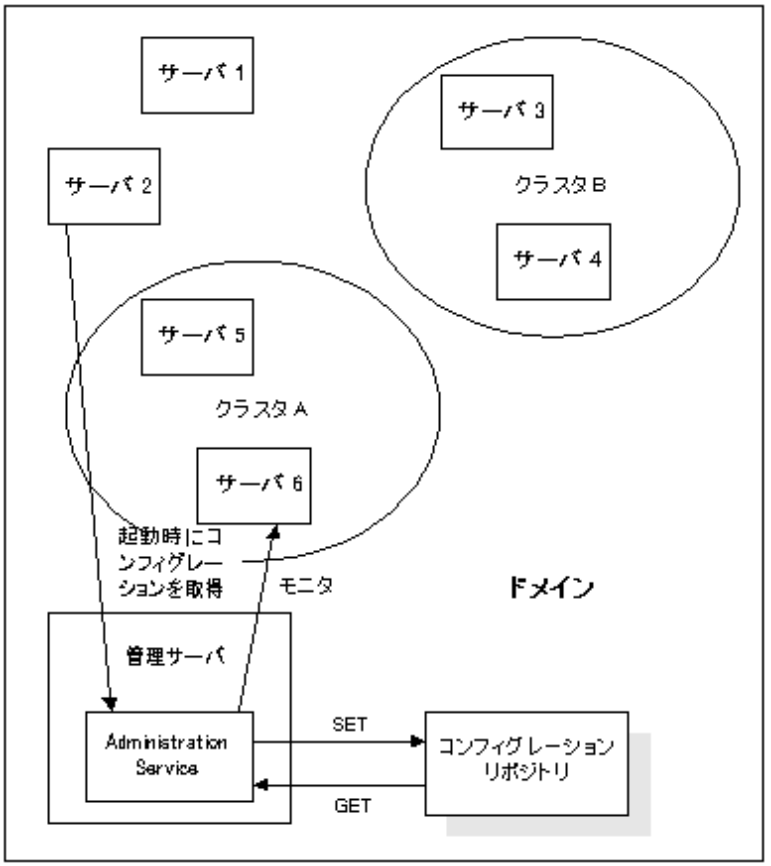
管理サーバが正常に起動するたびに、config.xml.booted という名前のバックアップ用コンフィグレーション ファイルがドメイン専用のディレクトリに作成されます。万が一、サーバが終了するまでの間に config.xml ファイルが壊れるようなことがあっても、この正常なコンフィグレーションに戻ることができます。

ドメインは、1 つの WebLogic Server だけで構成することもできます。ただし、各ドメインには少なくとも（そして最大で）1 つの管理サーバが必要なので、その場合には、その WebLogic Server が管理サーバになります。

図 4-1 は、1 つの管理サーバと複数の WebLogic Server が存在する典型的なプロダクション環境を示しています。そのようなドメインでサーバを起動するときには、管理サーバを最初に起動します。その他の各サーバは、起動時に、コンフィグレーション情報を取得するために管理サーバにアクセスするように指示されます。このように、管理サーバはドメイン全体のコンフィグレーションの一元的な制御エンティティとして動作します。1 つのドメイン内で 複数の管理サーバがアクティブになることはできません。管理サーバだけが、動作中にコンフィグレーション ファイルを修正できます。

注意： 共有ファイルシステムと 1 つのインストールを使用して、異なるマシン上で複数の WebLogic Server インスタンスを実行しないでください。共有ファイルシステムを使用すると、シングル ポイントの競合が発生します。共有ファイルシステムにアクセスする場合（たとえば、個々のログファイルに書き込みを行う場合など）に、すべてのサーバが競合することになります。さらに、共有ファイルシステムに障害が発生した場合には、管理対象のサーバまたはクラスタ化されたサーバを起動できなくなることもあります。

図 4-1 WebLogic Server のコンフィグレーション



Administration Console の起動

管理サーバには、Administration Console を通じてアクセスできます。Administration Console を開くには、次の操作を行います。

1. 次の URL を入力します。
   `http://host:port/console`

host は管理サーバが動作しているマシンのホスト名または IP アドレス、*port* は管理サーバが要求をリスンするポートのアドレス（デフォルトは 7001）です。

2. ユーザ ID とパスワードの入力を要求されます。ユーザ ID とパスワードを入力します。認証と認可のチェックが実行されます。チェックでは、ユーザ データベースと照らし合わせてユーザ ID とパスワードが確認されます。

   コンソールの使用が許可されている場合は、システム管理者によって割り当てられているアクセス モード（ReadOnly または Read/Write）でコンソールが表示されます。

動的コンフィグレーションの仕組み

WebLogic Server では、ドメイン リソースのコンフィグレーション属性を動的に（サーバの動作中に）変更できます。ほとんどの場合では、変更を有効にするために WebLogic Server を再起動する必要はありません。属性をコンフィグレーションし直すと、新しい値は、属性の現在の実行時値と、XML コンフィグレーション ファイルに格納されている永続的な値の両方に直ちに反映されます。

ただし、例外もあります。たとえば、WebLogic Server のリスン ポートを変更した場合、新しいアドレスは対象サーバを次に再起動するまで使用されません。その場合は、値を変更すると、XML ファイルに格納されている永続的な値が変更され、その属性の現在の実行時コンフィグレーション値と永続的に格納されている値が同じでなくなる場合があります。Administration Console では、アイコンを使用して、コンフィグレーション属性の永続的な値と実行時値が同じであるかどうかが示されます。そのアイコンが、⚠ という警告に変化した場合は、変更を有効にするためにサーバを再起動する必要があります。

Administration Console では、ユーザが変更した各属性に対して検証が行われます。サポートされているエラーは、範囲外のエラーとデータ型の不一致エラーです。両方の場合で、エラーが発生したことをユーザに通知するエラー ダイアログ ボックスが表示されます。

Administration Console の起動後に、別のプロセスが管理サーバに割り当てられたリスン ポートを獲得した場合、サーバを獲得したプロセスを削除しなければなりません。管理サーバに割り当てられているリスン ポートを獲得したプロセスを削除できない場合は、`Config.XML` ファイルを編集して、割り当てられたリ

スン ポートを変更する必要があります。Config.XML ファイルの編集については、『BEA WebLogic Server コンフィグレーション リファレンス』を参照してください。

クラスタ コンフィグレーションのプランニング

クラスタ コンフィグレーションをプランニングするときは、ネットワーク環境とクラスタ コンフィグレーションに関して、以下の制約があることに注意してください。

1. クラスタの WebLogic ホストとして使用するマシンには、静的な IP アドレスが永続的に割り当てられている必要があります。クラスタ化環境では、動的に割り当てられる IP アドレスは使用できません。サーバとクライアントの間にファイアウォールがある場合、各サーバには、クライアントがアクセスできるパブリックな静的 IP アドレスが割り当てられている必要があります。
2. クラスタ内の WebLogic Server は、すべて同じローカル エリア ネットワーク（LAN）上にあり、IP マルチキャストを通じてアクセス可能でなければなりません。
3. クラスタ内のすべてのサーバは同じバージョンの WebLogic Server を実行する必要があります。

   クラスタ内のサーバは、提供するサービスの構成をサポートするようにコンフィグレーションしてください。

   - EJB で JDBC 接続が使用される場合は、特定の EJB をデプロイするすべてのサーバでデプロイメントと永続性のコンフィグレーションが同じでなければなりません。つまり、各サーバで同じ JDBC 接続プールをコンフィグレーションします。
   - サーブレットのホストとなるすべてのマシンでは、同じ ACL（アクセス制御リスト）を使用して同じサーブレットのリストを維持しなければなりません。
   - クライアント アプリケーションで JDBC 接続プールが直接使用される場合は、各 WebLogic Server で同じ接続プール（ACL も同じ）を作成しなければなりません。つまり、クラスタ内のすべてのマシンで使用できる

接続プールを作成しなければりません。たとえば、WebLogic が動作している Windows NT サーバで Microsoft SQL Server データベースへの接続プールをコンフィグレーションする場合、Windows 以外のマシン（Microsoft SQL Server 接続をサポートできないマシン）が存在するクラスタではこの接続プールは使用できません。

- その他のコンフィグレーションの詳細は、クラスタ内のさまざまなメンバーによって異なることがあります。たとえば、小規模な Windows NT のワークステーションよりも多くのログイン要求を処理するように Solaris サーバをコンフィグレーションする場合があり、そうした相違は許容されます。従ってこの例では、すべてのメンバーのサービス コンフィグレーションが同じである場合に限り、個々のクラスタ メンバーのパフォーマンスに関する属性は異なる値でコンフィグレーションできます。実際には、クラスタ内の WebLogic Server は、WebLogic サービス、クラス ファイル、および外部リソース（データベースなど）に関連するすべての領域で同じようにコンフィグレーションすることになります。

サーバ コンフィグレーションの作業

Administration Console では、以下のサーバ コンフィグレーションを行うことができます。

- Administration Console の [サーバ] ノードを使用してサーバを個別にコンフィグレーションできます。このノードを使用して変更できる属性には、サーバ名、リスン ポート、およびリスン アドレスなどがあります。
- Administration Console の [サーバ] ノードを使用してサーバを個別に複製できます。サーバは元のサーバの属性値を維持して複製され、新しいサーバの名前は [サーバ] ノードの [コンフィグレーション | 一般] で設定します。
- Administration Console の [サーバ] ノードを使用してサーバを削除できます。削除するサーバの削除アイコンをクリックします。削除の確認を求めるメッセージが表示されます。[はい] をクリックして削除を確定するとサーバが削除されます。
- Administration Console の [サーバ] ノードを使用してサーバ ログを表示できます。モニタするサーバをクリックします。[モニタ] タブを選択します。[

サーバ ログを見る] リンクをクリックし、Administration Console の右ペインでサーバ ログをモニタします。

- Administration Console の [サーバ] ノードを使用してサーバの JNDI ツリーを表示できます。モニタするサーバをクリックします。[モニタ] タブを選択します。[JNDI ツリーを見る] リンクをクリックし、Administration Console の右ペインでツリーを表示します。
- Administration Console の [サーバ] ノードを使用してサーバの実行キューを表示できます。モニタするサーバをクリックします。[実行キューを見る] リンクをクリックし、Administration Console の右ペインでテーブルを表示します。
- Administration Console の [サーバ] ノードを使用してサーバの実行スレッドを表示できます。モニタするサーバをクリックします。[実行スレッドを見る] リンクをクリックし、Administration Console の右ペインでテーブルを表示します。
- Administration Console の [サーバ] ノードを使用してサーバのソケットを表示できます。モニタするサーバをクリックします。[ソケットを見る] リンクをクリックし、Administration Console の右ペインでテーブルを表示します。
- Administration Console の [サーバ] ノードを使用してサーバの接続を表示できます。モニタするサーバをクリックします。[接続を見る] リンクをクリックし、Administration Console の右ペインでテーブルを表示します。
- Administration Console の [サーバ] ノードを使用してサーバでガベージ コレクションを強制できます。モニタするサーバをクリックします。[パフォーマンス] タブを選択します。[ガーベジコレクションを強制する] をクリックします。ガベージ コレクションが行われたことを確認するメッセージが表示されます。
- Administration Console の [サーバ] ノードを使用してサーバのセキュリティをモニタできます。モニタするサーバをクリックします。[モニタ] タブを選択します。[セキュリティ] タブを選択します。セキュリティ情報が表示されます。
- Administration Console の [サーバ] ノードを使用してサーバのバージョンを表示できます。モニタするサーバをクリックします。[バージョン] タブを選択します。このサーバのバージョン データが表示されます。

- Administration Console の [サーバ] ノードを使用してサーバのクラスタをモニタできます。モニタするサーバをクリックします。[クラスタ] タブを選択します。このサーバのクラスタ データが表示されます。
- Administration Console の [サーバ] ノードを使用してサーバで EJB をデプロイできます。EJB をデプロイするサーバをクリックします。[デプロイメント | EJB] タブをクリックします。デプロイする EJB をクリックし、移動コントロールを使用して [選択済み] カラムに移動します。[適用] をクリックして選択を保存します。
- Administration Console の [サーバ] ノードを使用してサーバのすべての EJB デプロイメントをモニタできます。EJB をモニタするサーバをクリックします。[デプロイメント | EJB] タブをクリックします。[すべてのアクティブな EJB のモニタ] リンクをクリックして [アクティブな EJB] テーブルを表示します。
- Administration Console の [サーバ] ノードを使用してサーバで Web アプリケーション コンポーネントをデプロイできます。Web アプリケーションをデプロイするサーバをクリックします。[デプロイメント | Web アプリケーション] タブをクリックします。デプロイする Web アプリケーションをクリックし、移動コントロールを使用して [選択済み] カラムに移動します。[適用] をクリックして選択を保存します。
- Administration Console の [サーバ] ノードを使用してサーバのすべての Web アプリケーション コンポーネントをモニタできます。Web アプリケーションをモニタするサーバをクリックします。[すべてのアクティブな Web アプリケーションのモニタ] リンクをクリックしてテーブルを表示します。
- Administration Console の [サーバ] ノードを使用してサーバでスタートアップ クラスとシャットダウン クラスをデプロイできます。スタートアップ クラスをデプロイするサーバをクリックします。[デプロイメント | 起動 / 停止] タブをクリックします。デプロイするスタートアップ クラスをクリックし、移動コントロールを使用して [選択済み] カラムに移動します。[適用] をクリックして選択を保存します。シャットダウン クラスの場合も、[停止クラス] コントロールを使用して同じ手順を行います。
- Administration Console の [サーバ] ノードを使用してサーバに JDBC 接続プールを割り当てることができます。JDBC 接続プールを割り当てるサーバをクリックします。[サービス | JDBC] タブをクリックします。サーバに割り当てる 1 つまたは複数の JDBC 接続プールを [選択可] カラムでクリック

し、移動コントロールを使用して [選択済み] カラムに移動します。[適用] をクリックして割り当てを保存します。

- Administration Console の [サーバ] ノードを使用してサーバに WLEC 接続プールを割り当てることができます。WLEC 接続プールを割り当てるサーバをクリックします。[サービス | WLEC] タブをクリックします。サーバに割り当てる 1 つまたは複数の WLEC 接続プールを [選択可] カラムでクリックし、移動コントロールを使用して [選択済み] カラムに移動します。
- Administration Console の [サーバ] ノードを使用してサーバのすべての WLEC 接続プールをモニタできます。WLEC 接続プールをモニタするサーバをクリックします。[サービス | WLEC] タブをクリックします。[すべてのアクティブ プールのモニタ] テキスト リンクをクリックします。このサーバに割り当てられているすべての接続プールを示す [アクティブ WLEC 接続プール] テーブルが表示されます。
- Administration Console の [サーバ] ノードを使用してサーバに XML レジストリを割り当てることができます。XML レジストリを割り当てるサーバをクリックします。[サービス | XML] タブをクリックします。[XML レジストリ] ドロップダウン リスト ボックスからレジストリをクリックします。[適用] をクリックして選択を保存します。
- Administration Console の [サーバ] ノードを使用してサーバにメール セッションを割り当てることができます。メール セッションを割り当てるサーバをクリックします。サーバに割り当てる 1 つまたは複数のメール セッションを [選択可] カラムでクリックします。移動コントロールを使用して、選択したメール セッションを [選択済み] カラムに移動します。[適用] をクリックして選択を保存します。
- Administration Console の [サーバ] ノードを使用してサーバに FileT3 を割り当てることができます。FileT3 を割り当てるサーバをクリックします。[サービス | File T3] タブをクリックします。サーバに割り当てる 1 つまたは複数の FileT3 を [選択可] カラムでクリックします。移動コントロールを使用して、選択した FileT3 を [選択済み] カラムに移動します。[適用] をクリックして選択を保存します。

クラスタ コンフィグレーションの作業

Administration Console では、以下のクラスタ コンフィグレーションを行うことができます。

- Administration Console の [クラスタ] ノードを使用してサーバのクラスタをコンフィグレーションできます。このノードを使用して変更できる属性には、クラスタ名、クラスタ アドレス、デフォルトのロード バランス アルゴリズム、およびサービス期間しきい値があります。
- Administration Console の [クラスタ] ノードを使用してサーバのクラスタを複製できます。クラスタは元のクラスタの属性値とサーバを維持して複製され、新しいクラスタの名前は [サーバ] ノードの [コンフィグレーション] で設定します。
- Administration Console の [クラスタ] ノードを使用してクラスタ内のサーバをモニタできます。サーバをモニタするクラスタをクリックします。[モニタ] タブをクリックします。[このクラスタを構成するサーバをモニタ] テキスト リンクをクリックします。このクラスタに割り当てられているすべてのサーバを示すサーバ テーブルが表示されます。
- Administration Console の [クラスタ] ノードを使用してクラスタにサーバを割り当てることができます。サーバを割り当てるクラスタをクリックします。[サーバ] タブをクリックします。クラスタに割り当てる 1 つまたは複数のサーバを [選択可] カラムでクリックします。移動コントロールを使用して、選択したサーバを [選択済み] カラムに移動します。[適用] をクリックして選択を保存します。
- Administration Console の [クラスタ] ノードを使用してクラスタを削除できます。削除するクラスタの行にある [削除] アイコンをクリックします。削除要求の確認を求めるメッセージが右ペインに表示されます。[はい] をクリックして削除を確定するとクラスタが削除されます。

5 WebLogic Server ドメインのモニタ

以下の節では、WebLogic Server ドメインをモニタする方法について説明します。

- モニタの概要
- サーバのモニタ
- JDBC 接続プール のモニタ

モニタの概要

WebLogic Server ドメインの状態とパフォーマンスをモニタするためのツールは Administration Console です。Administration Console では、サーバ、HTTP、JTA サブシステム、JNDI、セキュリティ、CORBA 接続プール、EJB、JDBC、JMS といった WebLogic Server リソースのステータスと統計を表示できます。

モニタ情報は、Administration Console の右ペインに表示されます。ページにアクセスするには、左ペインの階層的なドメイン ツリーでコンテナまたはサブシステム、あるいはコンテナの下の特定のエンティティを選択します。

Administration Console には、モニタ情報を表示する以下の 3 種類のページがあります。

- 特定のエンティティ（JDBC 接続プールのインスタンスや特定のサーバのパフォーマンスなど）のモニタ タブ ページ。
- 特定の種類のすべてのエンティティに関するデータのテーブル（WebLogic Server テーブルなど）。

- ドメイン ログおよびローカル サーバ ログのビュー。ログ メッセージについては、「ログ メッセージを使用した WebLogic Server の管理」を参照してください。

Administration Console では、ドメイン リソースについての情報が管理サーバから取得されます。管理サーバでは、Sun の Java Management Extension（JMX）規格に基づく Management Bean（MBean）が使用されます。JMX 規格は、管理を目的としてドメイン リソースにアクセスする方法を定めています。

管理サーバには、ドメインのコンフィグレーションを管理するコンフィグレーション MBean と実行時 MBean があります。実行時 MBean では、JVM のメモリ使用率や WebLogic Server のステータスといったドメイン リソースに関する特定の時点での情報が提供されます。ドメインの特定のリソース（Web アプリケーションなど）がインスタンス化されると、その特定のリソースについての情報を収集する MBean のインスタンスが生成されます。

Administration Console で特定のリソースのモニタ ページにアクセスすると、管理サーバでは現在の属性値を取り出すための GET 処理が実行されます。

以降の節では、WebLogic Server ドメインの管理に便利なモニタ ページをいくつか選んで説明します。それらのページは、ここでは、Administration Console の機能説明を目的として取り上げています。

サーバのモニタ

サーバ テーブルおよび個別サーバのモニタ タブ ページでは、WebLogic Server をモニタできます。サーバ テーブルでは、ドメイン内のすべてのサーバのステータスが簡潔に表示されます。サーバからログ メッセージの一部しかドメイン ログに転送されない場合は、ローカル サーバ ログにアクセスすると、トラブルシューティングやイベントの調査に便利です。

ログ ファイルとロギング サブシステムの詳細については、「ログ メッセージを使用した WebLogic Server の管理」を参照してください。

各 WebLogic サーバのモニタ データには、そのサーバのモニタ タブからアクセスできます。ロギング タブからは、サーバのローカル ログ（サーバが稼働しているマシン上のログ）にアクセスできます。

[モニタ｜一般] タブ ページでは、現在の状態とアクティブ化時刻が表示され、アクティブ キュー テーブル、アクティブ ソケット テーブル、および接続テーブルにアクセスできます。アクティブ実行キュー テーブルは、保留中の最も古い要求や、キューのスループットといったパフォーマンス情報を提供します。

パフォーマンス

[モニタ｜パフォーマンス] タブは、JVM メモリ ヒープの使用率、要求スループット、およびキューの長さに関するリアルタイム データをグラフで示します。このタブ ページでは、メモリ ヒープでのガベージ コレクション実行を JVM に強制することもできます。

Java ヒープは、ライブ Java オブジェクトおよびデッド Java オブジェクトのリポジトリです。通常は、ガベージ コレクションを手動で実行する必要はなく、JVM で自動的に行われます。JVM でメモリが不足し始めると、すべての実行が停止され、ガベージ コレクション アルゴリズムを使用して Java アプリケーションで使用されなくなったスペースが解放されます。

その一方で、アプリケーションをデバッグする開発者には、ガベージ コレクションを手動で強制しなければならない場合もあります。手動のガベージ コレクションは、たとえば JVM メモリを急速に消費するメモリ リークをテストする場合に便利です。

サーバのセキュリティ

[モニタ｜セキュリティ] タブでは、不正なログインの試行およびロックされているユーザとロックが解除されているユーザについての統計が表示されます。

JMS

[モニタ｜ JMS] タブでは、JMS サーバおよび接続に関する統計が表示されます。また、このページは、アクティブな JMS 接続とアクティブな JMS サーバのテーブルへのリンクも提供します。これらは、現在のセッション総数などの属性をモニタします。

JT

[モニタ | JTA] タブでは、トランザクション総数やロールバック総数などの Java トランザクション サブシステムに関する統計が表示されます。このページは、リソースと名前によってリストされるトランザクションのテーブルと、実行中のトランザクションのテーブルへのリンクを提供します。

JDBC 接続プール のモニタ

Java Database Connectivity（JDBC）サブシステムのリソースは、Administration Console を使用してモニタできます。JDBC 接続プールの [モニタ] タブを使用すると、そのプールのインスタンスに関する統計を示す表にアクセスできます。Administration Console の他のエンティティ テーブルと同様に、テーブルをカスタマイズして表示する属性を選択できます。

それらの属性は、クライアントのデータベース アクセスを管理するための重要な情報を提供します。

[最大待ち] フィールドは、一度に接続を待つクライアントの最大数を示します。[待ち] フィールドは、現在接続を待機中のクライアント数を示します。[最大接続数] フィールドは、一度に発生した接続の最大数を示します。[最大待ち時間（秒）] フィールドは、クライアントがデータベース接続を待つ最長時間を示します。これらの属性から、クライアント要求への応答に関して、現在のコンフィグレーションの効果を判断できます。

[最大接続数] フィールドの値が [最大容量] フィールドの値（[コンフィグレーション | 接続] タブで設定）に近い場合は、[最大容量]（同時接続の最大数）の値を増やすことを検討することがあります。[最大待ち] フィールドの値がクライアントがデータベース アクセスを長時間待たなければならないことを示す場合、プールのサイズを増やすことがあります。

[縮小間隔] フィールドの値は、プールが最大のサイズから縮小するまでに JDBC サブシステムが待つ時間です。サブシステムがプールを縮小するとき、データベース接続は破棄されます。データベース接続を作成するとリソースが消費されて時間もかかることがあります。システムでクライアント要求の発生が断続的に集中する場合、縮小間隔が短いと、データベース接続が絶えず再作成されパフォーマンスが低下することがあります。

6　ログ メッセージを使用した WebLogic Server の管理

以下の節では、ロギング サブシステムの機能について説明します。

- ロギング サブシステムの概要
- ローカル サーバのログ ファイル
- メッセージの属性
- メッセージ カタログ
- メッセージの重要度
- ログ ファイルの参照
- ドメイン ログ フィルタの作成

ロギング サブシステムの概要

ログ メッセージは、システムの管理に便利なツールです。ログ メッセージを利用すると、問題の検出、傷害の発生源の特定、およびシステム パフォーマンスの監視ができます。WebLogic Server ソフトウェアで生成されるログ メッセージは、以下のように 2 つの場所に格納されます。

- WebLogic Server コンポーネント サブシステムでは、ローカル ファイル（サーバが動作しているマシン上のファイル）に記録されるメッセージが生成されます。マシン上に複数のサーバがある場合は、各サーバ用に別々のログ ファイルが用意されます。WebLogic Server にデプロイされたアプリケーションのメッセージもサーバのローカル ログ ファイルに記録されます。
- また、ローカルで記録されるメッセージの一部は、管理サーバで管理されるドメイン全体のログ ファイルにも格納されます。

WebLogic Server に組み込まれている Java Management Extension（JMX）の機能は、ログ メッセージを WebLogic Server から管理サーバに送信するために使用します。ローカル WebLogic Server の判断に基づいて他のエンティティに転送されるメッセージは、JMX の用語で通知と呼ばれます。

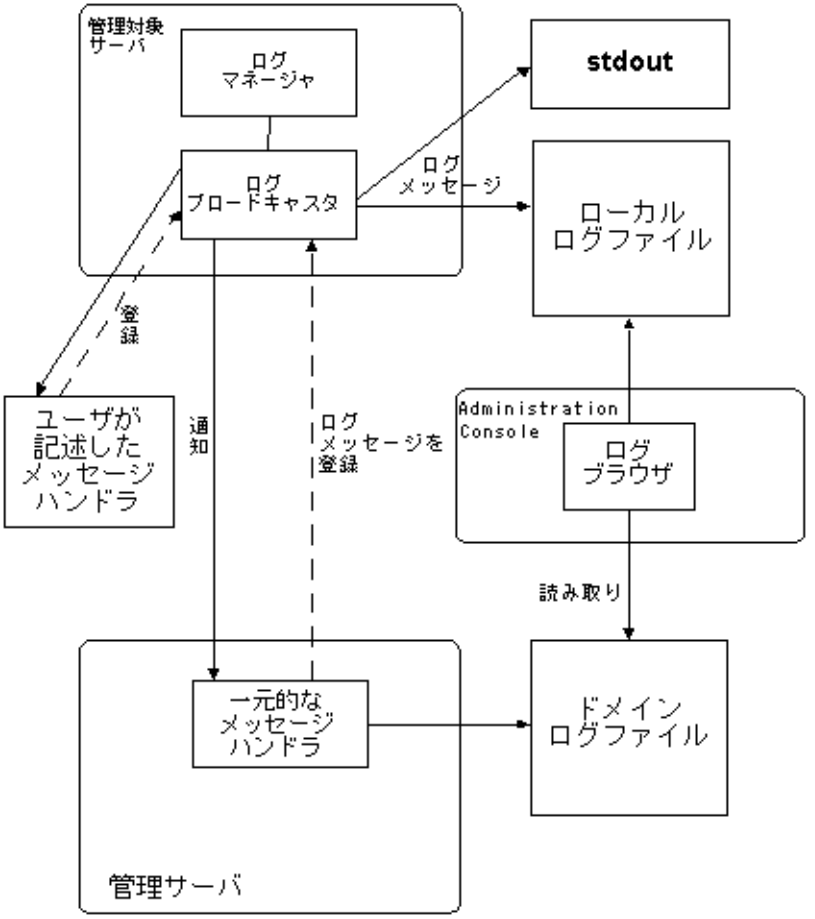
WebLogic サーバが起動すると、ログ メッセージを受信するために管理サーバのメッセージ ハンドラがそのサーバに登録されます。登録時には、管理サーバに転送するメッセージを選択するためにローカル サーバで使用されるフィルタ（ユーザによる修正が可能）が提供されます。それらのメッセージは、ドメイン ログに収集されます。

デフォルトでは、最も重要なログ メッセージだけがローカル サーバからドメイン ログに転送されます（この章の「メッセージの重要度」を参照）。ドメイン ログを参照すると、最も重要なメッセージだけに注目してドメイン全体を調べることができます。

ローカル サーバから受信するメッセージを変更するためにフィルタを修正する必要がある場合は、Administration Console を使用して動的に修正できます。変更を有効にするためにローカル サーバを再起動する必要はありません。詳細については、この章の「ドメイン ログ フィルタの作成」を参照してください。

WebLogic Server に登録できるカスタム メッセージ ハンドラをビルドして、JMX 通知を通じてログ メッセージを受信することもできます。

図 6-1　WebLogic Server のロギング サブシステム



ローカル サーバのログ ファイル

6.0 より前のバージョンの WebLogic Server では、ログ ファイルが最大のサイズに達すると新しいログ ファイルが作成されます。このようなログ ファイルの自動作成は、ログ ローテーションと呼ばれます。現在のリリースでは、サイズまたは時間に基づいてログ ファイルをローテーションできます。ローテーションをコンフィグレーションするには、Administration Console を開いて次の操作を行います。

1. 左ペインでサーバを選択します。
2. 右ペインで、[ログ｜ローテーション] を選択します。
3. [ローテーション タイプ] フィールドで時間またはサイズを選択します。

   このフィールドの値が [なし] の場合、ログ ローテーションは行われません。時間でログ ファイルをローテーションする場合は、指定された時間（[ファイル ローテーション間隔]）が経過すると新しいログ ファイルが作成されます。

デフォルトでは、ローカル サーバのログ ファイルは *weblogic*.log という名前が付けられ、WebLogic Server が起動されたディレクトリに作成されます。ファイル名は、サーバの [ログ｜一般] ページでも設定できます。

ローテーションされたファイルを蓄積できる最大数を指定するには、[ファイル数] フィールドで適切な値を設定します。ログ ファイル数がその値に達すると、ログ ファイルがローテーションされるたびに一番古いログ ファイルが削除されます。ローテーションされるログ ファイルには、*filename*nnnnn のように作成順の番号が付けられます。*filename* は、ログ ファイルにコンフィグレーションされている名前です。たとえば、weblogic.log00007 です。

ローカル サーバのログには、常に、記録されたすべてのメッセージが格納されています。

ローカル サーバによるロギングのコンフィグレーションでは、どのメッセージが stdout に記録されるのかを指定することもできます。記録される最低の重要度を指定すると、重要度の低いメッセージを除外することができます。デバッグ メッセージの stdout への記録を有効または無効にすることもできます。

注意： サーバ ログ ファイルは、手作業の編集で修正しないようにしてください。時間によるログのローテーションは、ファイルのタイムスタンプに基づきます。ファイルを修正すると、タイムスタンプが変更されてログのローテーションが混乱する可能性があります。手作業でファイルを編集すると、ファイルがロックされて、サーバによるファイルの更新が妨げられる可能性があります。

クライアントのロギング

WebLogic のロギング機能を利用する Java クライアントでもログ メッセージが生成される場合があります。ただし、クライアントによって記録されるメッセージはドメイン ログには転送されません。クライアントのロギング プロパティをコンフィグレーションするには、次のコマンドラインで適切な引数を入力します。

`-Dweblogic.log.`*`attribute`*`=`*`value`*

attribute は、LogMBean 属性です。

デフォルトでは、クライアントについてはログ ファイルへの記録は行われず、メッセージは stdout に記録されます。コマンドラインで次の引数を使用すると、ファイルへのロギングを有効にし、ログのファイル名を設定できます。

`-Dweblogic.log.FileName=`*`logfilename`*

logfilename は、クライアント ログ ファイルの名前です。

次のコマンドライン引数もクライアントのロギングで使用できます。

`-Dweblogic.StdoutEnabled=`*`boolean`*

`-Dweblogic.StdoutDebugEnabled=`*`boolean`*

`-Dweblogic.StdoutSeverityLevel = [64 | 32 | 16 | 8 | 4 | 2 | 1 ]`

boolean は、true または false のいずれかです。

ログ ファイル フォーマット

ログ ファイルの各メッセージの先頭行は #### で始まり、その後にメッセージ ヘッダが続きます。メッセージ ヘッダは、メッセージの実行時コンテキストを示します。メッセージの各属性は、山括弧で囲まれます。

メッセージ本文以降の行は例外を記録するメッセージの場合にのみ存在し、例外のスタック トレースを示します。メッセージがトランザクションのコンテキストで記録されたものではない場合、Transaction ID は存在しませんが、Transaction ID のための山括弧（セパレータ）は配置されます。

次に、ログ メッセージの例を示します。

```
####<Jun 2, 2000 10:23:02 AM PDT> <Info> <SSL> <bigbox> <myServer>
    <SSLListenThread> <harry> <> <004500> <Using exportable strength SSL>
```

この例では、メッセージの属性は、Timestamp、Severity、Subsystem、Machine Name、Server Name、Thread ID、User ID、Transaction ID、Message ID、および Message Text です。

注意： クライアントによって記録されるログ メッセージには、Server Name 属性または Thread ID 属性はありません。

注意： ログ ファイルの記述で使用される文字エンコーディングは、ホスト システムのデフォルトの文字エンコーディングです。

メッセージの属性

サーバのログ ファイルに保存される各ログ メッセージでは、次の表にある属性が定義されます。Message Id では、メッセージ カタログに格納された追加属性（Probable Cause や Recommended Action など）とメッセージを関連付けることもできます。

属性	説明
Timestamp	メッセージが発生した時刻と日付。書式はロケールに基づく。
Severity	メッセージで報告されるイベントの影響または深刻さの度合いを示す。「メッセージの重要度」を参照。
Subsystem	メッセージの発生源である WebLogic Server の特定のサブシステムを示す。たとえば EJB、RMI、JMS など。

属性	説明
Server Name Machine Name Thread ID Transaction ID	これら 4 つの属性はメッセージの発生源を識別する。Transaction ID は、トランザクションのコンテキストで記録されたメッセージの場合のみ存在する。 **注意：** Server Name と Thread ID は、Java クライアントで生成され、クライアント ログに記録されるログ メッセージにはない。
User ID	メッセージが生成されたときのセキュリティ コンテキストからのユーザ。
Message ID	ユニークな 6 桁の識別子。499999 までのメッセージ ID は、WebLogic Server のシステム メッセージ用に予約されている。
Message Text	WebLogic Server メッセージの場合は、システムのメッセージ カタログで定義されている短い説明（この章の「メッセージ カタログ」を参照）。他のメッセージの場合は、プログラムの開発者によって定義されたテキスト。

メッセージ カタログ

ログ メッセージに含まれる情報に加えて、WebLogic Server システム コンポーネント（またはユーザが記述したコード）で生成されるメッセージには、メッセージ カタログに格納された定義済みの追加情報も含まれます。メッセージ カタログに格納されている追加属性は以下のとおりです。

属性	説明
Message Body	報告される状況についての短いテキストによる説明。これはメッセージの Message Text と同じ。
Message Detail	メッセージで報告される状況についてのより詳しい説明。
Probable Cause	なぜメッセージが記録されたのかについての説明。メッセージで報告される状況の考えられる原因。

属性	説明
Recommended Action	メッセージで報告される状況を解決または回避するために管理者が行うべきアクション。

これらの追加メッセージ属性には、Administration Console のログ ビューからアクセスできます。

メッセージの重要度

WebLogic Server ログ メッセージには、**重要度**という属性があります。この属性は、メッセージで報告されるイベントまたは状況のユーザに対する重要度または影響を示します。

定義されている重要度は以下のとおりです。重要度は、`Emergency` を最高とした重要度の順で示されています。

Severity	デフォルトでドメイン ログに転送されるかどうか	意味
Informational	転送されない	通常の処理を報告する。
Warning	転送されない	不審な処理またはコンフィグレーションが行われたが、通常の処理に影響する恐れはない。
Error	転送される	ユーザ エラーが発生した。システムまたはアプリケーションでは、サービスの中断や停止なしでエラーに対処できる。
Notice	転送される	警告メッセージ。サーバの通常の処理に影響する恐れのない不審な処理またはコンフィグレーションが行われた。

Severity	デフォルトでドメイン ログに転送されるかどうか	意味
Critical	転送される	システム エラーまたはサービス エラーが発生した。システムは回復できるが、サービスが一時的に停止するか、永続的に停止する恐れがある。
Alert	転送される	システムの特定のサービスだけが使用不能の状態にある。自動回復できないので、管理者が直ちに問題を解決する必要がある。
Emergency	転送される	サーバが使用不能な状態であることを示す。深刻なシステム障害または危機的状態を示す。

デバッグ メッセージ

debug という重要度のメッセージは特別なメッセージです。デバッグ メッセージは、ドメイン ログには転送されません。デバッグ メッセージには、アプリケーションまたはサーバについての詳しい情報が含まれる場合があります。デバッグ メッセージは、アプリケーションがデバッグ モードで動作している場合にのみ発生します。

ログ ファイルの参照

Administration Console のログ参照機能を使用すると、以下のことができます。

- サーバのローカル ログ ファイルを表示する
- ドメイン全体のログ ファイルを表示する

ドメイン ログまたはローカル サーバ ログを表示すると、以下のことができます。

- 発生時刻、ユーザ ID、サブシステム、メッセージの重要度、またはメッセージの短い説明に基づいて、表示されるログ メッセージを選択する
- 記録どおりにメッセージを表示するか、または過去のログ メッセージを検索する
- Administration Console に表示されるログ メッセージ属性および属性が表示される順序を選択する

ログの表示

ドメイン ログおよびローカル サーバのログ ファイルには、Administration Console からアクセスできます。アクセス方法については、Console オンライン ヘルプの以下のトピックを参照してください。

- 「ドメイン ログの表示」
- 「サーバのログの表示」

ドメイン ログ フィルタの作成

WebLogic Server によってドメイン ログに転送されるログ メッセージは、デフォルトでは、ローカルで記録されるメッセージの一部です。メッセージの重要度、サブシステム、またはユーザ ID に基づいて、転送されるログ メッセージを選択するログ フィルタをコンフィグレーションできます。デバッグ メッセージは特殊なメッセージであり、ドメイン ログには転送されません。ドメイン ログ フィルタは、ドメイン ログ フィルタ テーブルで作成または修正できます。ドメイン ログ フィルタ テーブルには、ドメインのモニタ タブ ページからアクセスできます。ドメイン ログ フィルタ作成の詳細については、Administration Console のオンライン ヘルプを参照してください。

7　アプリケーションのデプロイメント

以下の節では、アプリケーションとアプリケーション コンポーネントを WebLogic Server にインストールおよびデプロイする方法について説明します。

- デプロイメントのサポート形式
- Administration Console を使用したアプリケーションのデプロイ
- 自動デプロイメント

デプロイメントのサポート形式

J2EE アプリケーションは、エンタープライズ アプリケーション アーカイブ（EAR）ファイルとして、または展開ディレクトリ形式で WebLogic Server にデプロイできます。

ただし、アプリケーションを展開形式でデプロイする場合、Web アプリケーション コンポーネント以外のコンポーネントは展開形式にしないようにすることをお勧めします。アプリケーションをアーカイブ形式でデプロイする場合は、アプリケーションのすべてのコンポーネントをアーカイブ形式にしてください。

アーカイブ コンポーネントは、EJB アーカイブ（JAR）ファイル、Web アプリケーション アーカイブ（WAR）ファイル、またはリソース アダプタ アーカイブ（RAR）ファイルとしてパッケージ化されます。

Web アプリケーションの詳細については、「WebLogic Server Web コンポーネントのコンフィグレーション」を参照してください。

リソース アダプタ コンポーネントの詳細については、「WebLogic J2EE コネクタ アーキテクチャの管理」を参照してください。

Administration Console を使用したアプリケーションのデプロイ

Administration Console を使用すると、アプリケーションまたはアプリケーション コンポーネント（EJB JAR ファイルなど）をインストールおよびデプロイし、アプリケーション コンポーネントのインスタンスを対象となる WebLogic Server にデプロイできます。この作業を行うには、次の手順に従います。

手順 1: アプリケーションのコンフィグレーションとデプロイ

そのためには、次の操作を行います。

1. [デプロイメント | アプリケーション] を選択して、アプリケーション テーブルを開きます。
2. [新しい Application のコンフィグレーション] リンクをクリックして、[新しい Application の作成] ページを開きます。
3. アプリケーションのこのコンフィグレーション エントリについて、以下の情報をフィールドに入力します。
   - アプリケーション エントリの名前
   - アプリケーション（EAR ファイル）のパス
   - このアプリケーションがデプロイされるのかどうか
4. [作成] をクリックして、新しいエントリを作成します。

Administration Console を使用してアプリケーション（またはアプリケーション コンポーネント）をインストールすると、ドメインのコンフィグレーション ファイル（`\config\`*`domain_name`*`\config.xml`）にそのアプリケーションとアプリケーション コンポーネントのエントリが作成されます。また、管理サーバは、アプリケーションとアプリケーション コンポーネントのコンフィグレーションとモニタを可能にする JMX Management Bean （MBean）も生成します。

手順 2: アプリケーション コンポーネントのデプロイメント

デプロイ可能なコンポーネントには、Web アプリケーション コンポーネント、EJB、リソース コネクタ コンポーネントの 3 種類があります。

注意： クラスタ内の管理対象サーバにアプリケーション コンポーネント（EJB や、WAR または RAR ファイルなど）をデプロイする場合、そのクラスタ内のすべてのサーバに同じアプリケーション コンポーネントがデプロイされるようにしておく必要があります。そのためには、そのクラスタをデプロイメントの対象として選択します。

Web アプリケーション コンポーネントのデプロイメント

管理対象サーバに Web アプリケーション コンポーネントをデプロイするには、次の操作を行います。

1. [デプロイメント | Web アプリケーション] を選択して、[Web アプリケーション] テーブルを開きます。
2. [新しい Web Application のコンフィグレーション] リンクをクリックして、[新しい WebApp Component の作成] コンフィグレーション ページを開きます。
3. 以下の情報をフィールドに入力します。
   - このコンポーネントのコンフィグレーション エントリの名前。
   - このコンポーネントを示す Universal Resource Identifier（URI）。
   - WAR ファイルのパス、またはディレクトリのパス（Web アプリケーションが展開形式の場合）。
   - デプロイメントの順序を選択します。これで、サーバの起動時に Web アプリケーションがデプロイされる順序が決まります。詳細については、この章の「デプロイ順」を参照してください。
   - このコンポーネントがデプロイされるのかどうかを指定します。
4. [作成] をクリックして、新しいコンポーネント エントリを作成します。

5. コンポーネントのデプロイ先として、管理対象サーバまたはクラスタを選択できます。管理対象サーバにコンポーネントをデプロイする場合は、[対象｜サーバ] をクリックします。クラスタにコンポーネントをデプロイする場合は、[対象｜クラスタ] をクリックします。
6. [選択可] フィールドに管理対象サーバ（[対象｜クラスタ] を選択した場合はクラスタ）のリストが表示されます。矢印ボタンを使用して [選択済み] フィールドに移動することで、Web アプリケーションをデプロイする管理対象サーバ（またはクラスタ）を選択します。[適用] をクリックして変更を有効にします。

Web アプリケーションのコンフィグレーションの詳細については、「WebLogic Server Web コンポーネントのコンフィグレーション」を参照してください。

EJB コンポーネントのデプロイメント

管理対象サーバに EJB をデプロイするには、次の操作を行います。

1. [デプロイメント｜ EJB] を選択して、[EJB デプロイメント] テーブルを開きます。
2. [新しい EJB のコンフィグレーション] リンクをクリックして、[新しい EJB Component の作成] ページを開きます。
3. 以下の情報をフィールドに入力します。
   - このコンポーネントのコンフィグレーション エントリの名前。
   - このコンポーネントを示す Universal Resource Identifier（URI）。
   - JAR ファイルのパス。
   - デプロイメントの順序を選択します。これで、サーバの起動時に EJB がデプロイされる順序が決まります。詳細については、この章の「デプロイ順」を参照してください。
   - このコンポーネントがデプロイされるのかどうかを指定します。
4. [作成] をクリックして、新しいコンポーネント エントリを作成します。
5. コンポーネントのデプロイ先として、管理対象サーバまたはクラスタを選択できます。管理対象サーバにコンポーネントをデプロイする場合は、[対象｜サーバ] をクリックします。クラスタにコンポーネントをデプロイする場合は、[対象｜クラスタ] をクリックします。

6. [選択可] フィールドに管理対象サーバ（[対象｜クラスタ] を選択した場合はクラスタ）のリストが表示されます。矢印ボタンを使用して [選択済み] フィールドに移動することで、Web アプリケーションをデプロイする管理対象サーバ（またはクラスタ）を選択します。[適用] をクリックして変更を有効にします。

リソース アダプタ コンポーネントのデプロイメント

管理対象サーバにリソース コネクタ コンポーネントをデプロイするには、次の操作を行います。

1. [デプロイメント｜コネクタ] を選択して、[リソース コネクタ] テーブルを開きます。
2. [新しい Connector Component のコンフィグレーション] リンクをクリックして、[新しい Connector Component の作成] コンフィグレーション ページを開きます。
3. 以下の情報をフィールドに入力します。
   - このコンポーネントのコンフィグレーション エントリの名前。
   - このコンポーネントを示す Universal Resource Identifier（URI）。
   - RAR ファイルのパス。
   - デプロイメントの順序を選択します。これで、サーバの起動時にリソース コネクタがデプロイされる順序が決まります。詳細については、この章の「デプロイ順」を参照してください。
   - このコンポーネントがデプロイされるのかどうかを指定します。
4. [作成] をクリックして、新しいコンポーネント エントリを作成します。
5. コンポーネントのデプロイ先として、管理対象サーバまたはクラスタを選択できます。管理対象サーバにコンポーネントをデプロイする場合は、[対象｜サーバ] をクリックします。クラスタにコンポーネントをデプロイする場合は、[対象｜クラスタ] をクリックします。
6. [選択可] フィールドに管理対象サーバ（[対象｜クラスタ] を選択した場合はクラスタ）のリストが表示されます。矢印ボタンを使用して [選択済み] フィールドに移動することで、Web アプリケーションをデプロイする管理対象サーバ（またはクラスタ）を選択します。[適用] をクリックして変更を有効にします。

リソース コネクタの詳細については、「WebLogic J2EE コネクタ アーキテクチャの管理」を参照してください。

アプリケーションまたはアプリケーション コンポーネント（EAR または WAR ファイル、または EJB JAR ファイルなど）を特定の WebLogic Server にデプロイすると、対象となる WebLogic Server にある `\config\domain_name\applications` の下の `.wlnotdelete` ディレクトリにファイルがコピーされます。WebLogic Administration Service はファイル配布サーブレットを呼び出して、対象サーバにファイルをコピーします。

デプロイ順

EJB などの同じ種類のコンポーネントの中では、サーバの起動時にそれらがデプロイされる順序を指定できます。コンポーネントのデプロイ時に [デプロイ順] フィールドで指定した整数は、同じ種類の他のコンポーネントと相対的な優先順位（EJB のデプロイ順など）を示します。デプロイ順が 0 のコンポーネントは、その種類のコンポーネントの中で一番最初にデプロイされます。

ただし、WebLogic Server は同じ種類のコンポーネント間でのこのユーザ定義の順序によって影響を受けない種類間での順序を使用します。WebLogic Server が起動すると、次に示すクラスレベルの順序がデプロイメントで使用されます。

1. JDBC 接続プール
2. JDBC マルチ プール
3. JDBC データ ソース
4. JDBC トランザクション データ ソース
5. JMS 接続ファクトリ
6. JMS サーバ
7. コネクタ コンポーネント
8. EJB コンポーネント
9. Web アプリケーション コンポーネント

自動デプロイメント

自動デプロイメントは、管理サーバにアプリケーションを迅速にデプロイするための手段です。自動デプロイメントは、アプリケーションをテストするための開発環境でのみ使用してください。プロダクション環境で使用したり、管理対象サーバにコンポーネントをデプロイするために使用することは避けてください。java コマンドラインで次の引数を使用すると、管理サーバの起動時に自動デプロイメントが無効になります。

```
-Dweblogic.ProductionModeEnabled=true
```

対象の WebLogic Server ドメインについて自動デプロイメントが有効な場合は、アプリケーションが WebLogic 管理サーバの \config*domain_name*\applications ディレクトリにコピーされると、その管理サーバは新しいアプリケーションの存在を検出し、それを自動的にデプロイします（管理サーバが動作している場合）。サブディレクトリ *domain_name* は、管理サーバの起動時に使用された WebLogic Server ドメインの名前です。アプリケーションを \applications ディレクトリにコピーしたときに WebLogic Server が稼働していない場合、そのアプリケーションは WebLogic Server が次に起動したときにデプロイされます。

自動デプロイされたアプリケーションのコンフィグレーションを Administration Console で変更した場合、その変更は永続的には格納されません。つまり、アクティブ ドメインの config.xml で定義されているコンフィグレーションは変更されません。自動デプロイされたアプリケーションのコンフィグレーションを変更した場合、その変更は管理サーバを再起動すると失われます。

自動デプロイメントの有効化または無効化

デフォルトでは、自動デプロイメントは有効になっています。

自動デプロイメントが有効になっているかどうかを調べるには、Administration Console を起動して、対象ドメインのドメイン アプリケーション設定ページを開きます（[*domain_name* | コンフィグレーション | アプリケーション]）。このページでは、自動デプロイメントを有効または無効にすることができ、また WebLogic Server が \applications サブディレクトリに新しいアプリケーショ

ンが存在するかどうかをチェックする間隔（単位：ミリ秒）を設定できます。デフォルトでは、自動デプロイメントをオンにすると、管理サーバは \applications ディレクトリ内の変更を 3 秒ごとにチェックします。

展開ディレクトリ形式によるアプリケーションの自動デプロイメント

アプリケーションまたはアプリケーション コンポーネントの自動デプロイメントは、展開ディレクトリ形式で行うか、エンタープライズ アプリケーション アーカイブ（EAR）ファイル、Web アプリケーション アーカイブ（WAR）ファイル、または Java アーカイブ（JAR）ファイルにパッケージングして行います。

アプリケーションを展開形式で動的にデプロイするには、次の手順に従います。

1. 展開されたアプリケーション用に作成したディレクトリ名が、アプリケーションのコンテキスト パスと同じであることを確認します。
2. このサブディレクトリを、\config*domain_name*\applications の下にコピーします。ここで *domain_name* は、アプリケーションのデプロイ先ドメイン名です。自動デプロイメントが有効にされている場合、これでアプリケーションが自動的にデプロイされます。

自動デプロイメント アプリケーションのアンデプロイメントと再デプロイメント

自動デプロイされたアプリケーションまたはアプリケーション コンポーネントは、サーバの稼働時に動的に再デプロイできます。これは、デプロイしたアプリケーションまたはアプリケーション コンポーネントを WebLogic 管理サーバを停止および再起動せずに更新する場合に便利です。JAR、WAR、または EAR ファイルを動的に再デプロイするには、このファイルの新バージョンを、\applications ディレクトリ内の既存のファイルに上書きコピーするだけです。

この機能を使用すると、開発者はメイクファイルの最後のステップとして \applications ディレクトリへのコピーを追加して、サーバを更新できます。

展開形式で自動デプロイされたアプリケーションの再デプロイメント

展開形式で自動デプロイされたアプリケーションまたはアプリケーション コンポーネントも、動的に再デプロイできます。アプリケーションが展開形式でデプロイされている場合、管理サーバは、展開されたアプリケーションのディレクトリ内で REDEPLY というファイルを定期的に検索します。このファイルのタイムスタンプが変更されている場合、管理サーバは展開ディレクトリを再デプロイします。

展開されたアプリケーション ディレクトリ内のファイルを更新する場合は、次の手順に従います。

1. 展開されたアプリケーションを最初にデプロイするときに、展開されたアプリケーションを格納するディレクトリに REDEPLOY という名前の空のファイルを作成します。
2. 展開されたアプリケーションを更新するには、更新されたファイルをそのディレクトリ内の既存のファイルに上書きコピーします。
3. 新しいファイルをコピーしたら、展開ディレクトリ内の REDEPLOY ファイルのタイプスタンプを更新します。

管理サーバは、タイムスタンプの変更を検出すると、展開ディレクトリのコンテンツを再デプロイします。

8 WebLogic Server Web コンポーネントのコンフィグレーション

以下の節では、WebLogic Server Web コンポーネントをコンフィグレーションする方法について説明します。

- 8-2 ページの「概要」
- 8-2 ページの「HTTP パラメータ」
- 8-4 ページの「リスンポートのコンフィグレーション」
- 8-4 ページの「Web アプリケーション」
- 8-7 ページの「仮想ホスティングのコンフィグレーション」
- 8-11 ページの「WebLogic Server による HTTP リクエストの解決方法」
- 8-14 ページの「HTTP アクセス ログの設定」
- 8-25 ページの「POST サービス拒否攻撃の防止」
- 8-26 ページの「HTTP トンネリングのための WebLogic Server の設定」
- 8-28 ページの「静的ファイルを提供するネイティブ I/O の使用（Windows のみ）」

概要

WebLogic Server は、動的な Java ベース分散アプリケーションのホストとなる他にも、大容量 Web サイトを処理できる高機能 Web サーバとして、HTML ファイルや画像ファイルなどの静的ファイル、およびサーブレットと JavaServer Pages（JSP）を提供します。WebLogic Server は、HTTP 1.1 規格をサポートしています。

HTTP パラメータ

サーバまたは仮想ホストごとに、Administration Console を使用して HTTP 操作パラメータをコンフィグレーションできます。

属性	説明	指定できる値	デフォルト値
[デフォルト サーバ名]	WebLogic Server がリクエストをリダイレクトするときには、[デフォルト サーバ名] で指定された文字列を使用して HTTP 応答ヘッダで返されるホスト名が設定される。 ファイアウォールまたはロード バランサを使用し、ブラウザからのリダイレクト リクエストで元のリクエストで送信された同じホスト名を参照するようにしたい場合に便利。	文字列	Null
[Keep Alive を有効化]	HTTP キープアライブが有効かどうかを設定する。	ブール True = 有効 False = 無効	選択
[Send Server Header を有効化]	false の場合は、サーバ名が HTTP 応答で送信されない。ヘッダのスペースが限られている無線アプリケーションで便利。	ブール True = 有効 False = 無効	True

属性	説明	指定できる値	デフォルト値
[持続時間] ([仮想ホスト] パネルでは [Keep Alive 時間] と表示)	非アクティブな HTTP 接続を閉じるまで WebLogic Server が待機する秒数。	整数	30
[HTTPS 持続時間] ([仮想ホスト] パネルでは [Https Keep Alive 時間] と表示)	非アクティブな HTTPS 接続を閉じるまで WebLogic Server が待機する秒数。	整数	60
[WAP 有効化]	選択すると、セッション ID に JVM 情報が含まれなくなる。これは、URL のサイズを 128 文字に制限する WAP デバイスで URL 書き換えを使用する場合に必要になる。[WAP 有効化] を選択すると、クラスタのレプリケート セッションの使用に影響する場合がある。	有効 無効	無効
[POST タイムアウト秒]	HTTP POST データに含まれる大量のデータを WebLogic Server が受信する際のタイムアウト（単位 : 秒）を設定する。これは、POST データを使用してサーバを過負荷状態にしようとするサービス拒否攻撃を防ぐために使用する。	整数	0
[最大 POST 時間]	HTTP POST データに含まれる大量のデータを WebLogic Server が待ち受ける時間（単位 : 秒）を設定する。	整数	0
[最大 POST サイズ]	HTTP POST データに含まれるデータの最大サイズを設定する。	整数	0

属性	説明	指定できる値	デフォルト値
[外部 DNS 名]	クラスタ化した WebLogic Server と Netscape（プロキシ）プラグインなど Web サーバ フロントエンドのプラグインとの間にアドレス変換ファイアウォールを配置したシステムの場合、この属性を、プラグインがこのサーバとの通信に使用するアドレスに設定する。		

リスンポートのコンフィグレーション

各 WebLogic Server が HTTP リクエストをリスンするポートを指定できます。任意の有効なポート番号を指定できますが、ポート 80 を指定した場合、HTTP を介してリソースにアクセスするために使用する HTTP リクエストからポート番号を省略できます。たとえば、リスン ポートとしてポート 80 を定義した場合、`http://hostname:portnumber/myfile.html` ではなく、`http://hostname/myfile.html` という形式を使用できます。

リスン ポートは、通常のリクエストとセキュアな（SSL を使用した）リクエストで別個に定義します。通常のリスン ポートは Administration Console の サーバ ノードの [コンフィグレーション｜一般] タブで定義し、SSL リスン ポートは [コンフィグレーション｜ SSL] タブで定義します。

Web アプリケーション

HTTP サービスと Web サービスは、Sun Microsystems のサーブレット仕様 2.2 に従ってデプロイされます。この仕様では、Web アプリケーション とは Web ベース アプリケーションのコンポーネントを 1 つにまとめるための標準化された方法であると定義されています。これらのコンポーネントには、JSP ページ、HTTP サーブレット 静的リソース（HTML ページや画像ファイルなど）が含まれます。また Web アプリケーションは、エンタープライズ EJB や JSP タグ ライ

ブラリなどの外部リソースにアクセスすることもできます。各サーバは、任意の数の Web アプリケーションのホストになることができます。通常、Web アプリケーションの名前は、その Web アプリケーションのリソースを要求するために使う URI の一部として使用します。

詳細については、『Web アプリケーションのアセンブルとコンフィグレーション』（http://edocs.beasys.co.jp/e-docs/wls61/webapp/index.html）を参照してください。

Web アプリケーションとクラスタ化

Web アプリケーションは、WebLogic Server のクラスタにデプロイできます。ユーザが Web アプリケーションのリソースを要求すると、そのリクエストはその Web アプリケーションがホストするクラスタの構成サーバの 1 つに転送されます。アプリケーションがセッション オブジェクトを使用する場合、そのセッションはクラスタ内の全サーバにレプリケートされなければなりません。セッションのレプリケートにはいくつかの方法があります。

詳細については、
http://edocs.beasys.co.jp/e-docs/wls61/cluster/index.html の
『WebLogic Server Clusters ユーザーズ ガイド』を参照してください。

デフォルト Web アプリケーションの指定

ドメイン内のすべてのサーバおよび仮想ホストで、デフォルト Web アプリケーションを宣言できます。デフォルト Web アプリケーションは、デプロイされている別の Web アプリケーションによって解決できない任意の HTTP リクエストに応答します。他のすべての Web アプリケーションとは異なり、デフォルト Web アプリケーションの名前は、URI の一部として使用されません。サーバまたは仮想ホストに割り当てられた Web アプリケーションを、デフォルト Web アプリケーションとして宣言することができます（Web アプリケーションの割り当てについては、この節で後述します。仮想ホストの詳細については、8-7 ページの「仮想ホスティングのコンフィグレーション」を参照してください）。

デフォルト ドメイン、および WebLogic Server に付属のサンプル ドメインでは、それぞれデフォルトの Web アプリケーションがすでにコンフィグレーションされています。それらのドメインのデフォルト Web アプリケーションは、DefaultWebApp という名前で各ドメインの applications ディレクトリに配置されています。

正常にデプロイされていないデフォルト Web アプリケーションを宣言すると、エラーがログに記録されるとともに、そのデフォルト Web アプリケーションにアクセスしようとしたユーザに対して HTTP 400 エラー メッセージが表示されます。

たとえば、shopping という Web アプリケーションが存在する場合、その Web アプリケーションの cart.jsp という JSP にアクセスするには、次の URL を使用します。

```
http://host:port/shopping/cart.jsp
```

しかし、shopping をデフォルト Web アプリケーションとして指定した場合、cart.jsp にアクセスするには次の URL を使用します。

```
http://host:port/cart.jsp
```

（host は WebLogic Server が稼働するマシンのホスト名、port は WebLogic Server がリクエストをリスンするポートの番号）

サーバまたは仮想ホストのデフォルト Web アプリケーションを宣言するには、Administration Console を使用して、次の手順を実行します。

1. 左ペインで [Web アプリケーション] ノードを展開します。
2. Web アプリケーションを選択します。
3. 右ペインで、[対象] タブを選択します。
4. [サーバ] タブを選択して、サーバ（または仮想ホスト）を [選択済み] カラムへ移動します。([クラスタ] タブを選択し、クラスタを [選択済み] カラムへ移動して、クラスタ内の全サーバを割り当てることもできます)。
5. [適用] をクリックします。
6. 左ペインの [サーバ]（または [仮想ホスト]）ノードを展開します。
7. 該当するサーバまたは仮想ホストを選択します。
8. 右ペインの [一般] タブを選択します。

9. [HTTP] タブを選択します。仮想ホストをコンフィグレーションする場合は、代わりに [一般] タブを選択します。
10. [デフォルト Web アプリケーション] ドロップダウン リストから Web アプリケーションを選択します。
11. [適用] をクリックします。
12. 複数の管理対象サーバのデフォルト Web アプリケーションを宣言する場合、各管理対象サーバについてこの手順を繰り返します。

仮想ホスティングのコンフィグレーション

仮想ホスティングを使用すると、サーバまたはクラスタが応答するホスト名を定義できます。仮想ホスティングを使用するときは、WebLogic Server またはクラスタの IP アドレスにマップする 1 つまたは複数のホスト名を、DNS を使って指定します。また、仮想ホストによって提供される Web アプリケーションを指定します。仮想ホスティングをクラスタ内で使用する場合、ロード バランシング機能により、DNS ホスト名の 1 つが他のホスト名より多くのリクエストを処理する場合でもハードウェアを最も効率的に使用できます。

たとえば、`books` という Web アプリケーションが仮想ホスト名 `www.books.com` のリクエストに応答し、これらのリクエストが WebLogic Server A、B、および C に向けられるよう指定し、一方、`cars` という Web アプリケーションが仮想ホスト名 `www.autos.com` に応答し、これらのリクエストが WebLogic Server D および E に向けられるよう指定できます。アプリケーションと Web サーバの条件に合わせて、仮想ホスト、WebLogic Server、クラスタ、および Web アプリケーションのさまざまな組み合わせをコンフィグレーションできます。

また、定義した各仮想ホストに対して、個別に HTTP パラメータと HTTP アクセス ログを定義できます。仮想ホストに対して設定された HTTP パラメータとアクセス ログは、サーバ に対して設定された HTTP パラメータとアクセス ログをオーバーライドします。指定できる仮想ホストの数に制限はありません。

仮想ホスティングをアクティブ化するには、仮想ホストをサーバまたはサーバ クラスタに割り当てます。クラスタに割り当てられた仮想ホスティングは、そのクラスタ内のすべてのサーバに適用されます。

仮想ホスティングとデフォルト Web アプリケーション

各仮想ホストに対して、デフォルト Web アプリケーション を指定することもできます。仮想ホストのデフォルト Web アプリケーションは、同じサーバまたはクラスタで仮想ホストとしてデプロイされている別の Web アプリケーションに解決できないすべてのリクエストに応答します。

他の Web アプリケーションとは異なり、デフォルト Web アプリケーションの名前（コンテキスト パスとも言う）は、そのデフォルト Web アプリケーションのリソースにアクセスするために使う URI の一部として使用されません。

たとえば、`www.mystore.com` という仮想ホスト名を定義し、`shopping` という Web アプリケーションをデプロイしたサーバにその仮想ホストを割り当てた場合、`shopping` の `cart.jsp` という JSP にアクセスするには、次の URI を使用します。

```
http://www.mystore.com/shopping/cart.jsp
```

しかし、`shopping` をこの仮想ホスト `www.mystore.com` のデフォルト Web アプリケーションとして指定した場合は、次の URI を使用して `cart.jsp` にアクセスします。

```
http://www.mystore.com/cart.jsp
```

詳細については、8-11 ページの「WebLogic Server による HTTP リクエストの解決方法」を参照してください。

仮想ホストの設定

仮想ホストを定義するには、Administration Console を使用して次の手順を実行します。

1. 仮想ホストを作成します。
   a. 左ペインの [サービス] ノードを展開します。ノードが展開され、サービスのリストが表示されます。
   b. 仮想ホスト ノードをクリックします。仮想ホストが定義されている場合、ノードが展開されて仮想ホストのリストが表示されます。

c. 右ペインの [新しい Virtual Host のコンフィグレーション] をクリックします。

d. この仮想ホストを表す名前を入力します。

e. 仮想ホスト名を 1 行に 1 つずつ入力します。これらの仮想ホスト名に一致するリクエストだけが、この仮想ホストとして指定された WebLogic Server またはクラスタによって処理されます。

f. （省略可能）この仮想ホストに対して、デフォルト Web アプリケーションを割り当てます。

g. [作成] をクリックします。

2. ロギングと HTTP パラメータを定義します。

a. （省略可能）[ログ] タブをクリックし、HTTP アクセス ログ属性を入力します（詳細については、8-14 ページの「HTTP アクセス ログの設定」を参照）。

b. [HTTP] タブを選択し、HTTP パラメータを入力します。

3. この仮想ホストに応答するサーバを定義します。

a. [対象] タブを選択します。

b. [サーバ] タブを選択します。使用可能なサーバのリストが表示されます。

c. [選択可] カラム内のサーバを選択し、右矢印ボタンを使ってサーバを [選択済み] カラムに移動します。

4. この仮想ホストに応答するクラスタを定義します（オプション）。すでに WebLogic Cluster が定義されている必要があります。詳細については、`http://edocs.beasys.co.jp/e-docs/wls61/cluster/index.html` の『WebLogic Server Clusters ユーザーズ ガイド』を参照してください。

a. [対象] タブを選択します。

b. [クラスタ] タブを選択します。使用可能なサーバのリストが表示されます。

c. [選択可] カラム内のクラスタを選択し、右矢印ボタンを使ってクラスタを [選択済み] カラムに移動します。仮想ホストは、クラスタ内のすべてのサーバに適用されます。

5. この仮想ホストの対象 Web アプリケーションを選択します。

a. 左ペインの [Web アプリケーション] ノードをクリックします。

b. ターゲットにする Web アプリケーションを選択します。

c. 右ペインの [対象] タブを選択します。

d. [仮想ホスト] タブを選択します。

e. [選択可] カラム内の仮想ホストを選択し、右矢印ボタンを使って仮想ホストを [選択済み] カラムに移動します。

WebLogic Server による HTTP リクエストの解決方法

WebLogic Server が HTTP リクエストを受信すると、WebLogic Server は、URL のさまざまな部分を解析し、その情報を利用してどの Web アプリケーションとサーバがそのリクエストを処理すべきかを決定することによって、そのリクエストを解決します。以下の例では、Web アプリケーション、仮想ホスト、サーブレット、JSP、および静的ファイルのリクエストのさまざまな組み合わせとその応答を示します。

注意： Web アプリケーションをエンタープライズ アプリケーションの一部としてパッケージ化する場合は、Web アプリケーションへのクエストの解決に使用する代わりの名前を指定できます。詳細については、`http://edocs.beasys.co.jp/e-docs/wls61/webapp/deployment.html#war-ear` の「エンタープライズ アプリケーションの一部としての Web 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。

次の表に、WebLogic Server によって提供される URL とファイルのサンプルを示します。「インデックス ディレクトリのチェック」カラムは、特定のファイルが要求されていない場合にディレクトリ リストを提供するかどうかを指定する [インデックス ディレクトリ] 属性に関するものです。[インデックス ディレクトリ] 属性は、Administration Console の [Web アプリケーション] ノードの [コンフィグレーション | ファイル] タブで設定します。

表 8-1 WebLogic Server による URL の解決例

URL	インデックス ディレクトリのチェック	応答で提供されるファイル
`http://host:port/apples`	変更しない	`apples` Web アプリケーションに定義されているウェルカム ファイル *
`http://host:port/apples`	変更する	`apples` Web アプリケーションの最上位ディレクトリのリスト

表 8-1 WebLogic Server による URL の解決例

URL	インデックス ディレクトリのチェック	応答で提供されるファイル
http://host:port/oranges/naval	関係なし	oranges Web アプリケーション内の /naval という <url-pattern> でマップされているサーブレット サーブレット マッピングでは、いくつか考慮すべきことがある。詳細については、「サーブレットのコンフィグレーション」（http://edocs.beasys.co.jp/e-docs/wls61/webapp/components.html#configuring-servlets）を参照。
http://host:port/naval	関係なし	oranges Web アプリケーション内の /naval という <url-pattern> にマップされているサーブレットがデフォルト Web アプリケーションとして定義されている。 詳細については、「サーブレットのコンフィグレーション」（http://edocs.beasys.co.jp/e-docs/wls61/webapp/components.html#configuring-servlets）を参照。
http://host:port/apples/pie.jsp	関係なし	apples Web アプリケーションの最上位ディレクトリにある pie.jsp
http://host:port	変更する	デフォルト Web アプリケーションの最上位ディレクトリのリスト

表 8-1 WebLogic Server による URL の解決例

URL	インデックス ディレクトリのチェック	応答で提供されるファイル
http://host:port	変更しない	デフォルト Web アプリケーションのウェルカム ファイル *
http://host:port/apples/myfile.html	関係なし	apples Web アプリケーションの最上位ディレクトリにある myfile.html
http://host:port/myfile.html	関係なし	デフォルト Web アプリケーションの最上位ディレクトリにある myfile.html
http://host:port/apples/images/red.gif	関係なし	apples Web アプリケーションの最上位ディレクトリの images サブディレクトリにある red.gif
http://host:port/myFile.html myfile.html が apples Web アプリケーションに存在せず、デフォルト サーブレットが定義されていない場合	関係なし	エラー 404 詳細については、http://edocs.beasys.co.jp/e-docs/wls61/webapp/components.html#error-page の「HTTP エラー応答のカスタマイズ」を参照
http://www.fruit.com/	変更しない	www.fruit.com というホスト名を持つ仮想ホストのデフォルト Web アプリケーションのウェルカム ファイル *
http://www.fruit.com/	変更する	www.fruit.com というホスト名を持つ仮想ホストのデフォルト Web アプリケーションの最上位ディレクトリのリスト

表 8-1 WebLogic Server による URL の解決例

URL	インデックス ディレクトリのチェック	応答で提供されるファイル
http://www.fruit.com/oranges/myfile.html	関係なし	www.fruit.com というホスト名の仮想ホストに関連付けられている oranges Web アプリケーションの myfile.html

* 詳細については、http://edocs.beasys.co.jp/e-docs/wls61/webapp/components.html#welcome_pages の「ウェルカム ページのコンフィグレーション」を参照してください。

HTTP アクセス ログの設定

WebLogic Server は、HTTP トランザクションのログを、共通ログ フォーマットまたは拡張ログ フォーマットのいずれかのフォーマットでテキスト ファイルに保存します。共通ログ フォーマットは、デフォルトの、標準規則に従った形式です。拡張ログ フォーマットでは、記録されている情報をカスタマイズできます。定義した各サーバまたは各仮想ホストに対して、HTTP アクセス ログの性質を定義する属性を設定できます。

ログ ローテーション

ログ ファイルは、そのファイルのサイズ、または指定した時間のいずれかに基づいてローテーションすることができます。これらの 2 つの条件のいずれかが満たされると、現在のアクセス ログ ファイルが閉鎖され、新しいログ ファイルが開始されます。ログ ローテーションを設定しないと、HTTP アクセス ログ ファ

イルは無限に大きくなります。アクセス ログ ファイルの名前には、ローテーションごとに増える数値が入ります。HTTP アクセス ログは、定義した Web Server ごとに保存されます。

Administration Console を使用した HTTP アクセス ログの設定

HTTP アクセス ログを設定するには、Administration Console を使用して、次の手順を実行します (`http://edocs.beasys.co.jp/e-docs/wls61/ConsoleHelp/virtualhost.html` を参照)。

1. 仮想ホストを設定してある場合
   a. 左ペインの [サービス] ノードを選択します。
   b. 仮想ホスト ノードを選択します。ノードが展開され、仮想ホストのリストが表示されます。
   c. 仮想ホストを選択します。

   仮想ホストを設定していない場合
   d. 左ペインの [サーバ] ノードを選択します。ノードが展開され、サーバのリストが表示されます。
   e. サーバを選択します。
   f. [ログ] タブを選択します。
   g. [HTTP] タブを選択します。
2. [ログを有効化] ボックスをチェックします。
3. ログ ファイルの名前を入力します。
4. [フォーマット] ドロップダウン リストから [common] または [extended] を選択します。
5. ローテーション タイプとして [サイズ] または [時間] を選択します。
   - [サイズ] : [ログ バッファ サイズ] パラメータに入力した値を超えたときにログをローテーションします。

- ［時間］：[ローテーション間隔] パラメータに指定した分数を超えたときにログをローテーションします。

6. [ローテーション タイプ] として [サイズ] を選択した場合、[最大ログファイル サイズ] フィールドを保存するログ ファイルの最大バイト数に設定します。
7. [更新間隔] パラメータに、アクセス ログがログ エントリを書き出す間隔を秒数で設定します。
8. [ローテーション タイプ] として [時間] を選択した場合、[ローテーション開始時間] に、ログ ファイルを最初にローテーションする日付を設定します（ローテーション タイプが時間に設定されている場合にだけ有効）。`java.text.SimpleDateFormat` の MM-dd-yyyy-k:mm:ss に従って日付を入力します。`java.text.SimpleDateFormat` クラスの詳細については、Javadoc を参照してください。
9. [ローテーション タイプ] として [時間] を選択した場合、[ローテーション間隔] にログ ファイルのローテーション間隔を設定します。

共通ログ フォーマット

HTTP 情報ログのデフォルト フォーマットは、共通ログ フォーマットです（`http://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format` を参照）。この標準フォーマットのパターンは以下のとおりです。

```
host RFC931 auth_user [day/month/year:hour:minute:second
    UTC_offset] "request" status bytes
```

各値の説明は次のとおりです。

host
: リモート クライアントの DNS 名または IP 番号。

RFC931
: リモート クライアントの IDENTD によって返された情報。WebLogic Server はユーザ識別をサポートしていません。

auth_user
: リモート クライアントが認証用にユーザ ID を送信した場合、そのユーザ名。それ以外の場合は 「-」。

day/month/year:hour:minute:second UTC_offset
: 日、月、年、時間（24 時間形式）、および現地時間と GMT の時差（角括弧で囲まれて示される）。

"request"
: リモート クライアントによって送信された HTTP リクエストの最初の行（二重引用符で囲まれて示される）。

status
: 使用可能な場合、サーバによって返された HTTP ステータス コード。それ以外の場合は「-」。

bytes
: 既知の場合、HTTP ヘッダのコンテンツ長として示されるバイト数（HTTP ヘッダは含まれない）。それ以外の場合は「-」。

拡張ログ フォーマットを使用した HTTP アクセス ログの設定

WebLogic Server は、W3C によって定義された拡張ログ フォーマット、バージョン 1.0 もサポートしています。このフォーマットは新しく登場した規格で、WebLogic Server は、W3C による草案仕様（`www.w3.org/TR/WD-logfile.html`）に準拠しています。最新バージョンは、「W3C Technical Reports and Publications」（`www.w3.org/pub/WWW/TR`）で参照できます。

拡張ログ フォーマットを使用すると、各 HTTP 通信に関する記録情報のタイプと順序を指定できます。拡張ログ フォーマットを有効にするには、Administration Console の [HTTP] タブで、フォーマットを [extended] に設定します（8-15 ページの「Administration Console を使用した HTTP アクセス ログの設定」の手順 4. を参照）。

このフォーマットでは、ログ ファイルに記録される情報のタイプをディレクティブによって指定します。ディレクティブは、実際のログ ファイルに組み込まれます。ディレクティブは、新しい行から「#」という記号で始まります。ログファイルが存在しない場合、デフォルト ディレクティブが記述された新しいログ ファイルが作成されます。しかし、サーバの起動時にログ ファイルがすでに存在する場合、そのファイルの先頭には有効なディレクティブが存在しなければなりません。

Fields ディレクティブの作成

ログ ファイルの最初の行には、そのログ ファイル フォーマットのバージョン番号を示すディレクティブが存在しなければなりません。また、ファイルの先頭の近くには、Fields ディレクティブが存在しなければなりません。

```
#Version: 1.0
#Fields: xxxx xxxx xxxx ...
```

ここで各 *xxxx* は、記録されるデータ フィールドを表します。フィールド タイプは、W3C 仕様に定義されているとおり、単純な識別子として指定されるか、またはプレフィックス - 識別子というフォーマットを取ります。次に例を示します。

```
#Fields:date time cs-method cs-uri
```

この識別子は、HTTP アクセスごとにトランザクションの日付と時間、クライアントが使用したリクエスト メソッド、およびリクエストの URI を記録するようサーバに指示します。各フィールドはスペースによって区切られ、各レコードは新しい行に書き込まれてログ ファイルに追加されます。

注意：ログ ファイル内の #Fields ディレクティブの後には新しい行が続かなければなりません。これは、最初のログ メッセージがディレクティブと同じ行に追加されないようにするためです。

サポートされるフィールド識別子

以下の識別子がサポートされています。プレフィックスは必要ありません。

date
: トランザクションが完了した日付。W3C 仕様で定義されているフィールド タイプは <date>

time
: トランザクションが完了した時間。W3C 仕様で定義されているフィールド タイプは <time>

time-taken
: トランザクションが完了するまでの時間。W3C 仕様で定義されているフィールド タイプは <fixed>

bytes
: 転送されたバイト数。フィールド タイプは <integer>

W3C 仕様で定義されている cached フィールドは、WebLogic Server ではサポートされていません。

以下の識別子はプレフィックスを必要とし、単独では使用できません。ここでは、サポートされている個々のプレフィックスの組み合わせについて説明します。

IP アドレス関連フィールド

これらのフィールドには、リクエストを行ったクライアントまたは応答したサーバのいずれかの IP アドレスとポートが記録されます。W3C 仕様で定義されているフィールド タイプは <address> です。サポートされるプレフィックスは以下のとおりです。

`c-ip`

クライアントの IP アドレス

`s-ip`

サーバの IP アドレス

DNS 関連フィールド

これらのフィールドには、クライアントまたはサーバのドメイン名が記録されます。W3C 仕様で定義されているフィールド タイプは <name> です。サポートされるプレフィックスは以下のとおりです。

`c-dns`

リクエストを送信したクライアントのドメイン名

`s-dns`

リクエストを受信したサーバのドメイン名

`sc-status`

応答のステータス コード。たとえば、(404) は「File not found」というステータスを表します。W3C 仕様で定義されているフィールド タイプは <integer> です。

`sc-comment`

ステータス コードと一緒に返されるコメント（「File not found」など）。このフィールド タイプは <text> です。

`cs-method`

リクエスト メソッド（GET や POST など）。W3C 仕様で定義されているフィールド タイプは <name> です。

`cs-uri`
: 完全なリクエスト URI。W3C 仕様で定義されているフィールド タイプは <uri> です。

`cs-uri-stem`
: URI の基本部分のみ（クエリを省略）。W3C 仕様で定義されているフィールド タイプは <uri> です。

`cs-uri-query`
: URI のクエリ部分のみ。W3C 仕様で定義されているフィールド タイプは <uri> です。

カスタム フィールド識別子の作成

拡張ログ フォーマットを使用する HTTP アクセス ログ ファイルに追加するために、ユーザ定義のフィールドを作成することもできます。カスタム フィールドを作成するには、ELF ログ ファイルで `Fields` ディレクティブを使用してフィールドを指定します。次に、そのフィールドに対応し、必要な出力が生成される Java クラスを作成します。フィールドごとに別々の Java クラスを作成することも、複数のフィールドを出力する Java クラスを作成することもできます。このようなクラスの Java ソースのサンプルをこのマニュアルの中で示します。8-24 ページの「カスタム ELF フィールドを作成する Java クラス」を参照してください。

カスタム フィールドを作成するには、次の手順に従います。

1. 次の形式を使用して、`Fields` ディレクティブにフィールド名を追加します。

   `X-`*`myCustomField`*.

   `myCustomField` は完全修飾クラス名です。

   `Fields` ディレクティブの詳細については、8-18 ページの「Fields ディレクティブの作成」を参照してください。

2. `Fields` ディレクティブで定義したカスタム フィールド（`myCustomField` など）と同じ完全修飾クラス名を持つ Java クラスを作成します。このクラスではカスタム フィールドにロギングする情報を定義します。Java クラスには次のインタフェースを実装する必要があります。

   `weblogic.servlet.logging.CustomELFLogger`

Java クラスでは、logField() メソッドを実装しなければなりません。このメソッドは、HttpAccountingInfo オブジェクトと FormatStringBuffer オブジェクトを引数として取ります。

- HttpAccountingInfo オブジェクトを使用して、HTTP リクエストとカスタム フィールドに出力できる応答データにアクセスします。この情報にアクセスするためのゲッター メソッドが提供されています。get メソッドの完全なリストについては、8-22 ページの「HttpAccountingInfo オブジェクトの get メソッド」を参照してください。
- FormatStringBuffer クラスを使用して、カスタム フィールドのコンテンツを作成します。適切な出力を作成するためのメソッドが提供されています。このメソッドの詳細については、FormatStringBuffer の Javadoc を参照してください (http://edocs.beasys.co.jp/e-docs/wls61/javadocs/weblogic/servlet/logging/FormatStringBuffer.html を参照)。

3. Java クラスをコンパイルして、WebLogic Server の起動に使用される CLASSPATH 文にクラスを追加します。WebLogic Server の起動に使用するスクリプト内の CLASSPATH 文を変更する必要があります。

   注意： このクラスを、展開形式または jar 形式で、Web アプリケーションまたはエンタープライズ アプリケーションの内部に配置しないでください。

4. 拡張ログ フォーマットを使用するように WebLogic Server をコンフィグレーションします。詳細については、8-17 ページの「拡張ログ フォーマットを使用した HTTP アクセス ログの設定」を参照してください。

注意： カスタム フィールドを定義する Java クラスの記述では、システムの処理速度を低下させるようなコードは実行しないでください（たとえば、DBMS へのアクセス、大量の I/O、または ネットワークの呼び出しなど）。HTTP アクセス ログ ファイルのエントリは HTTP リクエストごとに作成されます。

注意： 複数のフィールドを出力する場合は、タブでフィールドを区切ります。フィールドの区切り方およびその他の ELF フォーマットの詳細については、http://www.w3.org/TR/WD-logfile-960221.html の「Extended Log Format」を参照してください。

HttpAccountingInfo オブジェクトの get メソッド

次のメソッドは HTTP リクエストに関するさまざまなデータを返します。これらのメソッドは、javax.servlet.ServletRequest、javax.servlet.http.Http.ServletRequest、および javax.servlet.http.HttpServletResponse のさまざまなメソッドと似ています。

これらのメソッドの詳細については、次の表に示す Java インタフェースの対応するメソッドを参照するか、表内の特定の情報を参照してください。

表 8-2 HttpAccountingInfo のゲッター メソッド

HttpAccountingInfo のメソッド	メソッドに関する情報の参照先
Object getAttribute(String name);	javax.servlet.ServletRequest
Enumeration getAttributeNames();	javax.servlet.ServletRequest
String getCharacterEncoding();	javax.servlet.ServletRequest
int getResponseContentLength();	javax.servlet.ServletResponse.setContentLength() このメソッドは応答のコンテンツ長を取得し、*set*ContentLength() メソッドと共に設定する。
String getContentType();	javax.servlet.ServletRequest
Locale getLocale();	javax.servlet.ServletRequest
Enumeration getLocales();	javax.servlet.ServletRequest
String getParameter(String name);	javax.servlet.ServletRequest
Enumeration getParameterNames();	javax.servlet.ServletRequest
String[] getParameterValues(String name);	javax.servlet.ServletRequest
String getProtocol();	javax.servlet.ServletRequest
String getRemoteAddr();	javax.servlet.ServletRequest
String getRemoteHost();	javax.servlet.ServletRequest
String getScheme();	javax.servlet.ServletRequest

表 8-2 HttpAccountingInfo のゲッター メソッド（続き）

HttpAccountingInfo のメソッド	メソッドに関する情報の参照先
String getServerName();	javax.servlet.ServletRequest
int getServerPort();	javax.servlet.ServletRequest
boolean isSecure();	javax.servlet.ServletRequest
String getAuthType();	javax.servlet.http.Http.ServletRequest
String getContextPath();	javax.servlet.http.Http.ServletRequest
Cookie[] getCookies();	javax.servlet.http.Http.ServletRequest
long getDateHeader(String name);	javax.servlet.http.Http.ServletRequest
String getHeader(String name);	javax.servlet.http.Http.ServletRequest
Enumeration getHeaderNames();	javax.servlet.http.Http.ServletRequest
Enumeration getHeaders(String name);	javax.servlet.http.Http.ServletRequest
int getIntHeader(String name);	javax.servlet.http.Http.ServletRequest
String getMethod();	javax.servlet.http.Http.ServletRequest
String getPathInfo();	javax.servlet.http.Http.ServletRequest
String getPathTranslated();	javax.servlet.http.Http.ServletRequest
String getQueryString();	javax.servlet.http.Http.ServletRequest
String getRemoteUser();	javax.servlet.http.Http.ServletRequest
String getRequestURI();	javax.servlet.http.Http.ServletRequest
String getRequestedSessionId();	javax.servlet.http.Http.ServletRequest
String getServletPath();	javax.servlet.http.Http.ServletRequest
Principal getUserPrincipal();	javax.servlet.http.Http.ServletRequest
boolean isRequestedSessionIdFromCookie();	javax.servlet.http.Http.ServletRequest
boolean isRequestedSessionIdFromURL();	javax.servlet.http.Http.ServletRequest

表 8-2 HttpAccountingInfo のゲッター メソッド（続き）

HttpAccountingInfo のメソッド	メソッドに関する情報の参照先
boolean isRequestedSessionIdFromUrl();	javax.servlet.http.Http.ServletRequest
boolean isRequestedSessionIdValid();	javax.servlet.http.Http.ServletRequest
String getFirstLine();	HTTP リクエストの最初の行を返す。 例 : GET /index.html HTTP/1.0
long getInvokeTime();	サーブレットのサービス メソッドがデータをクライアントへ書き戻すのにかかる時間を返す。
int getResponseStatusCode();	javax.servlet.http.HttpServletResponse
String getResponseHeader(String name);	javax.servlet.http.HttpServletResponse

コード リスト 8-1 カスタム ELF フィールドを作成する Java クラス

```
import weblogic.servlet.logging.CustomELFLogger;
import weblogic.servlet.logging.FormatStringBuffer;
import weblogic.servlet.logging.HttpAccountingInfo;

/* この例では、User-Agent フィールドを
   MyCustomField というカスタム フィールドに出力する
*/

public class MyCustomField implements CustomELFLogger{

public void logField(HttpAccountingInfo metrics,
  FormatStringBuffer buff) {
  buff.appendValueOrDash(metrics.getHeader("User-Agent"));
  }
}
```

POST サービス拒否攻撃の防止

サービス拒否攻撃とは、偽りのリクエストによってサーバを過負荷状態にしようとする悪意ある試みです。一般的な攻撃の 1 つは、HTTP `POST` メソッドで膨大な量のデータを送信するというものです。WebLogic Server では、3 つの属性を設定して、この種の攻撃を防くことができます。3 つの属性は、コンソールの [サーバ] または [仮想ホスト] で設定します。これらの属性を仮想ホストに対して設定した場合、その値は [サーバ] で設定した値をオーバーライドします。

`[Post タイムアウト秒 ]`

HTTP POST に含まれる大量のデータを WebLogic Server が受信する間隔を制限できます。

`[ 最大 Post 時間 ]`

WebLogic Server が POST データを受信するために費やす総時間数を制限します。この制限を超えた場合、`PostTimeoutException` が送出され、次のメッセージがサーバ ログに記録されます。

```
Post time exceeded MaxPostTimeSecs.
```

`MaxPostSize`

単一の POST リクエストで受領するデータのバイト数を制限します。この制限を超えた場合、`MaxPostSizeExceeded` が送出され、次のメッセージがサーバ ログに記録されます。

```
POST size exceeded the parameter MaxPostSize.
```

HTTP エラー コード 413 （Request Entity Too Large）がクライアントに返されます。

クライアントがリスン モードの場合、クライアントはこれらのメッセージを取得します。クライアントがリスン モードでない場合は、接続は切断されます。

HTTP トンネリングのための WebLogic Server の設定

HTTP トンネリングとは、HTTP プロトコルしか使用できないときに、WebLogic Server と Java クライアントの間にステートフルなソケット接続をシミュレートするための手段です。HTTP トンネリングは、通常セキュリティ ファイアウォール内の HTTP ポートを「トンネリング」するために使用されます。HTTP はステートレスなプロトコルですが、WebLogic Server はトンネリング機能を提供して接続を通常の T3Connection のように見せかけます。しかし、通常のソケット接続に比べてパフォーマンスが若干低下する場合があります。

HTTP トンネリング接続の設定

HTTP プロトコルでは、クライアントはリクエストを送信し、サーバから応答を受信することしかできません。一方、サーバも自主的にクライアントと通信できません。つまり、HTTP プロトコルはステートレスであり、連続的な双方向接続を行うことができません。

WebLogic HTTP トンネリングは、HTTP プロトコルを通して T3Connection をシミュレートすることによって、こうした制限を乗り越えます。トンネリング接続を調整してパフォーマンスを向上させるには、Administration Console で 2 つの属性を設定します。これらの属性にアクセスするには、[サーバ] の [コンフィグレーション | チューニング] タブを開きます。接続に関する問題が発生しない限り、これらの属性はデフォルトのままにしておくことをお勧めします。これらの属性は、クライアント接続が有効かどうか、またはクライアントが生存しているかどうかをサーバが調べるために使用されます。

[トンネリングを有効化]
: HTTP トンネリングを有効または無効にします。HTTP トンネリングはデフォルトでは無効です。

[トンネリング クライアント Ping]
: HTTP トンネリング接続が設定されると、クライアントは自動的にリクエストをサーバに送信し、サーバは自主的にクライアントに応答できるようになります。また、クライアントはリクエストに指示を入れることができますが、この処理はクライアント アプリケーションがサーバと通

信する必要があるかどうかに関係なく発生します。この属性で設定された秒数以内にサーバがクライアントのリクエストに（アプリケーションコードの一部として）応答しない場合、クライアントはその処理を行います。クライアントは応答を受信し、自動的に別のリクエストを即座に送信します。

デフォルトは 45 秒で、有効な範囲は 20 ～ 900 秒です。

[トンネリング クライアント タイムアウト]

クライアントがサーバーに対して（応答に対する）リクエストを最後に送信してから、この属性で設定された秒 数が経過した場合、サーバはクライアントを応答なしと見なして HTTP トンネル接続を終了します。サーバはこの属性によって指定された間隔で経過時間をチェックし、それまでにクライアントからリクエストがあればそれに応答します。

デフォルトは 40 秒で、有効な範囲は 10 ～ 900 秒です。

クライアントからの WebLogic Server への接続

クライアントが WebLogic Server への接続を要求する場合、HTTP トンネリングを使用するために必要なことは URL に HTTP プロトコルを指定することだけです。次に例を示します。

```
Hashtable env = new Hashtable();
env.put(Context.PROVIDER_URL, "http://wlhost:80");
Context ctx = new InitialContext(env);
```

クライアント側では、特殊なタグが `http` プロトコルに付加されます。このため WebLogic Server は、これが通常の HTTP リクエストではなくトンネリング接続であることを認識します。この処理では、アプリケーション コードを変更する必要はありません。

クライアントは、ポートが 80 の場合でも URL にポートを指定しなければなりません。WebLogic Server では HTTP リクエスト用のリスン ポートを任意に設定できますが、ポート 80 を使用するのが最も一般的です。通常、ファイアウォールを介したポート 80 へのリクエストは許可されるからです。

WebLogic Server 用のリスン ポートは、Administration Console の [サーバ] ノードの [コンフィグレーション｜一般] タブで指定します。

静的ファイルを提供するネイティブ I/O の使用（Windows のみ）

Windows NT/2000 上で WebLogic Server を実行する場合、WebLogic Server で Java メソッドを使用する代わりにネイティブ オペレーティング システム呼び出しの TramsmitFile を使用するように指定して、HTML ファイル、テキスト ファイル、および画像ファイルなどの静的ファイルを提供することができます。ネイティブ I/O を使用すると、サイズの大きな静的ファイルを提供するときのパフォーマンスが向上します。

ネイティブ I/O を使用するには、ネイティブ I/O を使用して提供するファイルが含まれている Web アプリケーションの web.xml デプロイメント記述子に 2 つのパラメータを追加します。1 つ目のパラメータ、weblogic.http.nativeIOEnabled を TRUE に設定して、ネイティブ I/O ファイルの提供を有効にします。2 つ目のパラメータ、weblogic.http.minimumNativeFileSize にはネイティブ I/O を使用するファイルの最小サイズを設定します。提供するファイルがこの値より大きい場合にネイティブ I/O が使用されます。このパラメータを指定しない場合、400 バイトの値が使用されます。

通常、ネイティブ I/O では、提供するファイルが大きいほどパフォーマンスが向上します。ただし、WebLogic Server を実行するマシンの負荷が増大すると、この利点は小さくなります。weblogic.http.minimumNativeFileSize の適切な値を見つけるためにテストする必要があります。

以下の例では、web.xml デプロイメント記述子に追加するすべてのエントリを示します。このエントリは、web.xml ファイルで、<distributable> 要素の後、<servlet> 要素の前に配置しなければなりません。

```
<context-param>
  <param-name>weblogic.http.nativeIOEnabled</param-name>
  <param-value>TRUE</param-value>
</context-param>

<context-param>
  <param-name>weblogic.http.minimumNativeFileSize</param-name>
  <param-value>500</param-value>
</context-param>
```

デプロイメント記述子の記述の詳細については、
http://edocs.beasys.co.jp/e-docs/wls61/webapp/webappdeployment.html の「Web プリケーションのデプロイメント記述子の記述」を参照してください。

9 別の HTTP サーバへのリクエストのプロキシ

以下の節では、別の HTTP サーバに HTTP リクエストをプロキシする方法について説明します。

- 9-1 ページの「概要」
- 9-2 ページの「セカンダリ HTTP サーバへのプロキシの設定」
- 9-3 ページの「プロキシ サーブレットのデプロイメント記述子のサンプル」

概要

WebLogic Server をプライマリ Web サーバとして使用する場合、その WebLogic Server が特定のリクエストをセカンダリ HTTP サーバ（Netscape Enterprise Server、Apache、Microsoft Internet Information Server など）に受け渡す、つまりプロキシするようコンフィグレーションできます。プロキシされたリクエストは、特定の URL にリダイレクトされます。また、異なるマシン上の別の Web サーバにプロキシすることもできます。リクエストのプロキシは、受信するリクエストの URL に基づいて行われます。

`HttpProxyServlet`（配布キットの一部として提供）は、WebLogic Server を介して HTTP リクエストを取得し、プロキシ URL にリダイレクトして、その応答を クライアントのブラウザに送信します。プロキシを使用するには、Web アプリケーションでそのプロキシをコンフィグレーションして、リクエストをリダイレクトする WebLogic Server にデプロイします。

セカンダリ HTTP サーバへのプロキシの設定

セカンダリ HTTP サーバのプロキシを設定するには、次の手順に従います。

1. プロキシ サーブレットを Web アプリケーション デプロイメント記述子に登録します（9-3 ページの「ProxyServlet と共に使用する web.xml のサンプル」を参照）。Web アプリケーションは、リクエストに応答する WebLogic Server のデフォルト Web アプリケーションでなければなりません。プロキシ サーブレットのクラス名は、`weblogic.t3.srvr.HttpProxyServlet` です。詳細については、『Web アプリケーションのアセンブルとコンフィグレーション』（`http://edocs.beasys.co.jp/e-docs/wls61/webapp/index.html`）を参照してください。
2. `<param-name>` に `redirectURL` を、`<param-value>` にプロキシされるリクエストのリダイレクト先サーバの URL を指定して、`ProxyServlet` の初期化パラメータを定義します。
3. 定義したサーブレットを、`<url-pattern>` にマップします。特に、プロキシするファイルの拡張子（`*.jsp`、`*.html` など）をマップします。Web アプリケーション デプロイメント記述子 `web.xml` で `<servlet-mapping>` 要素を使用します。

   `<url-pattern>` を「`/`」に設定した場合、WebLogic Server によって解決できないリクエストはすべてリモート サーバにプロキシされます。しかし、拡張子が `*.jsp`、`*.html`、および `*.html` のファイルをプロキシする場合、これらの拡張子もマップしなければなりません。
4. 受信するリクエストをリダイレクトする WebLogic Server に Web アプリケーションをデプロイします。

プロキシ サーブレットのデプロイメント記述子のサンプル

次に、プロキシ サーブレットを使用するための Web アプリケーション デプロイメント記述子のサンプルを示します。

コード リスト 9-1　ProxyServlet と共に使用する web.xml のサンプル

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.
 //DTD Web Application 2,2//EN"
 "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>

<servlet>
  <servlet-name>ProxyServlet</servlet-name>
  <servlet-class>weblogic.t3.srvr.HttpProxyServlet</servlet-class
>

  <init-param>
    <param-name>redirectURL</param-name>
    <param-value>
       tehama1:7736:7737|tehama2:7736:7737|tehama:7736:7737
    </param-value>
  </init-param>

</servlet>

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

<servlet-mapping>
  <servlet-name>ProxyServlet</servlet-name>
  <url-pattern>*.jsp</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>ProxyServlet</servlet-name>
  <url-pattern>*.htm</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>ProxyServlet</servlet-name>
  <url-pattern>*.html</url-pattern>
</servlet-mapping>

</web-app>
```

10 WebLogic クラスタ へのリクエストのプロキシ

以下の節では、WebLogic Server のクラスタに HTTP リクエストをプロキシする方法について説明します。

- 10-1 ページの「概要」
- 10-2 ページの「HttpClusterServlet の設定」
- 10-5 ページの「HttpClusterServlet 用デプロイメント記述子のサンプル」

概要

`HttpClusterServlet` （WebLogic Server 配布キットの一部として提供）は、リクエストを WebLogic Server から WebLogic クラスタ内の別の WebLogic Server にプロキシします。`HttpClusterServlet` は、プロキシされる HTTP リクエストに対するロード バランシングとフェイルオーバを提供します。サーブレットと WebLogic クラスタの詳細については、`http://edocs.beasys.co.jp/e-docs/wls61/cluster/servlet.html` の「HTTP セッション ステートのレプリケーションについて」を参照してください。

HttpClusterServlet の設定

HttpClusterServlet を設定するには、次の手順を実行します。

1. リクエストを WebLogic Server クラスタにプロキシする WebLogic Server インスタンスをコンフィグレーションします。WebLogic Server Administration Console を使用します（Administration Console の使い方については、http://edocs.beasys.co.jp/e-docs/wls61/adminguide/index.html を参照）。

   a. ドメインに新しい Web アプリケーションを作成します。

   b. ドメインに新しいサーバを作成するか、デフォルトを使用します。

   c. 手順 a で作成した Web アプリケーションを、作成したサーバのデフォルト Web アプリケーションとして割り当てます。

2. 手順 1. で作成した Web アプリケーションの Web アプリケーション デプロイメント記述子に、HttpClusterServlet を登録します（10-5 ページの「HttpClusterServlet 用デプロイメント記述子のサンプル」を参照）。Web アプリケーションは、リクエストに応答する WebLogic Server のデフォルト Web アプリケーションでなければなりません。詳細については、8-5 ページの「デフォルト Web アプリケーションの指定」を参照してください。

   HttpClusterServlet のクラス名は、weblogic.servlet.internal.HttpClusterServlet です。HttpClusterServlet 用デプロイメント記述子のサンプルを以下に示します。

3. HttpClusterServlet の適切な初期化パラメータを定義します。初期化パラメータは、Web アプリケーション デプロイメント記述子 web.xml の <init-param> 要素を使って定義します。defaultServers パラメータ、および 表 10-1 「HttpClusterServlet パラメータ」で説明されているその他の適切なパラメータを定義する必要があります。

4. プロキシ サーブレットを、<url-pattern> にマップします。特に、プロキシするファイルの拡張子（*.jsp、*.html など）をマップします。

   <url-pattern> を「/」に設定した場合、WebLogic Server によって解決できないリクエストはすべてリモート サーバにプロキシされます。しかし、拡張子が *.jsp、*.html、および *.html のファイルをプロキシする場合、これらの拡張子もマップしなければなりません。

url-pattern を設定するもう 1 つの方法は、<url-pattern> として /foo などをマップして、pathTrim パラメータを foo に設定することです。こうしておけば、プロキシされる URL から foo が削除されます。

表 10-1 HttpClusterServlet パラメータ

<param-name>	<param-value>	デフォルト値
defaultServers	（必須）リクエストのプロキシ先サーバのホスト名とポート番号のリスト。フォームは次のとおり。 *host1:HTTP_Port:HTTPS_Port*| *host2:HTTP_Port:HTTPS_Port* （*host1* と *host2* はクラスタ内のサーバのホスト名、*HTTP_Port* はホストが HTTP リクエストをリスンするポート、*HTTPS_Port* はホストが HTTP SSL リクエストをリスンするポート。） ホストは | で区切る。 secureProxy パラメータ（secureProxy パラメータを参照）を ON に設定した場合、HTTPS ポートは HttpClusterServlet を実行している WebLogic Server とクラスタ内の WebLogic Server 間で SSL を使用する。HTTPS ポートは、secureProxy が OFF の場合でも常に定義する必要がある。	なし
secureProxy	ON/OFF。ON に設定した場合、HttpClusterServlet と WebLogic Server クラスタ メンバー間で SSL が使用される。	OFF

DebugConfigInfo	ON/OFF。ON に設定した場合、?__WebLogicBridgeConfig リクエスト パラメータをリクエストに追加することによって、HttpClusterServlet にデバッグ情報をクエリできる（注意 : ? の後にはアンダースコア（_）が 2 つある）。セキュリティ上の理由から、プロダクション システムでは DebugConfigInfo パラメータを OFF にしておくことが望ましい。	OFF
connectionTimeout	ソケットが大量のデータを読み込んでから次に読み込むまでの待ち時間（単位はミリ秒）。タイムアウトを経過すると、java.io.InterruptedIOException が送出される。	0 = タイムアウト制限なし
numOfRetries	HttpClusterServlet が失敗した接続を再試行する回数。	5
pathTrim	元の URL の先頭部分から取り除かれる文字列。	なし
trimExt	元の URL の最後から取り除かれるファイル拡張子。	なし
pathPrepend	PathTrim による文字列の削除の後、リクエストが WebLogic Server クラスタ メンバーに転送される前に、元の URL の先頭に付加される文字列。	なし

HttpClusterServlet 用デプロイメント記述子のサンプル

次に、HttpClusterServlet を使用するための Web アプリケーション デプロイメント記述子 web.xml のサンプルを示します。

コード リスト 10-1　HttpClusterServlet と共に使用する web.xml のサンプル

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.
 //DTD Web Application 2,2//EN"
 "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>

<servlet>
  <servlet-name>HttpClusterServlet</servlet-name>
    <servlet-class>
      weblogic.servlet.internal.HttpClusterServlet
    </servlet-class>

  <init-param>
    <param-name>defaultServers</param-name>
    <param-value>
      myserver1:7736:7737|myserver2:7736:7737|myserver:7736:7737
    </param-value>
  </init-param>

  <init-param>
    <param-name>DebugConfigInfo</param-name>
    <param-value>ON</param-value>
  </init-param>

</servlet>

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

<servlet-mapping>
  <servlet-name>HttpClusterServlet</servlet-name>
  <url-pattern>*.jsp</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>HttpClusterServlet</servlet-name>
  <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

```
<servlet-mapping>
  <servlet-name>HttpClusterServlet</servlet-name>
  <url-pattern>*.html</url-pattern>
</servlet-mapping>

</web-app>
```

11 Apache HTTP Server プラグインのインストールとコンフィグレーション

以降の節では、Apache HTTP Server プラグインをインストールおよびコンフィグレーションする方法について説明します。

- 概要
- プラットフォーム サポート
- Apache HTTP Server プラグインのインストール
- Apache HTTP Server プラグイン のコンフィグレーション
- Apache プラグインでの SSL の使用
- SSL-Apache コンフィグレーションに関する問題
- httpd.conf ファイルのテンプレート
- コンフィグレーション ファイルのサンプル
- 接続エラーとクラスタのフェイルオーバ

概要

Apache HTTP Server プラグインを使用すると、Apache HTTP サーバから WebLogic Server へリクエストをプロキシできます。このプラグインは、WebLogic Server の動的な機能を必要とするリクエストを WebLogic Server が処理できるようにすることによって Apache を拡張します。

このプラグインは、Apache サーバが静的ページを提供している環境で使用されることを想定しています。ドキュメント ツリーの他の部分（HTTP サーブレットや JavaServer Pages によって最も適切な状態で生成される動的ページ）は、別のプロセス（おそらく別のホスト）で動作している WebLogic Server に委託されます。それでも、エンド ユーザ（ブラウザ）では、WebLogic Server に委託される HTTP リクエストは同じソースから来ているものと認識されます。

HTTP トンネリングもこのプラグインを通じて機能でき、ブラウザ以外のクライアントが WebLogic Server サービスにアクセスすることを可能にします。

Apache HTTP Server プラグインは、Apache HTTP サーバ内の Apache モジュールとして機能します。Apache モジュールは起動時に Apache サーバによってロードされ、特定の HTTP リクエストがそこに委託されます。Apache モジュールは、HTTP サーブレットと似ていますが、プラットフォームにネイティブなコードで記述されています。

Apache バージョン 1.3.x のキープアライブ接続

Apache HTTP Server プラグインはリクエストごとにソケットを作成し、応答を読み込んでからソケットを閉じます。Apache HTTP サーバは多重処理されるため、WebLogic Server と Apache HTTP Server プラグインの間では接続プールとキープアライブ接続はサポートされていません。

Apache バージョン 2.x のキープアライブ接続

Apache HTTP Server プラグインは、WebLogic Server との接続の再利用可能なプールを使用してパフォーマンスを向上させます。このプラグインは、同じクライアントからの後続リクエストにプール内の同じ接続を再利用することで、

WebLogic Server との間で HTTP 1.1 キープアライブ接続を実装します。接続が 30 秒（またはユーザ定義の時間）を超えて非アクティブな場合、その接続は閉じて、プールに返されます。この機能は、必要に応じて無効にできます。詳細については、C-10 ページの「KeepAliveEnabled」を参照してください。

リクエストのプロキシ

このプラグインは、指定されたコンフィグレーションに基づいてリクエストを WebLogic Server にプロキシします。リクエストは、リクエストの URL（または URL の一部）に基づいてプロキシできます。この方法は、パスによるプロキシ、と呼びます。リクエストのプロキシは、要求されたファイルの MIME タイプに基づいて行うこともできます。さらに、前述の方法を組み合わせて利用することもできます。リクエストが両方の基準に一致する場合、そのリクエストはパスを基準にプロキシされます。リクエストの種類ごとに、プラグインの補足的な動作を定義する追加パラメータを指定することもできます。詳細については、11-9 ページの「Apache HTTP Server プラグイン のコンフィグレーション」を参照してください。

プラットフォーム サポート

Apache HTTP Server プラグインは、Linux、Solaris、および HPUX11 でサポートされています。Apache の特定バージョンのサポートについては、「BEA WebLogic Server プラットフォーム サポート ページ」を参照してください。

Apache HTTP Server プラグインのインストール

Apache HTTP Server プラグインは、Apache HTTP サーバの Apache モジュールとしてインストールします。モジュールは、動的共有オブジェクト（DSO）または静的リンク モジュールとしてインストールされます。静的リンク モジュールとしてのインストールは、Apache バージョン 1.3.x でのみ可能です。この節では、DSO と静的リンク モジュールを区別して説明を行います。

動的共有オブジェクトとしてのインストール

Apache HTTP サーバ プラグイを動的共有オブジェクトとしてインストールするには、次の操作を行います。

1. 使用しているプラットフォーム用の共有オブジェクト ファイルを見つけます。

   Apache プラグインは、Solaris、Linux、HPUX11 の各プラットフォーム上で使用する共有オブジェクト（.so）として提供されます。各共有オブジェクト ファイルは、プラットフォーム、クライアントと Apache の間での SSL の使用と不使用、および SSL 暗号化の強度（通常または 128 ビット）に応じて別々のバージョンとして配布されます。128 ビット バージョンは、128 ビット バージョンの WebLogic Server をインストールする場合のみインストールされます。共有オブジェクト ファイルは、WebLogic Server の以下のディレクトリに配置されています。

   Solaris
   ```
   lib/solaris
   ```

   Linux
   ```
   lib/linux
   ```

   HPUX11
   ```
   lib/hpux11
   ```

   Windows（Apache 2.0 のみ）
   ```
   bin\apache20
   ```

次の表を基準に適切な共有オブジェクトを選択してください。

Apache のバージョン	通常強度の暗号化	128 ビットの暗号化
標準の Apache バージョン 1.x	mod_wl.so	mod_wl128.so
Apache w/ SSL/EAPI バージョン 1.x (Stronghold、modssl など)	mod_wl_ssl.so	mod_wl128_ssl.so
Apache + Raven バージョン 1.x Raven で配布されるパッチによってプラグインが標準の共有オブジェクトと互換性がなくなるので必要となる。	mod_wl_ssl_raven.so	mod_wl128_ssl_raven.so
標準の Apache バージョン 2.x	mod_wl_20.so	mod_wl28_20.so

2. 共有オブジェクトを有効にします。

   Apache HTTP Server プラグインは、Apache 動的共有オブジェクト（DSO）としてインストールされます。Apache の DSO サポートは、mod_so.c というモジュールを基にしています。このモジュールは、mod_wl.so がロードされる前に有効になっている必要があります。提供されるスクリプトを使用して Apache をインストールした場合、mod_so.c はすでに有効になっているはずです。mod_so.c が有効であることを確認するには、次のコマンドを実行します。

   APACHE_HOME\bin\httpd -l

   APACHE_HOME は、Apache HTTP サーバをインストールしたディレクトリです。

   このコマンドを実行すると、すべての有効なモジュールのリストが表示されます。mod_so.c がリストにない場合は、ソース コードから Apache HTTP サーバを構築して、以下のオプションがコンフィグレーションされているようにします。

```
...
--enable-module=so
--enable-rule=SHARED_CORE
...
```

3. Apache ソース ツリーの外で DSO ベースのモジュールを構築する apxs（APache eXtenSion）というサポート プログラムで Apache HTTP Server プラグイン をインストールし、次の行を httpd.conf ファイルに追加します。

   ```
   AddModule mod_so.c
   ```

   WebLogic Server でコマンド シェルを使用して、使用しているプラットフォーム用の共有オブジェクトがあるディレクトリに移動し、次のコマンドを発行して weblogic_module をアクティブにします。Perl がインストールされていないと、この Perl スクリプトは実行できません。

   ```
   perl APACHE_HOME\bin\apxs -i -a -n weblogic mod_wl.so
   ```

   このコマンドでは、mod_wl.so ファイルが *APACHE_HOME*\libexec ディレクトリにコピーされます。また、httpd.conf ファイルに weblogic_module に関する指示が 2 行追加され、このモジュールがアクティブになります。*APACHE_HOME*\conf\httpd.conf ファイルに以下の行が追加されたことを確認します。

   ```
   LoadModule weblogic_module
   AddModule mod_weblogic.c
   ```

   注意： このプロシージャは、Apache 2.0 ベータ製品では異なる場合があります。詳細については、http://httpd.apache.org/docs-2.0/ にある Apache HTTP サーバ バージョン 2.0 のドキュメントを参照してください。

4. 次のコマンドで、*APACHE_HOME*\conf\httpd.conf ファイルの構文を検証します。

   ```
   APACHE_HOME/bin/apachect1 configtest
   ```

   このコマンドの出力は、httpd.conf ファイルのエラーを示します。

5. 11-9 ページの「Apache HTTP Server プラグイン のコンフィグレーション」で説明されているとおりに、Apache httpd.conf コンフィグレーション ファイルで追加パラメータをコンフィグレーションします。httpd.conf ファイルでは、Apache HTTP Server プラグイン の動作をカスタマイズできます。

6. Weblogic Server を起動します。

7. Apache HTTP サーバを起動（コンフィグレーションを変更した場合は再起動）します。
8. ブラウザを開き、Apache サーバの URL +「/weblogic/」を設定して Apache プラグインをテストします。この設定では、WebLogic Server でデフォルト Web アプリケーションとして定義されている、デフォルトの WebLogic Server HTML ページ、ウェルカム ファイル、またはデフォルト サーブレットが開くはずです。次に例を示します。

   ```
   http://myApacheserver.com/weblogic/
   ```

静的リンク モジュールとしてのインストール

Apache HTTP サーバ プラグインを静的リンク モジュールとしてインストールするには、次の操作を行います。

1. 使用しているプラットフォーム用のリンク ライブラリ ファイルを見つけます。

   各ライブラリ ファイルは、プラットフォームおよび SSL 暗号化の強度（通常または 128 ビット）に応じて別々のバージョンとして配布されます。128 ビット バージョンは、128 ビット バージョンの WebLogic Server をインストールする場合のみインストールされます。ライブラリ ファイルは、WebLogic Server の以下のディレクトリに配置されています。

   Solaris
   ```
   lib/solaris
   ```
   Linux
   ```
   lib/linux
   ```
   HPUX11
   ```
   lib/hpux11
   ```

   次の表を基準に適切な共有オブジェクトを選択してください。

Apache のバージョン	通常強度の暗号化	128 ビットの暗号化
標準の Apache バージョン 1.3.x	libweblogic.a	libweblogic128.a

2. 次のコマンドで Apache 配布キットを復元します。

   ```
   tar -xvf apache_1.3.x.tar
   ```

3. 復元した配布キットの中で src/modules ディレクトリに移動します。
4. weblogic という名前のディレクトリを作成します。
5. Makefile.libdir、Makefile.tmpl を WebLogic Server インストール ディレクトリの lib ディレクトリから src/modules/weblogic にコピーします。
6. libweblogic.a（128 ビットのセキュリティを使用する場合は libweblogic128.a）をリンク ライブラリ ファイルの格納されている同じディレクトリ（手順 1. を参照）から src/modules/weblogic にコピーします。
7. 通常の強度の暗号化を使用する場合は、Apache 1.3 のホーム ディレクトリから次のコマンドを実行します。

   ```
   configure --activate-module=src/modules/weblogic/libweblogic.a
   ```

8. 128 ビットの暗号化を使用する場合は、次のコマンドを（1 行で）実行します。

   ```
   configure--activate-module=
    src/modules/weblogic/libweblogic128.a
   ```

9. 次のコマンドを実行します。

   ```
   make
   ```

10. 次のコマンドを実行します。

    ```
    make install
    ```

11. 「動的共有オブジェクトとしてのインストール」の手順 4. 以降を行います。

Apache HTTP Server プラグイン のコンフィグレーション

プラグインをインストールした後は（11-4 ページの「Apache HTTP Server プラグインのインストール」を参照）、httpd.conf ファイルを編集して Apache プラグインをコンフィグレーションします。httpd.conf ファイルを編集して、プラグイン用のネイティブ ライブラリを Apache モジュールとしてロードしなければならないことを Apache Web サーバに通知し、どの要求をそのモジュールで処理しなければならないかを記述します。

httpd.conf ファイルの編集

httpd.conf ファイルを編集して Apache HTTP Server プラグイン をコンフィグレーションするには、次の操作を行います。

1. httpd.conf ファイルを開きます。このファイルは、*APACHE_HOME*\conf\ に配置されています。*APACHE_HOME* は、Apache をインストールしたルート ディレクトリです。
2. apxs ユーティリティを実行したときに以下の 2 行が httpd.conf ファイルに追加されたことを確認します。

   ```
   LoadModule weblogic_module    libexec/mod_wl.so
   AddModule mod_weblogic.c
   ```

3. 以下のいずれかを定義する IfModule ブロックを追加します。

   クラスタ化されていない WebLogic Server の場合

   WebLogicHost および WebLogicPort パラメータ

   WebLogic Server のクラスタの場合

   WebLogicCluster パラメータ

   次に例を示します。

   ```
   <IfModule mod_weblogic.c>
     WebLogicHost myweblogic.server.com
   ```

```
   WebLogicPort 7001
</IfModule>
```

4. MIME タイプを基準にリクエストをプロキシする場合は、MatchExpression 行も IfModule ブロックに追加します。パスを基準にリクエストをプロキシすることもできます。パスを基準としたプロキシは、MIME タイプを基準としたプロキシに優先します。パスのみを基準にリクエストをプロキシしたい場合は、手順 5. に進んでください。

   たとえば、クラスタ化されていない WebLogic Server 用の次の IfModule ブロックでは、MIME タイプが .jsp のすべてのファイルがプロキシされます。

```
<IfModule mod_weblogic.c>
  WebLogicHost myweblogic.server.com
  WebLogicPort 7001
  MatchExpression *.jsp
</IfModule>
```

   複数の MatchExpressions を使用することもできます。次に例を示します。

```
<IfModule mod_weblogic.c>
  WebLogicHost myweblogic.server.com
  WebLogicPort 7001
  MatchExpression *.jsp
  MatchExpression *.xyz
</IfModule>
```

   MIME タイプを基準にして WebLogic Server のクラスタにリクエストをプロキシする場合は、WebLogicHost と WebLogicPort パラメータの代わりに WebLogicCluster パラメータを使用します。次に例を示します。

```
<IfModule mod_weblogic.c>
  WebLogicCluster w1s1.com:7001,w1s2.com:7001,w1s3.com:7001
  MatchExpression *.jsp
  MatchExpression *.xyz
</IfModule>
```

5. パスを基準にリクエストをプロキシする場合は、Location ブロックと SetHandler 文を使用します。SetHandler は、Apache HTTP Server プラグイン モジュールのハンドラを指定します。たとえば、次の Location ブロックでは、URL に /weblogic が含まれているすべてのリクエストがプロキシされます。

```
<Location /weblogic>
  SetHandler weblogic-handler
</Location>
```

6. Apache HTTP Server プラグインの追加パラメータを定義します。

   Apache HTTP Server プラグインは、C-2 ページの「Web サーバ プラグインの一般的なパラメータ」で示されているパラメータを認識します。Apache HTTP Server プラグインの動作を修正するには、以下のいずれかでパラメータを定義します。

   - `Location` ブロック（パスを基準としたプロキシに適用されるパラメータの場合）
   - `IfModule` ブロック（MIME タイプを基準としたプロキシに適用されるパラメータの場合）

httpd.conf ファイルの編集に関する注意事項

- 11-9 ページの「httpd.conf ファイルの編集」のプロシージャの代わりとして、`IfModule` ブロックにインクルードされる `weblogic.conf` という独立したファイルでパラメータを定義できます。このインクルード ファイルを使用すると、コンフィグレーションをモジュール化できます。次に例を示します。

  ```
  <IfModule mod_weblogic.c>
    # パラメータを定義する WebLogic 用コンフィグレーション ファイル
    Include conf/weblogic.conf
  </IfModule>
  ```

  注意： インクルード ファイルでのパラメータの定義は、Apache HTTP Server プラグインと WebLogic Server の間で SSL を使用する場合はサポートされません。

- 各パラメータは、新しい行で入力する必要があります。パラメータとその値の間に「=」を挿入しないでください。次に例を示します。

  ```
  PARAM_1 value1
  PARAM_2 value2
  PARAM_3 value3
  ```

- リクエストが `IfModule` ブロックの `MatchExpression` で指定された MIME タイプと `Location` ブロックで指定されたパスの両方に一致する場合は、`Location` ブロックで指定された動作が優先されます。

- `CookieName` パラメータを定義する場合は、`IfModule` ブロックで定義する必要があります。

- <location> ブロックに加えて <files> ブロックもリクエストの照合に使用し、仮想ホストで Stronghold SSL を使用している場合、MatchExpression は無視され、<files> および <location> ブロックに定義されているルールがリクエストに適用されます。Stronghold を使用していない場合、<location> ブロックに定義されているルールが <files> ブロックの定義をオーバーライドします。
- <VirtualHost> ブロックを使用する場合、各仮想ホストのコンフィグレーション パラメータ（MatchExpression など）をそれぞれの <VirtualHost> ブロックに指定する必要があります。
- <files> ブロックの代わりに、MatchExpression 文を使用することをお勧めします。

Apache プラグインでの SSL の使用

セキュア ソケット レイヤ（SSL）プロトコルを使用すると、Apache HTTP Server プラグインと WebLogic Server の間の接続を保護できます。SSL プロトコルは、Apache HTTP Server プラグインと WebLogic Server の間でやり取りされるデータに機密性と整合性を提供します。また、SSL プロトコルを使用すると、プラグインでは、信頼性のあるプリンシパルに情報が渡されることを確認するために、WebLogic Server に対して自身を認証することができます。

Apache HTTP Server プラグインでは、Apache HTTP Server プラグインと WebLogic Server の接続を保護するために SSL プロトコルが使用されるのかどうかを、（通常はブラウザからの）HTTP リクエストで指定された転送プロトコル（http または https）では判断しません。

HTTP クライアントと Apache HTTP サーバ間では相互 SSL を使用できますが、Apache HTTP サーバと WebLogic Server 間では一方向の SSL が使用されます。

Apache と WebLogic Server 間で相互 SSL を実装するには、次の手順に従います。

1. クライアント証明書を要求するよう Apache HTTP サーバをコンフィグレーションします。証明書は、次のいずれかのリクエスト属性として格納されます。
   - javax.net.ssl.peer_certificates（weblogic.security.X509Certificate 証明書を返す）

- `java.security.cert.X509Certificate`
  (`java.security.cert.X509` 証明書を返す)

2. 次のようにリクエスト属性を読み出して、証明書にアクセスします。

```
request.getAttribute("javax.net.ssl.peer_certificates");
```

3. `weblogic.security.acl.certAuthenticator.authenticate()` メソッドを使用してユーザを認証します。

Apache HTTP Server プラグインと WebLogic Server の間の SSL のコンフィグレーション

Apache HTTP Server プラグインと WebLogic Server の間で SSL プロトコルを使用するには、次の操作を行います。

1. SSL 向けに WebLogic Server をコンフィグレーションします。詳細については、14-42 ページの「SSL プロトコルのコンフィグレーション」を参照してください。
2. WebLogic Server の SSL リスン ポートをコンフィグレーションします。詳細については、8-4 ページの「リスンポートのコンフィグレーション」を参照してください。
3. `httpd.conf` ファイルの `WebLogicPort` パラメータを 手順 2. でコンフィグレーションしたリスン ポートに設定します。
4. `httpd.conf` ファイルの `SecureProxy` パラメータを `ON` に設定します。
5. SSL 接続に関する情報を定義する追加パラメータを `httpd.conf` ファイルで設定します。パラメータのリストについては、C-13 ページの「Web サーバ プラグインの SSL パラメータ」を参照してください。

SSL-Apache コンフィグレーションに関する問題

SSL を使用するように Apache プラグインをコンフィグレーションする際には、以下の 2 つの点に注意してください。

- PathTrim（C-4 ページ参照）パラメータは、<Location> タグの内側でコンフィグレーションしなければなりません。

  次のコンフィグレーションは正しくありません。

  ```
  <Location /weblogic>
    SetHandler weblogic-handler
  </Location>
  ```

  ```
  <IfModule mod_weblogic.c>
   WebLogicHost localhost
   WebLogicPort 7001
   PathTrim /weblogic
  </IfModule>
  ```

  次のコンフィグレーションは正しい設定です。

  ```
  <Location /weblogic>
    SetHandler weblogic-handler
    PathTrim /weblogic
  </Location>
  ```

- Apache SSL では Include ディレクティブが機能しません。パラメータはすべて、httpd.conf ファイルの中で直接コンフィグレーションする必要があります。SSL を使用する際には次のコンフィグレーションは使用しないでください。

  ```
  <IfModule mod_weblogic.c>
    MatchExpression *.jsp
    Include weblogic.conf
  </IfModule>
  ```

接続エラーとクラスタのフェイルオーバ

WebLogic Server に接続するときに、Apache HTTP Server プラグインは複数のコンフィグレーション パラメータを使用して WebLogic Server ホストへの接続の待ち時間と、接続確立後の応答の待ち時間を判断します。接続できないか、応答がない場合、このプラグインはクラスタ内の別の WebLogic Server に接続してリクエストを送信しようとします。接続が失敗するか、クラスタ内のどの WebLogic Server からも応答がない場合は、エラー メッセージが送信されます。

図 11-1（11-17 ページの「接続のフェイルオーバ」）は、プラグインがどのようにフェイルオーバを処理するのかを示しています。

接続の失敗

接続要求にホストが応答できない場合は、ホスト マシンの問題やネットワークの問題など、サーバに障害があることが考えられます。

WebLogic Server が応答できない場合は、WebLogic Server が動作していないことや、サーバのハング、データベースの問題など、アプリケーションに障害があることが考えられます。

クラスタ化されていない単一 WebLogic Server でのフェイルオーバ

WebLogic Server が 1 つしか動作していない場合でも、ここで説明する同じ理論が適用されますが、プラグインは `WebLogicHost` パラメータで定義されたサーバにのみ接続しようとします。その試みが失敗すると、`HTTP 503` エラー メッセージが返されます。プラグインは、`ConnectTimeoutSecs` に達するまで WebLogic Server への接続を繰り返し試みます。

動的サーバ リスト

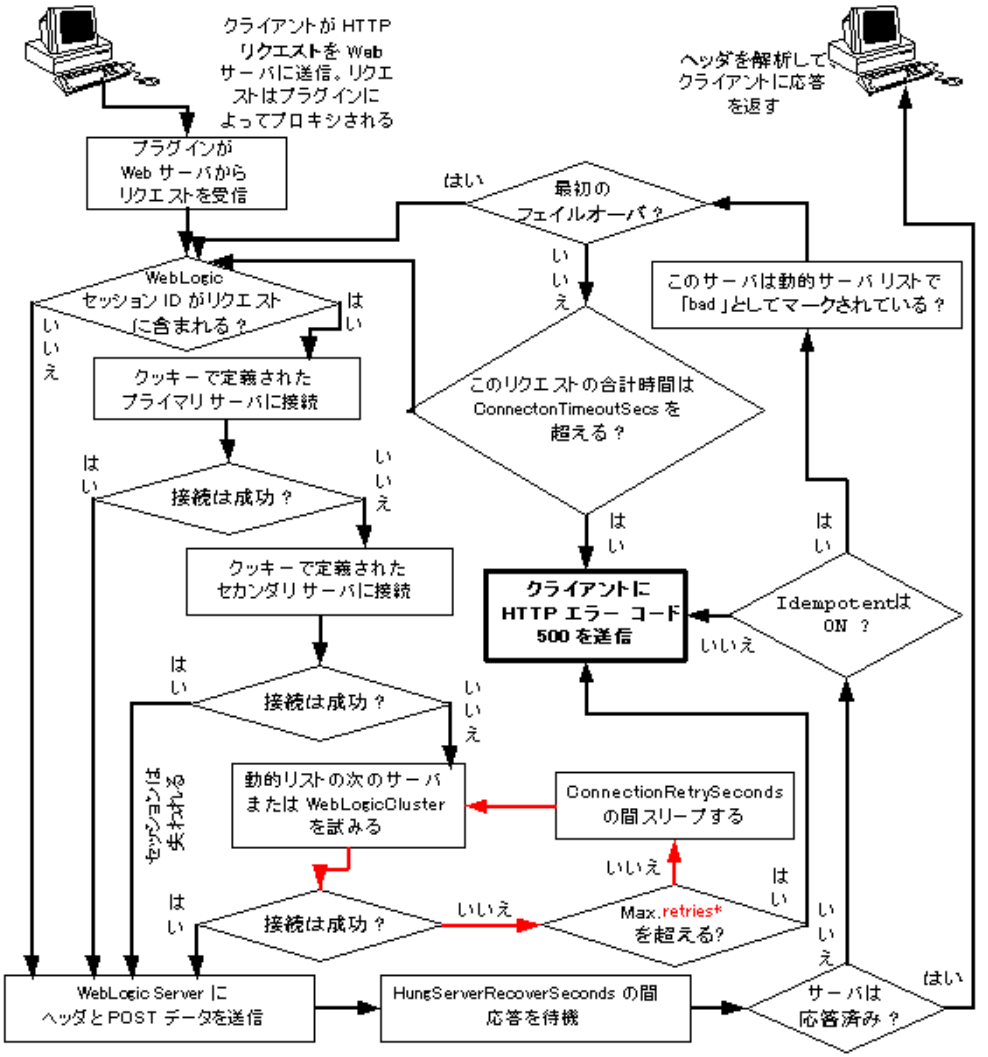
WebLogicCluster パラメータで WebLogic Server のリストを指定すると、プラグインではクラスタ メンバー間でのロード バランシングの起点としてそのリストが使用されます。最初のリクエストがそれらのサーバの 1 つに転送された後に、クラスタ内のサーバの更新されたリストを格納する動的サーバ リストが返されます。更新されたリストはクラスタ内の新しいサーバを追加し、すでにクラスタから外れているか、リクエストに応答できなかったサーバを削除します。このリストは、クラスタで変更が行われたときに HTTP 応答によって自動的に更新されます。

フェイルオーバ、クッキー、および HTTP セッション

リクエストがクッキー、POST データ、または URL エンコーディングを通じてセッション情報を格納している場合、そのセッション ID にはセッションが最初に確立された特定のサーバ（プライマリ サーバ）への参照と元のセッションがレプリケートされる追加サーバ（セカンダリ サーバ）への参照が含まれています。クッキーが含まれているリクエストは、プライマリ サーバに接続しようとします。その試行が失敗すると、リクエストはセカンダリ サーバに転送されます。プライマリ サーバとセカンダリ サーバが両方とも失敗すると、セッションが失われて、プラグインは動的クラスタ リストの別のサーバにあらためて接続しようとします。詳細については、図 11-1（ 11-17 ページの「接続のフェイルオーバ」）を参照してください。

注意： POST データが 64K を超える場合、プラグインは、セッション ID を取得するための POST データの解析を行いません。したがって、セッション ID を POST データに格納した場合、プラグインはリクエストを正しいプライマリまたはセカンダリ サーバにルーティングできないので、セッション データが失われる可能性があります。

図 11-1　接続のフェイルオーバ



* 赤いループで許可される再試行の限度は、次の式で計算されます。
`ConnectTimeoutSecs ÷ ConnectRetrySecs`

httpd.conf ファイルのテンプレート

この節では、httpd.conf ファイルのサンプルを紹介します。このサンプルをテンプレートとして使用し、ユーザの環境およびサーバに合うように変更できます。# で始まる行はコメントです。Apache HTTP サーバでは大文字と小文字は区別されません。また、apxs ユーティリティによって LoadModule および AddModule 行が自動的に追加されます。

```
########################################################

APACHE-HOME\conf\httpd.conf ファイル
########################################################

LoadModule weblogic_module    libexec/mod_wl.so

AddModule mod_weblogic.c

<Location /weblogic>
  SetHandler weblogic-handler
  PathTrim /weblogic
  ErrorPage http://myerrorpage1.mydomain.com
</Location>

<Location /servletimages>
  SetHandler weblogic-handler
  PathTrim /something
  ErrorPage http://myerrorpage1.mydomain.com
</Location>

<IfModule mod_weblogic.c>
  MatchExpression *.jsp
  WebLogicCluster w1s1.com:7001,w1s2.com:7001,w1s3.com:7001
  ErrorPage http://myerrorpage.mydomain.com
</IfModule>
```

コンフィグレーション ファイルのサンプル

httpd.conf ファイルの location ブロックでパラメータを定義する代わりに、必要に応じて、httpd.conf ファイルの IfModule によってロードされる weblogic.conf ファイルを使用できます。以下の例テンプレートとして使用して、ユーザの環境およびサーバに合うように変更できます。# で始まる行はコメントです。

WebLogic クラスタを使用した例

```
# このパラメータは、現在のモジュールに転送される
# すべての URL で共通。URL ごとにパラメータを
# オーバーライドする場合は、<Location> ブロック
# または <Files> ブロックで設定できる（WebLogicHost、
# WebLogicPort、WebLogicCluster、CookieName は除く）。


<IfModule mod_weblogic.c>
  WebLogicCluster w1s1.com:7001,w1s2.com:7001,w1s3.com:7001
  ErrorPage http://myerrorpage.mydomain.com
  MatchExpression *.jsp
</IfModule>
#######################################################
```

複数の WebLogic Cluster を使用した例

```
# このパラメータは、現在のモジュールに転送される
# すべての URL で共通。URL ごとにパラメータを
# オーバーライドする場合は、<Location> ブロック
# または <Files> ブロックで設定できる（WebLogicHost、
# WebLogicPort、WebLogicCluster、CookieName は除く）。


<IfModule mod_weblogic.c>
  MatchExpression *.jsp WebLogicHost=myHost|WebLogicPort=7001|Debug=ON
  MatchExpression *.html WebLogicCluster=myHost1:7282,myHost2:7283|ErrorPage=
    http://www.xyz.com/error.html
</IfModule>
```

WebLogic クラスタを使用しない例

```
# このパラメータは、現在のモジュールに転送される
# すべての URL で共通。URL ごとにパラメータを
# オーバーライドする場合は、<Location> ブロック
# または <Files> ブロックで設定できる（WebLogicHost、
# WebLogicPort、WebLogicCluster、CookieName は除く）。

<IfModule mod_weblogic.c>
  WebLogicHost myweblogic.server.com
  WebLogicPort 7001
```

```
  MatchExpression *.jsp
</IfModule>
```

IP ベースの仮想ホスティングのコンフィグレーション例

```
NameVirtualHost 172.17.8.1
<VirtualHost goldengate.domain1.com>
WebLogicCluster tehama1:4736,tehama2:4736,tehama:4736
PathTrim /x1
ConnectTimeoutSecs 30
</VirtualHost>
<VirtualHost goldengate.domain2.com>
WeblogicCluster green1:4736,green2:4736,green3:4736
PathTrim /y1
ConnectTimeoutSecs 20
</VirtualHost>
```

単一 IP アドレスによる名前ベースの仮想ホスティングのコンフィグレーション例

```
<VirtualHost 162.99.55.208>
        ServerName myserver.mydomain.com
         <Location / >
               SetHandler weblogic-handler
               WebLogicCluster 162.99.55.71:7001,162.99.55.72:7001
               Idempotent ON
               Debug ON
               DebugConfigInfo ON
        </Location>
</VirtualHost>

<VirtualHost 162.99.55.208>
        ServerName myserver.mydomain.com
       <Location / >
                SetHandler weblogic-handler
                WebLogicHost russell
                WebLogicPort 7001
                Debug ON
                DebugConfigInfo ON
        </Location>
</VirtualHost>
```

12 Microsoft Internet Information Server（ISAPI）プラグインのインストールとコンフィグレーション

以下の節では、Microsoft Internet Information Server プラグインをインストールおよびコンフィグレーションする方法について説明します。

- Microsoft Internet Information Server プラグインの概要
- Microsoft Internet Information Server プラグインのインストール
- iisproxy.ini ファイルのサンプル
- Microsoft Internet Information Server プラグインでの SSL の使用
- IIS から WebLogic Server へのサーブレットのプロキシ
- インストールのテスト
- 接続エラーとクラスタのフェイルオーバ

Microsoft Internet Information Server プラグインの概要

Microsoft Internet Information Server プラグインを使用すると、Microsoft Internet Information Server（IIS）から WebLogic Server へリクエストをプロキシできます。このプラグインは、WebLogic Server の動的な機能を必要とするリクエストを WebLogic Server が処理できるようにすることによって IIS を拡張します。

Microsoft Internet Information Server プラグインは、Internet Information Server（IIS）が HTML ページなどの静的ページを提供し、WebLogic Server が HTTP サーブレットまたは JavaServer Pages などの動的ページを提供する環境で使用されることを想定しています。WebLogic Server は別のプロセス（おそらく別のホスト）で動作しています。それでも、エンド ユーザ（ブラウザ）では、WebLogic Server に委託される HTTP リクエストは IIS から来ているものと認識されます。WebLogic クライアント / サーバ プロトコルの HTTP トンネリング機能もこのプラグインを介して動作するため、すべての WebLogic サービスへのアクセスを提供できます。

接続プールとキープアライブ

Microsoft Internet Information Server プラグインは、WebLogic Server との接続の再利用可能なプールを使用してパフォーマンスを向上させます。このプラグインは、同じクライアントからの後続リクエストにプール内の同じ接続を再利用することで、WebLogic Server との間で HTTP 1.1 キープアライブ接続を実装します。接続が 30 秒（またはユーザ定義の時間）を超えて非アクティブな場合、その接続は閉じて、プールに返されます。この機能は、必要に応じて無効にできます。詳細については、C-10 ページの「KeepAliveEnabled」を参照してください。

リクエストのプロキシ

このプラグインは、指定されたコンフィグレーションに基づいてリクエストを WebLogic Server にプロキシします。リクエストは、リクエストの URL（または URL の一部）に基づいてプロキシできます。この方法は、パスによるプロキシ、

と呼びます。リクエストのプロキシは、要求されたファイルの MIME タイプに基づいて行うこともできます（ファイル拡張子を基準としたプロキシ）。さらに、前述の方法を組み合わせて利用することもできます。リクエストが両方の基準に一致する場合、そのリクエストはパスを基準にプロキシされます。リクエストの種類ごとに、プラグインの補足的な動作を定義する追加パラメータを指定することもできます。詳細については、12-3 ページの「Microsoft Internet Information Server プラグインのインストール」を参照してください。

プラットフォーム サポート

Microsoft Internet Information Server プラグインと互換性のあるオペレーティング システムおよび IIS のバージョンに関する最新の情報については、`http://edocs.beasys.co.jp/weblogic/docs/platforms/index.html#iis` にあるプラットフォーム サポート ページを参照してください。

Microsoft Internet Information Server プラグインのインストール

Microsoft Internet Information Server プラグインをインストールするには、次の操作を行います。

1. WebLogic Server インストール ディレクトリの `\bin` ディレクトリにある `iisproxy.dll` ファイルを、IIS からアクセス可能なディレクトリにコピーします。このディレクトリには、`iisproxy.ini` ファイルも格納されていなければなりません。
2. [Microsoft IIS Start] メニューから選択して、IIS Internet Service Manager を起動します。
3. Service Manager の左側のパネルから、使用する Web サイト（デフォルトは「Default Web Site」）を選択します。
4. ツールバーの再生ボタンをクリックして起動します。

5. 左パネルの選択した Web サイトの上でマウスを右クリックし、Web サイトのプロパティを開きます。
6. [プロパティ] パネルで、[ホーム ディレクトリ] タブを選択し、[アプリケーションの設定] セクションの [構成] ボタンをクリックします。
7. ファイル拡張子によるプロキシをコンフィグレーションします。
   a. [アプリケーションのマッピング] タブで、[追加] ボタンをクリックしてファイル タイプを追加し、WebLogic Server に対してプロキシを実行するようにコンフィグレーションを変更します。
   b. ダイアログボックスで「iisproxy.dll」ファイルを検索します。
   c. 拡張子を、WebLogic Server にプロキシするファイルのタイプに設定します。
   d. [スクリプト エンジン] チェック ボックスを選択します。
   e. [ファイルの存在を確認する] チェック ボックスをオフにします。
   f. 安全なインストールを作成するために、必要に応じて [メソッド除外] を設定します。
   g. 完了したら、[OK] ボタンをクリックしてコンフィグレーションを保存します。WebLogic に対してプロキシを実行するファイル タイプごとにこの手順を繰り返します。
   h. ファイル タイプのコンフィグレーションが完了したら、[OK] ボタンをクリックしてプロパティ パネルを閉じます。

   注意： URL のサーバとポートの後ろに追加されたパス情報は、WebLogic Server にそのまま渡されます。たとえば、次の URL で IIS にファイルを要求した場合、

   ```
   http://myiis.com/jspfiles/myfile.jsp
   ```

   そのリクエストは次のような URL で WebLogic Server にプロキシされます。

   ```
   http://mywebLogic:7001/jspfiles/myfile.jsp
   ```

8. iisproxy.ini ファイルを作成します。

   `iisproxy.ini` ファイルには、プラグインのコンフィグレーション パラメータを定義する名前と値の組み合わせを格納します。パラメータのリストは、

C-2 ページの「Web サーバ プラグインの一般的なパラメータ」で参照できます。

注意： パラメータの変更は、コントロール パネルの [サービス] で「IIS Admin Service」を再起動するまで有効にはなりません。

BEA では、iisproxy.ini ファイルを iisproxy.dll ファイルと同じディレクトリに配置することをお勧めします。他のディレクトリを使用することも可能です。他のディレクトリに配置する場合は、iisproxy.ini が以下のディレクトリを以下の順序で検索されることに注意してください。

a. iisproxy.dll と同じディレクトリ。

b. Windows レジストリで参照されている最新バージョンの WebLogic Server のホーム ディレクトリ。そこで iisproxy.ini ファイルが見つからない場合、WebLogic Server は、Windows レジストリで WebLogic Server の旧バージョンを調べ、古いインストール環境のホーム ディレクトリで iisproxy.ini ファイルを探します。

c. c:\weblogic ディレクトリ（存在する場合）。

9. Microsoft Internet Information Server プラグインがリクエストをプロキシする WebLogic Server ホストとポート番号を定義します。コンフィグレーションに応じて、以下の 2 通りの方法でホストとポートを定義できます。

   - 1 つの WebLogic Server にリクエストをプロキシする場合は、iisproxy.ini ファイルで WebLogicHost パラメータと WebLogicPort パラメータを定義します。次に例を示します。

     ```
     WebLogicHost=localhost
     WebLogicPort=7001
     ```

   - WebLogic Server のクラスタにリクエストをプロキシする場合は、iisproxy.ini ファイルで WebLogicCluster パラメータを定義します。次に例を示します。

     ```
     WebLogicCluster=myweblogic.com:7001,yourweblogic.com:7001
     ```

     myweblogic.com と yourweblogic.com は、クラスタ内で動作している WebLogic Server のインスタンスです。

10. パスによるプロキシをコンフィグレーションします。ファイル タイプによるプロキシだけでなく、iisproxy.ini ファイルで追加パラメータを指定することにより、パスに基づいてファイルを提供するように Microsoft Internet

Information Server プラグインをコンフィグレーションすることもできます。パスを基準としたプロキシは、MIME タイプを基準としたプロキシに優先します。

また、パスを使用して IIS で定義されている複数の Web サイトをプロキシすることもできます。詳細については、12-7 ページの「IIS の複数の仮想 Web サイトのプロキシ」を参照してください。

パスによるプロキシをコンフィグレーションするには、次の手順に従います。

a. `iisforward.dll` ファイルを `iisproxy.dll` ファイルと同じディレクトリに置き、`iisforward.dll` ファイルをフィルタ サービスとして IIS に追加します（Web サイトの [プロパティ] パネルで、[ISAPI フィルタ] タブの [追加] をクリックして `iisforward dll` を追加します）。

b. `iisproxy.dll` で処理される特殊なファイル タイプとして `wlforward` を登録します。

c. `iisproxy.ini` で `WlForwardPath` プロパティを定義します。`WlForwardPath` では、次のように WebLogic Server にプロキシするパスを定義します。`WlForwardPath=/weblogic.`

d. `PathTrim` パラメータを設定して、必要に応じて `WlForwardPath` を削除します。次に例を示します。

```
WlForwardPath=/weblogic
PathTrim=/weblogic
```

この場合は、IIS から WebLogic Server へのリクエストが削除されます。したがって、`/weblogic/session` は `/session` に変更されます。

e. パス情報が含まれていない（ホスト名しか含まれていない）リクエストが必要な場合は、`DefaultFileName` パラメータを、リクエストがプロキシされる Web アプリケーションのウェルカム ページの名前に設定します。このパラメータの値は、URL に追加されます。

f. アプリケーションをデバッグする必要がある場合は、`iisproxy.ini` で `Debug=ON` パラメータを設定します。デバッグに利用できるプラグインのアクティビティのログを記録した `c:\tmp\iisforward.log` が生成されます。

11. HTTP トンネリング（オプション）を有効にする場合、パスによるプロキシのコンフィグレーション手順（上の手順 10）に従います。ただし、HTTP トンネリング リクエストを処理させる WebLogic Server ホスト名と WebLogic Server ポート番号、または WebLogic Cluster の名前を指定します。

    ```
    WlForwardPath=*/HTTPClnt*
    ```

    PathTrim パラメータを使用する必要はありません。

    注意： HTTP トンネリングを使用する必要があるのは、アプレットを使用して IIS/NES 経由で WebLogic Server に接続し、t3 の代わりのプロトコルとして http を使用する場合です（たとえば、プロバイダ URL のプロトコルとして t3:// の代わりに http:// を使用する場合）。

12. `iisproxy.ini` ファイルで追加パラメータを設定します。パラメータのリストについては、C-2 ページの「Web サーバ プラグインの一般的なパラメータ」で参照できます。

13. IIS から WebLogic Server にサーブレットをプロキシするが、パスを基準にしてプロキシしない場合は、12-11 ページの「IIS から WebLogic Server へのサーブレットのプロキシ」を参照してください。

IIS の複数の仮想 Web サイトのプロキシ

複数の Web サイト（IIS の仮想ディレクトリとして定義）を WebLogic Server にプロキシするには、次の手順に従います。

1. 各仮想ディレクトリ用に新しいディレクトリを作成します。このディレクトリには、プロキシを定義するための `dll` および `ini` ファイルが格納されます。
2. 手順 1 で作成した各ディレクトリに `iisforward.dll` をコピーします。
3. 各 Web サイトの `iisforward.dll` を IIS に登録します。
4. `iisforward.ini` というファイルを作成します。作成したファイルを、`iisforward.dll` と同じディレクトリに配置します。このファイルには、IIS に定義されている仮想 Web サイトごとに次のエントリを格納する必要があります。

```
vhostN=websiteName:port
websiteName:port=dll_directory/iisproxy.ini
```

各値の説明は次のとおりです。

- *N* は、仮想 Web サイトを表す整数です。最初に定義する仮想 Web サイトには整数 1 を使用し、以降の Web サイトには順に 1 ずつ増やします。
- *websiteName* は、IIS に登録した仮想 Web サイトの名前です。
- *port* は、IIS が HTTP リクエストをリスンしているポート番号です。
- *dll_directory* は、手順 1 で作成したディレクトリのパスです。

次に例を示します。

```
vhost1=strawberry.com:7001
strawberry.com:7001=c:\strawberry\iisproxy.ini
vhost2=blueberry.com:7001
blueberry.com:7001=c:\blueberry\iisproxy.ini
...
```

5. 「リクエストのプロキシ」の手順 8. で説明したように、仮想 Web サイトごとに iisproxy.ini ファイルを作成します。仮想 Web サイトごとに、手順 1 で作成したディレクトリにこの iispoxy.ini ファイルをコピーします。
6. 手順 1 で作成した各ディレクトリに iisproxy.dll をコピーします。
7. IIS で、[アプリケーション保護] オプションの値を [高（分離プロセス）] に設定します。

IIS を介した ACL の作成

認可ヘッダが IIS によって渡されない場合、ACL は Microsoft Internet Information Server プラグインを介して機能しません。次の説明に基づいて、認可ヘッダが IIS によって渡されるようにします。

基本認証を使用する場合、ユーザはローカルのログオン権限でログオンします。基本認証を使用するには、各ユーザにローカル ログオンのユーザ権利を与えます。ただし、基本認証でローカル ログオンを使用することで 2 つの問題が発生する可能性があります。

- FrontPage、IIS、および Windows NT のコンフィグレーションが正しい場合でも、ユーザがローカル ログオン権限を持っていないと、基本認証は機能しません。
- ローカル ログオン権限を持ち、IIS を実行しているホスト コンピュータに物理的にアクセス可能なユーザには、コンソールで対話セッションを開始するパーミッションが与えられます。

コンソールの [ディレクトリ セキュリティ] タブで基本認証を有効にするには、[匿名アクセスを許可する] オプションをオンに、その他のオプションをすべてオフにします。

iisproxy.ini ファイルのサンプル

ここでは、クラスタ化されていない 1 つの WebLogic Server で使用する iisproxy.ini ファイルのサンプルを示します。先頭に「#」が付いた行はコメントです。

```
# このファイルでは、IIS/WebLogic プラグイン用の
# 初期化パラメータの名前と値の組み合わせを指定する

WebLogicHost=localhost
WebLogicPort=7001
ConnectTimeoutSecs=20
ConnectRetrySecs=2
```

次に示すのは、クラスタ化された WebLogic Server で使用する iisproxy.ini ファイルのサンプルです。先頭に「#」が付いた行はコメントです。

```
# このファイルでは、IIS/WebLogic プラグイン用の
# 初期化パラメータの名前と値の組み合わせを指定する

WebLogicCluster=myweblogic.com:7001,yourweblogic.com:7001
ConnectTimeoutSecs=20
ConnectRetrySecs=2
```

注意： プラグインと WebLogic Server の間で SSL を使用する場合は、SSL リスン ポートとしてポート番号を定義する必要があります。

Microsoft Internet Information Server プラグインでの SSL の使用

セキュア ソケット レイヤ（SSL）プロトコルを使用すると、WebLogic Server プロキシ プラグインと Microsoft Internet Information Server の間の接続を保護できます。SSL プロトコルは、Microsoft Internet Information Server プラグインと WebLogic Server の間でやり取りされるデータに機密性と整合性を提供します。また、SSL プロトコルを使用すると、WebLogic Server プロキシ プラグインでは、信頼性のあるプリンシパルに情報が渡されることを確認するために、Microsoft Internet Information Server に対して自身を認証することができます。

Microsoft Internet Information Server プラグインでは、SSL プロトコルを使用してプロキシ プラグインと Microsoft Internet Information Server 間の接続を保護するかどうかを、転送プロトコル（`http` または `https`）によって決定できません。Microsoft Internet Information Server プラグインで SSL プロトコルを使用するには、プロキシされるリクエストを受け取る WebLogic Server で、SSL プロトコルを使用するようにコンフィグレーションする必要があります。WebLogic Server プロキシ プラグインでは、セキュアな SSL 通信を使用するようにコンフィグレーションされた WebLogic Server 上のポートを使用して Microsoft Internet Information Server と通信します。

Microsoft Internet Information Server プラグインと WebLogic Server の間で SSL プロトコルを使用するには、次の操作を行います。

1. SSL 向けに WebLogic Server をコンフィグレーションします。詳細については、14-42 ページの「SSL プロトコルのコンフィグレーション」を参照してください。
2. WebLogic Server の SSL リスン ポートをコンフィグレーションします。詳細については、8-4 ページの「リスンポートのコンフィグレーション」を参照してください。
3. `iisproxy.ini` ファイルの `WebLogicPort` パラメータを 手順 2. でコンフィグレーションしたリスン ポートに設定します。
4. `iisproxy.ini` ファイルの `SecureProxy` パラメータを `ON` に設定します。

5. SSL 接続を定義する追加パラメータを iisproxy.ini ファイルで設定します。パラメータのリストについては、C-13 ページの「Web サーバ プラグインの SSL パラメータ」を参照してください。

次に例を示します。

```
WebLogicHost=myweblogic.com
WebLogicPort=7002
SecureProxy=ON
```

IIS から WebLogic Server へのサーブレットのプロキシ

iisforward.dll がフィルタとして登録されている場合、サーブレットをパスでプロキシできます。その場合は、次の形式の URL でサーブレットを呼び出します。

```
http://IISserver/weblogic/myServlet
```

iisforward.dll がフィルタとして登録されていない場合にサーブレットをプロキシするには、ファイル タイプを基準としたプロキシをコンフィグレーションする必要があります。ファイル タイプでサーブレットをプロキシするには、次の操作を行います。

1. WebLogic Server にリクエストをプロキシできるように IIS に任意のファイル タイプ（拡張子）を登録します。詳細については、12-3 ページの「Microsoft Internet Information Server プラグインのインストール」の手順 7. を参照してください。
2. 適切な Web アプリケーションにサーブレットを登録します。サーブレット登録の詳細については、http://edocs.beasys.co.jp/e-docs/wls61/webapp/components.html#configuring-servlets の「サーブレットのコンフィグレーション」を参照してください。
3. 次のような形式の URL でサーブレットを呼び出します。

   http://www.myserver.com/*virtualName*/*anyfile.ext*

virtualName は、このサーブレットの Web アプリケーション デプロイメント記述子（web.xml）の <servlet-mapping> 要素で定義された URL パターンで、*ext* は WebLogic Server に対してプロキシするように IIS に登録したファイル タイプ（拡張子）です。URL の *anyfile* の部分は、このコンテキストでは無視されます。

注意：

- サーブレットから呼び出されるイメージ リンクが Web アプリケーションの一部である場合は、IIS に適切なファイル タイプ（.gif や .jpg など）を登録してイメージに対するリクエストも WebLogic Server にプロキシする必要があります。ただし、それらのイメージは IIS から直接提供することもできます。
- プロキシ対象のサーブレットに他のサーブレットを呼び出すリンクがある場合は、そのリンクも上記のパターンに従って WebLogic Server にプロキシする必要があります。

インストールのテスト

Microsoft Internet Information Server プラグインをインストールしてコンフィグレーションした後は、次の手順に従ってデプロイメントとテストを行います。

1. WebLogic Server と IIS が動作していることを確認します。
2. JSP ファイルをデフォルト Web アプリケーションのドキュメント ルートに保存します。
3. ブラウザを開き、次のように IIS + filename.jsp という形式の URL を設定します。

   ```
   http://myii.server.com/filename.jsp
   ```

   filename.jsp がブラウザに表示される場合は、プラグインが正常に機能しています。

接続エラーとクラスタのフェイルオーバ

WebLogic Server に接続するときに、Microsoft Internet Information Server プラグインは複数のコンフィグレーション パラメータを使用して WebLogic Server ホストへの接続の待ち時間と、接続確立後の応答の待ち時間を判断します。接続できないか、応答がない場合、このプラグインはクラスタ内の別の WebLogic Server に接続してリクエストを送信しようとします。接続が失敗するか、クラスタ内のどの WebLogic Server からも応答がない場合は、エラー メッセージが送信されます。

図 12-1 （12-15 ページの「接続のフェイルオーバ」）は、プラグインがどのようにフェイルオーバを処理するのかを示しています。

接続の失敗

接続要求にホストが応答できない場合は、ホスト マシンの問題やネットワークの問題など、サーバに障害があることが考えられます。

WebLogic Server が応答できない場合は、WebLogic Server が動作していないことや、サーバのハング、データベースの問題など、アプリケーションに障害があることが考えられます。

クラスタ化されていない単一 WebLogic Server でのフェイルオーバ

WebLogic Server が 1 つしか動作していない場合でも、ここで説明する同じ理論が適用されますが、プラグインは `WebLogicHost` パラメータで定義されたサーバにのみ接続しようとします。その試みが失敗すると、`HTTP 503` エラー メッセージが返されます。プラグインは、`ConnectTimeoutSecs` に達するまで WebLogic Server への接続を繰り返し試みます。

動的サーバ リスト

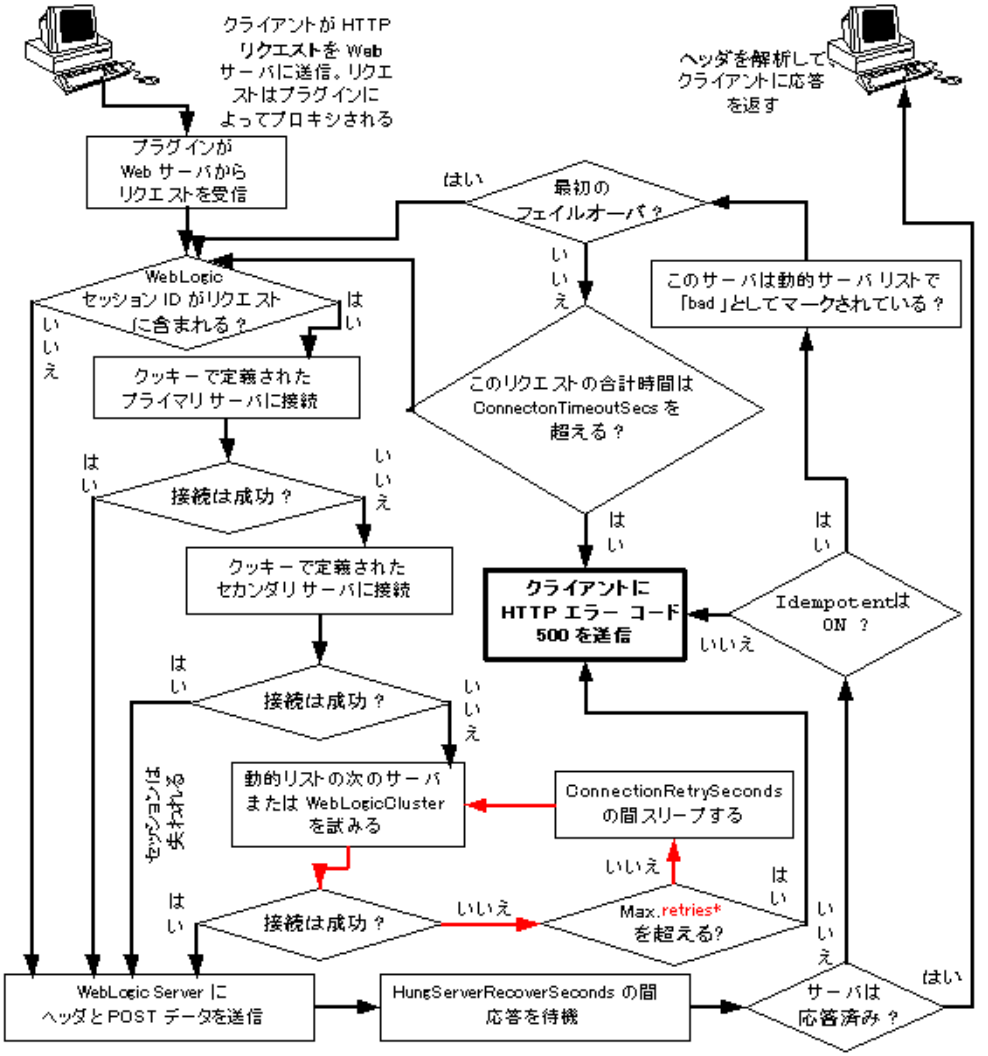
WebLogicCluster パラメータで WebLogic Server のリストを指定すると、プラグインではクラスタ メンバー間でのロード バランシングの起点としてそのリストが使用されます。最初のリクエストがそれらのサーバの 1 つに転送された後に、クラスタ内のサーバの更新されたリストを格納する動的サーバ リストが返されます。更新されたリストはクラスタ内の新しいサーバを追加し、すでにクラスタから外れているか、リクエストに応答できなかったサーバを削除します。このリストは、クラスタで変更が行われたときに HTTP 応答によって自動的に更新されます。

フェイルオーバ、クッキー、および HTTP セッション

リクエストがクッキー、POST データ、または URL エンコーディングを通じてセッション情報を格納している場合、そのセッション ID にはセッションが最初に確立された特定のサーバ（プライマリ サーバ）への参照と元のセッションがレプリケートされる追加サーバ（セカンダリ サーバ）への参照が含まれています。クッキーを含むリクエストはプライマリ サーバに接続しようとします。接続できない場合、リクエストはセカンダリ サーバにルーティングされます。プライマリ サーバとセカンダリ サーバが両方とも失敗すると、セッションが失われて、プラグインは動的クラスタ リストの別のサーバにあらためて接続しようとします。詳細については、図 12-1（12-15 ページの「接続のフェイルオーバ」）を参照してください。

注意： POST データが 64K を超える場合、プラグインは、セッション ID を取得するための POST データの解析を行いません。したがって、セッション ID を POST データに格納した場合、プラグインはリクエストを正しいプライマリまたはセカンダリ サーバにルーティングできないので、セッション データが失われる可能性があります。

図 12-1　接続のフェイルオーバ



* 赤いループで許可される再試行の限度は、次の式で計算されます。

`ConnectTimeoutSecs ÷ ConnectRetrySecs`

13 Netscape Enterprise Server プラグイン（NSAPI）のインストールとコンフィグレーション

以降の節では、Netscape Enterprise Server プラグイン（NES）プロキシ プラグインをインストールおよびコンフィグレーションする方法について説明します。

- Netscape Enterprise Server プラグインの概要
- Netscape Enterprise Server プラグインのインストールとコンフィグレーション
- NSAPI プラグインでの SSL の使用
- 接続エラーとクラスタのフェイルオーバ
- ファイアウォールとロード ディレクタを使用する場合のフェイルオーバの動作
- obj.conf ファイルのサンプル（WebLogic クラスタを使用しない場合）
- obj.conf ファイルのサンプル（WebLogic クラスタを使用する場合）

Netscape Enterprise Server プラグインの概要

Netscape Enterprise Server プラグインを使用すると、Netscape Enterprise Server（NES、または iPlanet とも呼ばれる）から WebLogic Server にリクエストをプロキシできます。このプラグインは、WebLogic Server の動的な機能を必要とするリクエストを WebLogic Server が処理できるようにすることによって NES を拡張します。

Netscape Enterprise Server プラグインは、Netscape Enterprise Server が静的ページを提供しており、別のプロセス（おそらく別のホスト）で動作している Weblogic Server が動的ページ（JSP や HTTP サーブレットで生成されたページ）を提供している環境で使用するためのものです。WebLogic Server と Netscape Enterprise Server プラグインは、クリア テキストまたはセキュア ソケット レイヤ（SSL）を使用して接続されています。エンド ユーザ、つまりブラウザには、WebLogic に委託された HTTP リクエストが静的ページと同じ場所から送られてきたように見えます。また、WebLogic Server の HTTP トンネリング機能も Netscape Enterprise Server プラグインを介して動作するため、動的ページだけでなく、すべての WebLogic Server サービスへのアクセスを提供できます。

Netscape Enterprise Server プラグインは、Netscape Enterprise Server の中で NSAPI モジュールとして機能します（`http://home.netscape.com/servers/index.html` を参照）。NSAPI モジュールは起動時に NES によってロードされ、特定の HTTP リクエストがそこに委託されます。NSAPI モジュールは、HTTP（Java）サーブレットと似ていますが、プラットフォームにネイティブなコードで記述されています。

Netscape Enterprise Server および iPlanet サーバのサポート対象のバージョンについては、「BEA WebLogic Server プラットフォーム サポート ページ」を参照してください。

接続プールとキープアライブ

WebLogic Server NSAPI プラグインでは、プラグインから WebLogic Server への接続に再利用できる接続プールを使用するため、効率的なパフォーマンスを実現します。NSAPI プラグインと WebLogic Server の間で「キープアライブ」接続が自動的に実装されます。接続が 30 秒（またはユーザ定義の時間）を超えて非アクティブな場合、その接続は閉じます。この機能は、必要に応じて無効にできます。詳細については、C-10 ページの「KeepAliveEnabled」を参照してください。

リクエストのプロキシ

このプラグインは、指定されたコンフィグレーションに基づいてリクエストを WebLogic Server にプロキシします。リクエストは、リクエストの URL（または URL の一部）に基づいてプロキシできます。この方法は、パスによるプロキシ、と呼びます。リクエストのプロキシは、要求されたファイルの MIME タイプに基づいて行うこともできます。さらに、前述の方法を組み合わせて利用することもできます。リクエストが両方の基準に一致する場合、そのリクエストはパスを基準にプロキシされます。リクエストの種類ごとに、プラグインの補足的な動作を定義する追加パラメータを指定することもできます。詳細については、次の節を参照してください。

Netscape Enterprise Server プラグインのインストールとコンフィグレーション

Netscape Enterprise Server プラグインをインストールしてコンフィグレーションするには、次の操作を行います。

1. ライブラリをコピーします。

   WebLogic NSAPI プラグイン モジュールは、UNIX では共有オブジェクト（`.so`）、Windows ではダイナミック リンク ライブラリ（`.dll`）として提供

されます。各ファイルはそれぞれ、WebLogic Server 配布キットの /lib または \bin ディレクトリにあります。

の「プラットフォーム サポート」表から環境に合った適切なライブラリ ファイルを選択し、そのファイルを NES が配置されているファイル システムにコピーします。

2. obj.conf ファイルを修正します。obj.conf ファイルでは、どのリクエストが WebLogic Server にプロキシされるのかといったコンフィグレーション情報を定義します。詳細については、13-5 ページの「obj.conf ファイルの修正」を参照してください。
3. MIME タイプを基準にしてリクエストをプロキシする場合は、以下の操作を行います。
   a. 適切な行を obj.conf ファイルに追加します。詳細については、13-5 ページの「obj.conf ファイルの修正」を参照してください。
   b. obj.conf ファイルで参照される新しい MIME タイプを MIME.types ファイルに追加します。MIME タイプは、Netscape サーバ コンソールを使用するか、MIME.types ファイルを直に編集することで追加できます。

      MIME.types ファイルを直接編集するには、編集するファイルを開いて次のように入力します。

      ```
      type=text/jsp          exts=jsp
      ```

      注意 : NES 4.0（iPlanet）の場合、JSP の MIME タイプを追加するのではなく、既存の MIME タイプを次のように変更する必要があります。

      ```
      magnus-internal/jsp
      ```

      この行を次のように変更します。

      ```
      text/jsp
      ```

      Netscape コンソールを使用するには、[Manage Preferences ｜ Mime タイプ] を選択して追加または編集します。
4. Netscape Enterprise Server プラグインをデプロイおよびテストします。
   a. WebLogic Server を起動します。
   b. Netscape Enterprise Server を起動します。NES が既に動作している場合、再起動するか、コンソールから新しい設定を適用して、新しい設定を有効にします。

c. Netscape Enterprise Server プラグインをテストするには、ブラウザを開き、Enterprise Server の URL + /weblogic/ を設定します。この設定では、WebLogic Server でデフォルト Web アプリケーションとして定義されている、デフォルトの WebLogic Server HTML ページ、ウェルカム ファイル、またはデフォルト サーブレットが開くはずです。次に例を示します。

```
http://myenterprise.server.com/weblogic/
```

obj.conf ファイルの修正

Netscape Enterprise Server プラグインを使用するには、NES obj.conf ファイルを修正する必要があります。その修正では、リクエストがどのように WebLogic Server にプロキシされるのかを指定します。リクエストは、URL または MIME タイプに基づいてプロキシできます。それぞれの手順はこの節で説明します。

Netscape の obj.conf ファイルは、テキストの配置に関して非常に厳密です。問題が起こらないようにするために、obj.conf ファイルに関する以下の点に注意してください。

- 前後の余分なスペースは削除します。余分なスペースがあると、Netscape サーバが機能しなくなることがあります。
- 入力文字が 1 行でおさまらない場合、行末に「\」を置いて次の行の入力を続けます。「\」があると、最初の行の末尾が次の行頭に直接つながります。行末と次行の語の間に空白が必要な場合は、最初の行の行末（「\」の前）か次行の先頭にスペースを 1 文字分だけ入れます。
- 属性は複数の行にまたがらないようにします。たとえば、クラスタのすべてのサーバは WebLogicCluster に続けて同じ行に列挙しなければなりません。
- 必須パラメータがコンフィグレーションで設定されていない場合は、オブジェクトを呼び出したときに HTML エラーが発行され、コンフィグレーションに含まれていないパラメータが通知されます。

obj.conf ファイルをコンフィグレーションするには、次の手順に従います。

1. obj.conf を探して開きます。

   NES インスタンス用の obj.conf ファイルは、次の場所にあります。

   NETSCAPE_HOME\https-*INSTANCE_NAME*\config\obj.conf

NETSCAPE_HOME は NES がインストールされているルート ディレクトリで、*INSTANCE_NAME* は特定のインスタンス、つまりユーザがサーバとしてコンフィグレーションしているマシンです。たとえば、myunixmachine という UNIX マシンでは、次の場所に obj.conf ファイルがあります。

```
/usr/local/netscape/enterprise-351/
 https-myunixmachine/config/obj.conf
```

2. ネイティブ ライブラリを NSAPI モジュールとしてロードするように NES に指示します。

   obj.conf ファイルの先頭に次の行を追加します。これらの行は、ネイティブ ライブラリ（.so または .dll ファイル）を NSAPI モジュールとしてロードするように NES に指示します。

   ```
   Init fn="load-modules" funcs="wl_proxy,wl_init"\
    shlib=/usr/local/netscape/plugins/SHARED_LIBRARY
   Init fn="wl_init"
   ```

   SHARED_LIBRARY は、13-3 ページの「Netscape Enterprise Server プラグインのインストールとコンフィグレーション」の手順 1. でインストールした共有オブジェクトまたは dll（libproxy.so など）です。関数「load-modules」は、NES 起動時に共有ライブラリにロード用のタグをつけます。値「wl_proxy」と「wl_init」は、Netscape Enterprise Server プラグインが実行する関数を識別します。

3. URL でリクエストをプロキシするには（パスによるプロキシとも言う）、プロキシする URL ごとに別々の <Object> タグを作成し、PathTrim パラメータを定義します（MIME タイプでリクエストをプロキシする場合は、手順 4. を参照）。パスを基準としたプロキシは、MIME タイプを基準としたプロキシに優先します。次に、文字列 */weblogic/* の含まれているリクエストをプロキシする <Object> タグの例を示します。

   ```
   <Object name="weblogic" ppath="*/weblogic/*">
   Service fn=wl_proxy WebLogicHost=myserver.com\
    WebLogicPort=7001 PathTrim="/weblogic"
   </Object>
   ```

   URL でリクエストをプロキシするために <Object> タグを作成するには、次の操作を行います。

   a. name 属性を使用して開始 <Object> タグの内側でこのオブジェクトの名前を指定します（省略可能）。name 属性は参照用であり、Netscape Enterprise Server プラグインでは使用されません。次に例を示します。

      ```
      <Object name=myObject ...>
      ```

b. <Object> タグの内側で ppath 属性を使用して、プロキシする URL を指定します。次に例を示します。

```
<Object name=myObject ppath="*/weblogic/*>
```

ppath 属性の値は、Weblogic Server 用のリクエストを示す任意の文字列です。ppath を使用すると、そのパスを含むリクエストはすべてリダイレクトされます。たとえば、「*/weblogic/*」という ppath では、「http://enterprise.com/weblogic」で始まるすべてのリクエストが Netscape Enterprise Server プラグイン（リクエストを指定された WebLogic ホストまたはクラスタに送信する）にリダイレクトされます。

c. <Object> タグと </Object> タグの内側で Service ディレクティブを追加します。Service ディレクティブでは、名前と値の組み合わせとして有効なパラメータを指定できます。名前と値の組み合わせが複数の場合は、1 つのスペースで区切ります。次に例を示します。

```
Service fn=wl_proxy WebLogicHost=myserver.com\
 WebLogicPort=7001 PathTrim="/weblogic"
```

パラメータのリストについては、C-2 ページの「Web サーバ プラグインの一般的なパラメータ」を参照してください。以下のパラメータを指定する必要があります。

クラスタ化されていない WebLogic Server の場合

WebLogicHost パラメータと WebLogicPort パラメータ

WebLogic Server のクラスタの場合

WebLogicCluster パラメータ

Service ディレクティブは、必ず、Service fn=wl_proxy の後にパラメータの名前と値の組み合わせが続くかたちでなければなりません。

次に示すのは、2 つの独立した ppath のオブジェクト定義の例です。各 ppath は、WebLogic Server の別々のインスタンスに送信されるリクエストを示します。

```
<Object name="weblogic" ppath="*/weblogic/*">
Service fn=wl_proxy WebLogicHost=myserver.com\
 WebLogicPort=7001 PathTrim="/weblogic"
</Object>

<Object name="si" ppath="*/servletimages/*">
Service fn=wl_proxy WebLogicHost=otherserver.com\
 WebLogicPort=7008
</Object>
```

注意： オプションのパラメータ（PathTrim など）を使用すると、Netscape Enterprise Server プラグインを通じて ppath がどのように渡されるのかをさらに細かくコンフィグレーションできます。プラグイン パラメータのリストについては、C-2 ページの「Web サーバ プラグインの一般的なパラメータ」を参照してください。

4. MIME タイプでプロキシする場合は、MIME.types ファイルで MIME タイプを指定しなければなりません。このファイルの修正方法については、13-3 ページの「Netscape Enterprise Server プラグインのインストールとコンフィグレーション」の手順 3. を参照してください。

   指定した MIME タイプの拡張子（.jsp など）を持つリクエストはすべて、URL に関係なく WebLogic Server にプロキシできます。

   特定のファイル タイプのすべてのリクエストを WebLogic Server にプロキシするには、次の操作を行います。

   a. Service ディレクティブを既存の default Object 定義（<Object name=default ...>）に追加します。

      たとえば、すべての JSP を WebLogic Server にプロキシするには、次に示す Service ディレクティブを次の文字列で始まる最後の行の後、

      ```
      NameTrans fn=....
      ```

      次の文字列で始まる行の前に追加する必要があります。

      ```
      PathCheck.
      ```

      ```
      Service method="(GET|HEAD|POST|PUT)" type=text/jsp
      fn=wl_proxy\
       WebLogicHost=192.1.1.4 WebLogicPort=7001
      PathPrepend=/jspfiles
      ```

      この Service ディレクティブでは、.jsp 拡張子を持つすべてのファイルを指定した WebLogic Server にプロキシします。その際の URL は次のとおりです。

      ```
      http://WebLogic:7001/jspfiles/myfile.jsp
      ```

      PathPrepend パラメータの値は、リクエストがプロキシされる WebLogic Server またはクラスタでデプロイされる Web アプリケーションのコンテキスト ルートと一致していなければなりません。

      Netscape Enterprise Server プラグインのエントリを追加した後、デフォルト Object 定義は次の例のようになります。追加部分は**太字**で示されています。

```
<Object name=default>
NameTrans fn=pfx2dir from=/ns-icons\
 dir="c:/Netscape/SuiteSpot/ns-icons"
NameTrans fn=pfx2dir from=/mc-icons\
 dir="c:/Netscape/SuiteSpot/ns-icons"
NameTrans fn="pfx2dir" from="/help" dir=\
 "c:/Netscape/SuiteSpot/manual/https/ug"
NameTrans fn=document-root root="c:/Netscape/SuiteSpot/docs"
Service method="(GET|HEAD|POST|PUT)" type=text/jsp\
 fn=wl_proxy WebLogicHost=localhost WebLogicPort=7001\
 PathPrepend=/jspfiles
PathCheck fn=nt-uri-clean
PathCheck fn="check-acl" acl="default"
PathCheck fn=find-pathinfo
PathCheck fn=find-index index-names="index.html,home.html"
ObjectType fn=type-by-extension
ObjectType fn=force-type type=text/plain
Service method=(GET|HEAD) type=magnus-internal/imagemap\
 fn=imagemap
Service method=(GET|HEAD)\
 type=magnus-internal/directory fn=index-common
Service method=(GET|HEAD)\
 type=*~magnus-internal\* fn=send-file
AddLog fn=flex-log name="access"
</Object>
```

b. WebLogic Server にプロキシする他のすべての MIME タイプについて、デフォルトのオブジェクト定義に同様の `Service` 文を追加します。

5. HTTP トンネリング（オプション）を有効にするには、次の操作を行います。

   次のオブジェクト定義を `obj.conf` ファイルに追加します。HTTP トンネリング リクエストを処理する実際の WebLogic Server ホスト名と WebLogic Server ポート番号、または WebLogic クラスタの名前を当てはめてください。

```
<Object name="tunnel" ppath="*/HTTPClnt*">
Service fn=wl_proxy WebLogicHost=192.192.1.4\
 WebLogicPort=7001
</Object>
```

NSAPI プラグインでの SSL の使用

セキュア ソケット レイヤ（SSL）プロトコルを使用すると、Netscape Enterprise Server プラグインと WebLogic Server の間の接続を保護できます。SSL プロトコルは、Netscape Enterprise Server プラグインと WebLogic Server の間でやり取りされるデータに機密性と整合性を提供します。また、SSL プロトコルを使用すると、WebLogic Server プロキシ プラグインでは、信頼性のあるプリンシパルに情報が渡されることを確認するために、Netscape Enterprise Server に対して自身を認証することができます。

WebLogic Server プロキシ プラグインでは、Netscape Enterprise Server プラグインと WebLogic Server の接続を保護するために SSL プロトコルが使用されるのかどうかを、（通常はブラウザからの）HTTP リクエストで指定された転送プロトコル（`http` または `https`）では判断しません。

Netscape Enterprise Server プラグインと WebLogic Server の間で SSL プロトコルを使用するには、次の操作を行います。

1. SSL 向けに WebLogic Server をコンフィグレーションします。詳細については、14-42 ページの「SSL プロトコルのコンフィグレーション」を参照してください。
2. WebLogic Server の SSL リスン ポートをコンフィグレーションします。詳細については、8-4 ページの「リスンポートのコンフィグレーション」を参照してください。
3. `obj.conf` ファイルの `Service` ディレクティブの `WebLogicPort` パラメータを 手順 2. でコンフィグレーションしたリスン ポートに設定します。
4. `obj.conf` ファイルの `Service` ディレクティブの `SecureProxy` パラメータを `ON` に設定します。
5. `obj.conf` ファイルの `Service` ディレクティブで SSL 接続に関する情報を定義する追加パラメータを設定します。パラメータのリストについては、C-13 ページの「Web サーバ プラグインの SSL パラメータ」を参照してください。

接続エラーとクラスタのフェイルオーバ

WebLogic Server に接続するときに、Netscape Enterprise Server プラグインは複数のコンフィグレーション パラメータを使用して WebLogic Server ホストへの接続の待ち時間と、接続確立後の応答の待ち時間を判断します。接続できないか、応答がない場合、このプラグインはクラスタ内の別の WebLogic Server に接続してリクエストを送信しようとします。接続が失敗するか、クラスタ内のどの WebLogic Server からも応答がない場合は、エラー メッセージが送信されます。

図 13-1 （13-13 ページの「接続のフェイルオーバ」）は、プラグインがどのようにフェイルオーバを処理するのかを示しています。

接続の失敗

接続要求にホストが応答できない場合は、ホスト マシンの問題やネットワークの問題など、サーバに障害があることが考えられます。

WebLogic Server が応答できない場合は、WebLogic Server が動作していないことや、サーバのハング、データベースの問題など、アプリケーションに障害があることが考えられます。

クラスタ化されていない単一 WebLogic Server でのフェイルオーバ

WebLogic Server が 1 つしか動作していない場合でも、ここで説明する同じ理論が適用されますが、プラグインは `WebLogicHost` パラメータで定義されたサーバにのみ接続しようとします。その試みが失敗すると、`HTTP 503` エラー メッセージが返されます。プラグインは、`ConnectTimeoutSecs` に達するまで WebLogic Server への接続を繰り返し試みます。

動的サーバ リスト

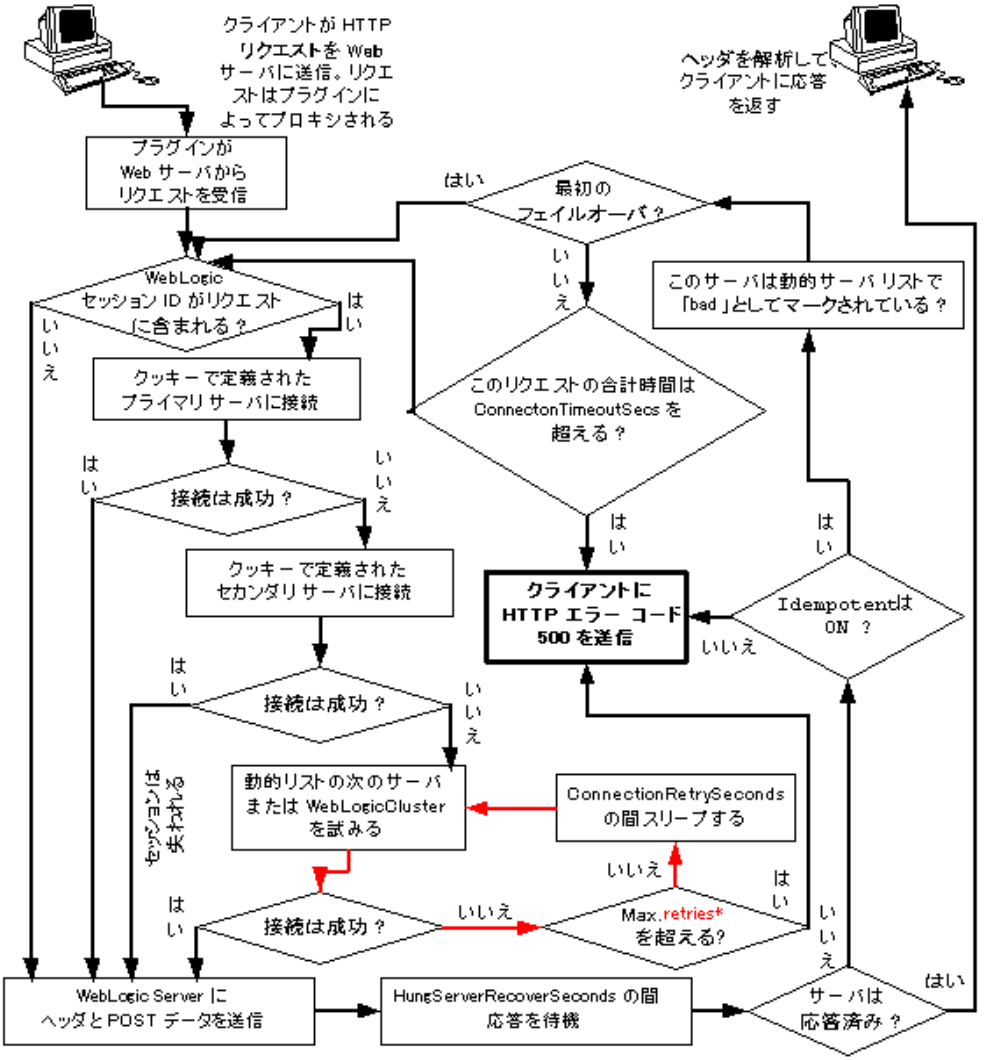
WebLogicCluster パラメータで WebLogic Server のリストを指定すると、プラグインではクラスタ メンバー間でのロード バランシングの起点としてそのリストが使用されます。最初のリクエストがそれらのサーバの 1 つに転送された後に、クラスタ内のサーバの更新されたリストを格納する動的サーバ リストが返されます。更新されたリストはクラスタ内の新しいサーバを追加し、すでにクラスタから外れているか、リクエストに応答できなかったサーバを削除します。このリストは、クラスタで変更が行われたときに HTTP 応答によって自動的に更新されます。

フェイルオーバ、クッキー、および HTTP セッション

リクエストがクッキー、POST データ、または URL エンコーディングを通じてセッション情報を格納している場合、そのセッション ID にはセッションが最初に確立された特定のサーバ（プライマリ サーバ）への参照と元のセッションがレプリケートされる追加サーバ（セカンダリ サーバ）への参照が含まれています。クッキーが含まれているリクエストは、プライマリ サーバに接続しようとします。その試行が失敗すると、リクエストはセカンダリ サーバに転送されます。プライマリ サーバとセカンダリ サーバが両方とも失敗すると、セッションが失われて、プラグインは動的クラスタ リストの別のサーバにあらためて接続しようとします。詳細については、図 13-1 （13-13 ページの「接続のフェイルオーバ」）を参照してください。

注意： POST データが 64K を超える場合、プラグインは、セッション ID を取得するための POST データの解析を行いません。したがって、セッション ID を POST データに格納した場合、プラグインはリクエストを正しいプライマリまたはセカンダリ サーバにルーティングできないので、セッション データが失われる可能性があります。

図 13-1　接続のフェイルオーバ



* 赤いループで許可される再試行の限度は、次の式で計算されます。
`ConnectTimeoutSecs ÷ ConnectRetrySecs.`

ファイアウォールとロード ディレクタを使用する場合のフェイルオーバの動作

ほとんどのコンフィグレーションでは、Netscape Enterprise Server プラグインはリクエストをクラスタのプライマリ インスタンスに送信します。そのインスタンスが利用できない場合、リクエストはセカンダリ インスタンスにフェイルオーバされます。ただし、ファイアウォールとロードディレクタを組み合わせて使う一部のコンフィグレーションでは、WebLogic Server のプライマリ インスタンスが利用できない場合でもどれか 1 つのサーバ（ファイアウォールまたはロードディレクタ）がリクエストを受け付けて、正常な接続を返すことができます。利用できない WebLogic Server のプライマリ インスタンスに送信しようとしたリクエストは、プラグインに「connection reset」として返されます。

ファイアウォールの組合せで実行するリクエストは、ロードディレクタの有無に関係なく、WebLogic Server が処理します。つまり、`connection reset` という応答は、WebLogic Server のセカンダリ インスタンスにフェイルオーバします。`connection reset` という応答がこれらのコンフィグレーションではフェイルオーバするため、サーブレットは多重呼び出し不変でなければなりません。それ以外の場合、トランザクションの重複処理になる可能性があります。

obj.conf ファイルのサンプル（WebLogic クラスタを使用しない場合）

次に示すのは、クラスタを使用しない場合に obj.conf ファイルに追加する行の例です。この例をテンプレートとして使用して、ユーザの環境およびサーバに合うように変更できます。# で始まる行はコメントです。

注意： obj.conf ファイルでは、意味のないスペースを挿入しないようにしてください。このサンプルをコピーして貼り付けると、余分なスペースが挿入されて、ファイルを読み取るときに問題が生じることがあります。

Enterprise Server のコンフィグレーション ファイルに関するマニュアルはすべて、Netscape Enterprise Server プラグインのマニュアルで参照できます。

```
## -------- ここから OBJ.CONF コンフィグレーションのサンプル --------
#（クラスタなし）

# 以下の行は、起動時にロードする NSAPI ライブラリを
# 指定し、ライブラリ内のどの関数が NSAPI 関数かを
# 示す。ライブラリへのパス（shlib=<...> パラメータの
# 値を検証し、ファイルが読み取り可能であることを確認する
# 読みとり可能でない場合、サーバは起動に失敗する

Init fn="load-modules" funcs="wl_proxy,wl_init"\
 shlib=/usr/local/netscape/plugins/libproxy.so
Init fn="wl_init"

# NSAPI モジュール（さらに WebLogic）で処理する
# HTTP リクエストの種類をコンフィグレーションする。これは
# 以下の例に示すように、1 つまたは複数の「<Object>」タグを使用して指定する

# ここでは、NSAPI モジュールをコンフィグレーションして
# 「/weblogic」のリクエストを、ホスト myweblogic.server.com の
# ポート 7001 でリスニングする WebLogic Server に渡す

<Object name="weblogic" ppath="*/weblogic/*">
Service fn=wl_proxy WebLogicHost=myweblogic.server.com\
 WebLogicPort=7001 PathTrim="/weblogic"
</Object>

# ここでは、プラグインをコンフィグレーションして
# 「/servletimages/」に一致するリクエストが
# plug-in/WebLogic で処理されるようにする

<Object name="si" ppath="*/servletimages/*">
Service fn=wl_proxy WebLogicHost=192.192.1.4 WebLogicPort=7001
</Object>
```

```
# この Object ディレクティブは、リクエストのパスではなく
# ファイル拡張子で機能する。このコンフィグレーションを使用するには、
# 次の mime.types ファイルにも行を追加する必要がある
#
# type=text/jsp              exts=jsp
#
# このコンフィグレーションでは、「.jsp」の拡張子が付いたファイルが
# WebLogic にプロキシ送信される。次に、
# この拡張子の Service 行を Object 定義「default」に追加する
# この定義は obj.conf ファイルに既に存在している必要がある

<Object name=default>
NameTrans fn=pfx2dir from=/ns-icons\
 dir="c:/Netscape/SuiteSpot/ns-icons"
NameTrans fn=pfx2dir from=/mc-icons\
 dir="c:/Netscape/SuiteSpot/ns-icons"
NameTrans fn="pfx2dir" from="/help" dir=\
 "c:/Netscape/SuiteSpot/manual/https/ug"
NameTrans fn=document-root root="c:/Netscape/SuiteSpot/docs"
Service method="(GET|HEAD|POST|PUT)" type=text/jsp fn=wl_proxy\
 WebLogicHost=localhost WebLogicPort=7001 PathPrepend=/jspfiles
PathCheck fn=nt-uri-clean
PathCheck fn="check-acl" acl="default"
PathCheck fn=find-pathinfo
PathCheck fn=find-index index-names="index.html,home.html"
ObjectType fn=type-by-extension
ObjectType fn=force-type type=text/plain
Service method=(GET|HEAD) type=magnus-internal/imagemap\
 fn=imagemap
Service method=(GET|HEAD)\
 type=magnus-internal/directory fn=index-common
Service method=(GET|HEAD) type=*~magnus-internal/* fn=send-file
AddLog fn=flex-log name="access"
</Object>

# 次のディレクティブは、NSAPI プラグインを介して
# WebLogic プロコトルの HTTP トンネリングを有効にする

<Object name="tunnel" ppath="*/HTTPClnt*">
Service fn=wl_proxy WebLogicHost=192.192.1.4 WebLogicPort=7001
</Object>

#
## ------- ここまで OBJ.CONF コンフィグレーションのサンプル -------
```

obj.conf ファイルのサンプル（WebLogic クラスタを使用する場合）

次に示すのは、WebLogic Server のクラスタを使用する場合に obj.conf ファイルに追加する行の例です。この例をテンプレートとして使用して、ユーザの環境およびサーバに合うように変更できます。# で始まる行はコメントです。

注意： obj.conf ファイルでは、意味のないスペースを挿入しないようにしてください。このサンプルをコピーして貼り付けると、余分なスペースが挿入されて、ファイルを読み取るときに問題が生じることがあります。

詳細については、Netscape から提供される Enterprise Server のコンフィグレーション ファイルに関するマニュアルを参照してください。

```
## -------- ここから OBJ.CONF コンフィグレーションのサンプル --------
# (WebLogic クラスタを使用)
#
# 以下の行は、起動時にロードする NSAPI ライブラリを
# 指定し、ライブラリ内のどの関数が NSAPI 関数かを
# 示す。ライブラリへのパス（shlib=<...> パラメータの
# 値を検証し、ファイルが読み取り可能であることを確認する。
# 読み取り可能でない場合、サーバは起動に失敗する

Init fn="load-modules" funcs="wl_proxy,wl_init"\
 shlib=/usr/local/netscape/plugins/libproxy.so
Init fn="wl_init"

# NSAPI モジュール（さらに WebLogic）で処理する
# HTTP リクエストの種類をコンフィグレーションする。これは
# 以下の例に示すように、1 つまたは複数の「<Object>」タグを使用して指定する

# ここでは、NSAPI モジュールをコンフィグレーションして
# 「/weblogic」のリクエストを WebLogic Server のクラスタに渡す

<Object name="weblogic" ppath="*/weblogic/*">
Service fn=wl_proxy\
 WebLogicCluster="myweblogic.com:7001,yourweblogic.com:7001,\
 theirweblogic.com:7001" PathTrim="/weblogic"
</Object>

# ここでは、プラグインをコンフィグレーションして
# 「/servletimages/」に一致するリクエストが
# plug-in/WebLogic で処理されるようにする

<Object name="si" ppath="*/servletimages/*">
Service fn=wl_proxy\
```

```
 WebLogicCluster="myweblogic.com:7001,yourweblogic.com:7001,\
 theirweblogic.com:7001"
</Object>

# この Object ディレクティブは、リクエストのパスではなく
# ファイル拡張子で機能する。このコンフィグレーションを使用するには、
# 次の mime.types ファイルにも行を追加する必要がある
#
# type=text/jsp             exts=jsp
#
# このコンフィグレーションでは、「.jsp」の拡張子が付いたファイルが
# WebLogic にプロキシ送信される。次に、
# この拡張子の Service 行を Object 定義「default」に追加する
# この定義は obj.conf ファイルに既に存在している必要がある

<Object name=default>
NameTrans fn=pfx2dir from=/ns-icons\
 dir="c:/Netscape/SuiteSpot/ns-icons"
NameTrans fn=pfx2dir from=/mc-icons\
 dir="c:/Netscape/SuiteSpot/ns-icons"
NameTrans fn="pfx2dir" from="/help" dir=\
 "c:/Netscape/SuiteSpot/manual/https/ug"
NameTrans fn=document-root root="c:/Netscape/SuiteSpot/docs"
Service method="(GET|HEAD|POST|PUT)" type=text/jsp fn=wl_proxy\
 WebLogicCluster="myweblogic.com:7001,yourweblogic.com:7001,\
 theirweblogic.com:7001",PathPrepend=/jspfiles
PathCheck fn=nt-uri-clean
PathCheck fn="check-acl" acl="default"
PathCheck fn=find-pathinfo
PathCheck fn=find-index index-names="index.html,home.html"
ObjectType fn=type-by-extension
ObjectType fn=force-type type=text/plain
Service method=(GET|HEAD) type=magnus-internal/imagemap\
 fn=imagemap
Service method=(GET|HEAD)\
 type=magnus-internal/directory fn=index-common
Service method=(GET|HEAD) type=*~magnus-internal/* fn=send-file
AddLog fn=flex-log name="access"
</Object>

# 次のディレクティブは、NSAPI プラグインを介して
# WebLogic プロコトルの HTTP トンネリングを有効にする

<Object name="tunnel" ppath="*/HTTPClnt*">
Service fn=wl_proxy WebLogicCluster="myweblogic.com:7001,\
 yourweblogic.com:7001,theirweblogic.com:7001"
</Object>

#
## ------- ここまで OBJ.CONF コンフィグレーションのサンプル -------
```

14 セキュリティの管理

以下の節では、WebLogic Server でセキュリティを実装する方法について説明します。

- セキュリティのコンフィグレーション手順
- システム パスワードの変更
- セキュリティ レルムの指定
- ユーザの定義
- グループの定義
- ACL の定義
- SSL プロトコルのコンフィグレーション
- 相互認証のコンフィグレーション
- SSL を使用した RMI over IIOP のコンフィグレーション
- パスワードの保護
- 監査プロバイダのインストール
- 接続フィルタのインストール
- Java セキュリティ マネージャの設定
- セキュリティ コンテキストの伝播のコンフィグレーション

セキュリティのコンフィグレーション手順

WebLogic Server のデプロイメントのセキュリティは、主にそのデプロイメントに合わせたセキュリティ ポリシーを定義する属性をコンフィグレーションすることで実装します。WebLogic Server には、デプロイメントのセキュリティ ポリシーを定義するための Administration Console が用意されています。Administration Console を使用して、デプロイメントの次の要素にセキュリティ固有の値を指定します。

- セキュリティ レルム
- ユーザとグループ
- WebLogic Server のリソースのアクセス制御リスト（ACL）およびパーミッション
- SSL プロトコル
- 相互認証
- ホスト名検証
- 監査プロバイダ
- カスタム フィルタ
- セキュリティ コンテキストの伝播

セキュリティ機能は互いに関連しているので、セキュリティをコンフィグレーションする場合に何から始めるべきか判断しにくいものです。実際、WebLogic Server のデプロイメントのセキュリティを定義する場合には、同じ作業を繰り返すこともあります。手順は 1 通りではありませんが、次の手順に従うことをお勧めします。

1. `system` ユーザのパスワードを変更して、WebLogic Server のデプロイメントを保護します。「システム パスワードの変更」を参照してください。
2. セキュリティ レルムを指定します。WebLogic Server のセキュリティ レルムは、デフォルトでファイル レルムです。ただし、代替セキュリティ レルムやカスタム セキュリティ レルムを指定することもできます。「セキュリティ レルムの指定」を参照してください。

3. セキュリティ レルムのユーザを定義します。セキュリティ レルムにグループを実装すると、ユーザを組織できます。「ユーザの定義」を参照。
4. WebLogic Server のデプロイメントのリソースの ACL およびパーミッションを定義します。「ACL の定義」を参照してください。
5. SSL プロトコルを実装することで、クライアントと WebLogic Server との間のネットワーク接続を保護します。SSL を実装すると、WebLogic Server は信頼された認証局によって発行されたデジタル証明書を使用して、クライアントを認証します。この手順は省略可能ですが、実行することをお勧めします。「SSL プロトコルのコンフィグレーション」を参照してください。
6. 相互認証を実装することで、WebLogic Server デプロイメントをさらに保護します。相互認証を実装すると、WebLogic Server はクライアントに対して自己認証してから、クライアントを認証し、WebLogic Server に対して自己認証しなければなりません。この手順も省略可能ですが、実行することをお勧めします。「相互認証のコンフィグレーション」を参照してください。

WebLogic Server のセキュリティ機能の詳細については、「WebLogic Security の概要」と「セキュリティの基礎概念」を参照してください。

注意： この節のコンフィグレーション手順はすべて Administration Console を使用して行います。

セキュリティ ロールの WebLogic EJB への割り当てについては、「WebLogic Server 6.1 デプロイメント プロパティ」を参照してください。

WebLogic Web アプリケーションでのセキュリティの詳細については、『Web アプリケーションのアセンブルとコンフィグレーション』を参照してください。

システム パスワードの変更

インストール中に、`system` ユーザのパスワードを指定します。指定されたパスワードは、WebLogic Server の `system` ユーザに関連付けられ、`\wlserver6.1\config\`*`domain`* ディレクトリの `fileRealm.properties` ファイルに保存されます。*`domain`* は、インストール時に WebLogic 管理ドメイン名として指定された名前です。指定されたパスワードは、ドメインの管理サーバと、その管理サーバに関連付けられているすべての管理対象サーバに対応しています。

注意： system ユーザは、WebLogic Server を起動できる唯一のユーザ アカウントです。

system ユーザのパスワードは、WebLogic Server がハッシュを適用するときに暗号化され、さらに保護されます。セキュリティを強化するため、インストール時に設定したシステム パスワードを頻繁に変更することをお勧めします。WebLogic Server の各デプロイメントでは、ユニークなパスワードが必要です。

システム パスワードを変更するには、次の手順に従います。

1. Administration Console の [セキュリティ | ユーザ] をクリックして [ユーザ] を開きます。
2. [ユーザ パスワードの変更] の [名前] 属性フィールドで system と入力します。
3. WebLogic Server のインストール時に指定したパスワードを [古いパスワード] 属性フィールドに入力します。
4. [新しいパスワード] 属性フィールドに新しいパスワードを入力します。
5. [パスワードの確認] 属性フィールドに新しいパスワードを再び入力します。

あるドメインの管理サーバと管理対象サーバを使用する場合、管理対象サーバは常にそのドメインの管理サーバのパスワードを使用しなければなりません。管理サーバのパスワードは、常に Administration Console を使用して変更してください。新しいパスワードは、そのドメインのすべての管理対象サーバに伝達されます。

WebLogic パスワードの機密性を維持することは、WebLogic Server のデプロイメントとデータの安全性を確保する上で極めて重要です。デプロイメントとデータを保護するために、WebLogic Server のパスワードの機密性を維持することをお勧めします。

セキュリティ レルムの指定

この節では、WebLogic Server デプロイメントのセキュリティ レルムをコンフィグレーションする方法について説明します。セキュリティ レルムの概要と WebLogic Server での使用方法については、『WebLogic Security プログラマーズ ガイド』の「セキュリティ レルム」を参照してください。以下の節では、セキュリティ レルムの指定について説明します。

- ファイル レルムのコンフィグレーション
- キャッシング レルムのコンフィグレーション
- LDAP セキュリティ レルムのコンフィグレーション
- Windows NT セキュリティ レルムのコンフィグレーション
- UNIX セキュリティ レルムのコンフィグレーション
- RDBMS セキュリティ レルムのコンフィグレーション
- カスタム セキュリティ レルムのインストール
- セキュリティ レルムの移行

ファイル レルムのコンフィグレーション

WebLogic Server のセキュリティ レルムは、デフォルトでファイル レルムです。ファイル レルムを使用する前に、ファイル レルムの使用を管理する属性を定義する必要があります。これらの属性は、Administration Console の [セキュリティ] ウィンドウの [ファイル レルム] タブで設定します。

次の表では、[ファイル レルム] タブの各属性について説明します。

表 14-1 ファイル レルムの属性

属性	説明
[キャッシング レルム]	使用されるキャッシング レルムの名前。 ■ ファイル レルムを使用するときには、この属性を [（なし）] に設定しなければならない。 ■ 代替セキュリティ レルムまたはカスタム セキュリティ レルムを使用する場合は、この属性を、使用するキャッシング レルムの名前に設定する。コンフィグレーションされているキャッシング レルムのリストはプルダウン メニューに表示される。
[最大ユーザ数]	ファイル レルムで使用するユーザの最大数。ファイル レルムのユーザ数は 10,000 以下。この属性の最小値は 1、最大値は 10,000、デフォルト値は 1,000。
[最大グループ数]	ファイル レルムで使用するグループの最大数。この属性の最小値は 1、最大値は 10,000、デフォルト値は 1,000。
[最大 ACL]	ファイル レルムで使用する ACL の最大数。この属性の最小値は 1、最大値は 10,000、デフォルト値は 1,000。

警告： `fileRealm.properties` ファイルが壊れたら、WebLogic Server のセキュリティ情報を再コンフィグレーションしなければなりません。WebLogic Server は、`fileRealm.properties` ファイルがないと起動できません。

`fileRealm.properties` ファイルには、WebLogic Server を起動するためのデフォルト ACL が含まれています。カスタム セキュリティ レルムは起動シーケンスでは呼び出されないので、カスタム セキュリティ レルムを記述する場合でも、WebLogic Server を起動するために `fileRealm.properties` ファイルは必要となります。

したがって、次の手順を実行することをお勧めします。

`fileRealm.properties` ファイルのバックアップを作成し、安全な場所に保管します。

WebLogic Server デプロイメントの管理者は読み書き特権を持ち、その他のユーザは何の特権も持たないように、fileRealm.properties ファイルにパーミッションを設定します。

注意： また、ファイル レルム用の SerializedSystemIni.dat ファイルのバックアップも作成する必要があります。SerializedSystemIni.dat ファイルの詳細については、「パスワードの保護」を参照してください。

ファイル レルムの代わりに WebLogic Server で提供される代替セキュリティ レルムまたはカスタム セキュリティ レルムを使用する場合は、必要なレルムに合わせて属性を設定し、WebLogic Server を再起動します。代替セキュリティ レルムを使用する場合は、キャッシング レルムを有効にする必要があります。

WebLogic Server のセキュリティ レルムの詳細については、「セキュリティ レルム」を参照してください。

キャッシング レルムのコンフィグレーション

キャッシング レルムはファイル レルム、代替セキュリティ レルム、またはカスタム セキュリティ レルムと連携し、適切な認証および認可を得たクライアントのリクエストを遂行します。キャッシング レルムは、成功したレルム ルックアップと失敗したレルム ルックアップの両方の結果を格納します。キャッシング レルムは、ユーザ、グループ、パーミッション、ACL、および認証リクエストのキャッシュを別々に管理します。キャッシング レルムによって、ルックアップがキャッシュされ、ほかのセキュリティ レルムへの呼び出し数が減るので、WebLogic Server のパフォーマンスが向上します。WebLogic Server のセキュリティ レルムの詳細については、「セキュリティ レルム」を参照してください。

キャッシング レルムは、WebLogic Server のインストール時に自動的にインストールされます。キャッシュはその他のセキュリティ レルムに権限を委託するよう設定されますが、キャッシュは有効化されていません。Administration Console を使用して、キャッシュを有効化する必要があります。代替セキュリティ レルムまたはカスタム セキュリティ レルムを使用する場合は、キャッシング レルムをコンフィグレーションして有効にする必要があります。

キャッシュを有効化すると、キャッシング レルムによって、レルム ルックアップの結果がキャッシュに保存されます。ルックアップの結果は、存続時間（TTL）属性に定義された秒数が経過する（ルックアップの結果の有効期限が切

れる）か、またはキャッシュがいっぱいになるまで、キャッシュ内に残ります。キャッシュがいっぱいになると、ルックアップの結果はキャッシュ内で最も古い結果と置き換えられます。TTL 属性によって、キャッシュされたオブジェクトの有効期間が決定されます。これらの属性に設定する値が大きいほど、キャッシング レルムが二次セキュリティ レルムを呼び出す回数が減ります。呼び出し回数が減ると、パフォーマンスは向上します。パフォーマンスが向上する代わりに、基のセキュリティ レルムへの変更は、キャッシュされたオブジェクトの有効期間が切れるまで認識されません。

注意： セキュリティ レルムからオブジェクトを取得した場合、オブジェクトはオブジェクトのスナップショットを反映しています。オブジェクトを更新するには、そのオブジェクトの `get()` メソッドをもう一度呼び出します。たとえば、グループのメンバシップは、`getGroup()` メソッドを呼び出してセキュリティ レルムからグループを取得したときに設定されます。グループのメンバを更新するには、`getGroup()` メソッドをもう一度呼び出さなければなりません。

デフォルトでは、キャッシング レルムは、代替セキュリティ レルムが大文字 / 小文字を区別することを前提にして処理します。大文字 / 小文字を区別するセキュリティ レルムでは、たとえば `bill` というユーザ名のオーナと `Bill` というユーザ名のオーナは、別々のユーザとして扱われます。大文字 / 小文字を区別しないセキュリティ レルムの例として、Windows NT セキュリティ レルムと LDAP セキュリティ レルムが挙げられます。大文字 / 小文字を区別しないセキュリティ レルムを使用する場合は、[キャッシュで大文字 / 小文字を区別] 属性を無効化しなければなりません。この属性を設定すると、キャッシング レルムでは、大文字 / 小文字を区別して比較した場合に WebLogic Server がセキュリティ レルムの正しい結果を返すように、ユーザ名が小文字に変換されます。大文字 / 小文字を区別するセキュリティ レルムのユーザまたはグループを定義したり参照したりする場合には、ユーザ名を小文字で入力します。

キャッシング レルムをコンフィグレーションするには、次の操作を行います。

1. Administration Console の左ペインで [セキュリティ｜キャッシング レルム] ノードを選択します。
2. Administration Console の右ペインで、[新しい Caching Realm のコンフィグレーション] リンクをクリックします。
3. [キャッシング レルム] ウィンドウの [コンフィグレーション] タブにある [一般] タブで属性を定義します。

   次の表では、[一般] タブで設定する属性について説明します。

表 14-2 [一般] タブのキャッシング レルム属性

属性	説明
[名前]	Administration Console で定義されているアクティブなセキュリティ レルムを表示する。この属性は変更できない。
[基本レルム]	キャッシング レルムと一緒に使用されている代替セキュリティ レルムまたはカスタム セキュリティ レルムのクラス名。コンフィグレーションされているレルムの名前はプルダウン メニューに表示される。
[キャッシュで大文字 / 小文字を区別]	指定されたセキュリティ レルムで大文字 / 小文字を区別するかどうかを定義する。デフォルトでは、この属性は有効、つまり、レルムで大文字 / 小文字を区別する。大文字 / 小文字を区別しないセキュリティ レルム（Windows NT および LDAP セキュリティ レルムなど）を使用するには、この属性を無効化しなければならない。

4. [作成] をクリックします。
5. [キャッシング レルム] ウィンドウの [コンフィグレーション] タブにある [ACL] タブの属性に値を定義して、ACL キャッシュをコンフィグレーションして有効化します。

   次の表では、[ACL] タブで設定する属性について説明します。

表 14-3 ACL キャッシュの属性

属性	説明
[ACL キャッシュを有効化]	ACL キャッシュを有効化するためのオプション。

表 14-3 ACL キャッシュの属性（続き）

属性	説明
[ACL キャッシュサイズ]	キャッシュする ACL ルックアップの最大数。ルックアップのパフォーマンスを最大限に引き出すには、この属性は素数でなければならない。デフォルトでは 211。
[成功時の ACL キャッシュ生存時間]	成功したルックアップの結果を保持する秒数。デフォルトでは 60 秒。
[失敗時の ACL キャッシュ生存時間]	失敗したルックアップの結果を保持する秒数。デフォルトでは 10 秒。

6. 変更を保存するには、[適用] ボタンをクリックします。
7. 認証キャッシュを有効化してコンフィグレーションするには、[キャッシング レルム] ウィンドウの [コンフィグレーション] タブにある [認証] タブの属性に値を定義します。

   次の表では、[認証] タブで設定する属性について説明します。

表 14-4 認証キャッシュの属性

属性	説明
[認証キャッシュを有効化]	認証キャッシュを有効化するためのオプション。
[認証キャッシュ サイズ]	キャッシュする認証リクエストの最大数。ルックアップのパフォーマンスを最大限に引き出すには、この属性は素数でなければならない。デフォルトでは 211。
[成功時の認証キャッシュ生存時間]	成功したルックアップの結果を保持する秒数。デフォルトでは 60 秒。
[失敗時の認証キャッシュ生存時間]	失敗したルックアップの結果を保持する秒数。デフォルトでは 10 秒。

8. 変更を保存するには、[適用] ボタンをクリックします。
9. グループ キャッシュを有効化してコンフィグレーションするには、[キャッシング レルム] ウィンドウの [コンフィグレーション] タブにある [グループ] タブの属性に値を定義します。

   次の表では、[グループ] タブで設定する属性について説明します。

表 14-5 グループ キャッシュの属性

属性	説明
[グループ キャッシュを有効化]	グループ キャッシュを有効化するためのオプション。
[グループ キャッシュ サイズ]	キャッシュするグループ ルックアップの最大数。ルックアップのパフォーマンスを最大限に引き出すには、この属性は素数でなければならない。デフォルトでは 211。
[成功時のグループ キャッシュ生存時間]	成功したルックアップの結果を保持する秒数。デフォルトでは 60 秒。
[失敗時のグループ キャッシュ生存時間]	失敗したルックアップの結果を保持する秒数。デフォルトでは 10 秒。
[グループ メンバシップ キャッシュ生存時間]	更新前にグループのメンバを保存する秒数。デフォルトでは 300 秒。

10. 変更を保存するには、[適用] ボタンをクリックします。
11. ユーザ キャッシュを有効化してコンフィグレーションするには、[キャッシング レルム] ウィンドウの [コンフィグレーション] タブにある [ユーザ] タブの属性に値を定義します。

    次の表では、[ユーザ] タブで設定する属性について説明します。

表 14-6 ユーザ キャッシュの属性

属性	説明
[ユーザ キャッシュを有効化]	ユーザ キャッシュを有効化するためのオプション。
[ユーザ キャッシュ サイズ]	キャッシュするユーザ ルックアップの最大数。ルックアップのパフォーマンスを最大限に引き出すには、この属性は素数でなければならない。デフォルトでは 211。
[成功時のユーザ キャッシュ生存時間]	成功したルックアップの結果を保持する秒数。デフォルトでは 60 秒。
[失敗時のユーザ キャッシュ生存時間]	失敗したルックアップの結果を保持する秒数。デフォルトでは 10 秒。

12. 変更を保存するには、[適用] ボタンをクリックします。

13. パーミッション キャッシュを有効化してコンフィグレーションするには、[キャッシング レルム] ウィンドウの [コンフィグレーション] タブにある [パーミッション] タブの属性に値を定義します。

    次の表では、[パーミッション] タブの各属性について説明します。

表 14-7 パーミッション キャッシュの属性

属性	説明
[パーミッション キャッシュを有効化]	パーミッション キャッシュを有効化するためのオプション。
[パーミッション キャッシュ サイズ]	キャッシュするパーミッション ルックアップの最大数。ルックアップのパフォーマンスを最大限に引き出すには、この属性は素数でなければならない。デフォルトでは 211。
[成功時のパーミッション キャッシュ生存時間]	成功したルックアップの結果を保持する秒数。デフォルトでは 60 秒。

表 14-7 パーミッション キャッシュの属性（続き）

属性	説明
[失敗時のパーミッション キャッシュ生存時間]	失敗したルックアップの結果を保持する秒数。デフォルトでは 10 秒。

14. 変更を保存するには、[適用] ボタンをクリックします。

15. キャッシング レルムの属性を定義した後は、WebLogic Server を再起動します。

LDAP セキュリティ レルムのコンフィグレーション

LDAP セキュリティ レルムでは、Lightweight Directory Access Protocol（LDAP）サーバを使用して認証を行います。このサーバを使用すると、組織内のすべてのユーザを LDAP ディレクトリだけで管理できます。LDAP セキュリティ レルムは、Open LDAP、Netscape iPlanet、Microsoft Site Server、および Novell NDS をサポートしています。

このリリースの WebLogic Server では、LDAP セキュリティ レルムを以下の 2 つのバージョンから選択できます。

- LDAP レルム V1 － WebLogic Server の以前のリリースでパッケージ化された LDAP セキュリティ レルムです。Microsoft Site Server という例外を除いて、LDAP セキュリティ レルム V1 はサポートされているすべての LDAP サーバと連係して機能し、WebLogic Server の旧リリースで LDAP セキュリティ レルムを使用しているユーザ向けに用意されています。ただし、LDAP レルム V1 はこのリリースでは非推奨になっているので、LDAP レルム V2 にアップグレードすることをお勧めします。
- LPAP レルム V2 － パフォーマンスが向上し、コンフィグレーションが容易になった LDAP セキュリティ レルムのアップグレード版です。これは、WebLogic Server 6.0 サービス パック 1.0 で提供されるものと同一の LDAP セキュリティ レルムです。LDAP レルム V2 は、`getUsers()` または `getGroups()` をサポートしていません。これらのリクエストを遂行するためにメモリを割り当てるとサービス拒否攻撃を受ける可能性があるからです。

これらの機能を使用する場合、LDAP レルム V1 を使用することをお勧めします。Windows 2000 を実行している場合は、LDAP レルム V2 を使用し、Windows 2000 ユーザおよびグループ ストアに照らして認証することをお勧めします。

注意： LDAP レルム V1 を使用する場合は、Administration Console を使用して LDAP ディレクトリ サーバに格納されているユーザおよびグループのメンバーを表示できます。ただし、LDAP レルム V2 を使用する場合は、LDAP ディレクトリ サーバに格納されているグループのみ Administration Console を使用して表示できます。

ユーザまたはグループの追加や削除、あるいはグループのメンバーの追加など、ユーザおよびグループを管理するためには、LDAP サーバで利用可能な管理ツールを使用する必要があります。LDAP ディレクトリ ストアで変更を行った場合は、ユーザ キャッシュおよびグループ キャッシュをリセットすると、Administration Console ですぐにその変更を表示できます。

LDAP セキュリティ レルムのパフォーマンスを向上させるためのヒントを次に示します。

- `ldaprealm.props` ファイルでフィルタを使用して、LDAP サーバから取得する結果セットをより具体的に絞り込みます（LDAP レルム V2 のみ）。
- LDAP サーバが LDAP レルム検索フィルタで検索キーとして使用するすべての属性にインデックスを付けるようにします。属性にインデックスを付けないと、パフォーマンスはリニア検索と同程度になります。
- キャッシング レルムを使用する場合には十分な注意を払います。LDAP サーバの情報の変更は、キャッシュがクリアされるまで、LDAP セキュリティ レルムに伝播されません。

LDAP セキュリティ レルムのコンフィグレーションでは、LDAP サーバと通信するために LDAP セキュリティ レルムを WebLogic Server で有効化する属性と、ユーザおよびグループを LDAP ディレクトリに保存する方法を指定する属性を定義します。LDAP ツリーおよびスキーマは、LDAP サーバごとに異なります。したがって、LDAP レルム V2 では、サポートされている LDAP サーバのデフォルト属性を定義するテンプレートのセットが提供されます。

LDAP セキュリティ レルム使用時の制限

LDAP セキュリティ レルムには以下の制限があります。

- Microsoft Site Server の LDAP サーバがインストールされ、LDAP ディレクトリのルートが作成されると、デフォルトによっていくつかの組織単位が作成されます。グループの下には、`Administrators` という空のデフォルト グループを持つ `NTGroups` というデフォルトの組織単位があります。デフォルトでは、WebLogic Server でも、`System`（WebLogic Server が起動されるユーザ）というメンバーが含まれる `Administrators` というグループが提供されます。Microsoft Site Server でデフォルトを使用し、デフォルト組織単位の下で独自のグループを作成し始めると、WebLogic Server は起動しなくなります。LDAP セキュリティ レルムを使用して WebLogic Server を起動するためには、LDAP ディレクトリで独自のユニークな組織単位を作成し、その組織単位の下で WebLogic Server デプロイメントのグループを作成する必要があります。
- LDAP ディレクトリに同じ名前のグループが 2 つある場合、WebLogic Server ではその 2 番目のグループでユーザを正しく認証できません。LDAP セキュリティ レルムでは、グループの識別名（DN）を使用して LDAP ディレクトリでグループを検索します。複数のグループを同じ名前で作成する場合、WebLogic Server は最初に発見したグループのユーザだけを認証します。LDAP セキュリティ レルムを使用するときは、ユニークなグループ名を使用する必要があります。
- LDAP レルム V2 では、LDAP レルム V1 で提供される以下の機能を利用できません。
  - すべてのユーザのリスト表示
  - グループのメンバーのリスト表示
  - authProtocol メカニズムと userAuthentication メカニズム。LDAP サーバにセキュリティ資格を渡すには、JNDI バインド メカニズムを使用する必要があります。

LDAP レルム V1 のコンフィグレーション

ファイル レルムの代わりに LDAP セキュリティ レルム V1 を使用するには、次の操作を行います。

1. Administration Console の左ペインで [セキュリティ｜レルム] ノードを選択します。
2. Administration Console の右ペインで、[新しい LDAP Realm V1 (Deprecated) のコンフィグレーション] リンクをクリックします。

   LDAP セキュリティ レルムを実装するクラスの名前が表示されます。
3. [作成] をクリックします。
4. LDAP サーバと WebLogic Server との通信を有効化するには、[新しい LDAP Realm の作成] ウィンドウの [LDAP レルム V1（非推奨）] タブの属性に値を定義します。

   次の表では、[LDAP レルム V1（非推奨）] タブで設定する属性について説明します。

表 14-8 [LDAP レルム V1（非推奨）] タブの LDAP セキュリティ レルムの属性

属性	説明
[LDAP URL]	LDAP サーバの場所。URL を、LDAP サーバが実行されているコンピュータの名前とリスンしているポートの番号に変更する。次に例を示す。`ldap://ldapserver:385` SSL プロトコルを使用して WebLogic Server を LDAP サーバと接続する場合は、URL に LDAP サーバの SSL ポートを指定する。
[プリンシパル]	WebLogic Server が LDAP サーバとの接続に使用する LDAP ユーザの識別名（Distinguished Name: DN）。このユーザは LDAP ユーザおよびグループをリストできなければならない。
[証明]	[プリンシパル] 属性に定義された、LDAP ユーザの認証用パスワード。

表 14-8 [LDAP レルム V1（非推奨）] タブの LDAP セキュリティ レルムの属性

属性	説明
[SSL の有効化]	LDAP サーバと WebLogic Server との通信を保護するために SSL プロトコルを使用できるようにするためのオプション。次のガイドラインに留意する。 ■ LDAP サーバが SSL プロトコルを使用するようコンフィグレーションされていない場合は、この属性を無効化する。 ■ [ユーザ] タブで [ユーザ認証] 属性を `external` に設定した場合は、この属性を有効にしなければならない。
[認証プロトコル]	LDAP サーバの認証に使用する認証のタイプ。この属性を次のいずれかの値に設定する。 ■ 認証を行わない場合の [(none)] ■ パスワード認証を行う場合の [simple] ■ 証明書認証用の [CRAM-MD5] Netscape iPlanet は `CRAM-MD5` をサポートしている。Microsoft Site Server、Netscape iPlanet、および OpenLDAP と Novell NDS は `Simple` をサポートしている。

5. 変更を保存するには、[適用] ボタンをクリックします。
6. LDAP ディレクトリにユーザを保存する方法を指定するには、[新しい LDAP Realm の作成] ウィンドウの [ユーザ] タブの属性に値を定義します。

次の表では、[ユーザ] タブで設定する属性について説明します。

表 14-9 [ユーザ] タブの LDAP セキュリティ レルムの属性

属性	説明
[ユーザ認証]	ユーザを認証するための方法を決定する。この属性を次のいずれかの値に設定する。 ■ [bind] に設定すると、LDAP セキュリティ レルムは LDAP サーバのパスワードなどのユーザ データを取得し、WebLogic Server でそのパスワードをチェックする。 ■ [external] に設定すると、LDAP セキュリティ レルムでは、LDAP サーバを、WebLogic Server クライアントから提供されるユーザ名およびパスワードとバインドすることでユーザを認証する。External に設定した場合は、SSL プロトコルを使用しなければならない。 ■ [local] に設定すると、LDAP セキュリティ レルムは LDAP ディレクトリで UserPassword プロパティを参照し、WebLogic Server のパスワードと照らし合わせることによってユーザを認証する。
[ユーザ パスワード属性]	[ユーザ認証] 属性が Local に設定されている場合は、この属性を使用してどの LDAP プロパティが LDAP ユーザのパスワードを格納しているのかを確認する。
[ユーザ DN]	[ユーザ名属性] 属性と組み合わされた場合に、LDAP ユーザをユニークに識別する属性とその値のリスト。
[ユーザ名属性]	LDAP ユーザのログイン名。この属性の値には LDAP ユーザの共通名を使用できるが、一般には共通名などの短縮した文字列を使用する。

7. 変更を保存するには、[適用] ボタンをクリックします。
8. LDAP ディレクトリにグループを保存する方法を指定するには、[新しい LDAP Realm の作成] ウィンドウの [グループ] タブの属性に値を定義します。

   次の表では、[グループ] タブで設定する属性について説明します。

表 14-10 [グループ] タブの LDAP セキュリティ レルムの属性

属性	説明
[グループ DN]	[グループ名属性] 属性と組み合わされた場合に、LDAP ディレクトリ内のグループをユニークに識別する属性と値のリスト。
[グループ名属性]	LDAP ディレクトリ内のグループの名前。通常は普通の名前。
[グループはコンテキスト]	LDAP ディレクトリにグループ メンバシップを記録する方法を指定する Boolean チェックボックス。 ■ 各グループが 1 ユーザを含む場合はこのチェックボックスをチェックする。デフォルトでは、このボックスは選択されている。 ■ 1 つのグループ エントリが各グループ メンバの属性を含む場合はこのチェックボックスのチェックをはずす。
[グループ ユーザ名属性]	グループ エントリ内でグループ メンバを格納する LDAP 属性の名前。

9. 変更を保存するには、[適用] ボタンをクリックします。
10. すべての属性の定義が終わったら、WebLogic Server を再起動します。
11. キャッシング レルムをコンフィグレーションします。詳細については、「キャッシング レルムのコンフィグレーション」を参照してください。

    キャッシング レルムをコンフィグレーションするときには、[一般] タブの [基本レルム] 属性のプルダウン メニューから LDAP レルム V1 を選択しま

す。[基本レルム] 属性では、キャッシング レルムと代替セキュリティ レルム（この場合は LDAP レルム V1）の関連付けを定義します。

12. [セキュリティ] ノードに移動します。

13. [ファイル レルム] タブを選択します。

14. [キャッシング レルム] 属性で、LDAP セキュリティ レルムで使用するキャッシング レルムの名前を選択します。コンフィグレーションされているキャッシング レルムのリストはプルダウン メニューに表示されます。

15. WebLogic Server を再起動します。

キャッシング レルムは、LDAP ディレクトリでのルックアップの回数を減らすためにユーザとグループを内部にキャッシュします。ユーザ キャッシュおよびグループ キャッシュ内の各オブジェクトには、キャッシング レルムをコンフィグレーションするときに設定する TTL 属性があります。LDAP ディレクトリ内で変更を加えた場合、キャッシュされているオブジェクトが有効期限切れになるか、フラッシュされるまで、それらの変更は LDAP セキュリティ レルムに反映されません。デフォルトの TTL は、ルックアップが失敗した場合に 60 秒、成功した場合に 10 秒です。ユーザ キャッシュとグループ キャッシュの TTL 属性を変更しないかぎり、LDAP ディレクトリ内の変更は 60 秒後に LDAP セキュリティ レルムに反映されます。

LDAP セキュリティ レルムで `getUser()` を呼び出すなどサーバ側コードで LDAP セキュリティ レルム内のルックアップを実行した場合、レルムから返されるオブジェクトは、コードで解放されるまで解放されません。したがって、WebLogic Server によって認証されたユーザは、LDAP ディレクトリから削除された場合でも、接続が持続しているかぎり有効なままです。

LDAP レルム V2 のコンフィグレーション

LDAP レルム V2 を使用する場合、WebLogic Server はサポートされている LDAP サーバのテンプレートを提供します。それらのテンプレートでは、サポートされている各 LDAP サーバでユーザとグループを表すために使用するデフォルトのコンフィグレーション情報を指定しています。使用する LDAP サーバに対応するテンプレートを選択して、LDAP サーバのホストとポート、および GroupDN、UserDN、Principal、Credential 属性を入力します。それらの属性については、「LDAP レルム V1 のコンフィグレーション」を参照してください。

LDAP セキュリティ レルム V2 を使用するには、次の操作を行います。

1. Administration Console の左ペインで [セキュリティ | レルム] ノードを選択します。
2. WebLogic Server と一緒に使用する LDAP サーバを選択します。以下のオプションがあります。
   - defaultLDAPRealmforOpenLDAPDirectoryServices
   - defaultLDAPRealmforNovellDirectoryServices
   - defaultLDAPRealmforMicrosoftSiteServer
   - defaultLDAPRealmforNetscapeDirectoryServer

   LDAP サーバを選択すると、そのサーバのコンフィグレーション ウィンドウが表示されます。
3. [コンフィグレーション情報] ボックスの `server.host` 属性および `server.port` 属性に LDAP サーバのホストとポートを入力します。
4. 必要に応じて、[コンフィグレーション情報] ボックスで LDAP ディレクトリ サーバの GroupDN、UserDN、Principal、および Credential 属性に定義されている情報を更新します。
5. 必要に応じて、LDAP サーバのパスワードを定義します。[パスワード] 属性では、Principal のパスワードを定義します。パスワードを定義すると、そのパスワードは WebLogic Server によって暗号化されます。
6. 変更を保存するには、[適用] ボタンをクリックします。
7. 属性の定義が終わったら、WebLogic Server を再起動します。
8. キャッシング レルムをコンフィグレーションします。詳細については、「キャッシング レルムのコンフィグレーション」を参照してください。

   キャッシング レルムをコンフィグレーションするときには、[一般] タブの [基本レルム] 属性のプルダウン メニューから LDAP レルム V2 を選択します。[基本レルム] 属性では、キャッシング レルムと代替セキュリティ レルム（この場合は LDAP レルム V2）の関連付けを定義します。
9. [セキュリティ] ノードに移動します。
10. [ファイル レルム] タブを選択します。

11. [キャッシング レルム] 属性で、LDAP セキュリティ レルムで使用するキャッシング レルムの名前を選択します。コンフィグレーションされているキャッシング レルムのリストはプルダウン メニューに表示されます。

12. WebLogic Server を再起動します。

Windows NT セキュリティ レルムのコンフィグレーション

Windows NT セキュリティ レルムでは、Windows NT ドメイン向けに定義されたアカウント情報を使用して、ユーザとグループを認証します。Windows NT セキュリティ レルム内のユーザおよびグループは Administration Console で表示できますが、ユーザおよびグループを管理する場合は Windows NT の機能を使用しなければなりません。

Windows NT セキュリティ レルムでは、（ユーザとグループの）認証は行えますが、（ACL の）認可を行うことはできません。WebLogic Server が使用する `filerealm.properties` ファイルの ACL 情報を更新するには、ACL を変更した後に [セキュリティ] ノードの [一般] タブで [更新] ボタンをクリックします。ACL でグループを使用すれば、WebLogic Server の情報を更新する回数を減らすことができます。Windows NT グループのメンバを変更すると、WebLogic Server リソースへの個々のユーザのアクセスを動的に管理できます。

Windows NT セキュリティ レルムを使用して、Windows 2000 Active Directory プライマリ ドメイン コントローラに照らし合わせて認証することは可能です。ただし、ドメイン コントローラ自体ではなく、ドメインのメンバーとなっているマシンから認証を行う必要があります。Windows NT セキュリティ レルムを実行するマシンが別のドメインのメンバーの場合、ローカルのユーザおよびグループ ストアを認証する方法はありません。

Windows NT セキュリティ レルムは、プライマリ ドメイン コントローラ、Windows NT ドメインのメンバーとなっているマシン、またはその Windows NT ドメインのメンバーとなっており、相互に信頼されたドメインを使用するマシンで実行可能です。

Windows NT セキュリティ レルムを使用するには、次の操作を行います。

1. Administration Console の左ペインで [セキュリティ｜レルム] ノードを選択します。

2. Administration Console の右ペインで、[新しい NTRealm のコンフィグレーション] リンクをクリックします。
3. Windows NT セキュリティ レルムのコンフィグレーションでは、レルムの名前と、Windows NT ドメインが実行されているコンピュータの名前を設定します。レルム名とコンピュータを指定するには、Administration Console の [新しい NT Realm の作成] ウィンドウの属性に値を定義します。

   次の表では、[新しい NT Realm の作成] ウィンドウの [コンフィグレーション] タブで設定する属性について説明します。

表 14-11 Windows NT セキュリティ レルムの属性

属性	説明
[名前]	AccountingRealm などの Windows NT セキュリティ レルムの名前。
[プライマリ ドメイン]	Windows NT ドメイン向けのユーザとグループが定義されたコンピュータのホストおよびポート番号。複数のホストとポート番号を入力する場合は、カンマ区切りのリストを使用する。

4. 変更を保存するには、[適用] ボタンをクリックします。
5. 属性の定義が終わったら、WebLogic Server を再起動します。
6. キャッシング レルムをコンフィグレーションします。詳細については、「キャッシング レルムのコンフィグレーション」を参照してください。

   キャッシング レルムをコンフィグレーションするときには、[一般] タブの [基本レルム] 属性のプルダウン メニューから Windows NT セキュリティ レルムを選択します。[基本レルム] 属性では、キャッシング レルムと代替セキュリティ レルム（この場合は Windows NT セキュリティ レルム）の関連付けを定義します。
7. [セキュリティ] ノードに移動します。
8. [ファイル レルム] タブを選択します。
9. [キャッシング レルム] 属性で、Windows NT セキュリティ レルムで使用するキャッシング レルムの名前を選択します。コンフィグレーションされているキャッシング レルムのリストはプルダウン メニューに表示されます。

10. WebLogic Server を再起動します。

次のコマンドを使用して、指定された Windows NT ユーザとして WebLogic Server を実行するための正しい特権を持っていることを確認します。

```
java weblogic.security.ntrealm.NTRealm username password
```

username と *password* は、WebLogic Server を実行する Windows NT アカウントのユーザ名とパスワードです。

このコマンドの出力によって、指定されたユーザ名とパスワードが適切に認証されたかどうかわかります。

コマンドの出力	意味
auth?poppy	入力されたユーザ名とパスワードは正しく認証された。
auth?null	入力されたユーザ名とパスワードは正しく認証されなかった。

テストの結果、WebLogic Server を実行するクライアントまたはユーザが Windows NT セキュリティ レルムを実行する特権を持っていないことがわかった場合、WebLogic Server を実行する Windows ユーザのパーミッション（権利と呼ばれる）を更新する必要があります。

Windows NT で権利を更新するには、次の手順に従います。

1. [プログラム｜管理ツール] を選択します。
2. [ユーザー マネージャ] を選択します。
3. [原則] メニューから [ユーザーの権利] オプションを選択します。
4. [高度なユーザー権利の表示] オプションをチェックします。
5. WebLogic Server を実行する Windows ユーザに次の権利を付与します。
   - [オペレーティング システムの一部として機能]
   - [トークン オブジェクトの作成]
   - [プロセス レベル トークンの置き換え]
6. WebLogic Server を実行する Windows ユーザが Administrators グループのメンバーであることを確認します。

7. Windows NT を再起動して、すべての変更を有効にします。
8. [Logon as System Account] オプションがチェックされていることを確認します。[Allow System to Interact with Desktop] オプションをチェックする必要はありません。Windows NT セキュリティ レルムを特定の Windows NT ユーザアカウントで実行することはできません。

Windows 2000 で権利を更新するには、次の手順に従います。

1. [プログラム｜管理ツール] を選択します。
2. [ローカル セキュリティ ポリシー] を選択します。
3. [ローカル ポリシー｜ユーザー権利の割り当て] を選択します。
4. WebLogic Server を実行する Windows ユーザに次の権利を付与します。
   - [オペレーティング システムの一部として機能]
   - [トークン オブジェクトの作成]
   - [プロセス レベル トークンの置き換え]
5. WebLogic Server を実行する Windows ユーザが Administrators グループのメンバーであることを確認します。
6. Windows 2000 を再起動して、すべての変更を有効にします。
7. [Logon as System Account] オプションがチェックされていることを確認します。[Allow System to Interact with Desktop] オプションをチェックする必要はありません。Windows NT セキュリティ レルムを特定の Windows NT ユーザアカウントで実行することはできません。

Windows NT セキュリティ レルムを使用する場合に発生する Windows NT の一般的なエラーを以下に示します。

エラー コード	意味
1326	セキュリティ レルムを実行するホスト マシンは、プライマリ ドメイン コントローラとの信頼が確立されていない。ホスト マシンがドメインのメンバーになっていないか、ドメインがホスト マシンを信頼していない可能性がある。

エラー コード	意味
53	プライマリ ドメイン コントローラのパスが見つからなかったことを示すをネットワーク エラーが発生した。このエラーは、ドメイン名が間違っている場合、またはプライマリ ドメイン コントローラのホスト名ではなく、ドメイン名が指定されている場合に発生する。

Windows NT のエラー コードについては、winerror.h ファイルで詳しく説明されています。

UNIX セキュリティ レルムのコンフィグレーション

注意： UNIX セキュリティ レルムは、Solaris および Linux プラットフォーム上でのみ動作します。

UNIX セキュリティ レルムは小さなネイティブ プログラム（wlauth）を実行して、ユーザとグループを検索し、UNIX ログイン名とパスワードに基づいてユーザを認証します。wlauth プログラムは PAM（Pluggable Authentication Modules）を使用します。これにより、オペレーティング システムの認証サービスを、このサービスを使用するアプリケーションを変更することなくコンフィグレーションできます。

ACL を変更した後は、[セキュリティ] の [一般] タブで [更新] ボタンをクリックして、WebLogic Server が使用する filerealm.properties ファイルの情報を更新します。ACL でグループを使用すれば、WebLogic Server の情報を更新する回数を減らすことができます。UNIX グループのメンバを変更すると、WebLogic Server リソースへの個々のユーザのアクセスを動的に管理できます。

wlauth プログラムは、setuid root を実行します。wlauth プログラムの所有権とファイル属性を変更し、wlauth に合わせて PAM コンフィグレーション ファイルを設定するには、ルート パーミッションが必要です。

UNIX セキュリティ レルムの wlauth プログラムを設定するには、次の操作を行います。

1. WebLogic Server がネットワーク ドライブにインストールされている場合は、wlauth ファイルを、WebLogic Server を実行するコンピュータのファイル システムの /usr/sbin ディレクトリなどにコピーします。wlauth ファイルは weblogic/lib/*arch* ディレクトリにあり、*arch* は使用しているプラットフォームの名前です。

2. ルート ユーザとして次のコマンドを実行して、wlauth のオーナとパーミッションを変更します。

```
# chown root wlauth
# chmod +xs wlauth
```

3. wlauth の PAM コンフィグレーションを設定します。

   Solaris の場合は、/etc/pam.conf ファイルに次の行を追加します。

```
# Solaris マシンの WebLogic 認証の設定
#
wlauth auth required      /usr/lib/security/pam_unix.so.1
wlauth password required  /usr/lib/security/pam_unix.so.1
wlauth account required   /usr/lib/security/pam_unix.so.1
```

   Linux の場合は、次の行を含むファイルを /etc/pam.d/wlauth という名前で作成します。

```
#%PAM-1.0
#
# ファイル名 :
# /etc/pam.d/wlauth
#
# シャドウ パスワードを使用しない場合は「shadow」を削除する
auth required     /lib/security/pam_pwdb.so shadow
account required  /lib/security/pam_pwdb.so
```

   注意： シャドウ パスワードを使用しない場合は、shadow を省略します。

UNIX セキュリティ レルムを使用するには、次の操作を行います。

1. Administration Console の左ペインで [セキュリティ｜レルム] ノードを選択します。

2. Administration Console の右ペインで、[新しい UNIX Realm のコンフィグレーション] リンクをクリックします。

3. UNIX セキュリティ レルムのコンフィグレーションでは、レルムの名前と、UNIX セキュリティ レルムの認証サービスを提供するプログラムの名前を定義する属性を設定します。これらの名前を定義するには、Administration Console の [新しい UnixRealm の作成] ウィンドウの属性に値を指定します。

   次の表では、[新しい UnixRealm の作成] ウィンドウで設定する属性について説明します。

表 14-12 UNIX セキュリティ レルムの属性

属性	説明
[名前]	AccountingRealm などの UNIX セキュリティ レルムの名前。
[認証プログラム]	UNIX セキュリティ レルムでユーザの認証に使用するプログラムの名前。ほとんどの場合、プログラムの名前は `wlauth`。
[レルム クラス名]	UNIX セキュリティ レルムを実装する Java クラスの名前。Java クラスは WebLogic Server のクラス パスに入っていなければならない。

4. 変更を保存するには、[適用] ボタンをクリックします。
5. 属性の定義が終わったら、WebLogic Server を再起動します。
6. キャッシング レルムをコンフィグレーションします。詳細については、「キャッシング レルムのコンフィグレーション」を参照してください。

   キャッシング レルムをコンフィグレーションするときには、[一般] タブの [基本レルム] 属性のプルダウン メニューから UNIX セキュリティ レルムを選択します。[基本レルム] 属性では、キャッシング レルムと代替セキュリティ レルム（この場合は UNIX セキュリティ レルム）の関連付けを定義します。
7. [セキュリティ] ノードに移動します。
8. [ファイル レルム] タブを選択します。
9. [キャッシング レルム] 属性で、UNIX セキュリティ レルムで使用するキャッシング レルムの名前を選択します。コンフィグレーションされているキャッシング レルムのリストはプルダウン メニューに表示されます。

10. WebLogic Server を再起動します。

wlauth が WebLogic Server のクラスパスに入っていない場合、または wlauth 以外のプログラム名を指定した場合は、WebLogic Server を起動したときに Java コマンドライン プロパティを追加しなければなりません。使用するスクリプトを編集し、WebLogic Server を起動して、java コマンドの後に次のオプションを追加します。

```
-Dweblogic.security.unixrealm.authProgram=wlauth_prog
```

wlauth_prog を wlauth プログラムの名前と置き換えます。プログラムが検索パスにない場合は、絶対パスも指定します。WebLogic Server を起動します。wlauth プログラムが WebLogic Server パスにあり、wlauth という名前の場合は、この手順は不要です。

RDBMS セキュリティ レルムのコンフィグレーション

RDBMS セキュリティ レルムは BEA 独自のカスタム セキュリティ レルムで、ユーザ、グループ、および ACL をリレーショナル データベースに保存します。RDBMS セキュリティ レルムはサンプルであり、プロダクション環境で使用するためのものではありません。Administration Console を使用して、RDBMS セキュリティ レルムの次の管理機能を実行できます。

管理機能	Administration Console のサポート
ユーザの作成	あり
ユーザの削除	あり
パスワードの変更	なし
グループの作成	なし
グループの削除	あり
グループ メンバーの追加	あり
グループ メンバーの削除	あり

管理機能	Administration Console のサポート
ACL の削除	なし
ACL の削除	なし
パーミッションの追加	なし
パーミッションの削除	なし

データベースに入力する SQL スクリプトを使用すると、RDBMS セキュリティ レルムのグループを作成できます。

RDBMS セキュリティ レルムを基にして、プロダクション セキュリティ レルムを作成できます。RDBMS セキュリティ レルムを拡張するには、`weblogic.security.acl` パッケージの次のインタフェースを使用して RDBMS セキュリティ レルムに管理機能を追加します。

- `ManageableRealm` ー グループの作成、ACL の作成と削除、ユーザ、グループ、および ACL のルックアップの実行
- `User` ー パスワードの変更
- `ACL` ー ユーザおよびグループのパーミッションの追加と削除

これらのインタフェースを使用して RDBMS セキュリティ レルムを拡張した場合、データベース スキーマの更新も必要になることがあります。

注意： サンプルの RDBMS は、自動コミットが有効になっているデータベースでは機能しません。サンプルの RDBMS をベースにして RDBMS を実装する場合、コードでは明示的にコミット文を使用し、データベースでは自動コミット機能を無効にしてください。

RDBMS セキュリティ レルムを使用するには、次の操作を行います。

1. Administration Console の左ペインで [セキュリティ｜レルム] ノードを選択します。
2. Administration Console の右ペインで、[新しい RDBMSRealm のコンフィグレーション] リンクをクリックします。
3. RDBMS セキュリティ レルムを実装するクラスの情報を定義します。

   次の表では、[一般] タブで設定する属性について説明します。

表 14-13 [一般] タブの RDBMS セキュリティ レルムの属性

属性	説明
[名前]	AccountingRealm などの RDBMS セキュリティ レルムの名前。
[レルム クラス]	RDBMS セキュリティ レルムを実装する WebLogic クラスの名前。Java クラスは WebLogic Server の CLASSPATH に入っていなければならない。

4. 変更を保存するには、[適用] ボタンをクリックします。
5. データベースへの接続に使用する JDBC ドライバの属性を定義します。

   次の表では、[データベース] タブで設定する属性について説明します。

表 14-14 [データベース] タブの RDBMS セキュリティ レルムの属性

属性	説明
[ドライバ]	JDBC ドライバの完全クラス名。このクラス名は、WebLogic Server の CLASSPATH に入っていなければならない。
[URL]	RDBMS レルムで使用するデータベースの URL。JDBC ドライバのマニュアルに従って指定する。
[ユーザ名]	データベースのデフォルト ユーザ名。
[パスワード]	データベースのデフォルト ユーザのパスワード。

6. 変更を保存するには、[適用] ボタンをクリックします。
7. [スキーマ] タブの [スキーマ プロパティ] ボックスで、ユーザ、グループ、および ACL をデータベースに格納するためのスキーマを定義します。

コード リスト 14-1 は、WebLogic Server に付属の RDBMS コード サンプル（\samples\examples\security\rdbmsrealm ディレクトリ）の Schema プロパティで入力されたデータベース文を示しています。

コード リスト 14-1　RDBMS セキュリティ レルムのサンプル スキーマ

```
"getGroupNewStatement=true;getUser=SELECT U_NAME, U_PASSWORD FROM
users WHERE U_NAME = ?;
getGroupMembers=SELECT GM_GROUP, GM_MEMBER from groupmembers WHERE
GM_GROUP = ?;
getAclEntries=SELECT A_NAME, A_PRINCIPAL, A_PERMISSION FROM
aclentries WHERE A_NAME = ? ORDER BY A_PRINCIPAL;
getUsers=SELECT U_NAME, U_PASSWORD FROM users;
getGroups=SELECT GM_GROUP, GM_MEMBER FROM groupmembers;
getAcls=SELECT A_NAME, A_PRINCIPAL, A_PERMISSION FROM aclentries
ORDER BY A_NAME, A_PRINCIPAL;
getPermissions=SELECT DISTINCT A_PERMISSION FROM aclentries;
getPermission=SELECT DISTINCT A_PERMISSION FROM aclentries WHERE
A_PERMISSION = ?;
newUser=INSERT INTO users VALUES ( ? , ? );
addGroupMember=INSERT INTO groupmembers VALUES ( ? , ? );
removeGroupMember=DELETE FROM groupmembers WHERE GM_GROUP = ? AND
GM_MEMBER = ?;
deleteUser1=DELETE FROM users WHERE U_NAME = ?;
deleteUser2=DELETE FROM groupmembers WHERE GM_MEMBER = ?;
deleteUser3=DELETE FROM aclentries WHERE A_PRINCIPAL = ?;
deleteGroup1=DELETE FROM groupmembers WHERE GM_GROUP = ?;
deleteGroup2=DELETE FROM aclentries WHERE A_PRINCIPAL = ?"
```

8. 変更を保存するには、[適用] ボタンをクリックします。
9. 属性の定義が終わったら、WebLogic Server を再起動します。
10. キャッシング レルムをコンフィグレーションします。詳細については、「キャッシング レルムのコンフィグレーション」を参照してください。

    キャッシング レルムをコンフィグレーションするときには、[一般] タブの [基本レルム] 属性のプルダウン メニューから RDBMS セキュリティ レルムを選択します。[基本レルム] 属性では、キャッシング レルムと代替セキュリティ レルム（この場合は RDBMS セキュリティ レルム）の関連付けを定義します。
11. [セキュリティ] ノードに移動します。
12. [ファイル レルム] タブを選択します。

13. [キャッシング レルム] 属性で、RDBMS セキュリティ レルムで使用するキャッシング レルムの名前を選択します。コンフィグレーションされているキャッシング レルムのリストはプルダウン メニューに表示されます。

14. WebLogic Server を再起動します。

カスタム セキュリティ レルムのインストール

ネットワーク上のディレクトリ サーバなどの既存のユーザ ストアからデータを抽出するカスタム セキュリティ レルムを作成できます。カスタム セキュリティ レルムを使用するには、`weblogic.security.acl.AbstractListableRealm` インタフェースまたは `weblogic.security.acl.AbstractManageableRealm` インタフェースの実装を作成し、Administration Console を使用してその実装をインストールします。

カスタム セキュリティ レルムをインストールするには、次の操作を行います。

1. Administration Console の左ペインで [セキュリティ｜レルム] ノードを選択します。
2. Administration Console の右ペインで、[新しい Custom Realm のコンフィグレーション] リンクをクリックします。
3. [コンフィグレーション] ウィンドウで、カスタム セキュリティ レルムの名前を定義し、そのレルムを実装するインタフェースを指定して、ユーザ、グループ、および ACL（オプション）をカスタム セキュリティ レルムに格納する方法を定義します。

   次の表では、[新しい CustomRealm の作成] ウィンドウの [コンフィグレーション] タブで設定する属性について説明します。

表 14-15 カスタム セキュリティ レルムの属性

属性	説明
[名前]	AccountingRealm などのカスタム セキュリティ レルムの名前。

表 14-15 カスタム セキュリティ レルムの属性（続き）

属性	説明
[レルム クラス名]	カスタム セキュリティ レルムを実装する WebLogic クラスの名前。Java クラスは WebLogic Server の CLASSPATH に入っていなければならない。
[コンフィグレーション情報]	セキュリティ ストアに接続するために必要な情報。
[パスワード]	カスタム セキュリティ レルムのパスワード。パスワードを指定すると、そのパスワードは WebLogic Server によって暗号化される。

4. 変更を保存するには、[作成] ボタンをクリックします。
5. 属性の定義が終わったら、WebLogic Server を再起動します。
6. キャッシング レルムをコンフィグレーションします。詳細については、「キャッシング レルムのコンフィグレーション」を参照してください。

   キャッシング レルムをコンフィグレーションするときには、[一般] タブの [基本レルム] 属性のプルダウン メニューからカスタム セキュリティ レルムを選択します。[基本レルム] 属性では、キャッシング レルムとカスタム セキュリティ レルムの関連付けを定義します。
7. [セキュリティ] ノードに移動します。
8. [ファイル レルム] タブを選択します。
9. [キャッシング レルム] 属性で、カスタム セキュリティ レルムで使用するキャッシング レルムの名前を選択します。コンフィグレーションされているキャッシング レルムのリストはプルダウン メニューに表示されます。
10. WebLogic Server を再起動します。

カスタム セキュリティ レルムの記述の詳細については、「カスタム セキュリティ レルムの記述」を参照してください。

セキュリティ レルムの移行

WebLogic Server は、セキュリティ レルム用の管理アーキテクチャを備えています。MBean で実装される管理アーキテクチャにより、Administration Console を使用してセキュリティ レルムを管理できます。以前のリリースの WebLogic Server でのセキュリティ レルムがある場合、以下の情報を使用して新しいアーキテクチャに移行します。

- Window NT、UNIX、または LDAP セキュリティ レルムを使用している場合、Administration Console の [weblogic.properties のコンバート] オプションを使用して、セキュリティ レルムを新しいアーキテクチャに変換します。Windows NT、UNIX、または LDAP セキュリティ レルムのユーザ、グループ、および ACL は Administration Console で表示できます。ただし、ユーザとグループを管理するためには、Windows NT、UNIX、または LDAP 環境のツールを使用する必要があります。
- カスタム セキュリティ レルムを使用している場合は、「カスタム セキュリティ レルムのインストール」の手順に従って、ユーザ、グループ、および ACL（省略可能）をカスタム セキュリティ レルムに保存する方法を指定します。
- 代理セキュリティ レルムはサポートされなくなりました。代理セキュリティ レルムを使用している場合は、他の種類のセキュリティ レルムを使用してユーザ、グループ、および ACL を保存する必要があります。
- RDBMS セキュリティ レルムを使用している場合は、以下のいずれかの方法でセキュリティ レルムを変換します。
  - RDBMS セキュリティ レルムのソースを変更しなかった場合は、「RDBMS セキュリティ レルムのコンフィグレーション」の手順に従って、既存の RDBMS セキュリティ レルムの新しいクラスをインスタンス化し、データベースへの接続に使用する JDBC ドライバとセキュリティ レルムで使用するスキーマの情報を定義します。この場合、RDBMS セキュリティ レルム用の MBean を WebLogic Server で作成します。
  - RDBMS セキュリティ レルムをカスタマイズした場合は、MBean で使用するためにソースを変換します。RDBMS セキュリティ レルムの変換のガイドとして `\samples\examples\security\rdbmsrealm` ディレクトリにあるコード例を使用します。RDBMS セキュリティ レルムを MBean へ変換したら、「RDBMS セキュリティ レルムのコンフィグレーション」の

手順に従って、データベースへの接続に使用する JDBC ドライバとセキュリティ レルムで使用するスキーマの情報を定義します。

ユーザの定義

注意： この節では、ファイル レルムにユーザを追加する方法について説明します。代替セキュリティ レルムを使用している場合、ユーザを定義するには、そのレルムで用意されている管理ツールを使用する必要があります。

ユーザとは、WebLogic Server セキュリティ レルムで認証されるエンティティのことです。ユーザは、個人または Java クライアントなどのソフトウェア エンティティでもかまいません。各ユーザには、WebLogic Server セキュリティ レルムでユニークな ID が与えられます。システム管理者は、同じセキュリティ レルム内で同一ユーザが重複しないようにする必要があります。

セキュリティ レルムのユーザの定義では、WebLogic Server セキュリティ レルム内のリソースにアクセスするユーザごとにユニークな名前とパスワードを、Administration Console の [ユーザ] ウィンドウで指定します。

WebLogic Server には、`system` と `guest` という 2 つの特別なユーザが定義されています。

- `system` ユーザとは、サーバの起動 / 停止やリソースのロック / ロック解除など、WebLogic Server のシステムレベルの操作を管理する管理者ユーザです。`system` ユーザとそのパスワードは、WebLogic Server のインストール手順の中で定義します。セキュリティ措置として、`system` ユーザのパスワードを変更することをお勧めします。詳細については、「システム パスワードの変更」を参照してください。
- `guest` ユーザは、WebLogic Server によって自動的に定義されます。許可が不要な場合、クライアントには、WebLogic Server によって `guest` ID が割り当てられるので、クライアントは `guest` ユーザが使用可能なすべてのリソースにアクセスできるようになります。クライアントからは、Web ブラウザから要求された場合にユーザ名にもパスワードにも `guest` と入力するか、Java クライアントで `guest` をユーザ名およびパスワードとして提供することで、`guest` ユーザとしてログインできます。デフォルトでは、`guest` アカウントが有効になってます。

デプロイメントの安全性を強化するために、WebLogic Server は guest アカウントを無効にして実行することをお勧めします。guest アカウントを無効にするには、[セキュリティ] ウィンドウの [一般] タブで [ゲスト不可] 属性を選択します。guest アカウントを無効にしても、アカウント guest にログインできなくなるだけであり、未認証ユーザが WebLogic Server デプロイメントにアクセスすることはできます。

system ユーザと guest ユーザは、WebLogic Server セキュリティ レルムのその他のユーザとほぼ同じです。

- WebLogic Server のリソースにアクセスするには、適切な ACL を必要とします。
- WebLogic Server のリソースに対して処理を実行するには、ユーザ名とパスワード（またはデジタル証明書）を提出する必要があります。

ユーザを定義するには、次の操作を行います。

1. Administration Console の左ペインで [セキュリティ｜ユーザ] ノードを選択します。

   [ユーザ] ウィンドウが表示されます。

2. [ユーザ] ウィンドウで [名前] 属性にユーザの名前を入力します。
3. [パスワード] 属性でユーザのパスワードを入力します。
4. [パスワードの確認] 属性にパスワードを再び入力します。
5. [作成] をクリックします。

ユーザを削除するには、次の操作を行います。

1. [ユーザ] ウィンドウの [ユーザの削除] ボックスでユーザの名前を入力します。
2. [削除] をクリックします。

ユーザのパスワードを変更するには、次の操作を行います。

1. [ユーザ] ウィンドウの [名前] 属性でユーザの名前を入力します。
2. [古いパスワード] 属性に古いパスワードを入力します。
3. [新しいパスワード] 属性に新しいパスワードを入力します。
4. 新しいパスワードを再度入力して、パスワードの変更を確定します。

WebLogic Server の使用時には、ユーザがロックされている場合があります。次の手順を実行すると、ユーザのロックを解除できます。

1. Administration Console で [ユーザ] ウィンドウを開きます。
2. [ユーザのロックを解除] リンクをクリックします。
3. [ユーザのロックを解除] フィールドで、ロックを解除するユーザの名前を入力します。
4. ユーザのロックを解除するサーバを選択します。
5. [ロック解除] をクリックします。

WebLogic Server のユーザとアクセス制御モデルの詳細については、「WebLogic Security の概要」と「セキュリティの基礎概念」を参照してください。

グループの定義

注意： この節では、ファイル レルムにグループを追加する方法について説明します。代替セキュリティ レルムを使用している場合、グループを定義するには、そのレルムで用意されている管理ツールを使用する必要があります。

グループは、通常、企業の同じ部門に所属しているなどの共通点を持つユーザの集合を表します。グループは、多数のユーザを効率的に管理する手段です。ACL でグループにパーミッションが付与された場合、そのグループのすべてのメンバがそのパーミッションを持つことになります。パーミッションは、個々のユーザに対してではなく、グループに対して割り当てることをお勧めします。

デフォルトの WebLogic Server には以下のグループがあります。

- 定義済みセキュリティ レルムのすべての定義済みユーザは自動的に `everyone` グループのメンバです。
- `system` ユーザは、`Administrators` グループのメンバーです。このグループには、サーバの起動と停止、および動作している WebLogic Server デプロイメントの管理を行うユーザに適切なパーミッションが割り当てられていなければなりません。このグループへのアクセスは制限する必要があります。

次の手順を実行すると、グループを WebLogic Server セキュリティ レルムに登録できます。

1. Administration Console の左ペインで [セキュリティ | グループ] ノードを選択します。
2. [新しい Group の作成] リンクをクリックします。

   [グループ] ウィンドウが表示されます。
3. [グループ] ウィンドウの [名前] 属性でグループの名前を入力します。グループ名は複数形にすることをお勧めします。たとえば、Administrator ではなく Administrators にします。
4. [追加ユーザ] 属性をクリックし、グループに追加する WebLogic Server ユーザを選択します。
5. [追加グループ] 属性をクリックし、グループに追加する WebLogic Server グループを選択します。
6. [適用] ボタンをクリックして新しいグループを作成します。

グループを削除するには、[グループ] ウィンドウのリスト ボックスでグループの名前を入力し、[削除] をクリックします。

WebLogic Server のグループとアクセス制御モデルの詳細については、「WebLogic Security の概要」と「セキュリティの基礎概念」を参照してください。

ACL の定義

ユーザは、WebLogic Server セキュリティ レルムのリソースにアクセスします。ユーザがリソースにアクセスできるかどうかは、そのリソースのアクセス制御リスト（ACL）によって決まります。ACL には、ユーザがリソースとの対話に用いるパーミッションが定義されています。ACL を定義するには、リソースの ACL を作成し、そのリソースに対するパーミッションを指定してから、そのパーミッションを付与するユーザおよびグループを指定します。ACL は、グループに対して割り当てることをお勧めします。

各 WebLogic Server リソースには、1 つまたは複数のパーミッションを付与できます。次の表は、ACL を使用してパーミッションを制限するさまざまな WebLogic Server リソースの機能の一覧を示しています。

表 14-16 WebLogic Server リソースの ACL

WebLogic Server リソース	ACL	付与するパーミッションの内容
WebLogic Server	`weblogic.server` `weblogic.server.`*`servername`*	boot
コマンドライン管理ツール	`weblogic.admin` **注意：** Administration Console を使用して ACL を追加するには、`weblogic.admin.acl.modify` を定義する必要がある。	shutdown lockServer unlockserver、 modify
MBean	`weblogic.admin.mbean.`*`mbeaninstancename`*	access
WebLogic Event	`weblogic.event.`*`topicName`*	submit receive
WebLogic JDBC 接続プール	`weblogic.jdbc.connectionPool.`*`poolname`*	reserve admin
WebLogic パスワード	`weblogic.passwordpolicy`	unlockuser
WebLogic JMS 送り先	`weblogic.jms.topic.`*`topicName`* `weblogic.jms.queue.`*`queueName`*	send、receive
WebLogic JNDI コンテキスト	`weblogic.jndi.`*`path`*	lookup modify list

WebLogic Server リソースの ACL を作成するには、Administration Console を起動して次の手順に従います。

1. Administration Console の左ペインで [セキュリティ｜ ACL] ノードを選択します。

2. Adminstration Console の右ペインで、[新しい ACL の作成] リンクをクリックします。

   [アクセス コントロール リスト] ウィンドウが表示されます。

3. [新しい ACL 名] フィールドで、ACL を使用して保護する WebLogic Server リソースの名前を指定します。

   たとえば、`demopool` という名前で、JDBC 接続プール用の ACL を作成します。

4. [作成] をクリックします。

5. [新しい Permssion を追加] リンクをクリックします。

6. リソースに対するパーミッションを指定します。

   リソースに対して設定可能なパーミッションごとに別々の ACL を作成することも、リソースに対するすべてのパーミッションを付与する 1 つの ACL を作成することもできます。たとえば、JDBC 接続プール、`demopool` に対して、`reserve` パーミッション用、`reset` パーミッション用、`shrink` パーミッション用にそれぞれ 1 つの ACL を作成できます。または、`reserve` および `reset` パーミッション用に 1 つの ACL を作成することもできます。

7. リソースに対して指定されたパーミッションを持つユーザまたはグループを指定します。

8. [適用] をクリックします。

WebLogic Server でリソースの ACL を作成する場合は、表 14-16 の構文に従ってリソースを参照しなければなりません。たとえば、`demopool` という JDBC 接続プールを、`weblogic.jdbc.connectionPool.demopool` と指定します。

既存の ACL を変更した場合は、[セキュリティ] ノードの [一般] タブで [更新] ボタンをクリックして、WebLogic Server が使用する `filerealm.properties` ファイルの情報を更新します。

WebLogic Server を起動できるようにするには、サーバを起動するためのパーミッションを特定のグループに付与する必要があります。このセキュリティ対策によって、許可のないユーザが WebLogic Server を起動できなくなります。

デフォルトでは、system ユーザだけが MBean を修正できます。MBean にアクセスして修正できるユーザの数は制限するようにしてください。すべての WebLogic Server MBean にアクセスするには、次のような ACL を使用します。

```
access.weblogic.admin.mbean=Group or User name
```

ユーザが MBean へアクセスしようとして失敗した場合、`weblogic.management.NoAccessRuntimeException` が返されます。サーバ ログには、アクセスしようとしたユーザと MBean を示す詳細が記録されます。

Administration Console を使用してユーザまたはグループにパーミッションを付与する前に、Administrators グループに次のパーミッションを付与する必要があります。

```
acl.modify.weblogic.admin=Administrators
```

SSL プロトコルのコンフィグレーション

以下の節では、デジタル証明書を取得する方法、および SSL プロトコルをコンフィグレーションする方法について説明します。

- プライベート キーとデジタル証明書の取得
- プライベート キーとデジタル証明書の保存
- 信頼された認証局の定義
- SSL プロトコル用の属性の定義
- SSL セッション キャッシングのパラメータの変更

SSL プロトコルの詳細については、「WebLogic Security の概要」と「セキュリティの基礎概念」を参照してください。

プライベート キーとデジタル証明書の取得

プライベート キーとデジタル証明書は、SSL プロトコルを使用する WebLogic Server のデプロイメントごとに必要です。認証局（CA）からデジタル証明書を取得するには、証明書署名リクエスト（CSR）と呼ばれる特定のフォーマットでリクエストを提出する必要があります。WebLogic Server には、CSR を作成する Certificate Request Generator サーブレットが入っています。Certificate Request Generator サーブレットはユーザから情報を収集して、プライベート キー ファイ

ルと証明書リクエスト ファイルを生成します。次に、VeriSign や Entrust.net などの認証局に CSR を提出します。Certificate Request Generator サーブレットを使用する前に、WebLogic Server をインストールして実行しておく必要があります。

注意： Certificate Request Generator サーブレット以外のソースからプライベート キーを入手した場合は、そのキーのフォーマットが PKCS#5/PKCS#8 PEM であることを確認します。

CSR を生成するには、次の手順に従います。

1. Certificate Request Generator サーブレットを起動します。サーブレットの `.war` ファイルは、`\wlserver6.1\config\applications` ディレクトリにあります。`.war` ファイルは、WebLogic Server を起動すると自動的にインストールされます。
2. Web ブラウザで、Certificate Request Generator サーブレットの URL を次のように入力します。

   `https://`*`hostname:port`*`/certificate/`

   URL の各要素は次のように定義します。

   - *`hostname`* は、WebLogic Server を実行しているマシンの DNS 名です。
   - *`port`* は、WebLogic Server が SSL 接続をリスンするポートの番号です。デフォルトでは 7002 です。

     たとえば、WebLogic Server が `ogre` というマシン上で動作しており、Certificate Request Generator サーブレットを実行するために SSL 通信をデフォルト ポートの `7002` でリスンするようコンフィグレーションされている場合は、Web ブラウザに次の URL を入力しなければなりません。

     `https://ogre:7002/certificate/`

3. Certificate Request Generator サーブレットによって、Web ブラウザからフォームがロードされます。次の表の情報を参照して、ブラウザに表示されたフォームに必要な情報を入力します。

表 14-17 Certificate Request Generator フォームのフィールド

フィールド	説明
[Country code]	国ごとの 2 文字の ISO コード。アメリカのコードは US。

表 14-17 Certificate Request Generator フォームのフィールド

フィールド	説明
[Organizational unit name]	組織の事業部、部、またはその他の運営単位の名前。
[Organization name]	組織の名前。認証局が、この組織に登録されているドメインに所属するホスト名をこの属性に入力するよう要求する場合がある。
[Email address]	管理者の E メール アドレス。このアドレスがデジタル証明書の送信先になる。
[Full host name]	デジタル証明書のインストール先となる WebLogic Server の完全修飾名。この名前は、WebLogic Server の DNS ルックアップ用の名前（たとえば `node..com`）である。Web ブラウザでは、URL のホスト名とデジタル証明書の名前を比較する。ホスト名を後で変更した場合は、新しいデジタル証明書を要求しなければならない。
[Locality name (city)]	市または町の名前。市で付与されたライセンスを使用して運用する場合は、この属性は必須、つまり、ライセンスを付与された市の名前を入力しなければならない。
[State name]	組織の所在地がアメリカまたはカナダの場合に、組織が業務を行っている州の名前。短縮してはならない。
[Private Key Password]	プライベート キーの暗号化に使用するパスワード。 WebLogic Server で保護されたキーを使用する場合は、このフィールドにパスワードを入力する。保護されたキーの使用を選択すると、キーの使用時にパスワードの入力が要求される。パスワードを指定した場合は、PKCS-8 で暗号化されたプライベート キーを受け取る。パスワードを使用してプライベート キーを保護することが望ましい。 保護されたキーを使用しない場合は、このフィールドに何も入力しない。 保護されたプライベート キーを使用するには、Administration Console の [サーバ] ウィンドウの [SSL] タブの [暗号化キーを使用] 属性を有効にする。

表 14-17 Certificate Request Generator フォームのフィールド

フィールド	説明
[Random String]	暗号化アルゴリズムによって使用される文字列。この文字列を覚えておく必要はない。この文字列は、暗号の解読をより困難にするために暗号化アルゴリズムに外部要因を追加する。したがって、簡単に推測されることのない文字列を入力する。大文字と小文字、数字、スペース、句読点をうまく組み合わせた長い文字列が望ましい。こうした文字列によって、暗号を強化できる。
[Strength]	生成するキーの長さ（ビット単位）。キーが長いほど、暗号の解読はより困難になる。 国内バージョンの WebLogic Server では、512 ビット、768 ビット、または 1024 ビットのキーを選択できる。1024 ビットのキーが望ましい。

4. [Generate Request] ボタンをクリックします。

   必須属性が空白の場合、または属性に無効な値が指定されている場合は、Certificate Request Generator サーブレットによってメッセージが表示されます。メッセージが表示された場合は、ブラウザの [戻る] ボタンをクリックして、エラーを修正します。

   すべての属性が受け付けられると、Certificate Request Generator サーブレットは次のファイルを WebLogic Server のスタートアップ ディレクトリに作成します。

   - `www__com-key.der` - プライベート キー ファイル。Administration Console の [SSL] タブの [サーバ キー ファイル名] 属性フィールドに入る名前です。
   - `www__com-request.dem` - バイナリ フォーマットの証明書リクエスト ファイル。
   - `www__com-request.pem` - 認証局に提出する CSR ファイル。このファイルの内容は `.dem` ファイルと同じデータですが、E メールにコピーしたり、Web フォームに貼り付けたりできるように、ASCII でエンコードされています。

5. 認証局を選択し、その認証局の Web サイトの指示に従って、デジタル証明書を購入します。
   - VeriSign, Inc. では、WebLogic Server 用に 2 つのオプションを用意しています。1 つは、国内および国外用の Web ブラウザ向けの強力な 128 ビット暗号化を特徴とする Global Site Services、もう 1 つは、国内用 Web ブラウザに 128 ビットの暗号を、国外用 Web ブラウザには 40 ビットの暗号を提供する Secure Site Services です。
   - Entrust.net のデジタル証明書は、国内用 Web ブラウザに 128 ビットの暗号を、国外用 Web ブラウザに 40 ビットの暗号を提供します。
6. サーバのタイプを選択するよう指示された場合は、WebLogic Server に対応したデジタル証明書を受け取れるように、`BEA WebLogic Server` を選択します。
7. 認証局からデジタル証明書を受け取ったら、`\wlserver6.1\config\` ディレクトリに保存する必要があります。
8. SSL プロトコルを使用するよう WebLogic Server をコンフィグレーションするには、[サーバ] ウィンドウの [コンフィグレーション] タブにある [SSL] タブで次の情報を入力する必要があります。
   - [サーバ認証ファイル名] 属性で、WebLogic Server の ID を確立するデジタル証明書の絶対パスと名前を入力します。
   - [信頼性のある CA ファイル名] 属性で、WebLogic Server のデジタル証明書に署名した認証局のデジタル証明書の絶対パスと名前を入力します。
   - [サーバ キー ファイル名] 属性で、WebLogic Server 用のプライベート キーの絶対パスと名前を入力します。

   SSL プロトコルのコンフィグレーションの詳細については、「SSL プロトコル用の属性の定義」を参照してください。
9. 保護されたプライベート キーを使用する場合は、次のコマンドライン オプションを使用して WebLogic Server を起動します。

   `-Dweblogic.management.pkpassword=`*`password`*

   `password` はプライベート キーのパスワード。

プライベート キーとデジタル証明書の保存

プライベート キーとデジタル証明書を取得したら、Certificate Request Generator サーブレットによって生成されたプライベート キーと認証局から入手したデジタル証明書を、\wlserver6.1\config\ ディレクトリにコピーします。

プライベート キーとデジタル証明書は、PEM または Definite Encoding Rules（DER）フォーマットで生成されます。デジタル証明書ファイルのフォーマットは、ファイル名拡張子で識別します。

PEM（.pem）フォーマットのプライベート キー ファイルは、先頭と末尾がそれぞれ次のような行になっています。

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
-----END ENCRYPTED PRIVATE KEY-----
```

PEM（.pem）フォーマットのデジタル証明書は、先頭と末尾がそれぞれ次のような行になっています。

```
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
```

注意： 使用するデジタル証明書は、ファイル内で BEGIN CERTIFICATE および END CERTIFICATE 行によってそれぞれが区切られた複数のデジタル証明書のうちのいずれかでもかまいません。通常、WebLogic Server 用のデジタル証明書は 1 つのファイル（拡張子は .pem または .der）に入っており、WebLogic Server 認証チェーン ファイルは別のファイルに入っています。2 つのファイルを使用する理由は、異なる WebLogic Server が同じ認証チェーンを共有する場合があるからです。

認証局ファイル内の最初のデジタル証明書は、WebLogic Server 認証チェーンの最初のデジタル証明書となります。ファイル内の次の証明書は、認証チェーン内の次のデジタル証明書になります。ファイル内の最後の証明書は、認証チェーン内の最後となる自己署名デジタル証明書です。

DER（.der）フォーマットのファイルにはバイナリ データが格納されます。WebLogic Server では、ファイル拡張子が認証ファイルの内容と一致する必要があるので、認証局から取得したファイルは正しい拡張子を付けて保存します。

プライベート キー ファイルとデジタル証明書には WebLogic Server の `system` ユーザだけが読み込み特権を持ち、その他のユーザがアクセスできないように、プライベート キーとデジタル証明書を保護します。複数の認証局のデジタル証明書を持つファイルまたは認証チェーンを格納するファイルを作成する場合は、PEM フォーマットを使用しなければなりません。WebLogic Server には、DER フォーマットと PEM フォーマットを互いに変換するツールが用意されています。詳細については、「WebLogic Server Java ユーティリティの使い方」を参照してください。

信頼された認証局の定義

SSL 接続が確立されると、WebLogic Server は、信頼された認証局リストと照らし合わせて認証局の ID をチェックして、使用中の認証局が信頼されていることを確認します。

認証局のルート証明書を WebLogic Server の `\wlserver6.1\config\` ディレクトリにコピーし、「SSL プロトコル用の属性の定義」で説明されている属性を設定します。

認証チェーンを利用する場合は、別の PEM エンコード済みデジタル証明書を、WebLogic Server 用のデジタル証明書を発行した認証局のデジタル証明書に追加します。ファイル内の最後のデジタル証明書は、自己署名デジタル証明書（つまり、rootCA 証明書）でなければなりません。

相互認証を利用する場合は、受け付ける認証局のルート証明書を取得して、信頼された CA ファイルにそれを含めます。

SSL プロトコル用の属性の定義

セキュア ソケット レイヤ（Secure Sockets Layer: SSL）では、ネットワーク接続している 2 つのアプリケーションが互いの ID を認証できるようにするとともに、アプリケーション間でやりとりされるデータを暗号化することで、セキュアな接続を実現します。SSL プロトコルは、サーバ認証と、必要に応じてクライアント認証、機密性、およびデータ整合性を提供します。

SSL プロトコル用の属性を定義するには、次の手順に従います。

1. Administration Console を起動します。

2. [サーバ] ウィンドウの [コンフィグレーション] タブを表示します。
3. [SSL] タブを選択します。値を入力したり、必須チェックボックスをチェックしたりして、このタブの属性を定義します（詳細については次の表を参照してください）。
4. [適用] ボタンをクリックして、変更を保存します。
5. WebLogic Server を再起動します。

注意： PKCS-8 で保護されたプライベート キーを使用している場合は、WebLogic Server を起動するときに、プライベート キーのパスワードをコマンド ラインで指定する必要があります。

次の表では、[サーバ] ウィンドウの [コンフィグレーション] タブにある [SSL] タブの各属性について説明します。

表 14-18 SSL プロトコルの属性

属性	説明
[有効化]	SSL プロトコルを有効化する。この属性はデフォルトで有効。
[リスン ポート]	WebLogic Server が SSL 接続をリスンする専用ポートの番号。デフォルトは 7002。
[サーバ キー ファイル名]	WebLogic Server 用のプライベート キーのパスと名前。 パスは WebLogic Server のインストールされたルート ディレクトリを起点とする。次に例を示す。 `\wlserver6.1\config\myapp\privatekey.pem` ファイル拡張子（`.DER` または `.PEM`）は、WebLogic Server がファイルの内容を読み込む方法を示す。
[サーバ認証ファイル名]	WebLogic Server の ID を確立するデジタル証明書ファイルの絶対パスと名前。 パスは WebLogic Server のインストールされたルート ディレクトリを起点とする。次に例を示す。 `\wlserver6.1\config\myapp\cert.pem` ファイル拡張子（`.DER` または `.PEM`）は、WebLogic Server がファイルの内容を読み込む方法を示す。

表 14-18 SSL プロトコルの属性（続き）

属性	説明
[サーバ認証チェーン ファイル]	WebLogic Server のデジタル証明書に署名するために使用するデジタル証明書の絶対パス。 パスは WebLogic Server のインストールされたルートディレクトリを起点とする。次に例を示す。 `\wlserver6.1\config\myapp\cacert.pem` ファイル拡張子（`.DER` または `.PEM`）は、WebLogic Server がファイルの内容を読み込む方法を示す。 WebLogic Server で証明書チェーンを使用する場合、そのファイルには WebLogic Server のデジタル証明書に署名するために使用されるデジタル証明書が最初のメンバーとして格納され、2 番目のメンバーには最初のデジタル証明書に署名するために使用されるデジタル証明書が格納されていなければならない。ファイル内の最後のデジタル証明書は自己署名でなければならない。 [サーバ認証チェーン ファイル] 属性では、少なくとも 1 つのデジタル証明書が必要。ファイルに 1 つのデジタル証明書しかない場合、そのデジタル証明書は自己署名でなければならない（つまり、ルート CA デジタル証明書でなければならない）。 認証局からデジタル証明書を取得する場合、認証局およびその他の上位のデジタル証明書を認証局から受け取る。
[クライアント認証を強制する]	クライアントが信頼できる認証局からのデジタル証明書を WebLogic Server に提示しなければならないかどうかを定義する。
[信頼性のある CA ファイル名]	WebLogic Server によって信頼された認証局のデジタル証明書を格納するファイルの名前。この属性で指定したファイルには、認証局の 1 つまたは複数のデジタル証明書が格納される。ファイル拡張子（`.DER` または `.PEM`）によって、WebLogic Server がファイルの内容を読み込む方法が決まる。

表 14-18 SSL プロトコルの属性（続き）

属性	説明
[認可済み認証機関]	CertAuthenticator インタフェースを実装する Java クラスの名前。`weblogic.security.acl.CertAuthenticator` インタフェースの使い方の詳細については、「WebLogic ユーザへのデジタル証明書のマップ」を参照。
[暗号化キーを使用]	WebLogic Server のプライベート キーがパスワードで暗号化されることを指定する。デフォルトでは無効。この属性を指定した場合は、保護されたキーを使用する必要がある。また、WebLogic Server を起動するときには、次のコマンドライン オプションを使用して WebLogic Server を起動する。`-Dweblogic.management.pkpassword=`*`password`* *`password`* はプライベート キーのパスワード。
[Java を使用]	この属性を選択すると、ネイティブ Java ライブラリを使用できるようになる。WebLogic Server は、SSL プロトコルの pure-Java 実装を提供する。ネイティブ Java ライブラリを使用すると、Solaris、Windows NT、および IBM AIX プラットフォーム上で SSL 処理のパフォーマンスが向上する。デフォルトでは、この属性は無効。

表 14-18 SSL プロトコルの属性（続き）

属性	説明
[ハンドラを有効化]	WebLogic Server が、次のいずれかの理由でクライアント認証に失敗した SSL 接続を拒否するかどうかを指定する。 ■ 必要なクライアント デジタル証明書が用意されていなかった。 ■ クライアントがデジタル証明書を提出しなかった。 ■ クライアントからのデジタル証明書の発行元が、[信頼性のある CA ファイル名] 属性に指定された認証局ではない。 SSL ハンドラのデフォルト設定では、WebLogic Server インスタンスから別の WebLogic Server インスタンスへの SSL 接続は 1 つしか許可されない。たとえば、WebLogic Server の EJB は別の Web サーバで HTTPS ストリームを開く場合がある。[ハンドラを有効化] 属性が有効な場合、WebLogic Server は SSL 接続のクライアントとして動作する。デフォルトでは、この属性は有効。 この属性は、SSL 接続を開始するために独自の実装を提供する場合にのみ無効にする。 **注意：** SSL ハンドラは、WebLogic Server が SSL 接続の受け付けを管理する機能には影響しない。
[キーの有効期間をエクスポート]	WebLogic Server がドメスティック サーバとエクスポータブル クライアントとの間で、新規のキーを生成する前に、エクスポータブル キーを使用する回数。新規のキーの生成前にキーを使用する回数が少ないほど、WebLogic Server のセキュリティが高くなる。デフォルトの使用回数は 500 回。

表 14-18 SSL プロトコルの属性（続き）

属性	説明
[ログイン タイムアウト ミリ秒]	WebLogic Server が SSL 接続のタイムアウトまで待機するミリ秒数。SSL 接続は、通常の接続よりも時間がかかる。クライアントがインターネット経由で接続する場合は、ネットワーク レイテンシに対応するためにデフォルト値を大きくする。デフォルト値は 25,000 ミリ秒。
[認可キャッシュ サイズ]	WebLogic Server がトークン化して保存するデジタル証明書の数。デフォルトは 3。
[ホスト名検証を無視]	デフォルトのホスト名検証を無効にする。WebLogic Server のホスト名検証では、デジタル証明書の Subject DN と SSL 接続を開始したサーバのホスト名を比較する。ホスト名検証を実行しない場合（たとえば、WebLogic Server 付属のデモ用デジタル証明書を使用する場合）、この属性をチェックする。この属性を無効にすると、WebLogic Server は介在者の攻撃に対して無防備になる。 プロダクション環境でデモ用デジタル証明書を使用したり、ホスト名検証を無効にしたりすることは望ましくない。
[ホスト名の検証]	ホスト名検証インタフェースを実装する Java クラスの名前。 `weblogic.security.SSL.HostNameVerifier` インタフェースの使い方については、「カスタム ホスト名検証の使い方」を参照。

注意： 以前のリリースの WebLogic Server では、自己署名されているが、[サーバ認証ファイル名] 属性（または `weblogic.security.certificate.server` プロパティ）で許可されてないデジタル証明書を定義することができました。ただし、これは優れたセキュリティ ポリシーではありませんでした。現在は、[サーバ認証ファイル名] 属性と [サーバ認証チェーン ファイル] 属性の両方を定義する必要があります。

SSL セッション キャッシングのパラメータの変更

WebLogic Server 6.1 サービス パック 2 では、SSL コードに SSL セッション キャッシングのパラメータが含まれています。キャッシュされた SSL セッションを使用すると、接続では再度 SSL ハンドシェークを行う必要がなくなります。接続は、中断されたところからそのまま再開されます。キャッシュされた SSL セッションを使用することで、アプリケーションでは SSL セッションの確立に要する時間を大幅に短縮できるので、パフォーマンスが大幅に向上します。キャッシュされた SSL セッションを使用するには、クライアントとサーバが SSL セッションをキャッシュする機能を持っている必要があります。ブラウザはすべて、SSL セッションをキャッシュする機能を持っています。

サーバセッション キャッシュは、TTL キャッシュに保存されます。TTL キャッシュの詳細については、「キャッシング レルムのコンフィグレーション」を参照してください。クライアントサイドの SSL セッション キャッシュでは、実行スレッドの SSL セッションを 1 つだけ保持します。

SSL セッション キャッシングはデフォルトで有効です。次のコマンド ライン フラグを使用すると、サーバセッション キャッシュのデフォルト サイズおよび存続期間を変更できます。

```
-Dweblogic.security.SSL.sessionCache.size=211
-Dweblogic.security.SSL.sessionCache.ttl=600
```

表 14-19 パラメータ

パラメータ	最小	最大	デフォルト
sessionCache.size	1	65537	211
sessionCache.ttl	1	max Integer.MAX_VALUE	600

相互認証のコンフィグレーション

WebLogic Server が相互認証向けにコンフィグレーションされている場合、クライアントは、信頼された認証局のリストと照らし合わせてデジタル証明書を検証する WebLogic Server に、デジタル証明書を提示する必要があります。

SSL プロトコルと証明書向けに WebLogic Server をコンフィグレーションするには、「SSL プロトコルのコンフィグレーション」の手順に従います。

WebLogic Server が使用する認証局のルート証明書を `\wlserver6.1\config` ディレクトリにコピーします。クライアントは、相互認証の際に、信頼された認証局のいずれかが発行したデジタル証明書を提示する必要があります。

相互認証をコンフィグレーションするには、Administration Console の [サーバ] ウィンドウの [コンフィグレーション] タブにある [SSL] タブで [クライアント認証を強制する] オプションをチェックします。デフォルトでは、このフィールドは無効です。

SSL を使用した RMI over IIOP のコンフィグレーション

SSL プロトコルを使用すると、RMI リモート オブジェクトへの IIOP 接続を保護できます。SSL プロトコルは、認証を通じて接続を保護し、オブジェクト間のデータ交換を暗号化します。SSL プロトコルを使用して RMI over IIOP 接続を保護するには、次の手順に従います。

1. SSL プロトコルを使用するよう WebLogic Server をコンフィグレーションします。詳細については、SSL プロトコル用の属性の定義を参照してください。
2. SSL を使用するよう Object Request Broker（ORB）をコンフィグレーションします。SSL プロトコルのコンフィグレーションの詳細については、クライアント ORB の製品マニュアルを参照してください。

3. host2ior ユーティリティを使用して、WebLogic Server IOR をコンソールに出力します。host2ior ユーティリティでは、SSL 接続用と非 SSL 用に 2 種類のインターオペラブル オブジェクト参照（IOR）が出力されます。IOR のヘッダは、IOR が SSL 接続で使用できるかどうかを示します。
4. SSL IOR は、WebLogic Server JNDI ツリーにアクセスする CosNaming サービスへの初期参照を取得するときに使用します。

RMI over IIOP の使い方の詳細については、『WebLogic RMI over IIOP プログラマーズ ガイド』を参照してください。

パスワードの保護

WebLogic Server のリソースにアクセスするためのパスワードを保護することは重要です。ユーザ名とパスワードは以前、WebLogic Server セキュリティ レルムにクリア テキストで保存されていました。現在、WebLogic Server では、すべてのパスワードがハッシュ化されています。クライアントのリクエストを受け取ると、WebLogic Server はクライアントが提示するパスワードをハッシュ化して、ハッシュ化済みパスワードと一致するかどうか比較します。

各 filerealm.properties ファイルは、パスワードをハッシュ化するために使用する SerializedSystemIni.dat ファイルに関連付けられます。SerializedSystemIni.dat ファイルは、インストール時に \wlserver6.1\config\ ディレクトリに置かれます。

何らかの理由で SerializedSystemIni.dat ファイルが破損した場合は、WebLogic Server を再コンフィグレーションしなければなりません。

以下の注意事項を考慮してください。

- SerializedSystemIni.dat ファイルのバックアップを作成し、関連する filerealm.properties ファイルのコピーと同じ場所に入れます。
- WebLogic Server デプロイメントの管理者は読み書き特権を持ち、その他のユーザは何の特権も持たないように、SerializedSystemIni.dat ファイルにパーミッションを設定します。
- ハッシュ化したいパスワードを持つ weblogic.properties ファイルがある場合は、Administration Console のメイン ウィンドウで Convert

weblogic.properties オプションを使用して、weblogic.properties ファイルを config.xml ファイルに変換します。ファイルが変換されると、既存のすべてのパスワードが保護されます。

config.xml ファイルには、クリア テキスト形式のパスワードが存在しなくなりました。クリア テキスト形式のパスワードに代わって、config.xml ファイルには暗号化およびハッシュ化されたパスワードが格納されます。暗号化パスワードは、別のドメインにコピーできません。その代わりに、config.xml ファイルを編集し、既存の暗号化およびハッシュ化されたパスワードをクリア テキストのパスワードで置換して、そのファイルを新しいドメインにコピーすることはできます。Administration Console は、次にそのファイルに書き込むときにパスワードを暗号化およびハッシュ化します。

セキュリティ攻撃では、パスワードを推測する方法が一般的です。ハッカーは、こうした攻撃でユーザ名とパスワードをさまざまに組み合わせてコンピュータにログ インしようとします。WebLogic Server では、パスワードを保護するための一連の属性を設けることで、パスワードの推測に対する保護を強化しています。

WebLogic Server デプロイメントでパスワードを保護するには、次の手順に従います。

1. Administration Console を起動します。
2. [セキュリティ] ノードをクリックします。
3. Administration Console の右ペインで [パスワード] タブをクリックします。
4. 指示に従って値を入力したり、必要なチェックボックスをチェックしたりすることで、このタブで必要な属性を定義します（詳細については次の表を参照してください）。
5. [適用] ボタンをクリックして、選択を保存します。
6. WebLogic Server を再起動します。

次の表では、[パスワード] タブの各属性について説明します。

表 14-20 パスワード保護の属性

属性	説明
[最小パスワード文字数]	パスワードに必要な文字数。パスワードは 8 文字以上でなければならない。デフォルトは 8。

表 14-20 パスワード保護の属性（続き）

属性	説明
[ロックアウト有効化]	ユーザ アカウントへの無効なログインが指定された [ロックアウトしきい値] を超えたときにそのユーザ アカウントのロックを要求する。この属性はデフォルトで有効。
[ロックアウトしきい値]	アカウントにログ インしようとする場合に、アカウントがロックアウトされるまでにユーザが間違ったパスワードを入力してもよい回数。この回数を超えてログインを試みると、(ユーザ名 / パスワードの組み合わせが正しい場合でも）セキュリティ例外が発生して、アカウントがロックアウトされる。システム管理者が明示的にロックを解除するか、またはロックアウト遅延時間が終了するまで、アカウントはロックアウトされたままとなる。ただし、無効なログインが [ロックアウト リセット遅延] 属性で定義された時間内に繰り返された場合。デフォルトは 5。
[ロックアウト遅延]	[ロックアウト リセット遅延] 属性で定義された時間内に無効なログインが一定回数以上繰り返されたためにユーザ アカウントがロックされた後、ユーザ アカウントにアクセスできるようになるまでの時間（分単位)。ユーザ アカウントをロック解除するには、`weblogic.passwordpolicy` の `unlockuser` パーミッションが必要。デフォルトでは 30 分。

表 14-20 パスワード保護の属性（続き）

属性	説明
[ロックアウト リセット遅延]	ここで指定した分単位の時間内に一定回数以上の無効なログインが試みられた場合に、ユーザのアカウントをロックする。 [ロックアウトしきい値] 属性で定義された無効なログインの試行回数が、この属性に定義された時間内に行われた場合、アカウントはロックアウトされる。たとえば、この属性の値が 5 分で、6 分間に 3 回ログインが失敗した場合、アカウントはロックされない。しかし、5 分以内に 5 回の無効なログインが繰り返された場合、アカウントはロックされる。 デフォルトでは 5 分。
[ロックアウト キャッシュ サイズ]	試行しなかったログインと試行した無効なログインのキャッシュ サイズを指定する。 デフォルトは 5。

監査プロバイダのインストール

WebLogic Server では、監査プロバイダを作成して、認証リクエストの受け取り、許可の成否、無効なデジタル証明書の提出などのセキュリティ イベントの通知を受け取ったり処理したりすることができます。

監査プロバイダを使用するには、`weblogic.security.audit.AuditProvider` インタフェースの実装を作成します。作成したら、Administration Console を使用してその実装をインストールし、アクティブにします。

監査プロバイダをインストールするには、Administration Console の [セキュリティ] ノードの [一般] タブの [監査プロバイダ クラス] 属性で、`AuditProvider` クラスの実装に名前を付けます。WebLogic Server を再起動します。

監査プロバイダの記述の詳細については、「セキュリティ イベントの監査」を参照してください。接続フィルタの作成例については、インストールされている WebLogic Server の `\samples\examples\security` ディレクトリに入っている `LogAuditProvider` のサンプルを参照してください。

接続フィルタのインストール

クライアントの出所やプロトコルに基づいてクライアントの接続を受け付けるか拒否するかを選択する接続フィルタを作成できます。クライアントが接続を完了し、何らかの処理を実行する前に、WebLogic Server は、クライアントの IP 番号とポート、プロトコル（HTTP、HTTPS、T3、T3S、または IIOP）、および WebLogic Server のポート番号を接続フィルタに渡します。この情報を調べることで、接続を許可するか、`FilterException` を生成して接続を終了するかを選択できます。

接続フィルタを使用するには、まず、`weblogic.security.net.ConnectionFilter` インタフェースの実装を作成する必要があります。作成したら、Administration Console を使用してその実装をインストールします。

接続フィルタをインストールするには、Administration Console の [セキュリティ] ウィンドウの [詳細設定] タブの [接続フィルタ] 属性で、`weblogic.security.net.ConnectionFilter` インタフェースの実装に名前を付けます。WebLogic Server を再起動します。

接続フィルタの記述の詳細については、「ネットワーク接続のフィルタリング」を参照してください。接続フィルタの作成例については、インストールされている WebLogic Server の `\samples\examples\security` ディレクトリに入っている `SimpleConnectionFilter` のサンプルを参照してください。

Java セキュリティ マネージャの設定

Java 2（JDK 1.2 または 1.3）環境で WebLogic Server を実行する場合、WebLogic Server では Java 2 の Java セキュリティ マネージャを使用して WebLogic Server リソースに追加のアクセス制御を提供できます。Java 仮想マシン（JVM）には、セキュリティ ポリシー ファイルから管理できるセキュリティ メカニズムが組み込まれています。Java セキュリティ マネージャを使用すると、CodeSource または SignedBy クラスに一連のパーミッションを強制的に付与できます。パーミッションによって、JVM のインスタンスで動作する特定のクラスが、特定の実行時の処理を行うかどうかを制御できます。多くの場合、脅威モデルでは、悪意あるコードが JVM で実行されることを想定していないため、Java セキュリティ マネージャは必要ありません。アプリケーション サービス プロバイダが WebLogic Server を使用し、未知のクラスが実行されるような場合では、Java セキュリティ マネージャが必要です。

注意： WebLogic Server の 6.0 より前のリリースでは、Java セキュリティ マネージャは、WebLogic Server の起動時に `-Dweblogic.security.manager` プロパティを使用することで有効になりました。WebLogic Server バージョン 6.0 以降ではプロパティが変更されていることに注意してください。

WebLogic Server で Java セキュリティ マネージャを使用するには、WebLogic Server の起動時に `-Djava.security.manager property` プロパティを指定します。

Java セキュリティ マネージャでは、パーミッションを定義するセキュリティ ポリシー ファイルを使用します。セキュリティ ポリシーの絶対パス名は、WebLogic Server の起動時に `-Djava.security.policy` プロパティで指定します。セキュリティ ポリシー ファイルを指定しないで Java セキュリティ マネージャを有効にする場合、Java セキュリティ マネージャでは、`$JAVA_HOME\lib\security` ディレクトリの `java.security` および `java.policy` ファイルで定義されるデフォルトのセキュリティ ポリシーを使用します。

WebLogic Server には `weblogic.policy` というサンプルのセキュリティ ポリシー ファイルがあります。このファイルにはデフォルトのパーミッションが含まれています。

WebLogic Server デプロイメントで Java セキュリティ マネージャのセキュリティ ポリシー ファイルを使用するには、次の操作を行います。

1. weblogic.policy ファイルの次の行を編集して、指定の場所を WebLogic Server のインストール先で置き換えます。

   ```
   grant codebase "file://BEA/-"{

     permission java.io.FilePermission "D:${/}BEA${/}=", ...
   ```

   注意： この変更は、インストール先ディレクトリの構造が、『BEA WebLogic Server インストール ガイド』で説明されているものと同じ構造であることを前提としています。

2. Administration Console を実行する場合は、weblogic.policy ファイルに次のような grant ブロックとパーミッションを追加します。

   ```
   grant {

       permission java.io.FilePermission
   "D:{/}BEA${/}wlserver6.1${/}weblogic${/}management${/}console${
   /}-", "read";

       permission java.io.FilePermission
   "D:{/}BEA${/}wlserver6.1${/}config${/}${/}applications${/}.wl_t
   emp_do_not_delete${/}weblogic${/}management${/}console${/}-",
   "read";

      permission java.util.PropertyPermission   "user.*",   "read";

      };
   ```

3. CLASSPATH に追加のディレクトリがある場合、または追加のディレクトリにアプリケーションをデプロイしている場合は、それらのディレクトリに対する特定のパーミッションを weblogic.policy ファイルに追加します。

4. 次のような注意事項を考慮することをお勧めします。

   - weblogic.policy ファイルのバックアップを作成し、安全な場所に保管します。
   - WebLogic Server デプロイメントの管理者は読み書き特権を持ち、その他のユーザは何の特権も持たないように、weblogic.policy ファイルにパーミッションを設定します。

5. WebLogic Server デプロイメントで Java セキュリティ マネージャと weblogic.policy ファイルを使用するには、WebLogic Server の起動時に次のようなプロパティを使用します。

   ```
   $java... -Djava.security.manager\
   ```

```
-Djava.security.policy==D:/BEA/wlserver6.1/lib/weblogic.policy
```

Java セキュリティ マネージャの詳細については、Java 2 に付属している Javadoc を参照してください。

レコーディング セキュリティ マネージャ ユーティリティの使い方

レコーディング セキュリティ マネージャ ユーティリティを使用すると、WebLogic Server の起動時または動作中に発生するパーミッションの問題を検出できます。ユーティリティでは、ユーティリティが見つけたパーミッションの問題を解決するために、セキュリティ ポリシー ファイルに追加できるパーミッションを出力します。レコーディング セキュリティ マネージャは BEA Developer Center で入手できます。

セキュリティ コンテキストの伝播のコンフィグレーション

セキュリティ コンテキストの伝播を使用すると、WebLogic Server 環境で動作している Java アプリケーションから、BEA Tuxedo ドメイン内のオブジェクトにアクセスして操作することができます。WebLogic Server の BEA WebLogic Enterprise Connectivity コンポーネントには、セキュリティ コンテキストの伝播機能があります。

セキュリティ コンテキストの伝播を使用する場合、WebLogic Server セキュリティ レルムで定義されているユーザのセキュリティ ID が、WLEC 接続プールの一部をなすネットワーク接続を通して BEA Tuxedo ドメインに送信される Internet Inter-ORB Protocol（IIOP）リクエストのサービス コンテキストの一部として伝播されます。WLEC 接続プール内の各ネットワーク接続は、定義済みのユーザ ID を使用して認証されます。

セキュリティ コンテキストの伝播を使用するには、WebLogic Server からアクセスする BEA Tuxedo ドメインごとに WLEC 接続プールを作成します。WebLogic Server は、IIOP 接続を各 WLEC 接続プールに追加します。WebLogic Server 環境の Java アプリケーションは、WLEC 接続プールから取得した IIOP 接続を使用して、BEA Tuxedo ドメインのオブジェクトを呼び出したり、処理を要求したりします。

セキュリティ コンテキストの伝播を使用する前に、*TUXDIR*\lib\wleorb.jar と *TUXDIR*\lib\wlepool.jar を、startAdminWebLogic.sh ファイルまたは startAdminWebLogic.cmd ファイルの CLASSPATH 変数に追加します。

詳細については、『WebLogic Enterprise Connectivity ユーザーズ ガイド』を参照してください。

セキュリティ コンテキストの伝播を実装するには、次の操作を行います。

1. Administration Console の左ペインで [サービス｜ WLEC] ノードを選択します。
2. Administration Console の右ペインで、[新しい WLEC Connection Pool のコンフィグレーション] リンクをクリックします。

3. 次の表の属性を定義します。

表 14-21 [一般] タブの WLEC 接続プールの属性

属性	説明
[名前]	WLEC 接続プールの名前。この名前は WLEC 接続プールごとにユニークでなければならない。

[プライマリ アドレス]	WLEC 接続プールと BEA Tuxedo ドメインとの接続を確立するために使用する IIOP リスナ / ハンドラのアドレスのリスト。各アドレスのフォーマットは、*//hostname:port*。
	アドレスは、UBBCONFIG ファイルに定義されている ISL アドレスと一致しなければならない。アドレスとアドレスの区切りにはセミコロンを使用する。たとえば、//main1.com:1024;//main2.com:1044 になる。
	SSL プロトコルを使用するよう WLEC 接続プールをコンフィグレーションするには、IIOP リスナ / ハンドラのアドレスに corbalocs プレフィックスを付ける。次に例を示す。corbalocs://*hostname:port*

[フェイルオーバー アドレス]	[プライマリ アドレス] 属性に定義されているアドレスを使って接続を確立できない場合に使用される IIOP リスナ / ハンドラのアドレスのリスト。アドレスとアドレスの区切りにはセミコロンを使用する。この属性は省略可能。
[ドメイン]	WLEC 接続プールの接続先 BEA Tuxedo ドメインの名前。WLEC 接続プールは、BEA Tuxedo ドメインにつき 1 つしか定義できない。ドメイン名は、BEA Tuxedo ドメインの UBBCONFIG ファイルの RESOURCES セクションの domainid パラメータに一致しなければならない。
[最小プール サイズ]	WebLogic Server が起動したときに、WLEC 接続プールに追加する IIOP 接続の数。デフォルトは 1。

表 14-21 [一般] タブの WLEC 接続プールの属性（続き）

属性	説明
[最大プール サイズ]	WLEC 接続プールから開始できる IIOP 接続の最大数。デフォルトは 1。

4. [作成] ボタンをクリックします。
5. Administration Console の [WLEC 接続プール] ウィンドウの [コンフィグレーション] タブにある [セキュリティ] タブで属性を定義して、WebLogic Server セキュリティ レルムのユーザのセキュリティ コンテキストを BEA Tuxedo ドメインに伝播します。次の表では、これらの属性について説明します。

表 14-22 [セキュリティ] タブの WLEC 接続プールの属性

属性	説明
[ユーザ名]	BEA Tuxedo ユーザ名。BEA Tuxedo ドメインのセキュリティ レベルが USER_AUTH、ACL、または MANDATORY_ACL の場合にのみ指定する。
[ユーザ パスワード]	[ユーザ名] 属性に定義したユーザのパスワード。[ユーザ名] 属性を定義する場合にのみ指定する。
[ユーザ ロール]	BEA Tuxedo ユーザ ロール。この属性は、BEA Tuxedo ドメインのセキュリティ レベルが APP_PW、USER_AUTH、ACL、または MANDATORY_ACL の場合にのみ指定する。
[アプリケーション パスワード]	BEA Tuxedo アプリケーション パスワード。BEA Tuxedo ドメインのセキュリティ レベルが APP_PW、USER_AUTH、ACL、または MANDATORY_ACL の場合にのみ指定する。

表 14-22 [セキュリティ] タブの WLEC 接続プールの属性（続き）

属性	説明
[最小暗号化レベル]	BEA Tuxedo ドメインと WebLogic Server との間で使用される SSL の最小暗号化レベル。指定できる値は、0、40、56、128。ゼロ（0）は、データを署名するが暗号化しないことを示す。40、56、および 128 は暗号キーの長さ（ビット単位）を指定する。最小暗号化レベルが満たされていない場合、BEA Tuxedo と WebLogic Server との SSL 接続は失敗する。デフォルトは 40。
[最大暗号化レベル]	BEA Tuxedo ドメインと WebLogic Server との間で使用される SSL の最大暗号化レベル。指定できる値は、0、40、56、128。ゼロ（0）は、データを署名するが暗号化しないことを示す。40、56、および 128 は暗号キーの長さ（ビット単位）を指定する。最小暗号化レベルが満たされていない場合、BEA Tuxedo と WebLogic Server との SSL 接続は失敗する。デフォルト値は 0。
[証明書を有効化]	証明書に基づく認証を有効にする。 デフォルトでは、証明書は無効。
[セキュリティ コンテキストを有効化]	WebLogic Server ユーザのセキュリティ コンテキストを BEA Tuxedo ドメインに渡せるようにする。 デフォルトでは、セキュリティ コンテキストは無効。

6. 変更を保存するには、[適用] ボタンをクリックして、WebLogic Server を再起動します。
7. `tpusradd` コマンドを実行して、WebLogic Server ユーザを WebLogic エンタープライズ ドメインで許可済みのユーザとして定義します。

8. ISL コマンドの -E オプションを設定して、WebLogic Server レルムから伝播されたセキュリティ コンテキストを認識して利用するよう IIOP リスナ / ハンドラをコンフィグレーションします。ISL コマンドの -E オプションでは、プリンシパル名を指定する必要があります。プリンシパル名によって、WLEC 接続プールが WebLogic エンタープライズ ドメインにログ インするために使用するプリンシパルが定義されます。プリンシパル名は、WLEC 接続プールを作成するときに [ユーザ名] 属性に定義した名前と一致しなければなりません。

WebLogic Server 環境と BEA Tuxedo 環境の間で証明書に基づく認証を使用する場合は、WebLogic Server 環境から BEA Tuxedo CORBA オブジェクトへの接続を確立するときに新しい SSL ハンドシェークが実行されます。同じ SSL ネットワーク接続を使用して複数のクライアント リクエストをサポートするには、そうした処理を行うように証明書を次のように設定しなければなりません。

1. WebLogic Server ユーザに対応するデジタル証明書を取得して、プライベート キーを BEA Tuxedo の TUXDIR\udataobj\security\keys ディレクトリに入れます。
2. BEA Tuxedo CORBA アプリケーションの UBBCONFIG ファイルでは、tpusradd コマンドを使用して WebLogic Server ユーザを BEA Tuxedo ユーザとして定義します。
3. -E オプションを使用して UBBCONFIG ファイルの IIOP リスナ / ハンドラを定義して、WebLogic Server ユーザが認証用に使用されることを示します。
4. WebLogic Server の Administration Console で WLEC 接続プールを作成するときに、[ユーザ名] 属性に WebLogic Server ユーザ名を定義します。
5. IIOP リスナ / ハンドラのデジタル証明書を取得します。
6. ISL コマンドの SEC_PRINCIPAL_NAME オプションでデジタル証明書を指定し、-S オプションを使用して、セキュア ポートを BEA Tuxedo ドメインと WebLogic Server セキュリティ レルムとの間で使用することを示します。

UBBCONFIG ファイルの詳細については、BEA Tuxedo のマニュアルの「Creating a Configuration File」を参照してください。

corbalocs プレフィックスの詳細については、BEA Tuxedo のマニュアルの「Understanding the Address Formats of the Bootstrap Object」を参照してください。

BEA Tuxedo のセキュリティ レベルの詳細については、BEA Tuxedo のマニュアルの「Defining a Security Level」を参照してください。

15　トランザクションの管理

以下の節では、トランザクション管理について説明するとともに、Administration Console でトランザクションをコンフィグレーションおよび管理する際のガイドラインを紹介します。

- トランザクション管理の概要
- トランザクションのコンフィグレーション
- トランザクションのモニタとログ
- 別のマシンへのサーバの移動

JDBC 接続プールをコンフィグレーションして JDBC ドライバを分散トランザクションに参加できるようにする方法については、16-1 ページの「JDBC 接続の管理」を参照してください。

トランザクション管理の概要

Administration Console を使用すると、JavaTransaction API（JTA）などの WebLogic Server 機能をコンフィグレーションおよび有効化するためのツールを利用できます。Administration Console を起動するには、「WebLogic Server とクラスタのコンフィグレーション」で説明されている手順を参照してください。トランザクションのコンフィグレーション プロセスでは、属性の値を指定する必要があります。指定した属性によって、以下のような、トランザクション環境のさまざまな側面を定義できます。

- トランザクションのタイムアウトと制限
- トランザクション マネージャの動作
- トランザクション ログ ファイルのプレフィックス

トランザクション環境をコンフィグレーションする前に、EJB、JDBC、JMS など、トランザクションに参加可能な J2EE コンポーネントについてよく理解しておく必要があります。

- EJB（Enterprise JavaBeans）では JTA を使用することでトランザクションがサポートされます。一部のデプロイメント記述子はトランザクション処理に関連しています。EJB と JTA を使用したプログラミングの詳細については、『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』を参照してください。
- JDBC（Java Database Connectivity）には、Java からリレーショナル データベース システムにアクセスするための標準インタフェースが用意されています。JDBC ドライバおよびトランザクション データ ソースを使用して取得された接続については、JTA によってトランザクションがサポートされます。JDBC と JTA を使用したプログラミングの詳細については、『WebLogic JDBC プログラミング ガイド』を参照してください。
- JMS（Java Messaging Service）では、JTA を使用することで複数のデータ リソースにわたるトランザクションがサポートされます。WebLogic JMS は、XA 準拠のリソース マネージャです。JMS と JTA を使用したプログラミングの詳細については、『WebLogic JMS プログラマーズ ガイド』を参照してください。

J2EE コンポーネントのコンフィグレーションの詳細については、このマニュアルの対応する章、および Administration Console オンライン ヘルプを参照してください。

トランザクションのコンフィグレーション

Administration Console では、すべての JTA コンフィグレーション属性に対して、デフォルト値が用意されています。コンフィグレーション属性に対して無効な値を指定すると、再起動時に WebLogic Server は起動しません。

JTA のコンフィグレーション設定は、ドメイン レベルで適用できます。つまり、コンフィグレーション属性の設定はドメイン内のすべてのサーバに適用されることになります。JTA のモニタ タスクおよびロギング タスクは、サーバ レベルで実行されます。

WebLogic JTA および任意のトランザクション参加コンポーネントをコンフィグレーションすると、システムは JTA API および WebLogic JTA 拡張機能を使用してトランザクションを実行できます。

アプリケーションを実行する前にトランザクション属性をコンフィグレーションすることもできますし（静的コンフィグレーション）、アプリケーションの実行時にトランザクション属性をコンフィグレーションすることもできます（動的コンフィグレーション）。ただし、後者の場合、例外が 1 つあります。[トランザクション ログファイルのプレフィックス] 属性は、アプリケーションの実行前に設定する必要があります。

トランザクション属性をコンフィグレーションするには、以下の手順に従います。

1. Administration Console を起動します。
2. 左ペインのドメイン ノードを選択します。デフォルトでは、そのドメインの [コンフィグレーション] タブが表示されます。
3. [JTA] タブを選択します。
4. 属性ごとに、値を指定するか、または適用可能な場合はデフォルト値をそのまま使用します。
5. [適用] をクリックして、新しい属性値を保存します。
6. サーバのコンフィグレーション時に [トランザクション ログファイルのプレフィックス] 属性が設定されていることを確認します。ロギングに関する属性の設定については、「トランザクションのモニタとログ」を参照してください。

表 15-1 では、WebLogic Server で使用可能なトランザクション属性について簡単に説明します。属性の詳細、および属性の有効な値とデフォルト値については、Administration Console オンライン ヘルプの「ドメイン」を参照してください。

表 15-1 トランザクション属性

属性	説明
[タイムアウト秒数]	強制ロールバックされるまで、トランザクションがアクティブ状態を継続できる時間（秒単位）。

表 15-1　トランザクション属性（続き）

属性	説明
[トランザクションを保持する最長時間]	トランザクション コーディネータがトランザクションの完了を試み続ける最長時間（秒単位）。
[beforeCompletion の反復上限]	強制ロールバックの前に処理される beforeCompletion コールバックの回数。
[最大トランザクション数]	特定のサーバ上で一度にアクティブにできるトランザクションの最大数。
[ユニーク名の最大数]	サーバが一度にトラッキングできるユニークなトランザクション名の最大数。
[ヒューリスティックを無視]	トランザクション マネージャが、ヒューリスティックな出力を得たトランザクションを無視するようリソースに指示するかどうかを指定するブール値。

トランザクションのモニタとログ

Administration Console を使用すると、トランザクションをモニタしたり、トランザクション ログ ファイルのプレフィックスを指定したりできます。モニタ タスクおよびロギング タスクは、サーバ レベルで実行されます。トランザクション統計は特定のサーバに表示され、トランザクション ログ ファイルは各サーバに格納されます。

トランザクション統計を表示し、トランザクション ログ ファイルのプレフィックスを設定するには、以下の手順に従います。

1. Administration Console を起動します。
2. 左ペインのサーバ ノードをクリックします。
3. 左ペインで特定のサーバを選択します。
4. [モニタ] タブを選択します。

5. [JTA] タブを選択します。トランザクション統計の総計が [JTA] ダイアログに表示されます。モニタに関するテキスト リンクをクリックすると、リソースや名前でトランザクションをモニタしたり、すべてのアクティブなトランザクションをモニタしたりすることもできます。
6. [ログ] タブを選択します。
7. [JTA] タブを選択します。
8. トランザクション ログ ファイルのプレフィックスを入力し、[適用] をクリックして属性値を保存します。

値と属性のモニタとログの詳細については、Administration Console オンライン ヘルプの「サーバ」を参照してください。

別のマシンへのサーバの移動

アプリケーション サーバが別のマシンに移動された場合、サーバは新しいディスクにあるトランザクション ログ ファイルを見つけられなければなりません。このため、トランザクション ログ ファイルを新しいマシンに移動してからサーバを起動することをお勧めします。そうすることによって、確実に適切な回復処理を実行できます。新しいマシンでパス名が異なる場合は、[トランザクション ログファイルのプレフィックス] 属性を新しいパス名で更新してからサーバを起動します。

サーバに障害が発生した後でトランザクション ログを移行する場合は、すべてのトランザクション ログ ファイルを新しいマシンで使用可能にしてから、マシンでサーバを起動します。このような移行は、両方のマシンで使用可能なデュアル ポート ディスクにトランザクション ログ ファイルを格納することで実行できます。計画的な移行の場合は、新しいマシンでパス名が異なるときに、[トランザクション ログファイルのプレフィックス] 属性を新しいパス名で更新してからサーバを起動します。必ず、新しいマシンですべてのトランザクション ログ ファイルが使用可能になっていることを確認してから、サーバを起動してください。そうしないと、クラッシュ時にコミット中だったトランザクションが適切に解決できず、その結果、アプリケーション データに矛盾が発生する場合があります。

16 JDBC 接続の管理

以下の節では、ローカル トランザクションと分散トランザクションの両方における、JDBC コンポーネント（データ ソース、接続プール、およびマルチプール）を介したデータベース接続のコンフィグレーションと管理のガイドラインを紹介します。

- 16-1 ページの「JDBC 管理の概要」
- 16-4 ページの「JDBC コンポーネント（接続プール、データ ソース、およびマルチプール）」
- 16-6 ページの「接続プール、マルチプール、およびデータソースの JDBC コンフィグレーションガイドライン」
- 16-21 ページの「Administration Console による JDBC 接続プール、マルチプール、およびデータソースのコンフィグレーションと管理」

JDBC 管理の概要

Administration Console には、JDBC（Java Database Connectivity）などの WebLogic Server 機能のコンフィグレーションと管理を可能にするツールへのインタフェースが用意されています。接続の作成、管理、およびモニタを含むほとんどの JDBC 管理機能において、システム管理者は Administrative Console またはコマンドライン インタフェースを使用します。アプリケーション開発者は JDBC API を使用することもできます。

以下に、接続を設定および管理するためによく行われるタスクを示します。

- WebLogic Server とデータベース管理システム間の JDBC 接続を制御する属性の定義
- 確立された接続の管理
- 確立された接続のモニタ

Administration Console について

JDBC 接続の設定と管理は、おもに Administration Console で行います。Administration Console を使用して、サーバを起動する前に接続を静的に設定します。詳細については、1-4 ページの「Administration Console の起動」を参照してください。

接続を設定するだけでなく、Administration Console では確立された接続を管理およびモニタすることもできます。

コマンドライン インタフェースについて

コマンドライン インタフェースでは、接続プールを動的に作成および管理できます。コマンドライン インタフェースの使い方については、B-1 ページの「WebLogic Server コマンドライン インタフェース リファレンス」を参照してください。

JDBC API について

プログラミングによる接続の設定と管理については、http://edocs.beasys.co.jp/e-docs/wls61/jdbc/index.html の『WebLogic JDBC プログラミング ガイド』を参照してください。

関連情報

ローカルおよび分散トランザクションで使用される JDBC ドライバは多くの WebLogic Server コンポーネントと連係して機能し、情報はさまざまなドキュメントに掲載されています。たとえば、JDBC ドライバについての情報は、JDBC、JTA、および WebLogic jDrivers のマニュアルで参照できます。

JDBC、JTA、および管理の追加リソースのリストを以下に示します。

管理

- Administration Console を開く手順については、4-1 ページの「WebLogic Server とクラスタのコンフィグレーション」を参照してください。
- JDBC 属性のリストについては、http://edocs.beasys.co.jp/e-docs/wls61/ConsoleHelp/index.html にある WebLogic Administration Console オンライン ヘルプの「JDBC 接続プール」、「JDBC データ ソース」、「JDBC マルチプール」、および「JDBC トランザクション データ ソース」を参照してください。
- コマンドライン インタフェースの使い方については、B-1 ページの「WebLogic Server コマンドライン インタフェース リファレンス」を参照してください。

JDBC と WebLogic jDrivers

以下のドキュメントは、おもにアプリケーション開発者向けに書かれています。システム管理者は、このドキュメントの内容を理解するための補助資料として、必要に応じて以下の初歩的な情報を参照してください。

- JDBC API については、『WebLogic JDBC プログラミング ガイド』を参照してください。「WebLogic JDBC の概要」という節で、JDBC および JDBC ドライバの概要が簡潔に紹介されています。
- WebLogic jDrivers の使い方については、http://edocs.beasys.co.jp/e-docs/wls61/oracle/index.html の『WebLogic jDriver for Oracle のインストールと使い方』、http://edocs.beasys.co.jp/e-docs/wls61/mssqlserver4/index.html の『WebLogic jDriver for Microsoft SQL Server のインストールと使い方』、または http://edocs.beasys.co.jp/e-docs/wls61/informix4/index.html の『WebLogic jDriver for Informix のインストールと使い方』を参照してください。

トランザクション（JTA）

- JTA の管理については、15-1 ページの「トランザクションの管理」を参照してください。
- サードパーティ ドライバの使い方については、http://edocs.beasys.co.jp/e-docs/wls61/jta/thirdpartytx.html の『WebLogic JTA プログラマーズ ガイド』の「WebLogic Server でのサードパーティ製 JDBC XA ドライバの使い方」を参照してください。

以下のドキュメントは、おもにアプリケーション開発者向けに書かれています。システム管理者は、この章の内容を理解するための補助資料として、必要に応じて以下の情報を参照してください。

- 分散トランザクションについては、http://edocs.beasys.co.jp/e-docs/wls61/jta/index.html の『WebLogic JTA プログラマーズ ガイド』を参照してください。
- WebLogic jDriver for Oracle/XA の使い方については、http://edocs.beasys.co.jp/e-docs/wls61/oracle/trxjdbcx.html の『WebLogic jDriver for Oracle のインストールと使い方』の「分散トランザクションでの WebLogic jDriver for Oracle/XA の使い方」を参照してください。

JDBC コンポーネント（接続プール、データ ソース、およびマルチプール）

以降の節では、JDBC 接続コンポーネント（接続プール、マルチプール、およびデータ ソース）の概要を説明します。

接続プール

接続プールとは、接続プールが登録されるとき（通常は WebLogic Server の起動時）に作成される JDBC 接続のグループに名前を付けたものです。アプリケーションはプールから接続を「借り」、使用後は接続をクローズしてプールに返し

ます。詳細については、http://edocs.beasys.co.jp/e-docs/wls61/jdbc/programming.html の『WebLogic JDBC プログラミング ガイド』の「接続プール」を参照してください。

Administration Console で行う設定はすべて静的なものです。つまり、すべての設定は WebLogic Server の起動前に行います。動的な接続プールは、コマンドライン（B-1 ページの「WebLogic Server コマンドライン インタフェース リファレンス」を参照）または API（『WebLogic JDBC プログラミング ガイド』の「動的接続プールの作成」を参照）を使用することで（サーバの起動後に）作成できます。

マルチプール

単一または複数の WebLogic Server コンフィグレーションのローカル（非分散）トランザクションで使用します。マルチプールは、以下の機能で役立ちます。

- ロードバランシング ― プールは特定の順序付けなしで追加され、ラウンドロビン方式でアクセスされます。接続を切り替えるときには、最後にアクセスされたプールのすぐ後の接続プールが選択されます。
- 高可用性 ― プールは、接続プールの切り替え順を指定する順序付きリストとして設定します。たとえば、リストの最初のプールが選択され、その後に次のプールが選択されます。

特定の接続プール内のすべての接続は同じです。つまり、それらは 1 つのデータベースにアタッチされています。ただし、マルチプール内の接続プールにはそれぞれ異なる DBMS を関連付けることができます。詳細については、http://edocs.beasys.co.jp/e-docs/wls61/jdbc/programming.html の『WebLogic JDBC プログラミング ガイド』の「マルチプール」を参照してください。

データ ソース

データ ソース オブジェクトによって、JDBC クライアントは DBMS 接続を取得できるようになります。各データ ソース オブジェクトは、接続プールまたはマルチプールを指します。データ ソース オブジェクトは、JTA を使用して定義することも、使用せずに定義することもできます。JTA によって分散トランザク

ションのサポートが提供されます。詳細については、http://edocs.beasys.co.jp/e-docs/wls61/jdbc/programming.html の『WebLogic JDBC プログラミング ガイド』の「データ ソース」を参照してください。

注意： マルチプールは分散トランザクションでサポートされていないので、トランザクション データ ソースはマルチプールを指すことができません（接続プールは可）。

接続プール、マルチプール、およびデータソースの JDBC コンフィグレーションガイドライン

この節では、ローカル トランザクションおよび分散トランザクションに対応する JDBC コンフィグレーションのガイドラインについて説明します。

JDBC コンフィグレーションの概要

JDBC 接続を設定するには、Administration Console（動的接続プールの場合はコマンドライン）で属性を定義して、接続プール、データ ソース オブジェクト（常に設定することが望ましいが、省略可能な場合もある）、およびマルチプール（省略可能）をコンフィグレーションします。トランザクションのタイプは以下の 3 つです。

- ローカル トランザクション ― 非分散トランザクション
- 分散トランザクション（XA 対応ドライバ）― 2 フェーズ コミット
- 分散トランザクション（XA 非対応ドライバ）― 単一リソース マネージャと単一データベース インスタンス

次の表にローカルおよび分散トランザクションでのこれらのオブジェクトの使い方を示します。

表 16-1 JDBC コンフィグレーション ガイドラインの概要

説明 / オブジェクト	ローカル トランザクション	分散トランザクション XA 対応ドライバ	分散トランザクション XA 非対応ドライバ
JDBC ドライバ	■ WebLogic jDriver for Oracle、WebLogic jDriver for Microsoft SQL Server、および WebLogic jDriver for Informix ■ 準拠するサードパーティ ドライバ	■ WebLogic jDriver for Oracle/ XA ■ 準拠するサードパーティ ドライバ	■ WebLogic jDriver for Oracle、WebLogic jDriver for Microsoft SQL Server、および WebLogic jDriver for Informix ■ 準拠するサードパーティ ドライバ
データ ソース	データ ソース オブジェクト推奨（データ ソースがない場合は JDBC API を使用）。	トランザクション データ ソースが必須。	トランザクションデータ ソース必須。 複数のリソースの場合は `enable two-phase commit=true` を設定する。16-19 ページの「分散トランザクション用の XA 非対応 JDBC ドライバのコンフィグレーション」を参照。
接続プール	Administration Console でコンフィグレーションするときにはデータ ソース オブジェクトが必須。	トランザクション データ ソースが必須。	トランザクションデータ ソースが必須。
マルチプール	接続プールとデータ ソースが必須。	分散トランザクションではサポートされていない。	分散トランザクションではサポートされていない。

注意： 分散トランザクションでは WebLogic jDriver for Oracle のトランザクション モードである WebLogic jDriver for Oracle/XA を使用します。

ローカル トランザクションをサポートするドライバ

- JDBC コア 2.0 API（`java.sql`）をサポートする JDBC 2.0 ドライバ。WebLogic jDrivers for Oracle、WebLogic jDrivers for Microsoft SQL Server、および WebLogic jDrivers for Informix など。この API を使用すると、データ ソースへの接続を確立し、クエリを送信し、その結果を処理するのに必要なクラス オブジェクトを作成できます。

分散トランザクションをサポートするドライバ

- JDBC 2.0 分散トランザクション標準拡張インタフェース（`javax.sql.XADataSource`、`javax.sql.XAConnection`、`javax.transaction.xa.XAResource`）をサポートする JDBC 2.0 ドライバ。WebLogic jDriver for Oracle/XA など。
- JDBC 2.0 コア API はサポートするが、JDBC 2.0 分散トランザクション標準拡張インタフェースはサポートしない JDBC ドライバ。XA 非対応の JDBC ドライバは、一度に 1 つしか分散トランザクションに参加できません。16-19 ページの「分散トランザクション用の XA 非対応 JDBC ドライバのコンフィグレーション」を参照してください。

JDBC ドライバのコンフィグレーション

この節では、ローカルおよび分散トランザクションに対応するドライバのコンフィグレーション方法について説明します。

ローカル トランザクション用の JDBC ドライバのコンフィグレーション

ローカル トランザクションに対応する JDBC ドライバをコンフィグレーションするには、次の手順に従って JDBC 接続プールを設定します。

- Driver Classname 属性に、`java.sql.driver` インタフェースをサポートしているクラスの名前を指定します。
- データ プロパティを指定します。これらのプロパティは、指定した `Driver` にデータ ソース プロパティとして渡されます。

WebLogic 2 層 JDBC ドライバの詳細については、使用するドライバについての BEA のマニュアル（http://edocs.beasys.co.jp/e-docs/wls61/oracle/index.html の『WebLogic jDriver for Oracle のインストールと使い方』、http://edocs.beasys.co.jp/e-docs/wls61/mssqlserver4/index.html の『WebLogic jDriver for Microsoft SQL Server のインストールと使い方』、または http://edocs.beasys.co.jp/e-docs/wls61/informix4/index.html の『WebLogic jDriver for Informix のインストールと使い方』）を参照してください。サードパーティ ドライバを使用する場合は、http://edocs.beasys.co.jp/e-docs/wls61/jta/thirdpartytx.html の『WebLogic JTA プログラマーズ ガイド』の「分散トランザクションでの WebLogic jDriver for Oracle/XA の使い方」およびベンダ提供のマニュアルを参照してください。以下の表では、WebLogic jDrivers を使用した JDBC 接続プールおよびデータ ソースのサンプル コンフィグレーションを示します。

次の表では、WebLogic jDriver for Oracle を使用した接続プールのサンプル コンフィグレーションを示します。

注意： 新しいプロパティ名として「Password」があります。この値は、名前と値のペアで定義されたプロパティ内のパスワードをオーバーライドします。この属性は、物理的なデータベース接続の作成時に 2 層 JDBC ドライバに渡されます。値は、暗号化されて config.xml に格納され、そのファイルにクリアテキスト パスワードが格納されるのを防止するために使用できます。

表 16-2　WebLogic jDriver for Oracle：接続プールのコンフィグレーション

属性名	属性値
Name	`myConnectionPool`
Targets	`myserver`
DriverClassname	`weblogic.jdbc.oci.Driver`
Initial Capacity	`0`
MaxCapacity	`5`
CapacityIncrement	`1`
Properties	`user=scott;server=localdb`
Password	`Tiger`（この値は Properties で名前と値の組み合わせとして定義されているすべてのパスワードをオーバーライドする）

次の表では、WebLogic jDriver for Oracle を使用したデータ ソースのサンプル コンフィグレーションを示します。

表 16-3　WebLogic jDriver for Oracle：データ ソースのコンフィグレーション

属性名	属性値
Name	`myDataSource`
Targets	`myserver`
JNDIName	`myconnection`
PoolName	`myConnectionPool`

次の表では、WebLogic jDriver for Microsoft SQL Server を使用した接続プールのサンプル コンフィグレーションを示します。

表 16-4 WebLogic jDriver for Microsoft SQL Server : 接続プールのコンフィグレーション

属性名	属性値
Name	`myConnectionPool`
Targets	`myserver`
DriverClassname	`weblogic.jdbc.mssqlsvr4.Driver`
Initial Capacity	`0`
MaxCapacity	`5`
CapacityIncrement	`1`
Properties	`user=sa;password=secret;db=pubs;server=myHost:1433;appname=MyApplication;hostname=myhostName`

次の表では、WebLogic jDriver for Microsoft SQL Server を使用したデータ ソースのサンプル コンフィグレーションを示します。

表 16-5 WebLogic jDriver for Microsoft SQL Server : データ ソースのコンフィグレーション

属性名	属性値
Name	`myDataSource`
Targets	`myserver`
JNDIName	`myconnection`
PoolName	`myConnectionPool`

次の表では、WebLogic jDriver for Informix を使用した接続プールのサンプル コンフィグレーションを示します。

表 16-6　WebLogic jDriver for Informix：接続プールのコンフィグレーション

属性名	属性値
Name	`myConnectionPool`
Targets	`myserver`
DriverClassname	`weblogic.jdbc.informix4.Driver`
Initial Capacity	`0`
MaxCapacity	`5`
CapacityIncrement	`1`
Properties	`user=informix;password=secret;server=myDBHost;port=1493;db=myDB`

次の表では、WebLogic jDriver for Informix を使用したデータ ソースのサンプル コンフィグレーションを示します。

表 16-7　WebLogic jDriver for Informix：データ ソースのコンフィグレーション

属性名	属性値
Name	`myDataSource`
Targets	`myserver`
JNDIName	`myconnection`
PoolName	`myConnectionPool`

分散トランザクション用の XA 対応 JDBC ドライバのコンフィグレーション

XA 対応 JDBC ドライバを分散トランザクションに参加させるには、以下のように JDBC 接続プールをコンフィグレーションします。

- `Driver Classname` 属性に、`javax.sql.XADataSource` インタフェースをサポートしているクラスの名前を指定します。
- データベース プロパティが指定されていることを確認します。これらのプロパティは、指定した `XADataSource` にデータ ソース プロパティとして渡されます。WebLogic jDriver for Oracle のデータ ソース プロパティについては、「WebLogic jDriver for Oracle/XA のデータ ソース プロパティ」を参照してください。サード パーティ製ドライバのデータ ソース プロパティについては、ベンダが提供するマニュアルを参照してください。

以下の属性は、XA モードで WebLogic jDriver for Oracle を使用する場合の JDBC 接続プールのコンフィグレーションの例です。

表 16-8 WebLogic jDriver for Oracle/XA：接続プールのコンフィグレーション

属性名	属性値
Name	`fundsXferAppPool`
Targets	`myserver`
DriverClassname	`weblogic.jdbc.oci.xa.XADataSource`
Initial Capacity	`0`
MaxCapacity	`5`
CapacityIncrement	`1`
Properties	`user=scott;password=tiger;server=localdb`

以下の属性は、XA モードで WebLogic jDriver for Oracle を使用する場合のトランザクション データ ソースのコンフィグレーションの例です。

表 16-9 WebLogic jDriver for Oracle/XA：トランザクション データ ソース

属性名	属性値
Name	fundsXferData Source
Targets	myserver
JNDIName	myapp.fundsXfer
PoolName	fundsXferAppPool

また、JDBC 接続プールをコンフィグレーションして、XA モードでサード パーティ ベンダ製ドライバを使用することもできます。この場合、データ ソース プロパティは、JavaBeans 設計パターンを使用し、XADataSource インスタンスに反映して設定します。つまり、abc というプロパティの場合、XADataSource インスタンスは、getAbc という名前の取得メソッドと、setAbc という名前の設定メソッドをサポートする必要があります。

以下の属性は、Oracle Thin Driver を使用する場合の JDBC 接続プールのコンフィグレーションの例です。

表 16-10 Oracle Thin ドライバ：接続プールのコンフィグレーション

属性名	属性値
Name	jtaXAPool
URL	jdbc:oracle:thin:@baybridge:1521:bay817
Targets	myserver,server1
DriverClassname	oracle.jdbc.xa.client.OracleXADataSource
Initial Capacity	1
MaxCapacity	20
CapacityIncrement	2
Properties	user=scott;password=tiger

以下の属性は、Oracle Thin Driver を使用する場合のトランザクション データ ソースのコンフィグレーションの例です。

表 16-11 Oracle Thin ドライバ : トランザクション データ ソースのコンフィグレーション

属性名	属性値
Name	jtaXADS
Targets	myserver,server1
JNDIName	jtaXADS
PoolName	jtaXAPool

Cloudscape ドライバで使用する JDBC 接続プールは、以下のようにコンフィグレーションします。

表 16-12 Cloudscape : 接続プールのコンフィグレーション

属性名	属性値
Name	jtaXAPool
Targets	myserver,server1
DriverClassname	COM.cloudscape.core.XADataSource
Initial Capacity	1
MaxCapacity	10
CapacityIncrement	2
Properties	databaseName=CloudscapeDB
SupportsLocalTransaction	true

Cloudscape ドライバで使用するトランザクション データ ソースは、以下のようにコンフィグレーションします。

表 16-13 Cloudscape：トランザクション データ ソースのコンフィグレーション

属性名	属性値
Name	jtaZADS
Targets	myserver,myserver1
JNDIName	JTAXADS
PoolName	jtaXAPool

WebLogic jDriver for Oracle/XA のデータ ソース プロパティ

表 16-14 に、WebLogic jDriver for Oracle でサポートされているデータ ソース プロパティを示します。「JDBC 2.0」カラムは、特定のデータ ソース プロパティが JDBC 2.0 の標準データ ソース プロパティ（Y）か、または JDBC に対する WebLogic Server の拡張（N）かを示します。

「省略可能」カラムは、特定のデータ ソース プロパティが省略可能かどうかを示します。「Y*」マークが付いたプロパティは、表 16-14 に示された、Oracle の xa_open 文字列（openString プロパティの値）の対応するフィールドにマップされます。それらのプロパティが指定されない場合、デフォルト値は openString プロパティから取得されます。それらのプロパティが指定される場合、その値は openString プロパティで指定された値と一致する必要があります。プロパティが一致しない場合、XA 接続を確立しようとすると SQLException が送出されます。

「N*」マークが付いた必須プロパティも、Oracle の xa_open 文字列の対応するフィールドにマップされます。これらのプロパティは、Oracle の xa_open 文字列を指定するときに指定します。プロパティが指定されない場合や、指定されていても一致しない場合は、XA 接続を確立しようとすると SQLException が送出されます。

「**」マークが付いたプロパティ名はサポートされていますが、WebLogic Server では使用されません。

表 16-14 WebLogic jDriver for Oracle/XA のデータ ソース プロパティ

プロパティ名	タイプ	説明	JDBC 2.0	省略可能	デフォルト値
databaseName**	String	サーバ上の特定のデータベース名。	Y	Y	なし
dataSourceName	String	データ ソース名。基になる XADataSource に名前を付ける場合に使用される。	Y	Y	接続プール名
description	String	このデータ ソースの説明。	Y	Y	なし
networkProtocol**	String	サーバと通信する場合に使用されるネットワーク プロトコル。	Y	Y	なし
password	String	データベースのパスワード。	Y	N*	なし
portNumber**	int	サーバがリクエストをリスンしているポート番号。	Y	Y	なし
roleName**	String	初期 SQL ロール名。	Y	Y	なし
serverName	String	データベース サーバ名。	Y	Y*	なし
user	String	ユーザのアカウント名。	Y	N*	なし
openString	String	Oracle の XA オープン文字列。	N	Y	なし

表 16-14　WebLogic jDriver for Oracle/XA のデータ ソース プロパティ（続き）

プロパティ名	タイプ	説明	JDBC 2.0	省略可能	デフォルト値
oracleXATrace	String	XA トレース出力が有効かどうかを示す。有効（true）の場合、xa_poolnamedate.trc 形式の名前を持つファイルがサーバの起動ディレクトリに配置される。	N	Y	true

表 16-15 に、Oracle の xa_open 文字列フィールドとデータ ソース プロパティの間のマッピングを示します。

表 16-15　xa_open 文字列名と JDBC データ ソース プロパティとのマッピング

Oracle xa_open 文字列フィールド名	JDBC 2.0 データ ソース プロパティ	省略可能
acc	user、password	N
sqlnet	ServerName	

また、ユーザは Oracle の xa_open 文字列で「Threads=true」と指定する必要があります。Oracle の xa_open 文字列フィールドの詳細については、Oracle のドキュメントを参照してください。

分散トランザクション用の XA 非対応 JDBC ドライバのコンフィグレーション

JDBC 接続プールをコンフィグレーションして、XA 非対応の JDBC ドライバを別のリソースと共に分散トランザクションに参加させる場合は、JDBC Tx Data Source に [2 フェーズ コミットを有効化] 属性を指定します（このパラメータは、`XAResource` インタフェースををサポートしているリソースには無視されます）。非 XA 接続プールは、一度に 1 つしか分散トランザクションに参加できません。

XA 非対応ドライバ / 単一リソース

XA 非対応ドライバを 1 つだけ使用し、それがトランザクションで唯一のリソースである場合は、Console において [2 フェーズ コミットを有効化] オプションを選択されていないままにします（デフォルトの enableTwoPhaseCommit = false を受け入れる）。この場合は、トランザクション マネージャが 1 フェーズの最適化を実行します。

XA 非対応ドライバ / 複数リソース

他の XA リソースとともに XA 非対応 JDBC ドライバを 1 つ使用する場合は、Console で [2 フェーズ コミットを有効化] を選択します（`enableTwoPhaseCommit = true`）。

`enableTwoPhaseCommit` が `true` に設定されている場合、XA 非対応の JDBC リソースは常に `XAResource.prepare()` メソッド呼び出し中に `XA_OK` を返します。リソースは、以降の `XAResource.commit()` 呼び出しまたは `XAResource.rollback()` 呼び出しに応答して、そのローカル トランザクションをコミットまたはロールバックしようとします。リソースのコミットまたはロールバックが失敗すると、ヒューリスティック エラーが発生します。ヒューリスティック エラーの結果、アプリケーション データは矛盾した状態のまま残される場合があります。

Console で [2 フェーズ コミットを有効化] が選択されていない（`enableTwoPhaseCommit` が `false` に設定されている）場合、XA 非対応の JDBC リソースによって `XAResource.prepare()` が失敗します。この場合、`commit()` が `SystemException` を送出するので、トランザクションには参加コンポーネントが 1 つしか存在しないことになります。トランザクションの参加コ

ンポーネントが 1 つしか存在しない場合、1 つのフェーズ最適化は XAResource.prepare() を無視し、トランザクションはほとんどのインスタンスで正常にコミットします。

次の表では、XA 非対応 JDBC ドライバを使用するサンプル JDBC 接続プールのコンフィグレーション属性を示します。

表 16-16 WebLogic jDriver for Oracle：接続プールのコンフィグレーション

属性名	属性値
Name	fundsXferAppPool
Targets	myserver
URL	jdbc:weblogic:oracle
DriverClassname	weblogic.jdbc.oci.Driver
Initial Capacity	0
MaxCapacity	5
CapacityIncrement	1
Properties	user=scott;password=tiger;server=localdb

次の表では、XA 非対応 JDBC ドライバを使用するサンプル トランザクション データ ソースのコンフィグレーション属性を示します。

表 16-17 WebLogic jDriver for Oracle: トランザクション データ ソースのコンフィグレーション

属性名	属性値
Name	fundsXferDataSource
Targets	myserver,server1
JNDIName	myapp.fundsXfer
PoolName	fundsXferAppPool
EnableTwoPhaseCommit	true

Administration Console による JDBC 接続プール、マルチプール、およびデータソースのコンフィグレーションと管理

以降の節では、JDBC コンポーネント（接続プール、データ ソース、およびマルチプール）をコンフィグレーションしてデータベース接続を設定する方法について説明します。接続がいったん確立されたら、Administration Console またはコマンドライン インタフェースを使用して接続を管理およびモニタできます。コンフィグレーション タスクの説明および Administration Console オンライン ヘルプのリンクについては、表 16-19 を参照してください。

JDBC コンフィグレーション

ここでは、コンフィグレーションとは以下のプロセスのことです。

JDBC オブジェクトの作成

Administration Console を使用し、属性とデータベース プロパティを指定して JDBC コンポーネント（接続プール、データ ソース、およびマルチプール）を作成します。16-23 ページの「Administration Console を使用した JDBC 接続のコンフィグレーション」を参照してください。

まず接続プールまたはマルチプールを作成してから、データ ソースを作成します。データ ソース オブジェクトを作成する場合、データ ソースの属性の 1 つとして接続プールまたはマルチプールを作成します。これにより、データ ソースが特定の接続プールまたはマルチプール（「プール」）と永続的に関連付けられます。

JDBC オブジェクトの割り当て

データ ソースと接続プール（またはマルチプール）のコンフィグレーションと関連付けを行ったら、各オブジェクトを同じサーバまたはサーバまたはクラスタに割り当てます。一般的なシナリオをいくつか以下に示します。

- クラスタでは、データ ソースをクラスタに割り当て、関連付けられている接続プールをクラスタ内の各管理対象サーバに割り当てます。
- 単一サーバのコンフィグレーションでは、各データ ソースとその関連付けられている接続プールをサーバに割り当てます。
- マルチプールを使用する場合は、接続プールをマルチプールに割り当て、データ ソースとすべての接続プールおよびマルチプールをサーバまたはクラスタに割り当てます。

実行するタスクの説明については、16-23 ページの「Administration Console を使用した JDBC 接続のコンフィグレーション」を参照してください。

コンフィグレーション プロセスの関連付けと割り当ての詳細については、次の表を参照してください。

表 16-18 関連付けと割り当てのシナリオ

シナリオ番号	関連付け	割り当て	対象の説明
1	データ ソース A と接続プール A を関連付ける。	1. データ ソース A を管理対象サーバ 1 に割り当てる。 2. 接続プール A を管理対象サーバ 1 に割り当てる。	データ ソースと接続プールは同じ対象に割り当てられる。
2	データ ソース B と接続プール B を関連付ける。	1. データ ソース B をクラスタ X に割り当てる。 2. 接続プール B をクラスタ X の管理対象サーバ 2 に割り当てる。	データ ソースと接続は関連するサーバ / クラスタの対象に割り当てられる。

表 16-18 関連付けと割り当てのシナリオ（続き）

シナリオ番号	関連付け	割り当て	対象の説明
3	データ ソース C と接続プール C を関連付ける。	■ データ ソース A と接続プール A を管理対象サーバ 1 に割り当てる。 および ■ データ ソース A をクラスタ X に割り当て、接続プール A をクラスタ X の管理対象サーバ 2 に割り当てる。	データ ソースと接続プールは 1 つのまとまりとして 2 つの異なる対象に割り当てられる。

（プールには複数のデータ ソースを割り当てることができますが、実際の効果はありません。）これらのデータ ソースとプールの組み合わせを複数のサーバまたはクラスタに割り当てることができますが、それらは組み合わせとして割り当てなければなりません。たとえば、関連付けられている接続プールがサーバ B にのみ割り当てられている場合は、データ ソースを管理対象サーバ A に割り当てることはできません。

コマンドライン インタフェースを使用すると、動的な接続プールを（サーバの起動後に）コンフィグレーションできます。16-25 ページの「コマンドライン インタフェースを使用した JDBC コンフィグレーション タスク」を参照してください。動的な接続プールは、API を使用してプログラミングによってコンフィグレーションすることもできます（『WebLogic JDBC プログラミング ガイド』の「動的接続プールの作成」を参照）。

Administration Console を使用した JDBC 接続のコンフィグレーション

Administration Console では、JDBC 接続をコンフィグレーション、管理、およびモニタできます。タスクに使用するタブを表示するには、次の操作を行います。

1. Administration Console を起動します。
2. 左ペインで [サービス] ノードを選択し、[JDBC] ノードを展開します。

3. コンフィグレーションまたは管理するコンポーネント（接続プール、マルチプール、データ ソース、またはトランザクション データ ソース）のタブを選択します。
4. オンライン ヘルプの指示に従います。オンライン ヘルプへのリンクについては、表 16-19 を参照してください。

次の表では、接続タスクを一般的な実行順序で示します。この順序は変更してもかまいません。ただし、オブジェクトは関連付けおよび割り当ての前にコンフィグレーションする必要があります。

表 16-19　JDBC のコンフィグレーション タスク

タスク番号	JDBC コンポーネント / タスク	説明
1	接続プールのコンフィグレーション	[コンフィグレーション] タブで、名前、URL、データベース プロパティなどの接続プールの属性を設定する。
2	接続プールのクローンの作成（省略可能）	このタスクでは接続プールをコピーする。[コンフィグレーション] タブで、プールの名前をユニークな名前に変更し、それ以外の属性をそのまま使用するか変更する。この機能は、別々の名前で複数の同じプール コンフィグレーションが必要な場合に便利。たとえば、各データベース管理者に、特定のプールを使用してデータベースへの個々の変更をトラッキングさせることができる。
3	マルチプールのコンフィグレーション（省略可能）	[コンフィグレーション] タブで、名前およびアルゴリズム（高可用性またはロードバランシング）の属性を設定する。[プール] タブで、接続プールをこのマルチプールに割り当てる。
4	データ ソースのコンフィグレーション（およびプールとの関連付け）	[コンフィグレーション] タブを使用して、名前、JNDI 名、およびプール名（これでデータ ソースが特定の接続プールまたはマルチプールと関連付けられる）といったデータ ソースの属性を設定する。

表 16-19 JDBC のコンフィグレーション タスク（続き）

タスク番号	JDBC コンポーネント / タスク	説明
5	トランザクション データ ソースのコンフィグレーション（および接続プールとの関連付け）	[コンフィグレーション] タブを使用して、名前、JNDI 名、および接続プール名（これでデータ ソースが特定のプールと関連付けられる）といったトランザクション データ ソースの属性を設定する。 **注意：** トランザクション データ ソースはマルチプールとは関連付けない。マルチプールは分散トランザクションでサポートされていない。
6	接続プールのサーバ / クラスタへの割り当て	[対象] タブを使用して、接続プールを 1 つまたは複数のサーバまたはクラスタに割り当てる。表 16-18「関連付けと割り当てのシナリオ」を参照。
7	マルチプールのサーバまたはクラスタへの割り当て	[対象] タブを使用して、コンフィグレーションされたマルチプールをサーバまたはクラスタに割り当てる。

コマンドライン インタフェースを使用した JDBC コンフィグレーション タスク

次の表では、動的な接続プールを作成する方法を示します。

表 16-20 接続の設定 — 動的

目的とする作業	使用するツール
動的接続プールの作成	■ コマンドライン B-20 ページの「CREATE_POOL」、または ■ API — 『WebLogic JDBC プログラミング ガイド』の「WebLogic JDBC 機能のコンフィグレーション」を参照

詳細については、B-1 ページの「WebLogic Server コマンドライン インタフェース リファレンス」、および『WebLogic JDBC プログラミング ガイド』の「動的接続プールの作成」を参照してください。

接続の管理とモニタ

接続の管理では、確立された JDBC コンポーネントを有効化、無効化、および削除します。

Administration Console を使用した JDBC の管理

JDBC 接続を管理およびモニタするには、次の表を参照してください。

表 16-21　JDBC 管理タスク

目的とする作業	Administration Console での操作
接続プールの割り当て先サーバまたはクラスタの変更	「接続プールのサーバ / クラスタへの割り当て」の手順に従って、[対象] タブで対象を選択解除（[選択済み] から [使用可能] に移動）し、新しい対象に割り当てる。
マルチプールの割り当て先クラスタの変更	「マルチプールのサーバまたはクラスタへの割り当て」の手順に従って、[対象] タブで対象を選択解除（[選択済み] から [使用可能] に移動）し、新しい対象に割り当てる。
JDBC 接続プールの削除	オンライン ヘルプの「JDBC 接続プールの削除」を参照。

表 16-21 JDBC 管理タスク（続き）

目的とする作業	Administration Console での操作
マルチプールの削除	1. 左ペインの [マルチプール] ノードを選択する。右ペインに [マルチプール] テーブルが表示され、ドメインで定義された全マルチプールが示される。 2. 削除するマルチプールの行にある [削除] アイコンをクリックする。削除要求の確認を求めるメッセージが右ペインに表示される。 3. [はい] をクリックしてマルチプールを削除する。[マルチプール] ノードの下の [マルチプール] アイコンが削除される。
データ ソースの削除	1. 左ペインの [データ ソース] ノードを選択する。右ペインに [データ ソース] テーブルが表示され、ドメインで定義された全データ ソースが示される。 2. 削除するデータ ソースの行にある [削除] アイコンをクリックする。削除要求の確認を求めるメッセージが右ペインに表示される。 3. [はい] をクリックしてデータ ソースを削除する。[データ ソース] ノードの下の [データ ソース] アイコンが削除される。
接続プールのモニタ	1. 左ペインでプールを選択する。 2. 右ペインで [モニタ] タブを選択し、[すべてのアクティブなプールのモニタ] リンクを選択する。

表 16-21 JDBC 管理タスク（続き）

目的とする作業	Administration Console での操作
接続プール、マルチプール、またはデータ ソースの属性の修正	1. 左ペインで JDBC オブジェクト（接続プール、マルチプール、またはデータ ソース）を選択する。 2. 右ペインの [対象] タブを選択し、各サーバからそのオブジェクトの割り当てを解除する（[選択済み] カラムから [使用可能] カラムにオブジェクトを移動する）。[適用] をクリックする。これで、対応するサーバで JDBC オブジェクト（接続プール、マルチプール、またはデータ ソース）が停止する。 3. 属性を修正するタブを選択する。 4. [対象] タブを選択し、オブジェクトをサーバに再び割り当てる。これで、対応するサーバで JDBC オブジェクト（接続プール、マルチプール、またはデータ ソース）が開始される。

コマンドライン インタフェースを使用した JDBC の管理

次の表では、コマンドライン インタフェースを使用した接続プールの管理について説明します。詳細情報が必要な場合は、目的のコマンドを選択してください。

接続プールのコマンドの使い方については、B-1 ページの「WebLogic Server コマンドライン インタフェース リファレンス」を参照してください。

表 16-22 コマンドライン インタフェースを使用した接続プールの管理

目的とする作業	使用するコマンド
接続プールの無効化	**DISABLE_POOL**
無効な接続プールの有効化	**ENABLE_POOL**
JDBC 接続プールの削除	**DESTROY_POOL**
接続プールが作成されたかどうかの確認	**EXISTS_POOL**
接続プールのリセット	**RESET_POOL**

17 JMS の管理

以下の節では、WebLogic Server の Java Message Service（JMS）を管理する方法について説明します。

- JMS と WebLogic Server
- JMS のコンフィグレーション
- JMS のモニタ
- JMS のチューニング
- WebLogic Server の障害からの回復

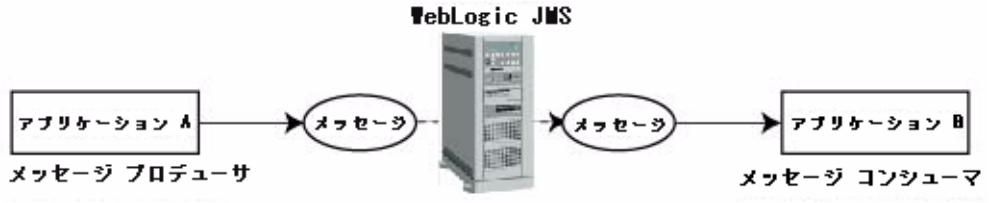
JMS と WebLogic Server

JMS は、エンタープライズ メッセージング システムにアクセスするための標準の API です。具体的な WebLogic JMS の機能は以下のとおりです。

- メッセージング システムを共有する Java アプリケーション同士でメッセージを交換できます。
- メッセージを作成、送信、および受信するための標準インタフェースによりアプリケーションの開発が容易になります。

次の図は、WebLogic JMS によるメッセージングの仕組みを示しています。

図 17-1　WebLogic Server JMS のメッセージング



図で示されているように、WebLogic JMS はプロデューサ アプリケーションからメッセージを受信し、受け取ったメッセージをコンシューマ アプリケーションに配信します。

JMS のコンフィグレーション

Administration Console を使用して、以下のコンフィグレーション属性を定義します。

- JMS を有効にします。
- JMS サーバを作成します。
- JMS サーバの値、接続ファクトリ、送り先（キューとトピック）、送り先テンプレート、（送り先キーを使用した）送り先のソート順指定、永続ストレージ、セッション プール、および接続コンシューマを作成またはカスタマイズします。
- カスタム JMS アプリケーションを設定します。
- しきい値と割当を定義します。
- サーバのクラスタ化、並行メッセージ処理、送り先のソート順指定、永続的なメッセージング、メッセージ ページングなど、必要な JMS 機能を有効にします。

WebLogic JMS では、一部のコンフィグレーション属性に対して、デフォルト値が用意されていますが、それ以外のすべての属性に対しては値を指定する必要があります。コンフィグレーション属性に対して無効な値を指定した場合や、デ

フォルト値が存在しない属性に対して値を指定しなかった場合は、再起動時に JMS が起動されません。製品には、JMS のサンプル コンフィグレーションが用意されています。

WebLogic Server アプリケーションを以前のリリースから移行する場合、コンフィグレーション情報は自動的に変換されます（『WebLogic JMS プログラマーズ ガイド』の「既存のアプリケーションの移行」を参照）。

WebLogic JMS の属性をコンフィグレーションするには、次の操作を行います。

1. Administration Console を起動します。
2. 左ペインの [サービス] の下の [JMS] ノードをクリックしてリストを展開します。
3. 以降の節、または Administration Console オンライン ヘルプで説明されている手順に従って、JMS オブジェクトを作成およびコンフィグレーションします。

WebLogic JMS をコンフィグレーションしたら、アプリケーションで JMS API を使用してメッセージの送受信ができるようになります。WebLogic JMS アプリケーションの開発の詳細については、『WebLogic JMS プログラマーズ ガイド』を参照してください。

注意： WebLogic JMS のコンフィグレーション プランを支援するために、『WebLogic JMS プログラマーズ ガイド』にはコンフィグレーション チェックリストがあります。このチェックリストを使用して、属性の要件や各種 JMS 機能をサポートするオプションを検討できます。

JMS サーバのコンフィグレーション

JMS サーバは、クライアントの代わりに接続およびメッセージ リクエストを管理するサーバです。

JMS サーバを作成するには、Administration Console の [JMS ｜サーバ] ノードを使用して、以下を定義します。

- 以下のコンフィグレーション属性。
  - JMS サーバの名前。

- 永続的なメッセージングに必要な永続ストレージ（ファイルまたは JDBC データベース）。JMS サーバに永続ストレージを割り当てない場合、そのサーバでは永続的なメッセージングはサポートされません。
- ページングに必要なページング ストア（ファイルを推奨）。JMS サーバにページング ストアを割り当てない場合、そのサーバでは永続的なメッセージングはサポートされません。
- 一時的なキューおよびトピックを含むすべての一時的な送り先を作成する場合に使用される一時的なテンプレート。
- メッセージおよびバイト数のしきい値と割当（最大数、最大しきい値と最小しきい値）。
- バイト ページングまたはメッセージ ページングを JMS サーバ上で有効にするかどうか。

■ JMS サーバと関連付けられている WebLogic Server インスタンスを対象とします。対象の WebLogic Server が起動すると、JMS サーバも起動します。対象の WebLogic Server が指定されていない場合、JMS サーバは起動しません。

注意： JMS サーバのデプロイメントは、接続ファクトリやテンプレートのデプロイメントとは異なります。JMS サーバは 1 つのサーバにデプロイされます。接続ファクトリやテンプレートは、複数のサーバで同時にインスタンス化されます。

JMS サーバを作成およびコンフィグレーションする手順については、Administration Console オンライン ヘルプの「JMS サーバ」を参照してください。

接続ファクトリのコンフィグレーション

接続ファクトリは、JMS クライアントが JMS 接続を作成することを可能にするオブジェクトです。接続ファクトリでは同時使用がサポートされており、複数のスレッドがオブジェクトに同時にアクセスできます。接続ファクトリを定義およびコンフィグレーションして、あらかじめ定義された属性で接続を作成します。WebLogic Server では、起動時に接続ファクトリが JNDI スペースに追加され、アプリケーションが WebLogic JNDI を使用して接続ファクトリを取り出します。

システム管理者は、複数の接続ファクトリをコンフィグレーションし、対象を使用してそれらを WebLogic サーバに割り当てることで、クラスタ内のあらゆるサーバから送り先へのクラスタワイドで透過的なアクセスを確立できます。各接続ファクトリは、複数の WebLogic サーバにデプロイできます。JMS クラスタ化の詳細については、http://edocs.beasys.co.jp/e-docs/wls61/jms/config.html の「WebLogic JMS のクラスタ化のコンフィグレーション」を参照してください。

接続ファクトリをコンフィグレーションするには、Administration Console の [接続ファクトリ] ノードを使用して、以下を定義します。

- 以下のコンフィグレーション属性。
  - 接続ファクトリの名前
  - JNDI ネームスペース内で接続ファクトリにアクセスする場合の名前
  - 恒久サブスクライバを持つクライアント用のクライアント ID（恒久サブスクライバの詳細については、『WebLogic JMS プログラマーズ ガイド』を参照）。
  - デフォルトのメッセージ配信属性（優先度、存続時間、配信時刻、およびモード）
  - 非同期セッション向けに存在する未処理のメッセージの最大数とオーバラン ポリシー（マルチキャスト セッションで最大数に達したときに実行されるアクション）
  - `close()` メソッドを `onMessage()` メソッドから呼び出せるかどうか
  - すべてのメッセージが確認応答されるのか、それとも受信したメッセージのみが確認応答されるのか
  - トランザクション属性（トランザクションのタイムアウト、Java Transaction API（JTA）ユーザ トランザクションが可能かどうか、およびトランザクション（XA）キューまたは XA トピック接続ファクトリが返されるのかどうか）
- 接続ファクトリに関連付けられ、クラスタ化をサポートする対象（WebLogic Server インスタンス）。対象を定義することにより、接続ファクトリがデプロイされる可能性のあるサーバ、グループ、およびクラスタのセットを限定できます。

WebLogic JMS では、デフォルトで次の 1 つの接続ファクトリが用意されています。`weblogic.jms.ConnectionFactory`. すべてのコンフィグレーション属性は、このデフォルトの接続ファクトリのデフォルト値に設定されています（Administration Console オンライン ヘルプの「JMS 接続ファクトリ」を参照）。

デフォルトの接続ファクトリの定義がアプリケーションに適用できる場合は、さらに接続ファクトリのコンフィグレーションを行う必要はありません。

注意： デフォルトの接続ファクトリを使用する場合は、接続ファクトリがデプロイされる可能性のある JMS サーバを限定することができません。特定の JMS サーバを対象にする場合は、新しい接続ファクトリを作成し、適切な JMS サーバの対象を指定してください。

接続ファクトリを作成およびコンフィグレーションする手順については、Administration Console オンライン ヘルプの「JMS 接続ファクトリ」を参照してください。

接続ファクトリの属性の中には、動的にコンフィグレーションできるものもあります。動的な属性が実行時に変更された場合、新しく設定された値は新規接続に対してのみ有効になります。既存の接続の動作には影響しません。

送り先のコンフィグレーション

送り先では、キューかトピックかが識別されます。JMS サーバを定義したら、その送り先をコンフィグレーションできます。JMS サーバごとに 1 つまたは複数の送り先をコンフィグレーションできます。

送り先は、明示的にコンフィグレーションすることも、送り先テンプレートを使用してコンフィグレーションすることもできます。送り先テンプレートを使用すると、似た属性設定を持つ複数の送り先を定義できます（17-7 ページの「JMS テンプレートのコンフィグレーション」を参照）。

送り先を明示的にコンフィグレーションするには、Administration Console の [送り先] ノードを使用して、以下のコンフィグレーション属性を定義します。

- 送り先の名前とタイプ（キューまたはトピック）
- JNDI ネームスペース内で送り先にアクセスする場合の名前
- 格納されている永続的メッセージに対するストアの有効化または無効化
- 送り先の作成に使用される JMS テンプレート
- 特定の送り先に対してソート順を定義する場合に使用されるキー
- メッセージおよびバイト数のしきい値と割当（最大数、最大しきい値と最小しきい値）

- バイト ページングまたはメッセージ ページングを送り先で有効にするかどうか
- オーバーライド可能なメッセージ属性（優先度、存続時間、配信時刻、および配信モード）
- 再配信遅延のオーバーライド、再配信制限、エラーの送り先などのメッセージ再配信属性
- マルチキャスト アドレス、存続時間（TTL）、ポートなどのマルチキャスト属性（トピックの場合のみ）

送り先を作成およびコンフィグレーションする手順については、Administration Console オンライン ヘルプの「JMS の送り先」を参照してください。

送り先の属性の中には、動的にコンフィグレーションできるものもあります。属性が実行時に変更された場合、変更は新しく配信されるメッセージにのみ適用され、格納されているメッセージには影響しません。

JMS テンプレートのコンフィグレーション

JMS テンプレートを使用することによって、似た属性設定を持つ複数の送り先を効率的に定義できます。JMS テンプレートには、以下のような利点があります。

- 新しい送り先を定義するたびにすべての属性設定を再入力する必要がありません（JMS テンプレートを使用しても、新しい値を割り当てる任意の設定をオーバーライドできます）。
- テンプレートを変更するだけで、共有される属性設定を動的に変更できます。

JMS テンプレートのコンフィグレーション属性を定義するには、Administration Console の [テンプレート] ノードを使用します。JMS テンプレートに対してコンフィグレーションできる属性は、送り先に対してコンフィグレーションされる属性と同じです。これらのコンフィグレーション属性は、それらを使用する送り先によって継承されます。ただし、以下の例外があります。

- JMS テンプレートを使用する送り先で属性のオーバーライド値が指定される場合は、そのオーバーライド値が使用されます。
- JMS テンプレートを使用する送り先で属性のメッセージ再配信値が指定される場合は、その再配信値が使用されます。
- [名前] 属性は、送り先によって継承されません。この名前は JMS テンプレートでのみ有効です。すべての送り先ではユニークな名前を明示的に定義しなければなりません。
- [JNDI 名]、[ストアを有効化]、[テンプレート] の各属性は、JMS テンプレートでは定義されません。
- [マルチキャスト] 属性は、トピックだけに適用されるので、JMS テンプレートでは定義されません。

送り先に対して明示的に定義されない属性には、デフォルト値が割り当てられます。デフォルト値が存在しない場合は、必ず、JMS テンプレートで値を指定するか、または送り先の属性のオーバーライド値として値を指定します。そうしないと、コンフィグレーション情報は不備な状態のままとなります。その場合、WebLogic JMS コンフィグレーションは失敗し、WebLogic JMS が起動しません。

JMS テンプレートを作成およびコンフィグレーションする手順については、Administration Console オンライン ヘルプの「JMS テンプレート」を参照してください。

送り先キーのコンフィグレーション

特定の送り先に対してソート順を定義するには、送り先キーを使用します。

送り先キーを作成するには、Administration Console の [送り先キー] ノードを使用して、以下のコンフィグレーション属性を定義します。

- 送り先キーの名前
- ソートするプロパティ名

- 予想されるキー タイプ
- ソートする方向（昇順または降順）

送り先キーを作成およびコンフィグレーションする手順については、Administration Console オンライン ヘルプの「JMS の送り先キー」を参照してください。

ストアのコンフィグレーション

永続ストレージは、永続的なメッセージングに使用されるファイルまたはデータベースで構成されます。

ファイル ストアまたはデータベース ストアを作成するには、Administration Console の [ストア] ノードを使用して、以下のコンフィグレーション属性を定義します。

- 永続ストレージの名前
- （ファイル ストアの場合）ファイル ストアが保存されるディレクトリ
- （JDBC データベース ストアの場合）複数のインスタンスと共に使用する JDBC 接続プールとデータベース テーブル名のプレフィックス

警告： JDBC データベース ストアでトランザクション（XA）接続プールをコンフィグレーションすることはできません。

注意： JMS 永続ストレージに格納されているメッセージ数が増加するにつれて、WebLogic Server の初期化に必要なメモリ量も増加します。WebLogic Server の再起動中にメモリ不足で初期化が失敗した場合は、Java 仮想マシン（JVM）のヒープ サイズを、現在 JMS 永続ストレージに格納されているメッセージ数に比例するよう増加させてから、再起動してください。

JDBC ストア

JMS では、JDBC を使用することで、指定された JDBC 接続プールからアクセスできるデータベースに永続的メッセージを格納できます。JMS データベースには、JDBC ドライバからアクセスできる任意のデータベースを指定できます。WebLogic Server では、以下のデータベースがサポートされており、それぞれに対応する JDBC ドライバが用意されています。

- Cloudscape
- Informix
- Microsoft SQL（MSSQL）Server
- Oracle
- Sybase

WebLogic Server によってサポートされているデータベース バージョンについては、「プラットフォーム サポート」ページ（http://edocs.beasys.co.jp/weblogic/docs/platforms/index.html）を参照してください。このページは頻繁に更新されており、さまざまなプラットフォームでの最新の動作確認情報が記載されています。

オプションとして、JDBC 接続プールに対してアクセス制御リスト（ACL）を制限することもできます。ACL を制限する場合は、WebLogic の *system* ユーザおよび JMS メッセージを送信するすべてのユーザが、このリストに含まれている必要があります。WebLogic Server セキュリティの管理の詳細については、14-1 ページの「セキュリティの管理」を参照してください。

注意： JMS サンプルは、Cloudscape Java データベースで動作するように設定されています。WebLogic Server には、Cloudscape の評価版が付属しており、*demoPool* データベースが用意されています。

JMS ストア

JMS データベースには、自動的に生成され、JMS 内部で使用されるシステム テーブルが 2 つあります。

- `<prefix>JMSStore`
- `<prefix>JMSState`

プレフィックス名は、この永続ストレージ内の JMS テーブルを識別します。ユニークなプレフィックスを指定すると、同一データベース内に複数のストアが存在できます。プレフィックスは、JDBC ストアをコンフィグレーションする際に Administration Console でコンフィグレーションします。プレフィックスは、DBMS で完全修飾名が必要な場合、または 2 つの WebLogic Server の JMS テーブルを区別する必要がある（1 つの DBMS で複数のテーブルを格納できるようにする）場合にテーブル名の前に付けられます。

警告： データに障害が発生するので、2 つの JMS ストアを同じデータベーステーブルで使用することはできません。

プレフィックスは、JMS テーブル名に付加されたときに有効なテーブル名になるように、次の形式で指定します。

```
[[[catalog.]schema.]prefix]JMSStore
```

catalog は DBMS が参照するシステム テーブルのセットを識別し、*schema* はテーブル オーナの ID に変換します。たとえば、JMS 管理者はプロダクション データベースで販売部門用の固有のテーブルを保持できます。

```
[[[Production.]JMSAdmin.]Sales]JMSStore
```

注意： Oracle などの一部の DBMS ベンダの場合、設定または選択するカタログがないので、このフォーマットは `[[schema.]prefix]` となります。詳細については、DBMS のマニュアルで完全修飾テーブル名の作成および使用方法を参照してください。

ストアを作成およびコンフィグレーションする手順については、Administration Console オンライン ヘルプの「JMS ファイル ストア」（ファイル ストアに関する情報）、および「JMS JDBC ストア」（JDBC データベース ストアに関する情報）をそれぞれ参照してください。

JMS ストア向けの JDBC 接続プールの推奨設定

DBMS を停止して再びオンラインに戻したときに JDBC ストアを使用していた場合、JMS では WebLogic Server を再起動するまでストアにアクセスできなくなります。この問題を回避するには、JMS ストアと関連付けられている JDBC 接続プールで以下の属性をコンフィグレーションします。

```
TestConnectionsOnReserve="true"
TestTableName="[[[catalog.]schema.]prefix]JMSState"
```

セッション プールのコンフィグレーション

サーバ セッション プールを使用すると、アプリケーションで複数のメッセージを並行して処理できます。JMS サーバを定義すると、各 JMS サーバに 1 つまたは複数のサーバ セッション プールをコンフィグレーションする選択ができます。

サーバ セッション プールをコンフィグレーションするには、Administration Console の [セッション プール] ノードを使用して、以下のコンフィグレーション属性を定義します。

- サーバ セッション プールの名前
- サーバ セッション プールが関連付けられ、セッションを作成する場合に使用される接続ファクトリ
- 並行して複数のメッセージを受信および処理する場合に使用されるメッセージ リスナ クラス
- トランザクション属性（確認応答モード、セッション プールでトランザクション セッションを作成するかどうか）
- 並行セッションの最大数

セッション プールを作成およびコンフィグレーションする手順については、Administration Console オンライン ヘルプの「JMS セッション プール」を参照してください。

セッション プールの属性の中には、動的にコンフィグレーションできるものもありますが、新しい値はセッション プールが再起動されるまで有効になりません。

接続コンシューマのコンフィグレーション

接続コンシューマは、サーバ セッションと取り出し、メッセージを処理するキュー（ポイント ツー ポイント）またはトピック（Pub/Sub）です。セッション プールを定義すると、各 JMS サーバに 1 つまたは複数の接続コンシューマをコンフィグレーションできます。

接続コンシューマをコンフィグレーションするには、Administration Console の [セッション プール] ノードを使用して、以下のコンフィグレーション属性を定義します。

- 接続コンシューマの名前
- 接続コンシューマによって蓄積されるメッセージの最大数
- メッセージをフィルタ処理する場合に使用される JMS セレクタ式。セレクタの定義の詳細については、『WebLogic JMS プログラマーズ ガイド』を参照してください。
- 接続コンシューマがリスン対象とする送り先

接続コンシューマを作成およびコンフィグレーションする場合に使用する、接続コンシューマの各コンフィグレーション属性の詳細については、Administration Console オンライン ヘルプの「JMS 接続コンシューマ」を参照してください。

JMS のモニタ

Administration Console を使用すると、JMS サーバ、接続、セッション、送り先、メッセージ プロデューサ、メッセージ コンシューマ、サーバ セッション プール、恒久サブスクライバといった JMS オブジェクトに関する統計をモニタできます。

サーバの実行中は、JMS 統計は増え続けます。統計は、サーバを再起動するときにのみリセットされます。

注意： WebLogic Server への JMS 接続のモニタについては、Administration Console オンライン ヘルプの「サーバ」を参照してください。

JMS オブジェクトのモニタ

JMS モニタ情報を表示するには、次の操作を行います。

1. Administration Console を起動します。
2. 左ペインの [サービス] の下にある [JMS] ノードをクリックし、JMS サービスのリストを展開します。
3. 左ペインの [JMS] の下にある [サーバ] ノードをクリックします。

   JMS サーバの情報が、右ペインに表示されます。
4. JMS サーバのリスト、または右ペインに表示されている JMS サーバから、モニタする JMS サーバを選択します。
5. [モニタ] タブを選択して、モニタ データを表示します。

モニタされている情報の詳細については、Administration Console オンライン ヘルプを参照してください。

恒久サブスクライバのモニタ

送り先トピックで動作している恒久サブスクライバを表示するには、次の操作を行います。

1. 17-14 ページの「JMS オブジェクトのモニタ」で説明されている手順 1 ～ 3 に従います。
2. 左ペインの [サーバ] の下にある [送り先] ノードをクリックし、JMS トピックおよびキューのリストを展開します。

   JMS 送り先情報が右ペインに表形式で表示されます。[Durable Subscribers] カラムには、表に示されている送り先トピックに対して実行されている恒久サブスクライバの数が表示されます。
3. 特定のトピックの恒久サブスクライバ情報を表示するには、目的のトピックの [Durable Subscribers] のアイコン（または実際の数）をクリックします。

モニタされている情報の詳細については、Administration Console オンライン ヘルプを参照してください。

JMS のチューニング

以降の節では、WebLogic Server JMS の管理者用パフォーマンス チューニング機能について説明します。

永続ストレージ

以降の節では、WebLogic Server JMS で永続ストレージを使用する場合のチューニング オプションについて説明します。

ファイル ストアへの同期書き込みの無効化

WebLogic Server JMS ファイル ストアでは、デフォルトで同期書き込みを使用することで最新のメッセージの整合性を保証します。通常、同期書き込みを無効にすると、ファイル ストアのパフォーマンスは大幅に向上します。その代わり、オペレーティング システムがクラッシュしたり、ハードウェアの障害が発生したりした場合には、メッセージがトランザクション対応であっても、送信したメッセージが失われたり、同じメッセージを重複して受信したりする可能性があります。オペレーティング システムでは通常のシャットダウン時に未処理の書き込みをすべてフラッシュするので、オペレーティング システムをシャット ダウンするだけではこうしたエラーは発生しません。こうしたエラーは、ビジー状態のサーバの電源を遮断することでエミュレートできます。

注意： 少なくとも 1 つの JMS ベンダでは同期書き込みをデフォルトで無効にしており、このベンダの場合のみ、受信に関して同期書き込みを無効にしたまま、送信に関して有効にすることができます。

WebLogic Server 上で実行されているすべての JMS ファイル ストアに対する同期書き込みを無効にするには、次のコマンドライン プロパティを設定します。

```
-Dweblogic.JMSFileStore.SynchronousWritesEnabled=false
```

JMS ファイル ストアに対する同期書き込みを無効にするには、次のように設定します。

```
-Dweblogic.JMSFileStore.store-name.SynchronousWritesEnabled=false
```

プロパティを両方とも設定すると、最初に設定したプロパティは後の設定でオーバーライドされます。同期書き込みが無効化された場合、ログ メッセージが生成されます。このメッセージを確認すると、コマンドライン プロパティが有効になっていることがわかります。

永続ストレージへのメッセージのページング

WebLogic Server 6.1 サービス パック 2 以上では、（容量が大きくなった場合に）仮想メモリ リソースを解放するために、メッセージを仮想メモリから永続ストレージにスワップ アウトします。パフォーマンスの点から見れば、メッセージをページングすると、今日のエンタープライズアプリケーションが必要とする大容量のメッセージ領域を使用するユーザには大きなメリットがあります。

メッセージのページングを開始および停止するタイミングの決定には、バイト ページングとメッセージ ページングという 2 つのメトリックが使用されます。各メトリックは、単一のページング モードの基準です。ページング モードは、指定したしきい値に達した場合に、JMS サーバまたはその送り先（トピックとキュー）に対して、個々に有効 / 無効を切り替えることも、同時に使用することもできます。

ページングのコンフィグレーション

Administration Console を使用して、新規または既存の JMS サーバまたはその送り先に対してページングをコンフィグレーションできます。[JMS ｜サーバ] ノードの属性を使用して、JMS サーバのページング ストアを指定したり、バイトまたはメッセージ ページングを有効にしたり、ページングを開始および停止するバイト / メッセージの最大および最小しきい値をコンフィグレーションしたりすることができます。

同様に、[送り先] ノードの属性を使用して、JMS サーバでコンフィグレーションされているすべてのトピックおよびキューのバイト / メッセージ ページングをコンフィグレーションできます。送り先は、JMS サーバ用にコンフィグレーションされているページング ストアを使用します。

また、JMS テンプレートを使用して複数の送り先をコンフィグレーションする場合、[テンプレート] ノードの属性を使用して、すべての送り先のページングをすばやくコンフィグレーションできます。特定の送り先に関してテンプレートのページング コンフィグレーションをオーバーライドする場合、どの送り先に対してもページングを有効または無効にできます。

新規の JMS サーバ、テンプレート、および送り先（トピックまたはキュー）のコンフィグレーション手順については、Administration Console オンライン ヘルプの「JMS サーバ」、「JMS 送り先」、および「JMS テンプレート」を参照してください。

注意： パフォーマンスをチューニングするために、ページングのしきい値をいつでも有効な値に変更できます。ただし、ページを有効にすると、バイトまたはメッセージしきい値を -1 にリセットしてページングを動的に無効にすることはできません。ページングの発生を防止するには、バイト / メッセージの最大しきい値を非常に大きな値（最大値は 2^{31} -1）に設定して、ページングが開始されないようにします。

JMS サーバのページング ストアのコンフィグレーション

注意： JDBC ページ ストアを使用することはできますが、お勧めしません。トラフィックが多いことと以後のパフォーマンスが低下することから、そうしたコンフィグレーションは望ましくないと言えます。

新しいページング ストアをコンフィグレーションするには、次の手順に従います。

1. Administration Console を起動します。
2. [JMS ｜ストア] ノードをクリックします。すべての JMS ストアが右ペインに表示されます。
3. [新しい JMSFile Store のコンフィグレーション] テキスト リンクをクリックします。新しいファイル ストアのコンフィグレーションに関連するタブが右ペインに表示されます。
4. 属性フィールドに値を入力します。
5. [作成] をクリックして、[名前] フィールドで指定した名前のファイル ストア インスタンスを作成します。新しいインスタンスが左ペインの [JMS ｜ストア] ノード下に追加されます。

6. ドメインに複数の JMS サーバがある場合、サーバ インスタンスごとに手順 3 ～ 5 を繰り返します。

JMS サーバのページングのコンフィグレーション

既存の JMS サーバでページングをコンフィグレーションして有効にするには、次の手順に従います。

1. [JMS ｜サーバ] ノードをクリックします。ドメインに定義されているすべてのサーバが右ペインに表示されます。
2. ページングをコンフィグレーションするサーバをクリックします。サーバのコンフィグレーションに関連するタブが右ペインに表示されます。
3. [一般] タブの [Paging Store] リスト ボックスで、ページングしたメッセージを格納するためのストアを選択します。ページング ストアのコンフィグレーション手順については、17-17 ページの「JMS サーバのページング ストアのコンフィグレーション」を参照してください。
4. [適用] をクリックして、ページング ストア属性値を保存します。
5. [しきい値と割当] タブをクリックします。
6. バイトまたはメッセージ ページング属性を有効にします。
   - [Bytes Paging Enabled] チェック ボックスを選択します。
   - [Messages Paging Enabled] チェック ボックスを選択します。
7. バイト ページングしきい値をコンフィグレーションします。
   - [最大バイトしきい値] フィールドで、バイト ページングを開始する基準値となる JMS サーバのバイト数を入力します。
   - [最小バイトしきい値] フィールドで、バイト ページングを停止する基準値となる JMS サーバのバイト数を入力します。
8. メッセージ ページングしきい値をコンフィグレーションします。
   - [最大メッセージしきい値] フィールドで、メッセージ ページングを開始する基準値となる JMS サーバのメッセージ数を入力します。
   - [最小メッセージしきい値] フィールドで、メッセージ ページングを停止する基準値となる JMS サーバのメッセージ数を入力します。
9. [適用] をクリックして、新しい属性値を保存します。

10. ドメインの JMS サーバのページングをさらにコンフィグレーションするには、手順 2 ～ 9 を繰り返します。

    注意： 各 JMS サーバは、それぞれ独自の永続ストレージを使用する必要があります。

11. JMS サーバのページングをコンフィグレーションしたら、次のいずれかの操作を行います。

    - JMS サーバの送り先のページングをコンフィグレーションしない場合、WebLogic Server を再起動してページングを有効にします。
    - サーバの送り先のページングをコンフィグレーションする場合、17-19 ページの「JMS テンプレートのページングのコンフィグレーション」または 17-20 ページの「送り先のページングのコンフィグレーション」の手順に従います。

JMS テンプレートのページングのコンフィグレーション

JMS テンプレートを使用することによって、似た属性設定を持つ複数の送り先（トピックまたはキュー）を効率的に定義できます。送り先用のテンプレートでページングをコンフィグレーションするには、次の手順に従います。

1. 左ペインの [JMS] ノードをクリックします。
2. [テンプレート] ノードをクリックします。ドメインに定義されているすべてのテンプレートが右ペインに表示されます。
3. ページングをコンフィグレーションするテンプレートをクリックします。テンプレートのコンフィグレーションに関連するタブが右ペインに表示されます。
4. [しきい値と割当] タブをクリックします。
5. バイトまたはメッセージ ページング属性を有効にします。
    - [Bytes Paging Enabled] チェック ボックスを選択します。
    - [Messages Paging Enabled] チェック ボックスを選択します。
6. バイト ページングしきい値をコンフィグレーションします。
    - [最大バイトしきい値] フィールドで、バイト ページングを開始する基準値となる送り先のバイト数を入力します。

- [最小バイトしきい値] フィールドで、バイト ページングを停止する基準値となる送り先のバイト数を入力します。

7. メッセージ ページングしきい値をコンフィグレーションします。
   - [最大メッセージしきい値] フィールドで、メッセージ ページングを開始する基準値となる送り先のメッセージ数を入力します。
   - [最小メッセージしきい値] フィールドで、メッセージ ページングを停止する基準値となる送り先のメッセージ数を入力します。
8. [適用] をクリックして、新しい属性値を保存します。
9. JMS テンプレートのページングをさらにコンフィグレーションするには、手順 3 ～ 8 を繰り返します。
10. ページングに関してすべての JMS テンプレートをコンフィグレーションしたら、WebLogic Server を再起動してページングを有効にします。

送り先のページングのコンフィグレーション

JMS テンプレートを使用しないで送り先のページングをコンフィグレーションする場合、以下の手順に従います。

1. [JMS ｜サーバ] をクリックして、ページングがコンフィグレーションされているサーバ インスタンスを展開します。
2. [送り先] ノードをクリックします。サーバのトピックおよびキューが右ペインにすべて表示されます。
3. ページングをコンフィグレーションするトピックまたはキューをクリックします。トピックまたはキューのコンフィグレーションに関連するタブが右ペインに表示されます。
4. [しきい値と割当] タブをクリックします。
5. バイトまたはメッセージ ページング属性を有効にします。
   - [Bytes Paging Enabled] リスト ボックスで [True] を選択します。
   - [Messages Paging Enabled] リスト ボックスで [True] を選択します。
6. バイト ページングしきい値をコンフィグレーションします。
   - [最大バイトしきい値] フィールドで、バイト ページングを開始する基準値となる送り先のバイト数を入力します。

- [最小バイトしきい値] フィールドで、バイト ページングを停止する基準値となる送り先のバイト数を入力します。

7. メッセージ ページングしきい値をコンフィグレーションします。
   - [最大メッセージしきい値] フィールドで、メッセージ ページングを開始する基準値となる送り先のメッセージ数を入力します。
   - [最小メッセージしきい値] フィールドで、メッセージ ページングを停止する基準値となる送り先のメッセージ数を入力します。
8. [適用] をクリックして、新しい属性値を保存します。
9. JMS 送り先のページングをさらにコンフィグレーションするには、手順 3 ～ 8 を繰り返します。
10. ページングに関してすべての送り先をコンフィグレーションしたら、WebLogic Server を再起動してページングを有効にします。

注意 : JMS テンプレートを使用して送り先をコンフィグレーションした場合、送り先のバイト / メッセージ ページングを明示的にコンフィグレーションすると、テンプレートのコンフィグレーションはオーバーライドされます。詳細については、17-21 ページの「JMS テンプレートのページングをオーバーライドする送り先のコンフィグレーション」および 17-2 ページの「JMS のコンフィグレーション」を参照してください。

JMS テンプレートのページングをオーバーライドする送り先のコンフィグレーション

テンプレートの設定をオーバーライドして特定の送り先のページングを有効または無効にする場合、次の手順に従います。

1. [JMS ｜サーバ] をクリックして、ページングがコンフィグレーションされているサーバ インスタンスを展開します。
2. [送り先] ノードをクリックします。サーバのトピックおよびキューが右ペインにすべて表示されます。
3. ページングをコンフィグレーションするトピックまたはキューをクリックします。サーバ インスタンスのトピックまたはキューが右ペインに表示されます。
4. [しきい値と割当] タブをクリックします。

5. JMS テンプレートをオーバーライドする方法に応じて、送り先の [Bytes Paging Enabled] または [Messages Paging Enabled] 属性をコンフィグレーションします。
   - 送り先のページングを無効にするには、[Bytes Paging Enabled] または [Messages Paging Enabled] リスト ボックスで [False] を選択します。
   - 送り先のページングを有効にするには、[Bytes Paging Enabled] または [Messages Paging Enabled] リスト ボックスで [True] を選択します。
6. [適用] をクリックして、新しい属性値を保存します。
7. 同じサーバ インスタンスの JMS 送り先のページングをさらにコンフィグレーションするには、手順 2 ～ 6 を繰り返します。
8. ページングに関してすべての送り先をコンフィグレーションしたら、WebLogic Server を再起動してページングを有効にします。

JMS のページング属性

以降の節では、WebLogic Server JMS で使用可能なページング属性について簡単に説明します。

JMS サーバのページング属性

表 17-1 では、JMS サーバでのページングをコンフィグレーションするときに定義するページング属性について説明します。JMS サーバの属性の詳細、および属性の有効な値とデフォルト値については、Administration Console オンラインヘルプの「ドメイン」を参照してください。

表 17-1 JMS サーバの属性

属性	説明
[Bytes Paging Enabled]	■ [Bytes Paging Enabled] チェック ボックスを選択しない場合（False）、サーバのバイト ページングは明示的に無効になる。 ■ [Bytes Paging Enabled] チェック ボックスを選択し（True）、ページング ストアがコンフィグレーションされており、[最小バイトしきい値] および [最大バイトしきい値] 属性が -1 より大きい場合、サーバのバイト ページングは有効になる。 ■ [最小バイトしきい値] または [最大バイトしきい値] 属性のいずれかが定義されていない場合、または -1 に設定されている場合、[Bytes Paging Enabled] が選択されていても（True）、サーバのバイト ページングは暗黙的に無効になる。

表 17-1 JMS サーバの属性

属性	説明
[Messages Paging Enabled]	■ [Messages Paging Enabled] チェック ボックスを選択しない場合（False）、サーバのメッセージ ページングは明示的に無効になる。 ■ [Messages Paging Enabled] チェック ボックスを選択し（True）、ページング ストアがコンフィグレーションされており、[最小メッセージしきい値] および [最大メッセージしきい値] 属性が ñ1 より大きい場合、サーバのメッセージ ページングは有効になる。 ■ [最小メッセージしきい値] または [最大メッセージしきい値] 属性のいずれかが定義されていない場合、または -1 に設定されている場合、[Messages Paging Enabled] が選択されていても（True）、サーバのメッセージ ページングは暗黙的に無効になる。
[Paging Store]	非永続メッセージをページングする永続ストレージの名前。ページング ストアは、永続メッセージまたは恒久サブスクライバ用と同じストアであってはならない。 2 つの JMS サーバは同じページング ストアを使用することができないので、サーバごとに固有のページング ストアをコンフィグレーションする必要がある。

JMS テンプレートのページング属性

表 17-3 では、JMS テンプレートで送り先のページングをコンフィグレーションするときに定義するページング属性について説明します。JMS テンプレートの属性の詳細、および属性の有効な値とデフォルト値については、Administration Console オンライン ヘルプの「JMS テンプレート」を参照してください。

表 17-2 JMS テンプレートの属性

属性	説明
[Bytes Paging Enabled]	■ [Bytes Paging Enabled] チェック ボックスを選択しない場合（Flase）、送り先レベルのバイト ページングは、送り先の設定でテンプレートをオーバーライドしない限り、JMS テンプレートの送り先に関して無効になる。 ■ [Bytes Paging Enabled] チェック ボックスを選択し（True）、JMS サーバのページング ストアがコンフィグレーションされており、[最小バイトしきい値] および [最大バイトしきい値] 属性が -1 より大きい場合、送り先レベルのバイト ページングは、送り先の設定でテンプレートをオーバーライドしない限り、JMS テンプレートの送り先に関して有効になる。 ■ JMS テンプレート Mbean に値が定義されてない場合、False がデフォルト値となるので、JMS テンプレートの送り先に関するバイト ページングは無効になる。

表 17-2 JMS テンプレートの属性

属性	説明
[Messages Paging Enabled]	■ [Messages Paging Enabled] チェック ボックスを選択しない場合（Flase）、送り先レベルのメッセージ ページングは、送り先の設定でテンプレートをオーバーライドしない限り、テンプレートの送り先に関して無効になる。 ■ [Messages Paging Enabled] チェック ボックスを選択し（True）、JMS サーバのページング ストアがコンフィグレーションされており、[最小バイトしきい値] および [最大バイトしきい値] 属性が -1 より大きい場合、送り先レベルのメッセージ ページングは、送り先の設定でテンプレートをオーバーライドしない限り、テンプレートの送り先に関して有効になる。 ■ JMS テンプレート Mbean に値が定義されてない場合、False がデフォルト値となるので、JMS テンプレートの送り先に関するメッセージ ページングは無効になる。

JMS 送り先のページング属性

表 17-3 では、送り先に関するページングをコンフィグレーションするときに定義するページング属性について説明します。JMS 送り先の属性の詳細、および属性の有効な値とデフォルト値については、Administration Console オンラインヘルプの「JMS の送り先」を参照してください。

表 17-3 JMS の送り先の属性

属性	説明
[Bytes Paging Enabled]	■ [Bytes Paging Enabled] を False に設定すると、送り先レベルのバイト ページングはその送り先に関して無効になる。 ■ [Bytes Paging Enabled] を True に設定し、JMS サーバのページング ストアがコンフィグレーションされており、[最小バイトしきい値] および [最大バイトしきい値] 属性が -1 より大きい場合、送り先レベルのバイト ページングはその送り先して有効になる。 ■ [Bytes Paging Enabled] をデフォルト設定にすると、この値はテンプレートの値を継承する（テンプレートが指定されている場合）。その送り先に関してテンプレートがコンフィグレーションされていない場合、デフォルト値は False となる。

表 17-3 JMS の送り先の属性

属性	説明
[Messages Paging Enabled]	■ [Messages Paging Enabled] を False に設定すると、送り先レベルのメッセージ ページングはその送り先に関して無効になる。 ■ [Messages Paging Enabled] を True に設定し、JMS サーバのページング ストアがコンフィグレーションされており、[最小バイトしきい値] および [最大バイトしきい値] 属性が -1 より大きい場合、送り先レベルのメッセージ ページングはその送り先して有効になる。 ■ [Messages Paging Enabled] をデフォルト設定にすると、この値はテンプレートの値を継承する（テンプレートが指定されている場合）。その送り先に関してテンプレートがコンフィグレーションされていない場合、デフォルト値は False となる。

注意： サーバのページングが有効で、送り先レベルのページングが指定した送り先に関して無効になっている場合、サーバのページングが開始されると、送り先のメッセージはページングされます。ただし、送り先レベルのページングが指定した送り先に関して無効になっている場合、送り先のメッセージ数がその送り先の最大しきい値を超えても、メッセージはページングされません。

ページングのしきい値属性

表 17-4 では、JMS サーバ、テンプレート、および送り先で使用可能なバイトおよびメッセージ ページングのしきい値について簡単に説明します。JMS サーバ、テンプレート、および送り先の属性の詳細と、属性の有効な値およびデフォルト値については、Administration Console オンライン ヘルプの「JMS サーバ」、「JMS テンプレート」、および「JMS の送り先」を参照してください。

表 17-4 ページングのしきい値属性

属性	説明
[最大バイトしきい値]	バイト数がこのしきい値を超えるとページングが開始される。
[最小バイトしきい値]	バイト数がこのしきい値を下回るとページングが停止される。
[最大メッセージしきい値]	メッセージ数がこのしきい値を超えるとページングが開始される。
[最小メッセージしきい値]	メッセージ数がこのしきい値を下回るとページングが停止される。

しきい値は、サーバ、テンプレート、および送り先に対して次のように定義します。

- 最大または最小バイトしきい値を定義しない場合（または ñ1 を定義した場合）、バイト数はページングを開始および停止するタイミングの決定に使用されません。
- 最大または最小メッセージしきい値を定義しない場合（または ñ1 を定義した場合）、メッセージ数はページングを開始および停止するタイミングの決定に使用されません。
- サーバまたはテンプレート / 送り先に関しては、ページングを有効にするために [Bytes Paging Enabled]/[Messages Paging Enabled] 属性を True に設定する必要があります。しきい値を設定し、ページングが有効になっていない場合、しきい値条件に達した時点でメッセージがサーバのログに記録されます。

WebLogic Server の障害からの回復

以降の節では、システムの障害発生時に WebLogic Server インスタンスを再起動または交換する方法と、そうした障害の後、JMS アプリケーションを正常に終了するためのプログラミングの考慮事項について説明します。

WebLogic Server の再起動または交換

WebLogic Server に障害が発生した場合、システムの回復方法には、以下の 3 種類があります。

- 障害が発生したサーバ インスタンスを再起動する
- 障害が発生したサーバ インスタンスと同じ IP アドレスを使用して新しいサーバを起動する
- 障害が発生したサーバ インスタンスとは異なる IP アドレスを使用して新しいサーバを起動する

障害が発生したサーバ インスタンスを再起動する、または障害が発生したサーバと同じ IP アドレスを使用して新しいサーバ インスタンスを起動する場合は、2-1 ページの「WebLogic Server の起動と停止」にある説明に従ってサーバを起動し、サーバ プロセスを開始します。

障害が発生したサーバとは異なる IP アドレスを使用して新しいサーバ インスタンスを起動するには、次の手順に従います。

1. サーバ エリアスが新しい IP アドレスを参照するように、ドメイン ネーム サービス（DNS）を更新します。
2. 2-1 ページの「WebLogic Server の起動と停止」の説明に従ってサーバを起動し、サーバ プロセスを開始します。
3. 必要に応じて、次の表のタスクを実行します。

JMS アプリケーションで使用している機能	実行するタスク
永続的なメッセージング—JDBC ストア	■ 障害が発生したサーバに JDBC データベース ストアが存在している場合は、データベースを新しいサーバに移行し、JDBC 接続プールの URL 属性が適切なロケーション参照を反映していることを確認する。 ■ 障害が発生したサーバに JDBC データベース ストアが存在していない場合は、データベースへのアクセスに影響はないので、変更は不要。
永続的なメッセージング — ファイル ストア	ファイルを新しいサーバに移行し、WebLogic Server ホーム ディレクトリ内のファイルのパス名が元のサーバにあったパス名と同じであることを確認する。
トランザクション	`<servername>*.tlog` という名前のすべてのファイルをコピーして、トランザクション ログを新しいサーバに移行する。このような移行は、一方のマシンに取り付け可能なデュアル ポート ディスクにトランザクション ログ ファイルを格納するか、または手動でファイルをコピーすることで実行できる。 ファイルが新しいサーバの異なるディレクトリにある場合は、サーバの [トランザクション ログファイルのプレフィックス] コンフィグレーション属性を更新してから新しいサーバを起動する。 **注意：** システムのクラッシュ後の移行では、サーバを新しい場所で再起動するときにトランザクション ログ ファイルが使用可能になっていることが特に重要である。そうしないと、クラッシュ時にコミット中だったトランザクションが適切に解決できず、その結果、アプリケーション データに矛盾が発生する場合がある。未確定のトランザクションはすべてロールバックされる。

注意： JMS 永続ストレージに格納されているメッセージ数が増加するにつれて、WebLogic Server の初期化に必要なメモリ量も増加します。WebLogic Server の再起動中にメモリ不足で初期化が失敗した場合は、Java 仮想マシン（JVM）のヒープ サイズを、現在 JMS 永続ストレージに格納されているメッセージ数に比例するよう増加させてから、再起動してください。

プログラミングの考慮事項

WebLogic Server の障害発生時に正常に終了するよう、JMS アプリケーションをプログラミングすることもできます。次に例を示します。

WebLogic Server の障害発生時の状態	対応
障害が発生した WebLogic Server インスタンスに接続していた。	`JMSException` が接続例外リスナに配信される。サーバを再起動または交換したらすぐに、アプリケーションを再起動する必要がある。
障害が発生した WebLogic Server インスタンスに接続していなかった。	サーバを再起動または交換したらすぐに、すべてを再確立する必要がある。
障害が発生した WebLogic Server インスタンスが JMS サーバの対象になっていた。	`ConsumerClosedException` がセッション例外リスナに配信される。失われたおそれがあるすべてのメッセージ コンシューマを再確立する必要がある。

18　JNDI の管理

以下の節では、JNDI を管理する方法について説明します。

- 18-1 ページの「JNDI 管理の概要」
- 18-2 ページの「JNDI ツリーの表示」
- 18-2 ページの「JNDI ツリーへのオブジェクトのロード」

JNDI 管理の概要

JNDI の管理には、Administration Console を使用します。JNDI API を使用すると、アプリケーションでデータ ソース、EJB、JMS、MailSession などを名前で検索できます。JNDI ツリーは、Administration Console の左ペインで表されます。

詳細については、http://edocs.beasys.co.jp/e-docs/wls61/jndi/index.html の『WebLogic JNDI プログラマーズ ガイド』を参照してください。

JNDI およびネーミング サービスの機能

JNDI は、LDAP（Lightweight Directory Access Protocol）や DNS（Domain Name System）など、既存のさまざまなネーミング サービスに対する共通インタフェースを提供します。これらのネーミング サービスは、バインディングのセットを管理します。名前はバインディングによってオブジェクトに関連付けられるので、オブジェクトを名前でルックアップできるようになります。したがって、JNDI を使用すると、分散アプリケーションのコンポーネントが互いを検索できます。

JNDI ツリーの表示

特定のサーバの WebLogic Server JNDI ツリーのオブジェクトを表示するには、次の手順に従います。

1. 左ペインのサーバ ノードを右クリックします。ポップアップ メニューが表示されます。
2. [JNDI ツリーを見る] を選択します。そのサーバの JNDI ツリーが、右ペインに表示されます。

JNDI ツリーへのオブジェクトのロード

Administration Console を使用して、WebLogic Server J2EE サービスおよびコンポーネント（RMI、JMS、EJB、JDBC データ ソースなど）を JNDI ツリーにロードします。

オブジェクトを JNDI ツリーにロードするには、オブジェクトを追加する JNDI ツリーの名前を選択します。オブジェクトを作成する場合は、[JNDI 名] 属性フィールドにオブジェクト名を入力します。オブジェクトがロードされると、JNDI はオブジェクトへのパスを提供します。

オブジェクトがロードされたかどうかを確認するには、「JNDI ツリーの表示」を参照してください。

オブジェクトのコンフィグレーションの詳細については、表 18-1「JNDI ツリーのオブジェクト」を参照してください。

表 18-1 JNDI ツリーのオブジェクト

サービス	バインドされたオブジェクト（オンライン ヘルプのリンク付き）
EJB	
JDBC データ ソース	JDBC データ ソースと JDBC トランザクション（Tx）データ ソース

表 18-1 JNDI ツリーのオブジェクト（続き）

サービス	バインドされたオブジェクト（オンライン ヘルプのリンク付き）
JMS 接続ファクトリ	JMS 接続ファクトリ
Web サービス	Web アプリケーション デプロイメント記述子エディタ
メール	メール セッション
デプロイメント記述子	BEA WebLogic J2EE コネクタ アーキテクチャ属性の説明

19 WebLogic J2EE コネクタ アーキテクチャの管理

Sun Microsystems J2EE コネクタ仕様バージョン 1.0 の最終草案バージョン 2 に基づく WebLogic J2EE コネクタ アーキテクチャは、J2EE プラットフォームと 1 つまたは複数の種類のエンタープライズ情報システム（EIS）を統合します。以下の節では、WebLogic J2EE コネクタ アーキテクチャを管理する方法について説明します。

- WebLogic J2EE コネクタ アーキテクチャの概要
- 新しいリソース アダプタのインストール
- 新しいコネクタのコンフィグレーションとデプロイメント
- モニタ
- コネクタの削除
- リソース アダプタのデプロイメント記述子の編集

BEA WebLogic J2EE コネクタ アーキテクチャの詳細については、『WebLogic J2EE コネクタ アーキテクチャ』を参照してください。

WebLogic J2EE コネクタ アーキテクチャの概要

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。その目的は、コンポーネント モデル、トランザクションやセキュリティのインフラストラクチャといった J2EE プラットフォームの機能を強化し、困難な EIS の統合を容易にすることです。

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。J2EE コネクタ アーキテクチャを使用すれば、EIS ベンダがアプリケーション サーバに合わせて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アプリケーション サーバ ベンダ（BEA WebLogic Server など）でも、アプリケーション サーバを拡張して新しい EIS への接続をサポートする場合にカスタム コードを追加する必要がありません。

J2EE コネクタ アーキテクチャを利用すると、EIS ベンダでは自社製 EIS 用の標準のリソース アダプタ（コネクタ）を提供できます。リソース アダプタは WebLogic Server などのアプリケーション サーバに接続され、EIS とアプリケーション サーバを統合するための基底のインフラストラクチャを提供します。

アプリケーション サーバ ベンダ（BEA WebLogic Server）は、J2EE コネクタ アーキテクチャをサポートし、複数の EIS との接続を保証するために 1 度だけそのシステムを拡張します。同様に、EIS ベンダは 1 つの標準リソース アダプタを提供し、そのアダプタは J2EE コネクタ アーキテクチャをサポートするどのアプリケーション サーバにでも接続できます。

新しいリソース アダプタのインストール

この節では、Administration Console を使用して新しいコネクタ（リソース アダプタ）を WebLogic Server に接続する方法を説明します。

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] を右クリックしてポップアップ メニューを表示します。
5. [新しい Connector Component をインストール] を選択します。
6. テキスト入力フィールドにリソース アダプタ `.rar` のパスを入力するか、[参照] ボタンをクリックしてファイル システムを参照し、インストールするリソース アダプタを選択します。
7. [Upload] ボタンをクリックしてリソース アダプタをインストールします。新しいリソース アダプタが左ペインの [コネクタ] ノード下に追加されます。

新しいコネクタのコンフィグレーションとデプロイメント

この節では、Administration Console を使用して新しいコネクタを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。

デプロイメント関連の詳細については、『WebLogic J2EE コネクタ アーキテクチャ』の「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。

リソース アダプタのコンフィグレーションとデプロイメント

WebLogic Server Administration Console を使用してコネクタをコンフィグレーションおよびデプロイするには、次の操作を行います。

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 アダプタの `.rar` ファイルの絶対パス、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を入力します。次に例を示します。
     `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。

デプロイされたリソース アダプタの表示

デプロイされたコネクタを Administration Console で表示するには、次の操作を行います。

1. Administration Console の左ペインで [デプロイメント] の下の [コネクタ] 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。

デプロイされたリソース アダプタのアンデプロイメント

WebLogic Server Administration Console を使用してデプロイされているコネクタをアンデプロイするには、次の操作を行います。

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 ボックスのチェックをはずします。
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。ただし、「デプロイされたリソース アダプタの更新」で説明されているように、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。

デプロイされたリソース 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。

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。

モニタ

コネクタのすべての接続中の接続プールをモニタするには、次の操作を行います。

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。

注意： この情報には、Administration Console の右ペインを使用してもアクセスできます。右ペインにあるコネクタのテーブルで、モニタする特定のコネクタを選択します。次に、[モニタ] タブを選択して、[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。

コネクタの削除

コネクタを削除するには、次の操作を行います。

1. Administration Console の左ペインで [デプロイメント | コネクタ | （コネクタ名）] を選択し、削除するコネクタを選択します。
2. 右ペインにあるコネクタのテーブルで、[削除] アイコンを選択します。

   右ペインに次のメッセージが表示されます。

   ドメインコンフィグレーションから < コネクタ名 > を本当に削除しますか？
3. [はい] をクリックしてコネクタを削除します。

リソース アダプタのデプロイメント記述子の編集

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次のリソース アダプタ（コネクタ）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 が稼働するコンピュータの名前、`port` は WebLogic Server がリスンするポートの番号です。
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。属性または下位要素を示すフォームが右ペインに表示されます。
   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。

20 WebLogic Server ライセンスの管理

WebLogic Server の実行には、有効なライセンスが必要です。以下の節では、WebLogic ライセンスのインストール方法と更新方法について説明します。

- WebLogic Server ライセンスのインストール
- ライセンスの更新

WebLogic Server ライセンスのインストール

WebLogic Server の評価版の有効期間は 30 日です。すぐに WebLogic Server の使用を開始できます。30 日間の評価期間を過ぎても WebLogic Server を使用する場合は、WebLogic Server を使用する IP アドレスごとに、評価期間の延長やライセンスの購入について販売担当者に問い合わせていただく必要があります。WebLogic Server の評価版では、ユニークな IP アドレスを持つクライアントが最大 3 つまでアクセスできる 1 つのサーバでの使用が許可されています。

BEA の Web サイトから WebLogic Server をダウンロードした場合は、配布キットに評価ライセンスが含まれています。WebLogic Server のインストール プログラムで、BEA ホーム ディレクトリの位置を指定できます。そのディレクトリに BEA ライセンス ファイル `license.bea` がインストールされます。

ライセンスの更新

以下のいずれかに該当する場合、BEA ライセンス ファイルを更新する必要があります。

- BEA ソフトウェアを追加購入した場合。
- 新製品を含む新しい配布キットを取得した場合。
- 30 日間の評価期間の延長を申し込み、その許可を受けた場合。

これらの場合のいずれかに該当するときには、ライセンス更新ファイルを電子メールの添付ファイルとして受け取る必要があります。BEA ライセンスを更新するには、次の手順に従います。

1. ライセンス更新ファイルを、`license.bea` 以外の名前で BEA ホーム ディレクトリに保存します。
2. java（Java 2）がパスに存在することを確認します。パスに JDK を追加するには、以下のいずれかのコマンドを入力します。
   - `set PATH=.\jdk130\bin;%PATH%`（Windows システム）
   - `set PATH=./jdk130/bin:$PATH`（UNIX システム）
3. コマンド シェルで、次のコマンドを BEA ホーム ディレクトリに入力します。

   `UpdateLicense` *`license_update_file`*

   `license_update_file` は、電子メールで受け取ったライセンス更新ファイルを保存したときの名前です。このコマンドを実行すると、`license.bea` ファイルが更新されます。
4. `license.bea` ファイルのコピーを WebLogic 配布キット以外の安全な場所に保存します。ライセンス ファイルを他人が使用することはできませんが、この情報を悪意あるまたは偶然による改ざんから保護された場所に保存する必要があります。

A WebLogic Java ユーティリティの使い方

WebLogic には、インストールおよびコンフィグレーション タスクを簡素化したり、サービスを提供したり、便利なショートカットを提供したりする Java プログラムが用意されています。以下の節では、WebLogic Server に用意されている各 Java ユーティリティについて説明します。ここでは、すべてのユーティリティのコマンドライン構文を示し、一部のユーティリティについては使用例を紹介します。

- AppletArchiver
- Conversion
- der2pem
- dbping
- deploy
- getProperty
- logToZip
- MulticastTest
- myip
- pem2der
- Schema
- showLicenses
- system
- t3dbping
- verboseToZip
- version
- writeLicense

これらのユーティリティを使用するには、CLASSPATH を正しく設定する必要があります。詳細については、「クラスパス オプションの設定」を参照してください。

AppletArchiver

AppletArchiver ユーティリティは、別のフレームにあるアプレットを実行し、ダウンロードされたクラスと、そのアプレットによって使用されたリソースの記録をすべて保持し、.jar ファイルまたは .cab ファイルにパッケージ化します（cabarc ユーティリティは、Microsoft から入手できます）。

構文

```
$ java utils.applet.archiver.AppletArchiver URL filename
```

引数	定義
URL	アプレットの URL
filename	.jar/.cab アーカイブの送り先であるローカル ファイル名

Conversion

以前のバージョンの WebLogic を使用していた場合は、`weblogic.properties` ファイルを変換する必要があります。変換スクリプトを使用してファイルを変換する手順については、Administration Console オンライン ヘルプの「変換」を参照してください。

der2pem

der2pem ユーティリティを使用すると、X509 証明書を DER 形式から PEM 形式に変換できます。.pem ファイルは、変換元の .der ファイルと同じディレクトリに書き込まれます。

構文

```
$ java utils.der2pem derFile [headerFile] [footerFile]
```

引数	説明
derFile	変換するファイルの名前。ファイル名は .der 拡張子で終わり、ファイルには .der 形式の有効な証明書が含まれている必要がある。
headerFile	PEM ファイルに配置されるヘッダ。デフォルトのヘッダは、"-----BEGIN CERTIFICATE-----"。 変換中の DER ファイルがプライベート キー ファイルの場合は、ヘッダ ファイルを使用する。以下のいずれかを含むヘッダ ファイルを作成する。 ■ "-----BEGIN RSA PRIVATE KEY-----"（暗号化されていないプライベート キーの場合） ■ "-----BEGIN ENCRYPTED PRIVATE KEY-----"（暗号化されているプライベート キーの場合） **注意：** ファイル内のヘッダ行の最後には、改行が必要になる。

引数	説明
footerFile	PEM ファイルに配置されるヘッダ。デフォルトのヘッダは、"-----END CERTIFICATE-----"。 変換中の DER ファイルがプライベート キー ファイルの場合は、フッタ ファイルを使用する。ヘッダに以下のいずれかを含むフッタ ファイルを作成する。 ■ "-----END RSA PRIVATE KEY-----"（暗号化されていないプライベート キーの場合） ■ "-----END ENCRYPTED PRIVATE KEY-----"（暗号化されているプライベート キーの場合） **注意：** ファイル内のヘッダ行の最後には、改行が必要になる。

例

```
$ java utils.der2pem graceland_org.der
Decoding
.....................................................................
```

dbping

dbping コマンドライン ユーティリティを使用すると、JDBC ドライバを使用した DBMS とクライアント マシンの間の接続をテストできます。このユーティリティを使用する前に、ドライバをインストールしておく必要があります。

構文

```
$ java -Dbea.home=WebLogicHome utils.dbping DBMS user password DB
```

引数	定義
WebLogicHome	WebLogic Server をインストールしたディレクトリ。たとえば d:\beaHome\wlserver6.1。BEA 提供の JDBC ドライバを使用する場合は必ず指定しなければならない。
DBMS	JDBC ドライバに合わせて以下のいずれかを選択する。 WebLogic jDriver for Microsoft SQL Server: MSSQLSERVER4 WebLogic jDriver for Oracle: ORACLE WebLogic jDriver for Informix: INFORMIX4 Oracle Thin Driver: ORACLE_THIN Sybase JConnect driver: JCONNECT
user	ログインに使用する有効なユーザ名。isql または sqlplus で使用する値と同じ値を使用する。
password	ユーザの有効なパスワード。isql または sqlplus で使用する値と同じ値を使用する。

引数	定義
DB	データベースの名前。使用する JDBC ドライバに応じて次の形式で指定する。 WebLogic jDriver for Microsoft SQL Server: *DBNAME*@*HOST*:*PORT* WebLogic jDriver for Oracle: *DBNAME* WebLogic jDriver for Informix: *DBNAME*@*HOST*:*PORT* Oracle Thin Driver: *HOST*:*PORT*:*DBNAME* Sybase JConnect driver: JCONNECT: *HOST*:*PORT*:*DBNAME* 各値の説明は次のとおり。 ■ *HOST* は、DBMS のホスト マシンの名前 ■ *PORT* は、DBMS が接続をリスンするデータベース ホストのポート ■ *DBNAME* は、DBMS のデータベースの名前

deploy

deploy ユーティリティは、アーカイブ（.jar、.war、または .ear）ファイルから J2EE アプリケーションを取得し、その J2EE アプリケーションを実行中の WebLogic Server にデプロイします。詳細については、『Web アプリケーションのアセンブルとコンフィグレーション』および『WebLogic Server アプリケーションの開発』を参照してください。

構文

```
$ java weblogic.deploy [options] [action] password {application
  name} {source}
```

アクション（以下のいずれかを選択）

アクション	説明
delete	アプリケーション名で指定されたアプリケーションを削除する。
deploy	J2EE アプリケーション（.jar、.war、.rar、または .ear）ファイルを、指定されたサーバにデプロイする。
list	指定された WebLogic Server 内のすべてのアプリケーションを一覧表示する。
undeploy	指定されたサーバから既存のアプリケーションを削除する。
update	アプリケーションを、指定されたサーバに再びデプロイする。

他の必須引数

引数	説明
password	その WebLogic Server 用のシステム パスワードを指定する。
application name	アプリケーションの名前を示す。このアプリケーション名は、デプロイメント時に deploy または console のいずれかのユーティリティを使って指定できる。
source	アプリケーション アーカイブ（.jar、.war、または .ear）ファイルの正確な場所、またはアプリケーション ディレクトリの最上位へのパスを指定する。

オプション

オプション	定義
-component *componentname:target1, target2*	さまざまな対象にデプロイされるコンポーネント。*componentname:target1,target2* のように指定する必要がある。 *componentname* は、拡張子なしの .jar、.rar、または .war ファイルの名前。このオプションは、コンポーネント（.jar、.rar、または .war）数に合わせて何回でも指定できる。 .ear ファイルをデプロイするには、このオプションを使用して各コンポーネントを個別に入力し、-source 引数を使用して .ear を指定する。たとえば、myDogApp.ear という .ear の jubilee.jar と wallance.war をデプロイするには、次のように入力する。 `weblogic.deploy -component jubilie:myserver -component wallance:myserver deploy gumby1234 appname myDogApp.ear` （このコマンドは 1 行で入力する。） コンポーネントが展開ディレクトリ形式の場合、アーカイブ ファイルの代わりにそれぞれのディレクトリ名を使用する。
-debug	デプロイメント処理中、詳細なデバッグ情報を stdout に出力する。
-help	deploy ユーティリティで使用できるすべてのオプションのリストを出力する。

オプション	定義
`-host host`	J2EE アプリケーション（`.jar`、`.war`、または `.ear`）ファイルをデプロイする場合に使用する WebLogic Server のホスト名を指定する。このオプションを指定しない場合、deploy ユーティリティでは、ホスト名 `localhost` を使用した接続が試行される。
`-jspRefreshComponentName`	更新されたファイルがコピーされる webapp コンポーネントを指定する。このオプションを `-jspRefreshFiles` オプションと一緒に使用すると静的なファイルを更新できる。このオプションの使い方については、「Web アプリケーションのデプロイメント」の「静的コンポーネント（JSP ファイル、HTML ファイル、画像ファイルなど）の更新」を参照。
`-jspRefreshFiles`	静的なファイル（JSP など）、HTML ファイル、画像ファイル（`.gif` や `.jpg` など）、およびテキスト ファイルを更新する。クラス ファイルは更新できない。クラス ファイルを更新するには、update フラグを使用してアプリケーションを再デプロイする。このオプションの使い方については、「Web アプリケーションのデプロイメント」の「静的コンポーネント（JSP ファイル、HTML ファイル、画像ファイルなど）の更新」を参照。

オプション	定義
-port *port*	J2EE アプリケーション（.jar、.war、または .ear）ファイルをデプロイする場合に使用する WebLogic Server のポート番号を指定する。 **注意：** -port オプションを指定しない場合、deploy ユーティリティでは、デフォルトのポート番号 7001 が接続に使用される。
-url *url*	Weblogic Server の URL を指定する。デフォルトは localhost:7001。
-username *username*	接続が行われるユーザの名前。デフォルトは system。
-version	deploy ユーティリティのバージョンを出力する。

例

deploy ユーティリティは、以下のようなさまざまな目的に役立ちます。

- デプロイされた J2EE アプリケーションの表示
- 新しい J2EE アプリケーションのデプロイ
- デプロイされた J2EE アプリケーションの削除
- デプロイされた J2EE アプリケーションの更新

デプロイされた J2EE アプリケーションの表示

ローカル WebLogic Server にデプロイされたアプリケーションを表示するには、次のコマンドを入力します。

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

ここで *password* は、WebLogic Server システム アカウント用のパスワードです。

リモート サーバにデプロイされたアプリケーションを一覧表示するには、次のように、port オプションと host オプションを指定します。

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

新しい J2EE アプリケーションのデプロイ

まだ WebLogic にデプロイされていない J2EE アプリケーション（.jar、.war、または .ear）ファイルまたはアプリケーション 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 application source
```

値は以下のとおりです。

- *application* は、このアプリケーションに割り当てる文字列
- *source* は、デプロイする J2EE アプリケーション ファイル（.jar、.war、.ear）のフルパス名か、アプリケーション ディレクトリのフルパス名

次に例を示します。

```
% java weblogic.deploy -port 7001 -host localhost deploy weblogicpwd Basic_example
   c:\mysamples\ejb\basic\BasicStatefulTraderBean.jar
```

注意： 管理サーバのアプリケーション ディレクトリにコピーする J2EE アプリケーション ファイル（.jar、.war、.ear）は、アプリケーションの名前を使ってリネームされます。このため、前述の例では、アプリケーション アーカイブ ディレクトリ　. .
.\config\mydomain\applications の名前は、
BasicStatefulTraderBean.jar から Basic_example.jar に変更されます。

デプロイされた J2EE アプリケーションの削除

デプロイされた J2EE アプリケーションを削除するには、割り当てられたアプリケーション名の参照のみが必要です。次の例のように入力します。

```
% java weblogic.deploy -port 7001 -host localhost undeploy
  weblogicpwd Basic_example
```

注意： J2EE アプリケーションを削除しても、アプリケーションは WebLogic Server から削除されません。deploy ユーティリティでそのアプリケーション名を再使用することはできません。次の項で説明するように、update 引数を使用してデプロイメントを更新する場合は、そのアプリケーション名を再使用できます。

デプロイされた J2EE アプリケーションの更新

J2EE アプリケーションを更新するには、次のように、update 引数を使用して、アクティブな J2EE アプリケーションの名前を指定します。

```
% java weblogic.deploy -port 7001 -host localhost update
   weblogicpwd Basic_example
   c:\updatesample\ejb\basic\BasicStatefulTraderBean.jar
```

1 つまたは複数のサーバにある特定のコンポーネントを更新する場合は、以下のコマンドを入力します。

```
% java weblogic.deploy -port 7001 -host localhost –component
   BasicStatefulTraderBean.jar:sampleserver,exampleserver update
   weblogicpwd Basic_example
   c:\updatesample\ejb\basic\BasicStatefulTraderBean.jar
```

getProperty

getProperty ユーティリティを使用すると、Java の設定およびシステムに関する詳細情報を表示できます。引数はありません。

構文

```
$ java utils.getProperty
```

例

```
$ java utils.getProperty
-- listing properties --
user.language=en
java.home=c:\java11\bin\..
awt.toolkit=sun.awt.windows.WToolkit
file.encoding.pkg=sun.io
java.version=1.1_Final
file.separator=\
line.separator=
user.region=US
file.encoding=8859_1
java.vendor=Sun Microsystems Inc.
user.timezone=PST
user.name=mary
os.arch=x86
os.name=Windows NT
java.vendor.url=http://www.sun.com/
user.dir=C:\weblogic
java.class.path=c:\weblogic\classes;c:\java\lib\cla...
java.class.version=45.3
os.version=4.0
path.separator=;
user.home=C:\
```

logToZip

logToZip ユーティリティは、HTTP サーバ ログ ファイルの内容（共通ログ形式）を検索し、その中でサーバによってロードされる Java クラスを検出してから、それらの Java クラスを含む非圧縮の .zip ファイルを作成します。このユーティリティは、HTTP サーバのドキュメント ルート ディレクトリから実行します。

このユーティリティを使用するには、HTTP サーバによって作成されたログ ファイルへのアクセスが必要です。

構文

```
$ java utils.logToZip logfile codebase zipfile
```

引数	定義
logfile	必須。ログ ファイルの完全修飾パス名。
codebase	必須。アプレットの CODEBASE、または CODEBASE がない場合は ""。CODEBASE をアプレットの完全パッケージ名と連結することで、HTTP ドキュメント ルートからアプレットへのフル パスを取得する。
zipfile	必須。作成する .zip ファイルの名前。.zip ファイルは、プログラムを実行しているディレクトリ内に作成される。入力されるファイル名のパスは、相対パスでも絶対パスでもよい。例では、相対パス名が使用されているので、.zip ファイルはカレント ディレクトリに作成される。

例

次の例に、ドキュメント ルート自体に存在するアプレット用の .zip ファイルの作成方法を示します（CODEBASE なしの例）。

```
$ cd /HTTP/Serv/docs
$ java utils.logToZip /HTTP/Serv/logs/access "" app2.zip
```

次の例に、ドキュメント ルートのサブディレクトリに存在するアプレット用の .zip ファイルの作成方法を示します。

```
C:\>cd \HTTP\Serv
C:\HTTP\Serv>java utils.logToZip \logs\applets\classes app3.zip
```

MulticastTest

MulticastTest ユーティリティは、WebLogic Cluster のコンフィグレーション時にマルチキャストに関する問題をデバッグする場合に便利です。このユーティリティは、マルチキャスト パケットを送信し、ネットワーク上で、マルチキャストがどのくらい効果的に機能してるかについての情報を返します。特に、MulticastTest は標準出力を通して以下のタイプの情報を表示します。

1. このサーバが送信する各メッセージの確認およびシーケンス ID
2. このサーバを含む、任意のクラスタ化されたサーバから受信した各メッセージのシーケンスと送信者 ID
3. メッセージを受信したがシーケンスがない場合は、シーケンス紛失警告
4. 予期されていたメッセージが受信されなかった場合は、メッセージ紛失警告

MulticastTest を使用するには、まず、マルチキャスト トラフィックのテストを行う各ノードにこのユーティリティをコピーします。

警告： 現在実行している WebLogic Cluster のアドレスと同じマルチキャスト アドレス（-a パラメータ）を指定して MulticastTest ユーティリティを実行しないでください。このユーティリティは、クラスタ化された WebLogic Server を起動する前に、マルチキャストが正しく機能することを確認することを目的にしています。

マルチキャストの設定に関する情報については、WebLogic Server ホストの特定のオペレーティング システムまたはハードウェアのコンフィグレーションに関するドキュメントを参照してください。クラスタの詳細については、『WebLogic Server Clusters ユーザーズ ガイド』を参照してください。

構文

```
$ java utils.MulticastTest -n name -a address [-p portnumber]
   [-t timeout] [-s send]
```

引数	定義
-n *name*	必須。シーケンスされたメッセージの送信者を示す名前。開始するテスト プロセスごとに、異なる名前を使用すること。

引数	定義
-a *address*	必須。シーケンスされたメッセージがブロードキャストされるマルチキャスト アドレス。または、クラスタ内のサーバが互いに通信するマルチキャスト アドレス（マルチキャスト アドレスが設定されていないクラスタのデフォルトは、237.0.0.1）。
-p *portnumber*	省略可能。クラスタ内のすべてのサーバが通信するマルチキャスト ポート（マルチキャスト ポートは、WebLogic Server に設定されたリスン ポートと同じである。設定されていない場合のデフォルトは、7001）。
-t *timeout*	省略可能。マルチキャスト メッセージが受け取れない場合のアイドル タイム アウト（秒単位）。この引数を設定しない場合、デフォルトは 600 秒（10 分）。タイムアウトを経過すると、タイムアウトの確認情報が stdout に出力される。
-s *send*	省略可能。送信間の時間間隔（秒単位）。この引数を設定しない場合、デフォルトは 2 秒。送信された各メッセージの確認情報が、stdout に出力される。

例

```
$ java utils.MulticastTest -N server100 -A 237.155.155.1
Set up to send and receive on Multicast on Address 237.155.155.1 on
port 7001
Will send a sequenced message under the name server100 every 2
seconds.
Received message 506 from server100
Received message 533 from server200
   I (server100) sent message num 507
Received message 507 from server100
Received message 534 from server200
   I (server100) sent message num 508
Received message 508 from server100
Received message 535 from server200
   I (server100) sent message num 509
Received message 509 from server100
Received message 536 from server200
   I (server100) sent message num 510
Received message 510 from server100
Received message 537 from server200
   I (server100) sent message num 511
Received message 511 from server100
Received message 538 from server200
```

```
   I (server100) sent message num 512
Received message 512 from server100
Received message 539 from server200
   I (server100) sent message num 513
Received message 513 from server100
```

myip

myip ユーティリティを使用すると、ホストの IP アドレスを取得できます。

構文

```
$ java utils.myip
```

例

```
$ java utils.myip
Host toyboat.toybox.com is assigned IP address: 192.0.0.1
```

pem2der

pem2der ユーティリティを使用すると、X509 証明書を PEM 形式から DER 形式に変換できます。.der ファイルは、変換元の .pem ファイルと同じディレクトリに書き込まれます。

構文

```
$ java utils.pem2der pemFile
```

引数	説明
pemFile	変換するファイルの名前。ファイル名は .pem 拡張子で終わり、ファイルには .pem 形式の有効な証明書が含まれている必要がある。

例

```
$ java utils.pem2der graceland_org.pem
Decoding
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
```

Schema

Schema ユーティリティを使用すると、WebLogic JDBC ドライバを使用してデータベースに SQL 文をアップロードできます。データベース接続の詳細については、『WebLogic JDBC プログラミング ガイド』を参照してください。

構文

```
$ java utils.Schema driverURL driverClass [-u username]
   [-p password] [-verbose SQLfile]
```

引数	定義
driverURL	必須。JDBC ドライバの URL。
driverClass	必須。JDBC ドライバ クラスのパス名。
-u *username*	省略可能。有効なユーザ名。
-p *password*	省略可能。ユーザの有効なパスワード。
-verbose	省略可能。SQL 文とデータベースのメッセージを出力する。
SQLfile	-verbose 引数を使用する場合は必須。SQL 文を記述したテキストファイル。

例

次のコードは、Schema コマンド ラインのサンプルです。

```
$ java utils.Schema "jdbc:cloudscape:demo;create=true"
   COM.cloudscape.core.JDBCDriver
   -verbose examples/utils/ddl/demo.ddl
```

次のコードは、.ddl ファイルのサンプルです。

```
    DROP TABLE ejbAccounts;
    CREATE TABLE ejbAccounts
      (id   varchar(15),
       bal  float,
       type varchar(15));
    DROP TABLE idGenerator;
    CREATE TABLE idGenerator
```

```
(tablename varchar(32),
 maxkey    int);
```

showLicenses

showLicenses ユーティリティを使用すると、このマシンにインストールされている BEA 製品に関するライセンス情報を表示できます。

構文

```
$ java utils.showLicenses
```

system

system ユーティリティを使用すると、コンピュータの操作環境に関する基本的な情報を表示できます。この情報には、JDK の製造メーカーとバージョン、CLASSPATH、オペレーティング システムに関する情報などがあります。

構文

```
$ java utils.system
```

例

```
$ java utils.system
* * * * * * * java.version * * * * * * *
1.1.6

* * * * * * * java.vendor * * * * * * *
Sun Microsystems Inc.

* * * * * * * java.class.path * * * * * * *
\java\lib\classes.zip;\weblogic\classes;
\weblogic\lib\weblogicaux.jar;\weblogic\license
...

* * * * * * * os.name * * * * * * *
Windows NT

* * * * * * * os.arch * * * * * * *
x86

* * * * * * * os.version * * * * * * *
4.0
```

t3dbping

t3dbping ユーティリティを使用すると、任意の 2 層 JDBC ドライバを使用した、DBMS への WebLogic JDBC 接続をテストできます。このユーティリティを使用するには、WebLogic Server と DBMS へのアクセスが必要です。

構文

```
$ java utils.t3dbping WebLogicURL username password DBMS
driverClass driverURL
```

引数	定義
WebLogicURL	必須。WebLogic Server の URL。
username	必須。有効な DBMS ユーザ名。
password	必須。有効な DBMS パスワード。
DBMS	必須。データベース名。
driverClass	必須。WebLogic 2 層ドライバの完全パッケージ名。
driverURL	必須。WebLogic 2 層ドライバの URL。

verboseToZip

verboseToZip ユーティリティは、HTTP サーバのドキュメント ルート ディレクトリから実行されると、verbose モードで実行されている Java アプリケーションから標準出力を取得し、参照されている Java クラスを検出してから、それらの Java クラスを含む非圧縮の .zip ファイルを作成します。

構文

```
$ java utils.verboseToZip inputFile zipFileToCreate
```

引数	定義
inputFile	必須。verbose モードで実行されているアプリケーションの出力が含まれる一時ファイル。
zipFileToCreate	必須。作成する .zip ファイルの名前。.zip ファイルは、プログラムを実行しているディレクトリ内に作成される。

UNIX の例

```
$ java -verbose myapplication > & classList.tmp
$ java utils.verboseToZip classList.tmp app2.zip
```

NT の例

```
$ java -verbose myapplication > classList.tmp
$ java utils.verboseToZip classList.tmp app3.zip
```

version

version ユーティリティは、インストールされている WebLogic に関する情報を stdout を介して表示します。

構文

```
$ java weblogic.Admin -url host:port -username username -password
password VERSION
```

例

```
$ java weblogic.Admin
-url localhost:7001 -username system -password foo VERSION
```

writeLicense

writeLicense ユーティリティを使用すると、WebLogic ライセンスすべてに関する情報を、カレント ディレクトリにある writeLicense.txt というファイルに書き込むことができます。このファイルは、たとえば WebLogic のテクニカル サポートなどへ電子メールで送信できます。

構文

```
$ java utils.writeLicense -nowrite -Dweblogic.system.home=path
```

引数	定義
-nowrite	必須。writeLicense.txt ではなく、stdout に出力を送る。
-Dweblogic.system.home	必須。WebLogic システム ホーム（インストールされている WebLogic のルート ディレクトリ）を設定する。 **注意：** この引数は、WebLogic システム ホームから writeLicense を実行しない場合に必要となる。

例

```
$ java utils.writeLicense -nowrite
```

UNIX の出力例

```
* * * * * * System properties * * * * * *

* * * * * * * java.version * * * * * * *
1.1.7

* * * * * * * java.vendor * * * * * * *
Sun Microsystems Inc.

* * * * * * * java.class.path * * * * * * *
c:\weblogic\classes;c:\weblogic\lib\weblogicaux.jar;
c:\java117\lib\classes.zip;c:\weblogic\license
...
```

Windows NT の出力例

```
* * * * * * * os.name * * * * * * *
Windows NT

* * * * * * * os.arch * * * * * * *
x86

* * * * * * * os.version * * * * * * *
4.0

* * * * * * IP * * * * * *
Host myserver is assigned IP address: 192.1.1.0

* * * * * * Location of WebLogic license files * * * * * *
No WebLogicLicense.class found

No license.bea license found in
weblogic.system.home or current directory

Found in the classpath: c:/weblogic/license/license.bea
Last Modified: 06/02/1999 at 12:32:12

* * * * * * Valid license keys * * * * * *
Contents:
Product Name    :WebLogic
 IP Address     : 192.1.1.0-255
 Expiration Date: never
 Units          : unlimited
 key            : b2fcf3a8b8d6839d4a252b1781513b9
...

* * * * * * All license keys * * * * * *
Contents:
Product Name    :WebLogic
 IP Address     : 192.1.1.0-255
 Expiration Date: never
 Units          : unlimited
 key            : b2fcf3a8b8d6839d4a252b1781513b9
...

* * * * * * WebLogic version * * * * * *
WebLogic Build: 4.0.x xx/xx/1999 10:34:35 #xxxxx
```

B WebLogic Server コマンドライン インタフェース リファレンス

以下の節では、WebLogic Server コマンドライン インタフェースの構文を示し、各 WebLogic Server の管理、接続プールの管理、および Mbean 管理コマンドについて説明します。

- B-1 ページの「コマンドライン インタフェースについて」
- B-2 ページの「WebLogic Server のコマンドの使い方」
- B-4 ページの「WebLogic Server 管理コマンドのリファレンス」
- B-18 ページの「WebLogic Server 接続プール管理コマンド リファレンス」
- B-28 ページの「Mbean 管理コマンド リファレンス」

コマンドライン インタフェースについて

Administration Console の代わりとして、WebLogic Server には、管理ツールやさまざまなコンフィグレーション MBean および実行時 MBean プロパティにアクセスするためのコマンドライン インタフェースが用意されています。

コマンドライン インタフェースは、以下の場合に使用します。

- 管理を効率的にするためのスクリプトを作成する場合
- ブラウザ経由で Administration Console にアクセスできない場合
- グラフィカル ユーザ インタフェースよりもコマンドライン インタフェースの方が使い慣れている場合

始める前に

この章で示す例は、以下のことを前提にしています。

- WebLogic Server が c:\weblogic ディレクトリにインストールされている
- JDK が c:\java ディレクトリにある
- WebLogic Server がインストール ディレクトリから起動されている

WebLogic Server コマンドを実行する前に、以下のことを行っておく必要があります。

1. 『WebLogic Server インストール ガイド』で説明されているとおりに、WebLogic Server ソフトウェアをインストールおよびコンフィグレーションします。
   http://edocs.beasys.co.jp/e-docs/wls61/install/index.html を参照してください。
2. CLASSPATH を正しく設定します。2-8 ページの「クラスパス オプションの設定」を参照してください。
3. 以下のいずれかの手順を実行して、コマンドライン インタフェースを有効にします。
   - サーバをインストール ディレクトリから起動します。
   - インストール ディレクトリからサーバを起動しない場合は、次のコマンドを入力し、c:\weblogic を WebLogic Server ソフトウェアがインストールされているディレクトリ名に変更します。

     ```
     -Dweblogic.system.home=c:\weblogic
     ```

WebLogic Server のコマンドの使い方

この節では、WebLogic Server のコマンドを使用するための構文と必須引数を示します。WebLogic Server コマンドでは、大文字と小文字は区別されません。

構文

```
java weblogic.Admin [-url URL] [-username username]
   [-password password] COMMAND arguments
```

引数

多くの WebLogic Server コマンドでは、以下の引数が必要となります。

引数	定義
URL	WebLogic Server ホストの URL（WebLogic Server がクライアントのリクエストをリスンする TCP ポートの番号を含む）。形式は、*hostname*:*port*。デフォルトは localhost:7001。 **注意：** 実行時およびコンフィグレーション Mbean コマンドで使用する URL が常に特定の管理サーバを参照するのに対して、サーバ コマンドで使用する URL は常に WebLogic Server を参照する。
username	省略可能。コマンドを実行できるように認証されているユーザ名。デフォルトは guest。
password	省略可能。コマンドを実行できるように認証されているパスワード。デフォルトは guest。

注意： 管理クライアントがサーバに接続できない場合、すべてのコマンドの終了コードは 1 です。

管理者は、実行時 Mbean を管理するコマンドを実行するための適切なアクセス制御パーミッションを持っている必要があります。

以下の節を参照してください。

- B-4 ページの「WebLogic Server 管理コマンドのリファレンス」
- B-18 ページの「WebLogic Server 接続プール管理コマンド リファレンス」
- B-28 ページの「Mbean 管理コマンド リファレンス」

WebLogic Server 管理コマンドのリファレンス

以降の節では、WebLogic Server 管理コマンドに関する情報を提供します。

表 B-1 は、WebLogic Server 管理コマンドの概要を示しています。以降の節では、コマンドの構文と引数を説明し、各コマンドの例を紹介します。

B-18 ページの「WebLogic Server 接続プール管理コマンド リファレンス」も参照してください。

Table B-1 WebLogic Server 管理コマンドの概要

タスク	コマンド	説明
WebLogic Server のシャットダウンの取り消し	CANCEL_SHUTDOWN	URL で指定された WebLogic Server の SHUTDOWN コマンドを取り消す。 B-6 ページの「CANCEL_SHUTDOWN」を参照。
WebLogic Server への接続	CONNECT	指定した数の接続を WebLogic Server に対して行い、各接続の合計時間と平均時間をミリ秒で示す。 B-7 ページの「CONNECT」を参照。
1 つまたは複数のコマンドのヘルプの表示	HELP	すべての WebLogic Server コマンドの構文と使用法に関する情報が返される（デフォルト）。HELP コマンド ラインで単一のコマンド値を指定した場合は、そのコマンドの情報が返される。 B-8 ページの「HELP」を参照。
WebLogic Server ライセンスの表示	LICENSES	特定のサーバにインストールされているすべての WebLogic Server インスタンスのライセンスを表示する。 B-9 ページの「LICENSES」を参照。
JNDI ネーミング ツリーのノードのバインドの表示	LIST	JNDI ネーミング ツリーのノードのバインドを示す。 B-10 ページの「LIST」を参照。

Table B-1 WebLogic Server 管理コマンドの概要 （続き）

タスク	コマンド	説明
WebLogic Server のロック	LOCK	特権を持たないログインに対して WebLogic Server をロックする。続けてログインが試行されると、オプションの文字列メッセージを含むセキュリティ例外が発生する。 B-11 ページの「LOCK」を参照。
WebLogic Server リスン ポートの検証	PING	WebLogic Server ポートでリスニングを行い、WebLogic クライアント リクエストを受け付ける準備ができていることを確認するためのメッセージを送信する。 B-12 ページの「PING」を参照。
サーバ ログ ファイルの表示	SERVERLOG	特定のサーバで生成されるログ ファイルを表示する。 B-13 ページの「SERVERLOG」を参照。
WebLogic Server のシャットダウン	SHUTDOWN	URL で指定した WebLogic Server をシャットダウンする。 B-14 ページの「SHUTDOWN」を参照。
スレッドの表示	THREAD_DUMP	実行中の WebLogic Server スレッドのスナップショットを提供する。 B-15 ページの「THREAD_DUMP」を参照。
WebLogic Server のロック解除	UNLOCK	LOCK 操作の後で WebLogic Server のロックを解除する。 B-16 ページの「UNLOCK」を参照。
WebLogic Server のバージョンの表示	VERSION	*URL* の値で指定したマシンで動作する WebLogic Server ソフトウェアのバージョンを示す。 B-17 ページの「VERSION」を参照。

注意： 管理クライアントがサーバに接続できない場合、すべてのコマンドの終了コードは 1 です。

CANCEL_SHUTDOWN

CANCEL_SHUTDOWN コマンドは、指定した WebLogic Server に対する SHUTDOWN コマンドを取り消します。

SHUT_DOWN コマンドを使用する場合、遅延時間（秒単位）を指定できます。管理者は、この遅延時間内にシャットダウンのコマンドを取り消すことができます。SHUTDOWN コマンドによってログインは無効になり、シャットダウンを取り消した後も無効のままになることに注意してください。ログインを再び有効にするには、UNLOCK コマンドを使用します。

B-14 ページの「SHUTDOWN」と B-16 ページの「UNLOCK」を参照してください。

構文

```
java weblogic.Admin [-url URL] [-username username]
   [-password password] CANCEL_SHUTDOWN
```

例

次の例では、ユーザ名が system、パスワードが gumby1234 のシステム ユーザが、localhost というマシンのポート 7001 でリスンする WebLogic Server のシャットダウンの取り消しを要求します。

```
java weblogic.Admin -url t3://localhost:7001 -username system
   -password gumby1234 CANCEL_SHUTDOWN
```

CONNECT

指定した数の接続を WebLogic Server に対して行い、各接続の合計時間と平均時間をミリ秒で示します。

構文

```
java weblogic.Admin [-url URL] [-username username]
   [-password password] CONNECT count
```

引数	定義
count	行われた接続の数。

例

次の例では、adminuser という名前と gumby1234 というパスワードを持つユーザが CONNECT コマンドを実行し、localhost というサーバに 25 回の接続を確立して、これらの接続に関する情報を取得します。

```
java weblogic.Admin -url localhost:7001 -username adminuser
-password gumby1234 CONNECT 25
```

HELP

すべての WebLogic Server コマンドの構文と使用法に関する情報が返されます(デフォルト)。HELP コマンド ラインで単一のコマンド値を指定した場合は、そのコマンドの情報が返されます。

構文

```
java weblogic.Admin HELP [COMMAND]
```

例

次の例では、PING コマンドの使い方に関する情報が要求されます。

```
java weblogic.Admin HELP PING
```

この場合、HELP コマンドは、以下の情報を stdout に返します。

```
Usage: weblogic.Admin [-url url] [-username username]
   [-password password] <COMMAND> <ARGUMENTS>

         PING <count> <bytes>
```

LICENSES

指定したサーバにインストールされたすべての WebLogic Server インスタンスのライセンスを示します。

構文

```
java weblogic.Admin [-url URL] [-username username]
[-password password] LICENSES
```

例

次の例では、デフォルトのユーザ名（guest）とパスワード（guest）を使用して、localhost というマシンのポート 7001 で動作する WebLogic Server のライセンス情報を要求します。

```
java weblogic.Admin -url localhost:7001 -username guest
-password guest LICENSES
```

LIST

JNDI ネーミング ツリーのノードのバインドを示します。

構文

```
java weblogic.Admin [-username username] [-password password]
   LIST context
```

引数	定義
context	必須。weblogic、weblogic.ejb、javax などのルックアップの JNDI コンテキスト。

例

この例では、ユーザ名 adminuser、パスワード gumby1234 のユーザが weblogic.ejb 内のノード バインドのリストを要求します。

```
java weblogic.Admin -username adminuser -password gumby1234
   LIST weblogic.ejb
```

LOCK

特権を持たないログインに対して WebLogic Server をロックします。続けてログインが試行されると、オプションの文字列メッセージを含むセキュリティ例外が発生します。

注意： これは、WebLogic Server 管理ユーザのパスワードを必要とする特権付きコマンドです。

構文

```
java weblogic.Admin [-url URL] [-username username]
   [-password password] LOCK "string_message"
```

引数	定義
"*string_message*"	省略可能。WebLogic Server がロックされているときに特権のないユーザがログインを試みると送出されるセキュリティ例外に付加するメッセージ。二重引用符で囲む。

例

次の例では、WebLogic Server がロックされます。

```
java weblogic.Admin -url localhost:7001 -username adminuser
   -password gumby1234
   LOCK "Sorry, WebLogic Server is temporarily out of service."
```

権限のないユーザ名とパスワードでログインするアプリケーションに対しては、「Sorry, WebLogic Server is temporarily out of service」というメッセージが表示されます。

PING

WebLogic Server ポートでリスニングを行い、WebLogic クライアント リクエストを受け付ける準備ができていることを確認するためのメッセージを送信します。

構文

```
java weblogic.Admin [-url URL] [-username username]
[-password password] PING [round_trips] [message_length]
```

引数	定義
round_trips	省略可能。ping の数。
message_length	省略可能。各 ping で送信されるパケットのサイズ。ping で送信されるパケットが 10 MB を超えると、例外が発生する。

例

次の例では、localhost というマシンのポート 7001 で動作する WebLogic Server を 10 回チェックします。

```
java weblogic.Admin -url localhost:7001 -username adminuser
-password gumby1234 PING 10
```

SERVERLOG

特定のサーバで生成されるサーバ ログ ファイルを表示します。

- URL を指定しない場合、管理サーバのサーバ ログがデフォルトによって表示されます。
- サーバ URL を指定した場合、管理サーバ以外のログを取得できます。
- 引数 `starttime` と `endtime` を省略すると、サーバ ログ全体の表示が開始されます。

構文

```
java.weblogic.Admin [-url URL] [-username username]
[-password password] SERVERLOG [[starttime]|[endtime]]
```

引数	定義
starttime	省略可能。どの時刻からメッセージを表示するかを指定する。指定しない場合、デフォルトによって SERVERLOG コマンドを実行したときにメッセージの表示が開始される。日付の書式は *yyyy/mm/dd*。時刻は 24 時間形式で示される。開始する日付と時刻は、次のように引用符の内側に入力する。"*yyyy/mm/dd hh:mm*"
endtime	省略可能。どの時刻までメッセージを表示するかを指定する。指定しない場合、SERVERLOG コマンドが実行された時間がデフォルトとなる。日付の書式は *yyyy/mm/dd*。時刻は 24 時間形式で示される。終了の日付と時刻は、次のように引用符の内側に入力する。"*yyyy/mm/dd hh:mm*"

例

次の例では、`localhost` というマシンのポート 7001 でリスンする WebLogic Server のログの表示が要求されます。

```
java weblogic.Admin -url localhost:7001
SERVERLOG "2001/12/01 14:00" "2001/12/01 16:00"
```

この要求では、ログの表示が 2001 年 12 月 1 日の午後 2 時に始まり、2001 年 12 月 1 日の午後 4 時に終わるよう指定されます。

SHUTDOWN

URL で指定した WebLogic Server をシャットダウンします。

構文

```
java weblogic.Admin [-url URL] [-username username]
   [-password password] SHUTDOWN [seconds] ["lockMessage"]
```

引数	定義
seconds	省略可能。このコマンドの実行時からサーバのシャットダウンまでの経過秒数。
"*lockMessage*"	省略可能。WebLogic Server がロックされているときにログインを試みると送出されるメッセージ。二重引用符で囲む。

例

次の例のコマンドでは、ユーザ名 adminuser、管理パスワード gumby1234 で、localhost というマシンのポート 7001 をリスンする WebLogic Server をシャットダウンします。

```
java weblogic.Admin -url localhost:7001 -username adminuser
-password gumby1234 SHUTDOWN 300 "Server localhost is shutting
down."
```

コマンドの発行後 5 分（300 秒）経過すると、指定したサーバがシャットダウンされ、次のメッセージが stdout に送られます。

```
   Server localhost is shutting down.
```

THREAD_DUMP

実行中の WebLogic Server スレッドのスナップショットを提供します。

構文

```
java weblogic.Admin [-url URL] [-username username]
[-password password] THREAD_DUMP
```

UNLOCK

LOCK 操作の後で WebLogic Server のロックを解除します。

構文

```
java weblogic.Admin [-url URL] [-username username]
   [-password password] UNLOCK
```

引数	定義
username	必須。このコマンドを使用するには、適切な管理ユーザ名を指定する必要がある。
password	必須。このコマンドを使用するには、適切な管理パスワードを指定する必要がある。

例

次の例では、ユーザ名が adminuser、パスワードが gumby1234 の管理者が、localhost というマシンのポート 7001 でリスンする WebLogic Server のロックの解除を要求します。

```
java weblogic.Admin -url localhost:7001 -username adminuser
   -password gumby1234 UNLOCK
```

VERSION

URL の値で指定したマシンで動作する WebLogic Server ソフトウェアのバージョンを示します。

構文

```
java weblogic.Admin -url URL -username username
   -password password VERSION
```

例

次の例では、あるユーザが localhost というマシンのポート 7001 で動作する WebLogic Server のバージョンを要求します。

```
java weblogic.Admin -url localhost:7001 -username guest
   -password guest VERSION
```

注意： この例では、引数 *username* と *password* の両方にデフォルト値の guest が使用されています。

WebLogic Server 接続プール管理コマンド リファレンス

表 B-2 は、接続プール用の WebLogic Server 管理コマンドの概要を示しています。以降の節では、コマンドの構文と引数を説明し、各コマンドの例を紹介します。

接続プールの詳細については、
http://edocs.beasys.co.jp/e-docs/wls61/jdbc/index.html の
『WebLogic JDBC プログラミング ガイド』および
http://edocs.beasys.co.jp/e-docs/wls61/adminguide/jdbc.html の『管理者ガイド』の「JDBC 接続の管理」を参照してください。

Table B-2 WebLogic Server 管理コマンドの概要 — 接続プール

タスク	コマンド	説明
動的接続プールの作成	CREATE_POOL	WebLogic Server の動作中に接続プールを作成できるようにする。動的に作成された接続プールは DataSources または TxDataSources では使用できない。 B-20 ページの「CREATE_POOL」を参照。
接続プールの破棄	DESTROY_POOL	接続はクローズされてプールから削除され、プールに残っている接続がなくなればプールは消滅する。接続プールを破棄できるのは、「system」ユーザか、またはそのプールに関連付けられている ACL によって「admin」パーミッションが与えられたユーザのみ。 B-23 ページの「DESTROY_POOL」を参照。
接続プールの無効化	DISABLE_POOL	接続プールを一時的に無効にして、クライアントがそのプールから接続を取得するのを防ぐことができる。接続プールを有効または無効にできるのは、「system」ユーザか、またはそのプールに関連付けられている ACL によって「admin」パーミッションが与えられたユーザのみ。 B-24 ページの「DISABLE_POOL」を参照。

Table B-2 WebLogic Server 管理コマンドの概要 — 接続プール

タスク	コマンド	説明
接続プールの有効化	ENABLE_POOL	無効にしたプールを再び有効にした場合、使用中だった各接続の JDBC 接続状態はその接続プールが無効にされたときと同じなので、クライアントはちょうど中断したところから JDBC 操作を続行できる。 B-25 ページの「ENABLE_POOL」を参照。
接続プールが存在するかどうかの確認	EXISTS_POOL	指定された名前の接続プールが WebLogic Server に存在するかどうかを調べる。このコマンドを使用すると、動的接続プールがすでに作成されているかどうかを調べ、作成する動的接続プールに固有の名前を付けることができる。 B-26 ページの「EXISTS_POOL」を参照。
接続プールのリセット	RESET_POOL	接続プール内に割り当てられている接続をすべてクローズしてから開き直す。これは、たとえば、DBMS が再起動されたあとに必要になることがある。接続プール内の 1 つの接続が失敗した場合は、プール内のすべての接続が不良であることがある。 B-27 ページの「RESET_POOL」を参照。

CREATE_POOL

WebLogic Server の動作中に接続プールを作成できるようにします。詳細については、http://edocs.beasys.co.jp/e-docs/wls61/jdbc/programming.html#programming004 の『WebLogic JDBC プログラミング ガイド』の「接続プールの動的作成」を参照してください。

構文

```
java weblogic.Admin [-url URL] [-username username]
   [-password password] CREATE_POOL poolName  aclName=aclX,
   props=myProps,initialCapacity=1,maxCapacity=1,
   capacityIncrement=1,allowShrinking=true,shrinkPeriodMins=15,
   driver=myDriver,url=myURL
```

引数	定義
poolName	必須。プールのユニークな名前。
aclName	必須。サーバの config ディレクトリにある fileRealm.properties 内の異なるアクセス リストを識別する。ペアになる名前は dynaPool でなければならない。
props	データベース接続プロパティ。通常は、「データベース ログイン名 ; データベース パスワード ; サーバ ネットワーク ID」の形式をとる。
initialCapacity	プール内の接続の初期数。このプロパティが定義済みで、0 より大きい正の数である場合、WebLogic Server は起動時にこの数の接続を作成する。デフォルトは 1。maxCapacity をより大きい値は指定できない。
maxCapacity	プールで許可される接続の最大数。デフォルトは 1。定義する場合、maxCapacity は =>1 でなければならない。
capacityIncrement	一度に追加できる接続の数。デフォルトは 1。
allowShrinking	接続が使用中でないことが検出されたときに、プールを縮小できるかどうかを指定する。デフォルトは true。

引数	定義
shrinkPeriodMins	必須。縮小の間隔。単位は分。最小値は 1。allowShrinking = True の場合、デフォルトは 15 分。
driver	必須。JDBC ドライバの名前。ローカル（非 XA）ドライバのみ参加できる。
url	必須。JDBC ドライバの URL。
testConnsOnReserve	予約される接続をテストすること示す。デフォルトは False。
testConnsOnRelease	解放されるときに接続をテストすることを示す。デフォルトは False。
testTableName	接続をテストするときに使用されるデータベース名。テストを正常に行うには、指定されている必要がある。testConnOnReserve または testConOnRelease を定義する場合は必須。
refreshPeriod	接続の更新間隔を設定する。未使用の接続がすべて TestTableName を使用してテストされる。テストに合格しない接続は閉じられ、有効な物理データベース接続を再確立する中で再び開かれる。TestTableName が設定されていない場合、テストは実行されない。
loginDelaySecs	各物理データベース接続を作成する前の遅延の秒数。この遅延は、プールの初期作成時とプールの有効期間の両方で物理データベース接続が作成されるときに必ず発生する。データベース サーバによっては、複数の接続リクエストが短い間隔で繰り返されると処理できないものもある。このプロパティを使用すると、データベース サーバの処理が追いつくように、少しの間隔をあけることができる。この遅延は、プールの初期作成時とプールの有効期間の両方で物理データベース接続が作成されるときに必ず発生する。

例

次の例では、名前が adminuser、パスワードが gumby1234 のユーザが、CREATE_POOL コマンドを実行して動的接続プールを作成します。

```
java weblogic.Admin -url localhost:7001 -username adminuser
   -password gumby1234 CREATE_POOL myPool
```

```
java weblogic.Admin -url t3://forest:7901 -username system
   -password gumby1234 CREATE_POOL dynapool6 "aclName=someAcl,
   allowShrinking=true,shrinkPeriodMins=10,
   url=jdbc:weblogic:oracle,driver=weblogic.jdbc.oci.Driver,
    initialCapacity=2,maxCapacity=8,
    props=user=SCOTT;password=tiger;server=bay816"
```

DESTROY_POOL

接続はクローズされてプールから削除され、プールに残っている接続がなくなればプールは消滅します。接続プールを破棄できるのは、「system」ユーザか、またはそのプールに関連付けられている ACL によって「admin」パーミッションが与えられたユーザだけです。

構文

```
java weblogic.Admin [-url URL] [-username username]
   [-password password] DESTROY_POOL poolName [true|false]
```

引数	定義
poolName	必須。プールのユニークな名前。
false (ソフト シャットダウン)	ソフト シャットダウンは、接続がプールに返されるのを待って、それらの接続をクローズする。
true (デフォルト — ハード シャットダウン)	ハード シャットダウンはすべての接続を即座に破棄する。プールから接続を利用している場合は、ハード シャットダウンの後に接続を使用しようとすると例外が生成される。

例

次の例では、名前が adminuser、パスワードが gumby1234 のユーザが、DESTROY_POOL コマンドを実行してアクティブなプール接続を一時的に凍結します。

```
java weblogic.Admin -url localhost:7001 -username adminuser
   -password gumby1234 DESTROY_POOL myPool false
```

DISABLE_POOL

接続プールを一時的に無効にして、クライアントがそのプールから接続を取得するのを防ぐことができます。接続プールを有効または無効にできるのは、「system」ユーザか、またはそのプールに関連付けられている ACL によって「admin」パーミッションが与えられたユーザだけです。

プールを無効化する方法には、後で有効化できるようにプール内の接続を凍結する方法と、接続を破棄する方法があります。

構文

```
java weblogic.Admin [-url URL] [-username username]
   [-password password] DISABLE_POOL poolName [true|false]
```

引数	定義
poolName	接続プールの名前。
false (無効化して**サスペンド**)	接続プールを無効化し、接続を使用しているクライアントをサスペンドする。データベース サーバと通信しようとすると、例外が送出される。ただし、クライアントは接続プールが無効になっている間に自分の接続をクローズできる。その場合、接続はプールに返され、プールが有効になるまでは別のクライアントから予約することはできない。
true (デフォルト — 無効化して**破棄**)	接続プールを無効化して、そのプールへのクライアントの JDBC 接続を破棄する。その接続で行われるトランザクションはすべてロールバックされ、その接続が接続プールに返される。

例

次の例では、名前が adminuser、パスワードが gumby1234 のユーザが、DISABLE_POOL コマンドを実行して、後で有効化する接続を凍結します。

```
java weblogic.Admin -url localhost:7001 -username adminuser
   -password gumby1234 DISABLE_POOL myPool false
```

ENABLE_POOL

プールを有効にした場合、使用中だった各接続の JDBC 接続状態はその接続プールが無効にされたときと同じなので、クライアントはちょうど中断したところから JDBC 操作を続行できます。

構文

```
java weblogic.Admin [-url URL] [-username username]
   [-password password] ENABLE_POOL poolName
```

引数	定義
poolName	接続プールの名前。

例

次の例では、名前が adminuser、パスワードが gumby1234 のユーザが、ENABLE_POOL コマンドを実行して、無効化（凍結）されている接続を再確立します。

```
java weblogic.Admin -url localhost:7001 -username adminuser
   -password gumby1234 ENABLE_POOL myPool
```

EXISTS_POOL

指定された名前の接続プールが WebLogic Server に存在するかどうかを調べます。このメソッドを使用すると、動的接続プールがすでに作成されているかどうかを調べ、作成する動的接続プールに固有の名前を付けることができます。

構文

```
java weblogic.Admin [-url URL] [-username username]
   [-password password] EXISTS_POOL poolName
```

引数	定義
poolName	接続プールの名前。

例

次の例では、名前が adminuser、パスワードが gumby1234 のユーザが、EXISTS_POOL コマンドを実行して、指定した名前のプールが存在するかどうかを確認します。

```
java weblogic.Admin -url localhost:7001 -username adminuser
   -password gumby1234 EXISTS_POOL myPool
```

RESET_POOL

このコマンドでは、登録されている接続プールの接続がリセットされます。

これは特権付きのコマンドです。このコマンドを使用するには、WebLogic Server 管理ユーザのパスワードを提示する必要があります。接続プールの名前（config.xml ファイルのエントリ）を知っていなければなりません。

構文

```
java weblogic.Admin URL RESET_POOL poolName system password
```

引数	定義
URL	WebLogic Server ホストの URL と、WebLogic がクライアントの要求をリスンする TCP ポートのポート番号（t3://host:port）。
poolName	接続プールの名前（WebLogic Server の config.xml ファイルに登録されている名前）。
password	ユーザ「system」の管理パスワード。この管理コマンドを使用するには、ユーザ名「system」と管理パスワードを提示しなければならない。

例

このコマンドでは、ホスト xyz.com のポート 7001 でリスンしている WebLogic Server の「eng」として登録されている接続プールが更新されます。

```
java weblogic.Admin t3://xyz.com:7001 RESET_POOL eng system gumby
```

Mbean 管理コマンド リファレンス

表 B-3 は、Mbean 管理コマンドの概要を示しています。以降の節では、コマンドの構文と引数を説明し、各コマンドの例を紹介します。

Table B-3 Mbean 管理コマンドの概要

タスク	コマンド	説明
コンフィグレーション Mbean の作成	CREATE	コンフィグレーション Mbean のインスタンスを作成する。成功した場合は、OK を stdout に返す。このコマンドは実行時 Mbean では使用できない。 B-29 ページの「CREATE」を参照。
コンフィグレーション Mbean の削除	DELETE	コンフィグレーション Mbean を削除する。成功した場合は、OK を stdout に返す。このコマンドは実行時 Mbean では使用できない。 B-31 ページの「DELETE」を参照。
実行時 Mbean の属性の表示	GET	実行時 Mbean の属性を表示する。 B-32 ページの「GET」を参照。
実行時 Mbean の呼び出し	INVOKE	属性を取得または設定するようには設計されていないメソッドを呼び出す。このコマンドでは実行時 Mbean のみを呼び出すことができる。 B-34 ページの「INVOKE」を参照。
実行時メトリックと統計の表示	INVOKE GET	INVOKE コマンドおよび GET コマンドを実行すると、実行時のメトリックと統計を表示できる。これらのコマンドでは実行時 Mbean のみを呼び出すことができる。 B-34 ページの「INVOKE」および B-32 ページの「GET」を参照。
コンフィグレーション Mbean の属性の設定	SET	指定したコンフィグレーション Mbean の指定した属性値を設定する。成功した場合は、OK を stdout に返す。このコマンドは実行時 Mbean では使用できない。 B-35 ページの「SET」を参照。

CREATE

コンフィグレーション Mbean のインスタンスを作成します。成功した場合は、OK を stdout に返します。このコマンドは実行時 Mbean では使用できません。Mbean インスタンスは、変更が行われた場所によって config.xml ファイルかセキュリティ レルムに保存されます。

注意： Mbean を作成すると、コンフィグレーション オブジェクトも作成されます。

Mbean 作成の詳細については、
http://edocs.beasys.co.jp/e-docs/wls61/programming/index.html の『WebLogic Server アプリケーションの開発』を参照してください。

構文

```
java weblogic.Admin [-url URL] [-username username]
[-password password] CREATE -name name -type mbean_type
[-domain domain_name]

java weblogic.Admin [-url URL] [-username username]
[-password password] CREATE -mbean mbean_name
```

引数	定義
name	必須。作成している Mbean を呼び出すときの名前を指定する。
mbean_type	必須。同じタイプの複数のオブジェクトに対するプロパティを作成するときに使用する。
mbean_name	必須。Mbean の完全修飾名を次の形式で指定する。 "*domain*:Type=*type*,Name=*name*" Type はオブジェクト グループのタイプ、Name は Mbean 名を示す。
domain_name	省略可能。ドメインの名前（mydomain など）を指定する。*domain_name* を指定しない場合、デフォルトのドメイン名が使用される。

例

```
java weblogic.Admin -url localhost:7001 -username adminuser
   -password gumby1234 CREATE -mbean
   "mydomain:Type=Server,Name=acctServer"
```

DELETE

コンフィグレーション Mbean を削除します。成功した場合は、OK を stdout に返します。このコマンドは実行時 Mbean では使用できません。

注意： Mbean を削除すると、コンフィグレーション オブジェクトも削除されます。

Mbean 削除の詳細については、
http://edocs.beasys.co.jp/e-docs/wls61/programming/index.html の『WebLogic Server アプリケーションの開発』を参照してください。

構文

```
java weblogic.Admin [-url URL] [-username username] [-password
password] DELETE {-type mbean_type|-mbean mbean_name}
```

引数	定義
mbean_type	必須。同じタイプの複数のオブジェクトに対する属性を削除するときに使用する。
mbean_name	必須。Mbean の完全修飾名を次の形式で指定する。 "*domain*:Type=*type*,Name=*name*" Type はオブジェクト グループのタイプ、Name は Mbean 名を示す。

例

```
java weblogic.Admin -url localhost:7001 -username adminuser
   -password gumby1234 DELETE -mbean
   "mydomain:Type:Server,Name=AcctServer"
```

GET

実行時 Mbean の属性を表示します。同じタイプの複数のオブジェクトに対する属性のリストを要求するには、次のように属性を要求します。

- 同じ Mbean タイプのすべての Mbean

```
GET {-pretty} -type mbean_type
```

- 特定の Mbean

```
GET {-pretty} -mbean mbean_name
```

指定した Mbean の名前が出力されます。-pretty を指定すると、各属性の名前と値が 1 組ずつ改行されて表示されます。

GET コマンドでは実行時 Mbean のみを呼び出すことができます。

各属性の名前と値のペアは、中括弧で囲んで指定します。この形式では、出力の解析を簡単にすることで、スクリプトの作成を容易にしています。

Mbean の名前は次のように出力されます。

```
{mbeanname mbean_name {property1 value} {property2 value} .. .}
{mbeanname mbean_name {property1 value} {property2 value} . . .}
. . .
```

-pretty を指定すると、各属性の名前と値が 1 組ずつ改行されて表示されます。指定した Mbean のそれぞれの名前も次のように出力されます。

```
mbeanname:mbean_name
property1: value
property2: value
.
.
.
mbeanname:mbean_name
property1: value
property2:value
```

構文

```
java weblogic.Admin [-url URL] [-username username] [-password
   password] GET {-pretty} {-type mbean_type|-mbean mbean_name}
   [-property property1] [-property property2]...
```

引数	定義
mbean_type	必須。同じタイプの複数のオブジェクトに対する属性を取得するときに使用する。Mbean 名が出力される。
mbean_name	Mbean の完全修飾名を次の形式で指定する。 *"domain*:Type=*type*,Location=*location*,Name=*name*" Type はオブジェクト グループのタイプ、Location は Mbean の位置、Name は Mbean 名を示す。
pretty	省略可能。適切にフォーマットされた出力を作成する。
property	省略可能。一覧表示される Mbean 属性の名前。 **注意：** 属性にこの引数を指定しなかった場合、すべての属性が表示される。

例

次の例では、あるユーザがポート 7001 でリスンする localhost というサーバの Mbean 属性の表示を要求します。

```
java weblogic.Admin -url localhost:7001 GET -pretty -type Server
```

INVOKE

指定した Mbean で、指定したメソッド（引数も含む）を呼び出します。このコマンドでは実行時 Mbean のみを呼び出すことができます。このコマンドは、Mbean 属性を取得または設定しないメソッドを呼び出すために使用します。

構文

```
java weblogic.Admin [-url URL] [-username username] [-password
   password] INVOKE {-type mbean_type|-mbean mbean_name} -method
methodname [argument . . .]
```

引数	定義
mbean_type	同じタイプの複数のオブジェクトに対する属性を取得するときに必要となる。次のように、Mbean 名の完全修飾名を指定する必要がある。 "domain:Name:name,Type=*type*,Application=*application*"
mbean_name	必須。Mbean の完全修飾名を次の形式で指定する。 "domain:Type=*type*,Location=*location*,Name=*name*" 各値の説明は次のとおり。 ■ Type ではオブジェクト グループのタイプを指定する。 ■ Location では Mbean の位置を指定する。 ■ Name は Mbean 名。 引数が文字列の配列の場合、その引数は以下の形式で渡されなければならない。 "*String1;String2;. . .* "
methodname	必須。呼び出すメソッドの名前を指定する。メソッド名の後に、次のようにメソッド呼び出しに渡す引数を指定できる。 "*domain*:Name=*name*,Type=*type*"

例

次の例では、getAttributeStringValue メソッドを使用して admin_one という管理者 Mbean を呼び出します。

```
java weblogic.Admin -username system -password gumby1234 INVOKE
   -mbean mydomain:Name=admin_one,Type=Administrator
   -method getAttributeStringValue PhoneNumber
```

SET

指定したコンフィグレーション Mbean の指定した属性値を設定します。成功した場合は、OK を stdout に返します。このコマンドは実行時 Mbean では使用できません。

新しい値は、その値が定義された場所によって config.xml ファイルかセキュリティ レルムに保存されます。

構文

```
java weblogic.Admin [-url URL] [-username username]
   [-password password] SET {-type mbean_type|-mbean mbean_name}
   -property property1 property1_value
[-property property2 property2_value] . . .
```

引数	定義
mbean_type	同じタイプの複数のオブジェクトに対する属性を取得するときに必要となる。次のように、Mbean 名の完全修飾名を指定する必要がある。 "domain:Name:name,Type=*type*,Application=*application*"
mbean_name	必須。Mbean の完全修飾名を次の形式で指定する必要がある。 "*domain*:Name=*name*,Location:*location*,Type=*type*" 各値の説明は次のとおり。 ■ Name は Mbean 名。 ■ Location では Mbean の位置を指定する。 ■ Type ではオブジェクト グループのタイプを指定する。
property	必須。設定する属性プロパティの名前を指定する。

引数	定義
property _value	必須。属性プロパティに設定する値を指定する。 ■ 引数が Mbean の配列の場合、その引数は以下の形式で渡されなければならない。 "*domain*:Name=*name*,Type=*type*;*domain*:Name=*name*,Type=*type*" ■ 引数が文字列の配列の場合、その引数は以下の形式で渡されなければならない。 "*String1*;*String2*;. . . " ■ JDBC 接続プールの属性を設定する場合、引数は以下の形式で渡さなければならない。 "user:*username*;password:*password*;server:*servername*"

C　Web サーバ プラグインのパラメータ

以下の節では、Apache、Netscape、および Microsoft IIS の Web サーバ プラグインをコンフィグレーションするために使用するパラメータを説明します。

- 概要
- Web サーバ プラグインの一般的なパラメータ
- Web サーバ プラグインの SSL パラメータ

概要

各 Web サーバ プラグインのパラメータは、特殊なコンフィグレーション ファイルに入力します。このコンフィグレーション ファイルは各 Web サーバで別々の名前を持ち、ファイルの形式にはそれぞれの規則があります。詳細については、各プラグインの以下の節を参照してください。

- 11-1 ページの「Apache HTTP Server プラグインのインストールとコンフィグレーション」
- 12-1 ページの「Microsoft Internet Information Server（ISAPI）プラグインのインストールとコンフィグレーション」
- 13-1 ページの「Netscape Enterprise Server プラグイン（NSAPI）のインストールとコンフィグレーション」

Web サーバ プラグインのパラメータは、次の表で説明するとおりに入力します。

Web サーバ プラグインの一般的なパラメータ

注意： パラメータでは大文字 / 小文字を区別します。

パラメータ	デフォルト値	説明
WebLogicHost （単一 WebLogic Server にプロキシする場合は必須）	none	HTTP リクエストの転送先となる WebLogic Server ホスト（または、WebLogic Server で動作している Web サーバで定義した仮想ホスト名）。 WebLogic クラスタを使用している場合は、WebLogicHost の代わりに WebLogicCluster パラメータを使用する。
WebLogicPort （単一 WebLogic Server にプロキシする場合は必須）	none	WebLogic 接続リクエストに対して WebLogic Server ホストがリスニングを行うポート。プラグインと WebLogic Server の間で SSL を使用する場合は、このパラメータを SSL リスン ポート（8-4 ページの「リスンポートのコンフィグレーション」を参照）に設定し、SecureProxy パラメータを ON に設定する。 WebLogic クラスタを使用している場合は、WebLogicPort の代わりに WebLogicCluster パラメータを使用する。

パラメータ	デフォルト値	説明
WebLogicCluster （WebLogic Server のクラスタにプロキシする場合は必須）	none	ロードバランシングのためにクラスタで使用できる WebLogic Server の一覧。クラスタ リストは、「ホスト：ポート」という形式のエントリをカンマで区切ったもの。次に例を示す。 `WebLogicCluster myweblogic.com:7001, yourweblogic.com:7001,theirweblogic.com:7001` プラグインと WebLogic Server の間で SSL を使用する場合は、ポート番号を SSL リスン ポート（8-4 ページの「リスン ポートのコンフィグレーション」を参照）に設定し、SecureProxy パラメータを ON に設定する。 WebLogicCluster は、WebLogicHost パラメータと WebLogicPort パラメータの代わりに使用する。WebLogic Server では、最初に WebLogicCluster パラメータがチェックされる。見つからなかった場合は、WebLogicHost および WebLogicPort が検索されて使用される。 プラグインは、使用可能な全クラスタ メンバの間で単純なラウンドロビンを行う。このプロパティで指定するクラスタ リストは、サーバおよびプラグインが保持する動的クラスタ リストの最初の状態。新しく追加されたり、障害が発生したり、障害から回復したクラスタ メンバがあると、WebLogic Server とプラグインは協力してクラスタ リストを自動的に更新する。 動的クラスタ リストを無効化するには、DynamicServerList パラメータを OFF に設定する（Microsoft Internet Information Server のみ）。 プラグインは、クッキーの含まれている HTTP リクエスト、URL エンコーディングされたセッションの含まれている HTTP リクエスト、または POST データにセッションを格納している HTTP リクエストをそのクッキーを元々作成したクラスタ内のサーバに転送する。

パラメータ	デフォルト値	説明
PathTrim	null	リクエストが WebLogic Server に転送される前に、元の URL の先頭からプラグインによって取り除かれる文字列。次に例を示す。 http://myWeb.server.com/weblogic/foo この URL が解析用にプラグインに渡され、その URL が WebLogic Server に渡される前に PathTrim が /weblogic を取り除くように設定されている場合、WebLogic Server に転送される URL は次のようになる。 http://myweblogic.server.com:7001/foo
PathPrepend	null	PathTrim が取り除かれた後、リクエストが WebLogic Server に転送される前に、元の URL の先頭にプラグインによって付加される文字列。
ConnectTimeoutSecs	10	プラグインが WebLogic Server ホストへの接続を試行する最大時間（秒）。この値は ConnectRetrySecs より大きくする。接続できないまま、何回か再試行（ConnectRetrySecs の項を参照）しても成功せず、ConnectTimeoutSecs の設定時間が切れた場合は、「HTTP 503/Service Unavailable」の応答がクライアントに返される。 エラー応答は ErrorPage パラメータを使用してカスタマイズできる。
ConnectRetrySecs	2	WebLogic Server ホスト（またはクラスタ内のすべてのサーバ）への接続試行の間にプラグインがスリープする間隔（秒）。この値は ConnectTimeoutSecs より小さくする。「HTTP 503/Service Unavailable」応答がクライアントに返されるまでにプラグインが接続を試行する回数は、ConnectTimeoutSecs を ConnectRetrySecs で除算することで算出される。 再試行しないようにするには、ConnectRetrySecs を ConnectTimeoutSecs と同じ値に設定する。ただし、プラグインは最低 2 回、接続を試みる。 エラー応答は ErrorPage パラメータを使用してカスタマイズできる。

パラメータ	デフォルト値	説明
Debug	OFF	デバッグで実行されるロギングの種類を設定する。これらのデバッグ オプションをプロダクション システムで切り替えることはお勧めできない。 デバッグ情報は、UNIX システムでは /tmp/wlproxy.log ファイル、Windows NT または 2000 システムでは c:\TEMP\wlproxy.log ファイルに書き込まれる。この位置とファイル名は、WLLogFile パラメータを別のディレクトリおよびファイルに設定することでオーバーライドできる。 以下のロギング オプションのいずれかを設定できる（HFC、HTW、HFW、および HTC オプションは、カンマ区切りで「HFC,HTW」というように組み合わせて設定できる）。 ON 情報メッセージとエラー メッセージのみのログが作成される。 OFF デバッグ情報のログは作成されない。 HFC クライアントのヘッダ、情報メッセージ、およびエラー メッセージのログが作成される。 HTW WebLogic Server に送信されるヘッダ、情報メッセージ、およびエラー メッセージのログが作成される。 HFW WebLogic Server から送信されるヘッダ、情報メッセージ、およびエラー メッセージのログが作成される。 HTC クライアントに送信されるヘッダ、情報メッセージ、およびエラー メッセージのログが作成される。 ALL クライアントとの間で送受信されるヘッダ、WebLogic Server との間で送受信されるヘッダ、情報メッセージ、およびエラー メッセージのログが作成される。

パラメータ	デフォルト値	説明
WLLogFile	Debug パラメータを参照。	Debug パラメータが ON に設定されている場合に生成されるログ ファイルのパスとファイル名を指定する。このディレクトリはこのパラメータの設定前に作成する必要がある。
DebugConfigInfo	OFF	特殊なクエリ パラメータ「__WebLogicBridgeConfig」を有効にする。このパラメータは、プラグインからコンフィグレーション パラメータに関する詳細を取得するのに使用する。 たとえば、DebugConfigInfo を設定して「__WebLogicBridgeConfig」を有効にし、クエリ文字列 ?__WebLogicBridgeConfig を含むリクエストを送信すると、コンフィグレーション情報と実行時統計が収集され、その情報がブラウザに返される。この場合、プラグインは WebLogic Server に接続しない。 このパラメータはデバッグにのみ使用するもので、出力メッセージの形式はリリースによって異なる。セキュリティ上の理由から、プロダクション システムではこのパラメータを OFF にしておくことが望ましい。
StatPath （Microsoft Internet Information Server プラグインでは利用できない）	false	true に設定した場合、プラグインでは、リクエストを WebLogic Server に転送する前に、リクエストの変換されたパス（Proxy-Path-Translated）の有無とパーミッションがチェックされる。 ファイルが存在しない場合、「HTTP 404 File Not Found」の応答がクライアントに返される。ファイルが存在するものの読み取れない場合は、「HTTP 403/Forbidden」の応答がクライアントに返される。どちらの場合も、Web サーバのデフォルトのメカニズムがこれらの応答を処理する。このオプションは、WebLogic Server Web アプリケーションのドキュメント ルートと Web サーバのドキュメント ルートが同じ場合に役立つ。 エラー応答は ErrorPage パラメータを使用してカスタマイズできる。

パラメータ	デフォルト値	説明
ErrorPage	none	Web サーバがリクエストを WebLogic Server に転送できなかった場合に表示されるユーザ独自のエラー ページを作成できる。 このパラメータは、以下の 2 通りの方法で設定できる。 ■ 相対 URI（ファイル名）として。プラグインは、エラーを返す Web アプリケーションのコンテキスト パスを自動的に URI に追加する。プロキシのコンフィグレーション（MIME タイプまたはパス）によっては、エラー ページのリクエストは WebLogic Server にプロキシされない場合がある。 ■ 絶対 URL として（推奨）。エラー ページへの絶対 URL を使用すると、リクエストが常に WebLogic Server 上の適切なリソースにプロキシされる。たとえば http://host:port/myWebApp/ErrorPage.html のように設定する。
HungServerRecoverSecs	300	WebLogic Server のリクエストへの応答に対するプラグインの待ち時間を定義する。プラグインは、サーバが応答するまで HungServerRecoverSecs で指定した秒数だけ待ってから、サーバの応答なしを宣言して、次のサーバにフェイルオーバする。この値は、大きな値にしておく必要がある。サーブレットの処理時間より短くした場合は、予期しない結果が発生する場合がある。 最小値 : 10 最大値 : 600
Idempotent	ON	ON に設定されている状態で、サーバが HungServerRecoverSecs の時間内に応答しない場合、プラグインはフェイルオーバする。 「OFF」に設定した場合、プラグインはフェイルオーバしない。Netscape Enterprise Server プラグインまたは Apache HTTP サーバを使用している場合は、異なる URL または MIME タイプごとにこのパラメータを別々に設定できる。

パラメータ	デフォルト値	説明
CookieName	JSESSIONID	WebLogic Server Web アプリケーションの WebLogic Server セッション クッキー名を変更する場合、プラグインの CookieName パラメータを同じ値に変更する必要がある。WebLogic セッション クッキー名は、<session-descriptor> 要素（http://edocs.beasys.co.jp/e-docs/wls61/webapp/weblogic_xml.html#session-descriptor を参照）の WebLogic 固有のデプロイメント記述子で設定される。
DefaultFileName	none	URI が「/」の場合、プラグインは以下の手順を実行する。 1. PathTrim パラメータで指定されたパスを取り除く。 2. DefaultFileName の値を付加する。 3. PathPrepend で指定された値を先頭に追加する。 これによって、WebLogic Server からリダイレクトされなくなる。 DefaultFileName は、リクエストがプロキシされる WebLogic Server の Web アプリケーションのデフォルト ウェルカム ページに設定する。たとえば、DefaultFileName を welcome.html に設定した場合、HTTP リクエストが「http://somehost/weblogic」であれば、「http://somehost/weblogic/welcome.html」になる。このパラメータが機能するためには、リクエストが転送される全 Web アプリケーションで同じウェルカム ファイルを指定する必要がある。詳細については、http://edocs.beasys.co.jp/e-docs/wls61/webapp/components の「ウェルカム ページのコンフィグレーション」を参照。 Apache を使用する場合の注意 : Stronghold バージョンまたは Raven バージョンを使用する場合は、IfModule ブロックではなく Location ブロックでこのパラメータを定義する。

パラメータ	デフォルト値	説明
MaxPostSize	-1	POST データの最大許容サイズ（バイト単位）。コンテキスト長が MaxPostSize を超えた場合、プラグインによってエラー メッセージが返される。-1 に設定した場合、POST データのサイズはチェックされない。これは、POST データを使用してサーバを過負荷状態にしようとするサービス拒否攻撃を防ぐのに役立つ。

MatchExpression
（Apache HTTP サーバのみ）
	none	MIME タイプによるプロキシを行う場合、MatchExpression パラメータを使用して IfModule ブロック内にファイル名のパターンを設定する。
		MIME タイプでプロキシする場合の例を次に示す。
		`<IfModule mod_weblogic.c>`
		`  MatchExpression *.jsp`
		`  WebLogicHost=myHost|paramName=value`
		`</IfModule>`
		パスでプロキシする場合の例を次に示す。
		`<IfModule mod_weblogic.c>`
		`  MatchExpression /weblogic`
		`  WebLogicHost=myHost|paramName=value`
		`</IfModule>`

FileCaching	ON	ON に設定されている状態で、リクエストの POST データのサイズが 2048 バイトより大きい場合、POST データはディスク上の一時ファイルに格納され、8192 バイト単位で WebLogic Server に転送される。ただし、FileCaching を ON に設定すると、ブラウザで表示される、ダウンロードの進捗状況を示すプログレス バーで問題が生じる可能性がある。ブラウザでは、ファイルがまだ転送中であるにもかかわらずダウンロードの完了が示される。 OFF に設定されている状態で、リクエストの POST データのサイズが 2048 バイトより大きい場合、POST データはメモリに格納され、8192 バイト単位で WebLogic Server に送信される。OFF に設定すると、リクエストの処理中にサーバがダウンした場合に問題が発生する（プラグインでフェイルオーバが不可能であるため）。

パラメータ	デフォルト値	説明
WlForwardPath （Microsoft Internet Information Server のみ）	null	WlForwardPath が「/」に設定されている場合は、すべてのリクエストがプロキシされる。特定の文字列で始まるリクエストを転送するには、WlForwardPath をその文字列に設定する。たとえば、WlForwardPath を /weblogic に設定すると、/weblogic で始まるすべてのリクエストが Weblogic Server に転送される。 このパラメータは、パスでプロキシを実行する場合に必要。カンマで文字列を区切れば、複数の文字列を設定できる。たとえば WlForwardPath=/weblogic,/bea のように設定する。
KeepAliveSecs （Apache HTTP サーバ バージョン 1.3.x には適用されない）	30	プラグインと WebLogic Server のアクティブではない接続が閉じられるまでの時間。このパラメータを有効にするには、KeepAliveEnabled を true に設定する必要がある。 このパラメータの値は、Administration Console の [サーバ ｜ HTTP] タブで設定される [持続時間] フィールドの値、または KeepAliveSecs 属性を使用して server Mbean で設定される値以下でなければならない。
KeepAliveEnabled （Apache HTTP サーバ バージョン 1.3.x には適用されない）	true	プラグインと WebLogic Server の間の接続のプールを有効化する。
QueryFromRequest （Apache HTTP サーバのみ）	OFF	ON に設定されている場合、Apache プラグインは (request_rec *)r->the request を使用して WebLogic Server にクエリ文字列を渡す（詳細については Apache のマニュアルを参照）。この動作は以下の状況において望ましい。 ■ Netscape バージョン 4.x ブラウザがクエリ文字列にスペースのあるリクエストを行う場合 ■ HP で Raven Apache 1.5.2 を使用する場合 OFF に設定されている場合、Apache プラグインは (request_rec *)r->args を使用して WebLogic Server にクエリ文字列を渡す。

パラメータ	デフォルト値	説明
MaxSkips （Apache 1.3.x では利用できない）	10	DynamicServerList が OFF に設定されている場合のみ有効。 WebLogicCluster パラメータまたは WebLogic Server から返される動的クラスタ リストにある WebLogic Server で障害が発生した場合、その障害の発生したサーバは「bad」とマークされ、プラグインはリスト内の次のサーバに接続しようとする。 MaxSkips は、プラグインが「bad」とマークされたサーバへの接続を再試行するまでの試行回数を設定する。プラグインは、ユニークなリクエスト（クッキーのないリクエスト）を受信するたびにリスト内の新しいサーバに接続しようとする。
DynamicServerList	ON	OFF に設定すると、プラグインからプロキシされるリクエストをロードバランシングするために使用される動的クラスタ リストが無視され、WebLogicCluster パラメータで指定された静的リストのみが使用される。通常、このパラメータは ON のままにする。 このパラメータを ON に設定する場合は、以下のことを考慮する必要がある。 ■ 静的リストの 1 つまたは複数のサーバで障害が発生した場合、プラグインは応答不能のサーバへの接続に時間を費やし、その結果としてパフォーマンスが低下する可能性がある。 ■ クラスタに新しいサーバを追加した場合は、このパラメータを再定義するまでプラグインはその新しいサーバにリクエストをプロキシできない。WebLogic Server は、新しいサーバがクラスタに追加されたときに動的サーバ リストに自動的にその新しいサーバを追加する。

パラメータ	デフォルト値	説明
WLProxySSL	OFF	次の条件に一致する場合に、プラグインと WebLogic Server 間の SSL 通信を維持する場合は、このパラメータを ON に設定する。 ■ HTTP クライアント リクエストが HTTPS プロトコルを指定している。 ■ リクエストが 1 つまたは複数のプロキシ サーバ（WebLogic Server プロキシ プラグインを含む）を経由して渡された。 ■ プラグインと WebLogic Server 間の接続が HTTP プロトコルを使用している。 WLProxySSL を ON に設定すると、WebLogic Server からクライアントに返されるロケーション ヘッダは HTTPS プロトコルを指定する。
WLLocalIP	none	プラグインがマルチホーム マシンで動作している WebLogic Server インスタンスに接続する場合のバインド先の IP アドレスを定義する。 WLLocalIP を設定しない場合、マルチホーム マシンで任意の IP アドレスが使用される。

Web サーバ プラグインの SSL パラメータ

注意： パラメータでは大文字 / 小文字を区別します。

パラメータ	デフォルト値	説明
SecureProxy	OFF	このパラメータを ON に設定すると、WebLogic Server プロキシ プラグインと WebLogic Server 間のすべての接続で SSL プロトコルの使用が有効になる。このパラメータを定義する前に、対応する WebLogic Server のポートを SSL プロトコル用にコンフィグレーションしておく必要がある。 このパラメータは、メイン サーバ用のコンフィグレーションと仮想ホスト用のコンフィグレーション（仮想ホストが定義されている場合）の 2 つのレベルで設定できる。仮想ホスト用のコンフィグレーションでこの設定がオーバーライドされない場合、メイン サーバ用のコンフィグレーションから SSL のコンフィグレーションを継承する。
TrustedCAFile	none	WebLogic Server プロキシ プラグインに対する信頼された認証局によるデジタル証明書が含まれるファイルの名前。SecureProxy パラメータが ON に設定されている場合はこのパラメータが必要。 filename にはファイルの絶対ディレクトリ パスを指定する。
RequireSSLHostMatch	true	WebLogic Server プロキシ プラグインが接続するホストの名前が、プロキシ プラグインが接続する WebLogic Server のデジタル証明書にある Subject Distinguished Name フィールドに一致する必要があるかどうかを指定する。

パラメータ	デフォルト値	説明
SSLHostMatchOID	22	ASN.1 Object ID (OID) を指定する。ホスト名の比較に使用されるピア デジタル証明書内の Subject Distinguished Name フィールドを示す。デフォルトでは、Subject Distinguished Name の CommonName フィールドに対応する。一般的な OID 値は以下のとおり。 ■ Sur Name － 23 ■ Common Name － 22 ■ Email － 13 ■ Organizational Unit － 30 ■ Organization － 29 ■ Locality － 26

索引

A

ADMIN_URL 環境変数 2-15
Administration Console
  SSL で使用するプライベート キー パスワードの指定 2-6
  WebLogic Server の停止 2-15
  起動 1-4
  使い方、アプリケーションのデプロイ 7-2
  テーブルのカスタマイズ 1-4
Administration Console の起動 4-4
Apache プラグイン 11-1
  httpd.conf ファイル 11-9
  httpd.conf ファイルのサンプル 11-18
  SSL 11-14
  インストール 11-4
  仮想ホスト 11-20
  クラスタ 11-19
  パラメータ 11-11
  リクエストのプロキシ 11-10

B

beasvc.exe 2-20

C

CANCEL_SHUTDOWN、WebLogic Server コマンド B-6
config.xml 1-2
config.xml.booted 2-10
CONNECT、WebLogic Server コマンド B-7
ConnectionRetrySecs C-4
ConnectionTimeoutSecs C-4
CREATE、WebLogic Server コマンド B-29
CREATE_POOL、WebLogic Server コマンド B-20

D

Debug C-5
DebugConfigInfo C-6
DefaultFileName C-8
DELETE、WebLogic Server コマンド B-31
DESTROY_POOL、WebLogic Server コマンド B-23
DISABLE_POOL、WebLogic Server コマンド B-24
DynamicServerList C-11

E

ENABLE_POOL、WebLogic Server コマンド B-25
ErrorPage C-7
EXISTS_POOL WebLogic Server コマンド B-26

F

FileCaching C-9

G

GET、WebLogic Server コマンド B-32

H

HELP、WebLogic Server コマンド B-8
HTTP 8-2
HTTP アクセス ログ 8-14
  拡張ログ フォーマット 8-17

共通ログ フォーマット 8-16
設定 8-15
ログ ローテーション 8-15
HTTP トンネリング 8-26
クライアント接続 8-27
コンフィグレーション 8-26
HTTP パラメータ 8-2
HTTP リクエスト 8-11
HttpClusterServlet 10-1
デプロイメント記述子のサンプル 10-5
HungServerRecoverSecs C-7

I

I/O 8-28
Idempotent C-7
INVOKE、WebLogic Server コマンド B-34

J

Java Management Extension
JMX を参照 1-1
Java ヒープ メモリ
最小値と最大値の指定 2-5
JDBC 接続プール
管理 5-4
モニタ 5-4
JMS
WebLogic Server の障害からの回復 17-30
コンフィグレーション
送り先 17-6
送り先キー 17-8
概要 17-2
サーバ 17-3
セッション プール 17-12
接続ファクトリ 17-4
テンプレート 17-7
バッキング ストア 17-9
フェイルオーバ手順 17-30
モニタ 17-13, 17-14
JMS サーバ 17-3
JMX、管理システムでの使用 1-1
JMX 通知
ログでの使用 1-8
JNDI ネーミング ツリー
ノードのバインドの表示 B-10

K

KeepAliveSecs C-10

L

LICENSES、WebLogic Server コマンド B-9
LIST、WebLogic Server コマンド B-10
LOCK、WebLogic Server コマンド B-11

M

Management Bean
MBean を参照 1-5
MatchExpression C-9
MaxPostSize 8-25, C-9
MaxSkips C-11
MBean
実行時とコンフィグレーション 1-5
Mbean 管理コマンド、概要 B-28
Mbean 情報の取得、GET コマンド B-32
Mbean の削除、DELETE コマンド B-31
Mbean の作成、CREATE コマンド B-29
Microsoft-IIS（プロキシ）プラグイン
コンフィグレーション 12-4
サーブレットのプロキシ 12-11
テスト 12-12
リクエストのプロキシ 12-3

N

Netscape（プロキシ）プラグイン 13-2
MIME タイプ 13-4
obj.conf ファイル 13-5
obj.conf ファイルのサンプル 13-15
クラスタ化 13-14

P

PathPrepend C-4
PathTrim C-4
PING、WebLogic Server コマンド B-12
Post タイムアウト秒 8-25
POST メソッド 8-25
Probable Cause 6-7
ProxyServlet 9-1
　デプロイメント記述子のサンプル 9-3

Q

QueryFromRequest C-10

R

Recommended Action 6-8
RequireSSLHostMatch C-13
RESET_POOL、WebLogic Server コマンド B-27
running-managed-servers.xml 2-11

S

SecureProxy C-13
SERVER_NAME 環境変数 2-15
SERVERLOG、WebLogic Server コマンド B-13
SET、WebLogic Server コマンド B-35
SHUTDOWN、WebLogic Server コマンド B-14
SSL
　サーバ起動時のプライベート キー パスワードの指定 2-6
SSL セッション キャッシング
　指定 2-7
SSLHostMatchOID C-14
StatPath C-6

T

THREAD_DUMP、WebLogic Server コマンド B-15
TransmitFile 8-28
TrustedCAFile C-13

U

UNLOCK、WebLogic Server コマンド B-16
URL の解決 8-11

V

VERSION、WebLogic Server コマンド B-17

W

Web アプリケーション 8-5
　URL 8-11
　デフォルト Web アプリケーション 8-5
WebLogic Server
　起動 2-2
　起動時のユーザ名の指定 2-6
　強制停止と停止、違い 3-18
　コマンドラインからの停止 2-16
　ライセンス、表示 B-9
WebLogic Server コマンドのヘルプ表示 B-8
WebLogic Server の起動
　Windows サービス 2-4
WebLogic Server のコマンド
　CANCEL_SHUTDOWN B-6
　CONNECT B-7
　CREATE B-29
　CREATE_POOL B-20
　DELETE B-31
　DESTROY_POOL B-23
　DISABLE_POOL B-24
　ENABLE_POOL B-25
　EXISTS_POOL B-26
　GET B-32
　HELP B-8
　INVOKE B-34
　LICENSES B-9

LIST B-10
LOCK B-11
Mbean 管理コマンドの概要 B-28
PING B-12
RESET_POOL B-27
SERVERLOG B-13
SET B-35
SHUTDOWN B-14
THREAD_DUMP B-15
UNLOCK B-16
VERSION B-17
管理コマンドの概要 B-4, B-18
構文と引数 B-2
コマンドライン インタフェースの有効化 B-2
接続プール コマンドの概要 B-18
WebLogic Server の停止 2-15
WebLogic Server のリモートでの起動 3-17
WebLogic Server のリモートでの停止 3-17
WebLogic Server リスン ポートの検証 B-12
WebLogic Server、リモートでの起動 3-10
WebLogicCluster C-3
WebLogicHost C-2
WebLogicPort C-2
Windows サービス
WebLogic Server の起動 2-4
WebLogic Server の削除 2-18
WLForwardPath C-10

あ

アクセス ログ 8-14
アプリケーション コンポーネント
デプロイメント 7-3
アプリケーションのデプロイメント 7-1

い

印刷、製品のマニュアル 1-xix

お

送り先、JMS 17-6
送り先キー、JMS 17-8

か

概要 4-2
拡張ログ フォーマット 8-14
カスタマ サポート情報 1-xx
仮想ホスティング 8-7
Apache プラグイン 11-20
設定 8-8
デフォルト Web アプリケーション 8-8
ガベージコレクション、強制 5-3
管理コマンド、概要 B-4, B-18
管理サーバ 4-2
管理対象サーバの検出 2-10
起動 2-2
起動時のクラスパスの指定 2-8
基本説明 1-2
コマンドラインからの起動 2-4
再起動 2-10
スクリプトによる起動 2-9
ドメインのモニタでのロール 5-2
管理サーバの起動スクリプト 2-9
管理サブシステム
図 1-5
管理サブシステム、概要 1-1
管理対象サーバ
起動 2-12
起動時における管理サーバの URL の指定 2-13
基本説明 1-2
コンフィグレーション エントリの追加 2-12
サスペンドと再開 2-17
スクリプトによる起動 2-14
管理対象サーバの起動スクリプト 2-14
管理対象サーバの検出 2-10
管理対象サーバの再開 2-17
管理対象サーバのサスペンド 2-17

き

キー
　ライセンス 20-2
起動、管理サーバ 2-2
共通ログ フォーマット 8-14

く

クラスタ コンフィグレーションの作業 4-11
クラスタ コンフィグレーションのプランニング 4-6
クラスタへのリクエストのプロキシ 10-1
クラスパス
　WebLogic Server 起動時の指定 2-8

こ

コマンドライン インタフェース
　Mbean 管理コマンドの概要 B-28
　管理コマンドの概要 B-4, B-18
　コマンドの構文と引数 B-2
　有効化 B-2
コンソール
　Administration Console を参照 1-4
コンフィグレーション
　Apache プラグイン 11-11
　HTTP パラメータ 8-2
　JMS
　　送り先 17-6
　　送り先キー 17-8
　　概要 17-2
　　サーバ 17-3
　　セッション プール 17-12
　　接続ファクトリ 17-4
　　テンプレート 17-7
　　バッキング ストア 17-9
　Microsoft-IIS（プロキシ）プラグイン 12-4
コンフィグレーション属性
　起動時の指定 2-8
コンフィグレーション ディレクトリ
　構造 2-8
コンフィグレーション ファイル、バックアップ 2-10

さ

サーバ起動メッセージ
　リモートでの起動時 3-2
サーバ コンフィグレーションの作業 4-7
サーバ障害の回復、JMS 17-30
サーバ セッション プール、JMS 17-12
サーバの強制停止
　停止との違い 3-18
サーバ名
　起動時の指定 2-5
サーバ ログ ファイルの表示、SERVERLOG コマンド B-13
サービス拒否攻撃、防止 8-25
最大 Post 時間 8-25
サポート
　技術情報 1-xx

し

システム ホーム ディレクトリ、WebLogic
　起動時の指定 2-6
自動デプロイメント 7-7
　applications ディレクトリのデフォルトのチェック周期 7-8
　有効化 7-7
シャットダウン クラス
　登録 2-21
障害、サーバ 17-30
証明書
　ノード マネージャでの使用 3-15

す

スクリプト
　JDK_HOME の設定 2-9, 2-14
スクリプトの JDK_HOME 設定 2-9, 2-14
スタートアップ クラス
　登録 2-21

スレッド、実行時の表示 B-15

せ

静的デプロイメント 7-2
接続ファクトリ、JMS 17-4
接続プール管理コマンド、概要 B-18
接続プールのリセット、RESET_POOL コマンド B-27

そ

属性値の設定、SET コマンド B-35

て

デフォルト Web アプリケーション 8-5
　仮想ホスティング 8-8
デプロイメント
　アプリケーション コンポーネント 7-3
デプロイメント、静的 7-2
デプロイメント、動的
　展開形式のアプリケーション 7-8
テンプレート、JMS 17-7

と

動的コンフィグレーション 4-5
動的デプロイメント 7-7
ドメイン
　基本説明 1-2
　モニタ 5-1
ドメイン、非アクティブ
　編集 1-3
ドメイン名
　起動時の指定 2-7
ドメイン ログ 1-7
　フィルタの変更 6-10
トランザクション、モニタ 5-4
トンネリング 8-26

ね

ネイティブ I/O 8-28

の

ノード マネージャ
　Windows サービスとしてインストール 3-19
　Windows サービスとして削除 3-21
　起動 3-14
　基本説明 3-1
　クラスパス引数 3-16
　デジタル証明書 3-9
　プラットフォーム サポート 3-11
　マシン用のコンフィグレーション 3-9

は

パスワード
　WebLogic Server 起動時の使用 2-3
バッキング ストア、JMS 17-9

ひ

評価ライセンス 20-1

ふ

フェイルオーバ手順、JMS 17-30
プラットフォーム サポート
　ノード マネージャ 3-11

ほ

ホスト名検証
　カスタム ホスト名検証の指定 2-7
　起動時の無効化 2-7

ま

マシン エントリ
　ノード マネージャで使用 3-9
マニュアル、入手先 1-xix

め

メッセージ カタログ 6-7
メッセージの属性
Machine Name 6-7
Message Body 6-7
Message Detail 6-7
Message Id 6-7
Probable Cause 6-7
Recommended Action 6-8
Server Name 6-7
Severity 6-6
Subsystem 6-6
Thread Id 6-7
Timestamp 6-6
Transaction Id 6-7
User Id 6-7

も

モニタ
Console のページの種類 5-1
JDBC 接続プール 5-4
JMS 17-13
JMS オブジェクト 17-14
恒久サブスクライバ 17-14
WebLogic ドメイン 5-1
仕組み 5-2
モニタ、WebLogic Server 5-2

ら

ライセンス
キー 20-2
更新 20-2
評価 20-1

り

リクエストのプロキシ 9-1
Apache プラグイン 11-10
Microsoft-IIS（プロキシ）プラグイン 12-3
リスン ポート 8-4
リスン ポート、検証 B-12
リソース、WebLogic
モニタ 5-1
リモートでの起動と停止
アーキテクチャ 3-4
コンフィグレーション 3-10

ろ

ローテーション、ログ ファイル 6-4
ログ ファイル
参照 6-9
ログ メッセージの属性
メッセージの属性を参照 6-6