



BEA WebLogic Server™ およ び WebLogic Express®

管理者ガイド

著作権

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

限定的権利条項

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

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

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

商標または登録商標

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

その他の商標はすべて、関係各社がその権利を有します。

管理者ガイド

パート番号	マニュアルの日付	ソフトウェアのバージョン
なし	2002年6月28日	BEA WebLogic Server バージョン 7.0

目次

このマニュアルの内容

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

1. WebLogic システムの概要

システム管理の概要.....	1-1
WebLogic Server のドメイン.....	1-2
システム管理のインフラストラクチャ.....	1-4
管理サーバと管理対象サーバ.....	1-6
管理サーバのフェイルオーバー.....	1-6
管理対象サーバのフェイルオーバー.....	1-7
ドメイン全体の管理ポート.....	1-7
サービスパックと WebLogic Server インスタンス.....	1-8
システム管理ツール.....	1-8
システム管理ツールのセキュリティ保護.....	1-8
システム Administration Console.....	1-8
コマンドラインインタフェース.....	1-9
JMX.....	1-10
コンフィグレーション ウィザード.....	1-11
Java ユーティリティ.....	1-11
ノードマネージャ.....	1-12
SNMP.....	1-12
ログ.....	1-12
config.xml の編集.....	1-13
WebLogic Server ドメインで管理できるリソース.....	1-13
サーバ.....	1-14
クラスタ.....	1-14
マシン.....	1-15

ネットワーク チャネル	1-15
JDBC	1-15
JMS.....	1-16
WebLogic メッセージング ブリッジ	1-17
Web サーバと Web コンポーネント	1-17
アプリケーション	1-17
アプリケーションの形式.....	1-18
Administration Console を使用したデプロイメント記述子の編集	1-19
WebLogic Builder ツールを使用したデプロイメント記述子の編集および作成	1-19
起動クラスと停止クラス	1-19
JNDI	1-20
トランザクション	1-20
XML	1-21
セキュリティ	1-21
WebLogic Tuxedo Connector	1-22
Jolt	1-22
メール	1-22
Administration Console の起動と使い方	1-22
Administration Console がサポートされているブラウザ	1-23
Administration Console の起動	1-23
Administration Console の使い方	1-24
Administration Console 内の移動	1-25
オブジェクトまたはリソースのコンフィグレーション	1-27
Administration Console を使用した複数ドメインの管理	1-27
Administration Console を使用したドメインのモニタ	1-27
Administration Console のタスクのモニタ	1-28
Administration Console を使用したヘルプの表示	1-28
Web サーバを伴う WebLogic Server の使用	1-28
モニタ	1-29
ライセンス	1-30

2. WebLogic Server の起動と停止

サーバのライフサイクル	2-1
サーバのライフサイクルの制御	2-5

ライフサイクル オペレーションのタイムアウト期間.....	2-6
サーバを起動するためのユーザ名とパスワードの指定	2-6
初期管理ユーザ名の指定.....	2-7
ユーザ名とパスワードのプロンプトの回避	2-8
管理サーバの起動 ID ファイルの作成.....	2-8
管理対象サーバの起動 ID ファイルの作成	2-10
起動 ID ファイルの使用.....	2-10
起動後の起動 ID ファイルの削除.....	2-12
代替手段：コマンドラインでの ID 情報の入力	2-12
管理サーバの起動	2-13
Windows の [スタート] メニューからの管理サーバの起動.....	2-14
スクリプトを使用した管理サーバの起動.....	2-14
コンフィグレーション ウィザード スクリプトを使用した管理サーバの起動.....	2-15
管理サーバを起動する独自のスクリプトの作成	2-15
WebLogic Server でのデフォルト以外の JVM の使用.....	2-18
weblogic.Server コマンドの使用.....	2-18
クラスパスの設定	2-19
weblogic.Server コマンドの構文.....	2-20
必須の引数	2-20
よく使用される任意指定の引数	2-21
その他の任意指定の引数.....	2-28
開発モードとプロダクション モード	2-29
管理ポートおよび weblogic.Admin ユーティリティの起動引数 ..	2-31
サーバのルート ディレクトリ	2-31
デフォルト コンフィグレーションを使用したサーバの起動.....	2-33
管理対象サーバの起動.....	2-34
管理対象サーバのドメインへの追加.....	2-35
Windows の [スタート] メニューからの管理対象サーバの起動.....	2-36
スクリプトを使用した管理対象サーバの起動	2-37
コンフィグレーション ウィザード スクリプトを使用した管理対象サーバの起動	2-37
管理対象サーバを起動する独自のスクリプトの作成.....	2-39
コマンドラインからの管理対象サーバの起動	2-39
管理サーバへの接続のコンフィグレーション	2-40

デフォルトの起動状態の指定	2-41
リモート管理対象サーバの起動	2-42
ドメインまたはクラスタ内のすべての WebLogic Server の起動および強制停止	2-42
ドメイン内のすべての管理対象サーバの起動	2-43
クラスタ内のすべての管理対象サーバの起動	2-43
ドメイン内のすべてのサーバの強制停止	2-44
クラスタ内のすべてのサーバの強制停止	2-44
WebLogic Server の停止	2-45
起動クラスと停止クラスのコンフィグレーション	2-46
WebLogic Server インスタンスの Windows サービスとしての設定	2-47
Windows サービスの設定 : 主な手順	2-48
サーバ固有のスクリプトの作成	2-49
管理対象サーバのその他の値の設定	2-52
管理サーバより後に管理対象サーバを起動する要求	2-53
コントロール パネルからの正常な停止の有効化	2-55
標準出力および標準エラーのファイルへのリダイレクト	2-58
クラスのクラスパスへの追加	2-62
サーバ固有のスクリプトの実行	2-63
設定の確認	2-64
サービスを実行するユーザ アカウントの確認	2-64
コントロール パネルを使用したサーバインスタンスの停止と再起動 ..	2-65
Windows サービスとしてのサーバの削除	2-66
Windows サービスとしてのサーバ設定に対する起動資格の変更	2-68

3. システム管理操作の保護

各ロールで使用可能な操作	3-2
デフォルト グループの関連付け	3-3
保護されたユーザ インタフェース	3-4
サーバリソースの階層化されたセキュリティ方式	3-5
サーバリソースのセキュリティ ポリシー	3-6
MBean による保護	3-6
WebLogic Security サービスによる階層化された保護の検証	3-7
例	3-8

パート 1 : MBean による保護	3-8
パート 2 : サーバリソースのセキュリティ ポリシー	3-9
一貫性のあるセキュリティ方式の維持	3-10
WebLogic Server の起動および停止に対するパーミッション	3-11
weblogic.Server コマンドを使用したパーミッション	3-11
ノード マネージャを使用したパーミッション	3-12
WebLogic Server インスタンスの停止	3-12

4. ログメッセージを使用した WebLogic Server の管理

WebLogic Server のログ メッセージ	4-2
メッセージの属性	4-2
メッセージの重要度	4-3
メッセージの出力	4-4
例外とスタック トレース	4-5
WebLogic Server のログ ファイル	4-5
ローカル ログ ファイルとドメイン ログ ファイル	4-6
ログ ファイルの名前と場所	4-8
ログ ファイル ローテーション	4-8
WebLogic ログ ファイル ビューア	4-10
標準出力への出力	4-11
System.out および System.err のファイルへのリダイレクト	4-12
ガベージ コレクション コメント	4-13
コンフィグレーション 監査	4-13
コンフィグレーション 監査の有効化	4-14
コンフィグレーション 監査メッセージ	4-14
追加のログ ファイル	4-18

5. 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	5-1
weblogic.deploy ユーティリティ (非推奨) を使用した Web アプリケーションのデプロイメント	5-2
デプロイメントに関するマニュアル	5-3

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

概要	6-1
----------	-----

HTTP パラメータ	6-2
リスンポートのコンフィグレーション	6-4
Web アプリケーション	6-5
Web アプリケーションとクラスタ化	6-5
デフォルト Web アプリケーションの指定	6-6
仮想ホスティングのコンフィグレーション	6-7
仮想ホスティングとデフォルト Web アプリケーション	6-9
仮想ホストの設定	6-9
WebLogic Server による HTTP リクエストの解決方法	6-11
HTTP アクセス ログの設定	6-14
ログ ローテーション	6-15
共通ログ フォーマット	6-15
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	6-16
Fields ディレクティブの作成	6-17
サポートされるフィールド識別子	6-17
カスタム フィールド識別子の作成	6-19
POST サービス拒否攻撃の防止	6-24
HTTP トンネリングのための WebLogic Server の設定	6-25
HTTP トンネリング接続の設定	6-25
クライアントからの WebLogic Server への接続	6-26
静的ファイルを提供するネイティブ I/O の使用 (Windows のみ)	6-27

7. トランザクションの管理

トランザクション管理の概要	7-1
トランザクションのコンフィグレーション	7-3
トランザクション管理用の追加の属性	7-4
ドメイン間トランザクションに対するドメインのコンフィグレーション	7-6
WebLogic Server 7.0 ドメインのドメイン間トランザクション	7-7
WebLogic Server 7.0 および WebLogic Server 6.x ドメインのドメイン間 トランザクション	7-7
トランザクションのモニタとログ	7-8
トランザクションのモニタ	7-9
トランザクション ログ ファイル	7-10
トランザクション ログ ファイル書き込みポリシーの設定	7-11
ヒューリスティックなログ ファイル	7-13

ヒューリスティックな終了の処理.....	7-14
トランザクションの破棄.....	7-15
別のマシンへのサーバの移動.....	7-16
サーバに障害が発生した後のトランザクションの回復.....	7-16
クラッシュ後のトランザクション回復サービスのアクション.....	7-18
クラスタ化されていないサーバで障害が発生した場合のトランザクシ ョンの回復.....	7-19
クラスタ化されたサーバで障害が発生した場合のトランザクションの回 復.....	7-20
トランザクション回復サービスの移行に対する制限.....	7-22
トランザクション回復サービスの移行準備.....	7-22

8. JDBC 接続の管理

JDBC 管理の概要.....	8-1
Administration Console について.....	8-2
コマンドライン インタフェースについて.....	8-2
JDBC API について.....	8-2
関連情報.....	8-3
管理.....	8-3
JDBC と WebLogic jDrivers.....	8-3
トランザクション (JTA).....	8-4
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). 8-4	
接続プール.....	8-5
アプリケーション スコープの JDBC 接続プール.....	8-6
マルチプール.....	8-7
データ ソース.....	8-7
JDBC データ ソース ファクトリ.....	8-7
JDBC 接続プールのセキュリティ.....	8-8
互換性モードにおける JDBC 接続プールのセキュリティ.....	8-8
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.....	8-10
JDBC コンフィグレーション.....	8-10
JDBC オブジェクトの作成.....	8-11
JDBC オブジェクトの割り当て.....	8-11
Administration Console を使用した JDBC 接続のコンフィグレーショ	

ン	8-13
接続プールのコンフィグレーションにおけるデータベースパスワード	8-15
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	8-17
接続の管理とモニタ	8-17
Administration Console を使用した JDBC の管理	8-17
コマンドラインインタフェースを使用した JDBC の管理	8-19
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	8-20
JDBC コンフィグレーションの概要	8-20
トランザクション データ ソースを使用すべき場合	8-22
ローカル トランザクションをサポートするドライバ	8-23
XA を使用する分散トランザクションをサポートするドライバ ..	8-23
XA を使用しない分散トランザクションをサポートするドライバ ..	8-23
正しい接続数によるサーバのロックアップの回避	8-24
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 ..	8-24
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 ..	8-28
WebLogic jDriver for Oracle/XA のデータ ソース プロパティ	8-34
追加の XA 接続プール プロパティ	8-36
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 ..	8-37
XA 非対応ドライバ / 単一リソース	8-38
XA 非対応ドライバ / 複数リソース	8-38
グローバル トランザクションで XA 非対応のドライバを使用する場合の制限とリスク	8-39
XA 非対応接続プールと Tx データ ソースのコンフィグレーション例	8-41
Prepared Statement キャッシュのパフォーマンスの向上	8-42
XA 非対応の Prepared Statement キャッシュ	8-43
XA 対応の Prepared Statement キャッシュ	8-44
Prepared Statement キャッシュに関する制限	8-46
データベース変更後の Prepared Statement 呼び出しはエラーの原因	

となる可能性がある	8-46
Prepared Statement における setNull の使用	8-46
キャッシュ内の Prepared Statement はデータベース カーソルを予約 可能.....	8-47
Prepared Statement キャッシュの適切なサイズの決定	8-47
起動クラスを使用した XA 非対応の Prepared Statement キャッシュの ロード	8-48

9. JMS の管理

JMS と WebLogic Server	9-1
JMS のコンフィグレーション	9-2
JMS コンフィグレーションにおける命名規則.....	9-3
WebLogic Server の起動と JMS のコンフィグレーション	9-4
デフォルトの WebLogic Server の起動.....	9-4
Administration Console の起動.....	9-4
基本的な JMS 実装のコンフィグレーション	9-5
JMS サーバのコンフィグレーション	9-8
接続ファクトリのコンフィグレーション	9-10
送り先のコンフィグレーション	9-12
JMS テンプレートのコンフィグレーション	9-13
送り先キーのコンフィグレーション	9-14
ストアのコンフィグレーション	9-15
JMS JDBC ストア について.....	9-16
JMS ストア テーブルのプレフィックス	9-18
JMS JDBC ストア向けの JDBC 接続プールの推奨設定.....	9-19
セッション プールのコンフィグレーション	9-19
接続コンシューマのコンフィグレーション	9-20
JMS のモニタ	9-20
JMS オブジェクトのモニタ	9-21
恒久サブスクリバのモニタ	9-22
分散送り先のシステム サブスクリプションとプロキシトピック メン バーのモニタ.....	9-22
JMS のチューニング	9-23
永続ストレージ	9-24
JMS ファイル ストアの同期書き込みポリシーのコンフィグレー ション.....	9-24

メッセージページングのコンフィグレーション	9-29
ページングのコンフィグレーション	9-29
JMS のページング属性	9-35
メッセージのフロー制御の確立	9-42
フロー制御のコンフィグレーション	9-42
フロー制御のしきい値	9-45
分散送り先のチューニング	9-46
分散送り先のメッセージロード バランシングのコンフィグレーション	9-46
分散送り先のサーバアフィニティのコンフィグレーション	9-48
分散送り先のコンフィグレーション	9-49
分散送り先のコンフィグレーション手順	9-49
分散トピックの作成とメンバーの自動作成	9-50
分散トピックの作成および既存の物理的トピックのメンバーとしての 手動追加	9-53
分散キューの作成とメンバーの自動作成	9-55
分散キューの作成および既存の物理キューのメンバーとしての手動 追加	9-58
JMS 分散キュー メンバーの作成	9-60
JMS 分散キュー メンバーの削除	9-61
JMS 分散トピック メンバーの作成	9-62
JMS 分散トピック メンバーの削除	9-63
分散送り先の削除	9-64
分散送り先のモニタ	9-64
WebLogic Server の障害からの回復	9-65
プログラミングの考慮事項	9-65
新しいサーバへの JMS データの移行	9-65

10. WebLogic メッセージング ブリッジの使い方

メッセージングブリッジとは	10-1
メッセージングブリッジのコンフィグレーション タスク	10-2
ブリッジのリソースアダプタについて	10-3
ブリッジのリソースアダプタのデプロイ	10-6
ソースと対象のブリッジ送り先のコンフィグレーション	10-7
JMS ブリッジ送り先のコンフィグレーション	10-7
一般ブリッジ送り先のコンフィグレーション	10-10

メッセージングブリッジインスタンスのコンフィグレーション	10-13
メッセージングブリッジを使用する WebLogic Server のさまざまなリリースおよびドメインとの相互運用	10-20
WebLogic Server およびドメインの名前を付ける際のガイドライン	10-21
WebLogic ドメインのセキュリティ相互運用性の有効化	10-21
メッセージングブリッジを使用するリリース 6.1 以降のドメインにおける送り先へのアクセス	10-22
メッセージングブリッジを使用するリリース 6.0 のドメインにおける送り先へのアクセス	10-24
メッセージングブリッジを使用するリリース 5.1 のドメインにおける送り先へのアクセス	10-25
メッセージングブリッジを使用するサードパーティのメッセージングプロバイダへのアクセス	10-26
メッセージングブリッジの管理	10-27
メッセージングブリッジの停止と再起動	10-27
メッセージングブリッジのモニタ	10-28
実行スレッドプールサイズのコンフィグレーション	10-28

11. JNDI の管理

JNDI 管理の概要	11-1
JNDI およびネーミングサービスの機能	11-1
JNDI ツリーの表示	11-2
JNDI ツリーへのオブジェクトのロード	11-2

12. WebLogic J2EE コネクタ アーキテクチャの管理

WebLogic J2EE コネクタの概要	12-1
デプロイメント用のリソースアダプタ (コネクタ) のコンフィグレーション	12-2
接続プロファイルを表示するためのコネクタのコンフィグレーション	12-4
リソースアダプタ (コネクタ) のデプロイメント	12-4
デプロイされたリソースアダプタ (コネクタ) の表示	12-6
デプロイされたリソースアダプタ (コネクタ) のアンデプロイメント	12-6
デプロイされたリソースアダプタ (コネクタ) の更新	12-6
接続のモニタ	12-7
はじめに	12-7
リークされた接続の表示	12-8

アイドル接続の表示.....	12-9
接続の削除.....	12-10
コネクタの削除.....	12-11
リソースアダプタのデプロイメント記述子の編集.....	12-11

13. WebLogic Server ライセンスの管理

WebLogic Server ライセンスのインストール.....	13-1
ライセンスの更新.....	13-2

A. WebLogic Java ユーティリティの使い方

AppletArchiver.....	A-3
構文.....	A-3
CertGen.....	A-4
構文.....	A-4
例.....	A-4
ClientDeployer.....	A-5
Conversion.....	A-6
der2pem.....	A-7
構文.....	A-7
例.....	A-8
dbping.....	A-9
構文.....	A-9
例.....	A-10
Deployer.....	A-12
構文.....	A-12
アクション (以下のいずれかを選択).....	A-12
オプション.....	A-13
例.....	A-15
EJBGen.....	A-17
getProperty.....	A-18
構文.....	A-18
例.....	A-18
ImportPrivateKey.....	A-19
構文.....	A-19
例.....	A-19

logToZip.....	A-21
構文.....	A-21
例.....	A-21
MulticastTest.....	A-23
構文.....	A-23
例.....	A-24
myip.....	A-26
構文.....	A-26
例.....	A-26
pem2der.....	A-27
構文.....	A-27
例.....	A-27
Schema.....	A-28
構文.....	A-28
例.....	A-28
showLicenses.....	A-29
構文.....	A-29
例.....	A-29
system.....	A-30
構文.....	A-30
例.....	A-30
verboseToZip.....	A-31
構文.....	A-31
UNIX の例.....	A-31
NT の例.....	A-31
version.....	A-32
構文.....	A-32
例.....	A-32
writeLicense.....	A-33
構文.....	A-33
例.....	A-33

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 の管理コマンドの使い方	B-2
構文.....	B-2
接続とユーザ資格の引数.....	B-3
ユーザ資格引数の概要.....	B-4
ユーザ資格の指定例.....	B-6
WebLogic Server 管理コマンドのリファレンス	B-6
CANCEL_SHUTDOWN	B-9
構文.....	B-9
例.....	B-9
CONNECT	B-10
構文.....	B-10
例.....	B-10
FORCESHUTDOWN	B-11
構文.....	B-11
例.....	B-11
GETSTATE	B-13
構文.....	B-13
例.....	B-13
HELP	B-14
構文.....	B-14
例.....	B-14
LICENSES	B-15
構文.....	B-15
例.....	B-15
LIST	B-16
構文.....	B-16
例.....	B-16
LOCK	B-17
構文.....	B-17
例.....	B-17
MIGRATE	B-18
構文.....	B-18
例.....	B-19
PING	B-20

構文	B-20
例	B-20
RESUME	B-21
構文	B-21
例	B-21
SERVERLOG	B-22
構文	B-22
例	B-22
SHUTDOWN	B-24
構文	B-24
例	B-25
START	B-26
構文	B-26
例	B-27
STARTINSTANDBY	B-28
構文	B-28
例	B-29
STOREUSERCONFIG	B-30
構文	B-30
デフォルト パス名のコンフィグレーション	B-32
ユーザ コンフィグレーション ファイルとキー ファイルの作成	B-33
複数のユーザ コンフィグレーション ファイルに 1 つのキー ファイルを使用する	B-34
例	B-35
THREAD_DUMP	B-37
構文	B-37
例	B-37
UNLOCK	B-38
構文	B-38
例	B-38
VERSION	B-39
構文	B-39
例	B-39
WebLogic Server 接続プール管理コマンドリファレンス	B-40
CREATE_POOL	B-42

構文	B-42
例	B-44
DESTROY_POOL	B-45
構文	B-45
例	B-45
DISABLE_POOL	B-46
構文	B-46
例	B-46
ENABLE_POOL	B-47
構文	B-47
例	B-47
EXISTS_POOL	B-48
構文	B-48
例	B-48
RESET_POOL	B-49
構文	B-49
例	B-49
MBean 管理コマンドリファレンス	B-50
MBean タイプの指定	B-50
サーバの指定	B-51
CREATE	B-53
構文	B-53
例	B-54
DELETE	B-55
構文	B-55
例	B-55
GET	B-57
構文	B-57
例	B-58
INVOKE	B-59
構文	B-59
例	B-59
SET	B-61
構文	B-61
例	B-62

例 : JDBC 接続プールの割り当て	B-63
weblogic.Admin コマンドを使用したユーザとグループの管理	B-64
AuthenticationProvider MBean のオブジェクト名の検索	B-66
ユーザの作成	B-66
グループへのユーザの追加	B-67
ユーザがグループのメンバーであるかどうかの検証	B-67
ユーザが属するグループのリスト	B-68
LDAP サーバでのグループ メンバーシップ検索の制限	B-69

C. Ant タスクを使用した WebLogic Server ドメインの コンフィグレーション

Ant タスクを使用したドメインのコンフィグレーションと起動の概要	C-1
wlserver Ant タスクを使用したサーバの起動とドメインの作成	C-2
wlserver Ant タスクの機能	C-2
wlserver を使用する基本的な手順	C-3
wlserver のサンプル build.xml ファイル	C-4
wlserver Ant タスクのリファレンス	C-5
wlconfig Ant タスクを使用した WebLogic Server ドメインのコンフィグレーション	C-7
wlconfig Ant タスクの機能	C-7
wlconfig を使用する基本的な手順	C-8
wlconfig のサンプル build.xml ファイル	C-9
完全な例	C-9
クエリと削除の例	C-11
複数の属性値の設定例	C-11
wlconfig Ant タスクのリファレンス	C-12
主な属性	C-12
ネストされた要素	C-14

D. WebLogic SNMP エージェント コマンドライン リファ レンス

SNMP コマンドライン インタフェースに必要な環境および構文	D-2
環境	D-2
共通の引数	D-2
WebLogic Server 属性の値を取得するためのコマンド	D-4

snmpwalk	D-5
構文	D-5
例	D-5
snmpgetnext	D-7
構文	D-7
例	D-7
snmpget	D-10
構文	D-10
例	D-10
トラップをテストするためのコマンド	D-11
snmpv1trap	D-12
構文	D-12
例	D-14
snmptrapd	D-15
構文	D-15
例	D-15
例：トラップデーモンへのトラップの送信	D-15

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 システムの概要**」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「**システム管理操作の保護**」では、ロールを使用してシステム管理タスクへのアクセスを制限する方法について説明します。
- 第 4 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 5 章「**アプリケーションのデプロイメント**」では、**WebLogic Server** 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 6 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 7 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 8 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 9 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 10 章「**WebLogic メッセージングブリッジの使い方**」では、2 つの JMS プロバイダ間におけるストアおよび転送機能をコンフィグレーションする方法について説明します。

-
- 第 11 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 12 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 13 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「Ant タスクを使用した WebLogic Server ドメインのコンフィグレーション」では、WebLogic Server ドメインの管理に役立つ WebLogic Server Ant タスクの構文および用法について説明します。
 - 付録 D「WebLogic SNMP エージェント コマンドライン リファレンス」では、WebLogic SNMP エージェントのコマンドラインインタフェースを使用して、WebLogic Server 属性の値を取得したり、WebLogic Server トラップを生成および取得したりする方法について説明します。

対象読者

このマニュアルは主に、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を一度に 1 章ずつ印刷できます。

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

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

サポート情報

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

電子メールのメッセージには、ご使用のソフトウェアの名前とバージョン、およびドキュメントのタイトルと日付をお書き添えください。本バージョンの BEA WebLogic Server について不明な点がある場合、または BEA WebLogic Server のインストールおよび動作に問題がある場合は、BEA WebSupport (www.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]	複数のキーを同時に押すことを示す。
<i>斜体</i>	強調または書籍のタイトルを示す。
等幅テキスト	コード サンプル、コマンドとそのオプション、データ構造体とそのメンバー、データ型、ディレクトリ、およびファイル名とその拡張子を示す。等幅テキストはキーボードから入力するテキストも示す。 例： <pre>import java.util.Enumeration; chmod u+w * config/examples/applications .java config.xml float</pre>
<i>斜体の等幅テキスト</i>	コード内の変数を示す。 例： <pre>String <i>CustomerName</i>;</pre>

表記法	適用
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
{ }	構文の中で複数の選択肢を示す。
[]	構文の中で任意指定の項目を示す。 例： <pre>java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]</pre>
	構文の中で相互に排他的な選択肢を区切る。 例： <pre>java weblogic.deploy [list deploy undeploy update] password {application} {source}</pre>
...	コマンドラインで以下のいずれかを示す。 <ul style="list-style-type: none"> ■ 引数を複数回繰り返すことができる。 ■ 任意指定の引数が省略されている。 ■ パラメータや値などの情報を追加入力できる。
.	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 システムの概要

以下の節では、WebLogic Server のシステム管理の概要について説明します。

- 1-1 ページの「システム管理の概要」
- 1-2 ページの「WebLogic Server のドメイン」
- 1-4 ページの「システム管理のインフラストラクチャ」
- 1-6 ページの「管理サーバと管理対象サーバ」
- 1-8 ページの「システム管理ツール」
- 1-13 ページの「WebLogic Server ドメインで管理できるリソース」
- 1-22 ページの「Administration Console の起動と使い方」
- 1-28 ページの「Web サーバを伴う WebLogic Server の使用」
- 1-29 ページの「モニタ」
- 1-30 ページの「ライセンス」

システム管理の概要

WebLogic Server のシステム管理ツールを使用すると、1 つまたは複数の WebLogic Server インストールをインストール、コンフィグレーション、モニタおよび管理できます。また、ツールを使用して、WebLogic Server にホストされているアプリケーションを管理したりモニタしたりできます。WebLogic Server インストールは、単一の WebLogic Server インスタンスまたは複数のインスタンスで構成され、各インスタンスは 1 つまたは複数の物理的なマシン上にホストされます。

Administration Console、コマンドラインユーティリティ、および API などのシステム管理ツールを使用すると、セキュリティ、データベース接続、メッセージング、およびトランザクション処理などのサービスを管理できます。ツールには、アプリケーションの最大の可用性を確保するために、WebLogic Server 環境の状態をモニタする機能も用意されています。

WebLogic Server のドメイン

WebLogic Server の基本的な管理単位を「ドメイン」と呼びます。ドメインは、論理的に関連付けられた WebLogic Server リソースの集合です。管理サーバとしてコンフィグレーションされた WebLogic Server インスタンスでは、ドメインを 1 つの単位として管理します。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。クラスタは WebLogic Server のグループで、連携して動作することにより、アプリケーションにスケーラビリティと高可用性を提供します。アプリケーションもドメインの一部としてデプロイおよび管理されます。

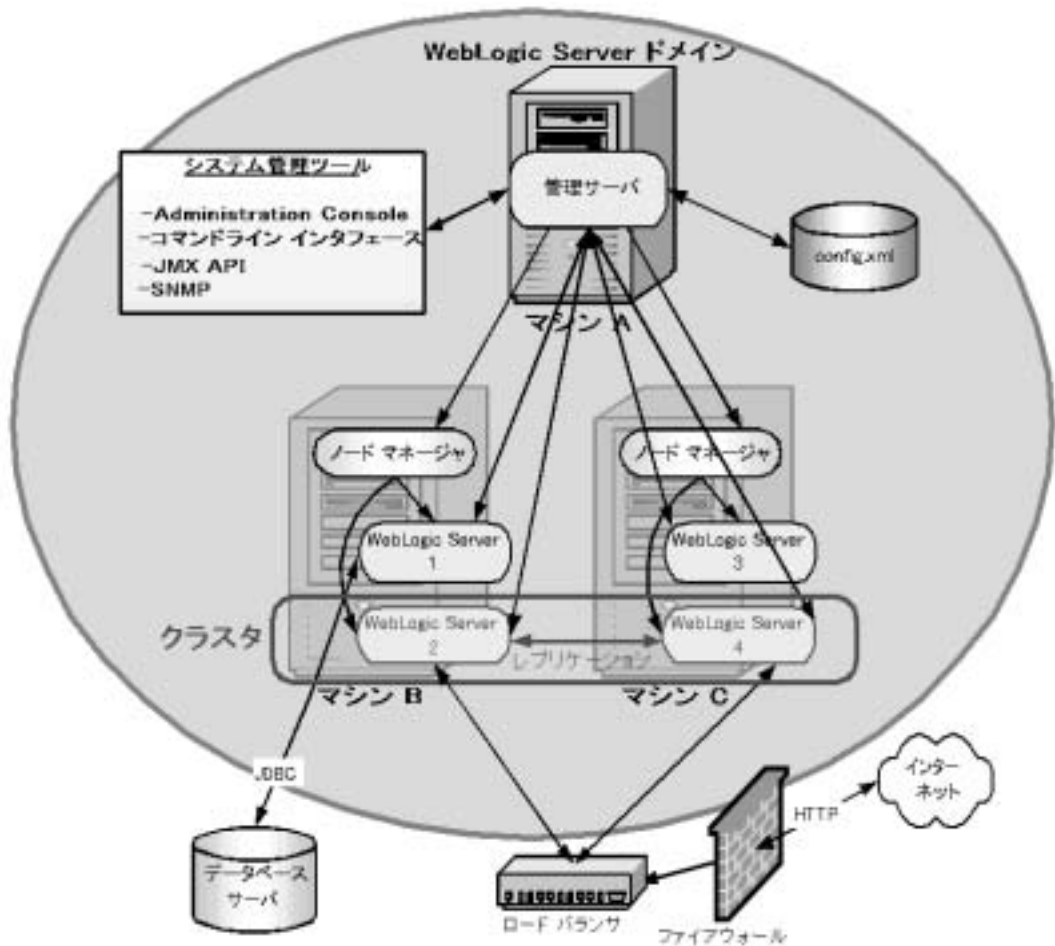
ドメインは、以下のような条件に基づいて編成できます。

- アプリケーションの論理的な区分。たとえば、ショッピング カートのようなエンドユーザ機能専用のドメインと、バックエンドの会計アプリケーション専用のドメインを持つことができます。
- 物理的な場所。企業の所在地や支店別にドメインを作成できます。
- サイズ。ドメインをより効率的に管理したり、別々のシステム管理者が管理したりできるよう、より小さな単位に分割できます。

注意： ドメイン内のすべての WebLogic Server インスタンスでは、同じバージョンの WebLogic Server ソフトウェアを実行する必要があります。管理サーバは、そのドメイン内の管理対象サーバにインストールされているものと同じかそれ以降のサービスパックを適用している必要があります。たとえば、管理対象サーバがサービスパック 1 を適用していないバージョン 7.0 を実行している場合に、管理サーバはバージョン 7.0 サービスパック 1 を実行できます。

ドメインの詳細については、『WebLogic Server ドメイン管理』を参照してください。

図 1-1 WebLogic Server ドメイン



WebLogic ドメインの概念を説明するため、WebLogic Server のコンフィギュレーションの例を図 1-1 に示します。

このドメインには、3つの物理的なマシンがあります。

マシン A は管理サーバとして表され、WebLogic Server の 1つのインスタンスをホストしています。システム管理ツールは管理サーバと通信して、サーバおよびドメイン内のアプリケーションのコンフィギュレーションとモニタを実行します。管理サーバは、システム管理ツールに代わって各管理対象サーバと通信します。

ドメイン内のすべてのサーバの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である `config.xml` に格納されます。このファイルは管理サーバをホストするマシン上に存在します。

マシン B および C は、WebLogic Server のインスタンスをそれぞれ 2 つずつホストしています (WebLogic Server 1 ~ 4)。これらのインスタンスは管理対象サーバと呼ばれます。管理サーバは各マシン上で実行されるノード マネージャのインスタンスと通信して、管理対象サーバの起動と停止を制御します。

WebLogic Server 2 および 4 は WebLogic クラスタ (赤で描かれた部分) の一部です。このクラスタでは、ハードウェアのロード バランサからクラスタヘルディングされる HTTP リクエストに応答するアプリケーションを実行します。ロード バランシングは、WebLogic Server のインスタンスを使用して提供することもできます。ロード バランサは、インターネットからの HTTP リクエストをファイアウォールの通過後に処理します。ロード バランサとファイアウォールはドメインには含まれません。HTTP セッションなどのオブジェクトがレプリケートされ、そのコピーを 2 つの クラスタ メンバー間で受け渡すことにより、フェイルオーバー機能が実現します。

WebLogic Server 1 は、JDBC を使用するアプリケーションを実行して、WebLogic ドメインに含まれていない他の物理的マシン上で稼動するデータベースサーバにアクセスします。

注意： 図のドメインは、WebLogic Server ドメインの概念とドメインの管理方法を示すことのみを目的としたものです。WebLogic Server ドメインでは、サーバ、クラスタ、およびアプリケーションから成るさまざまなコンフィグレーションが可能です。

システム管理のインフラストラクチャ

WebLogic Server のシステム管理インフラストラクチャは、Sun Microsystems の Java Management Extension (JMX) 仕様を使用して実装されています。JMX API では、MBean と呼ばれる Java オブジェクトを使用してシステム管理機能をモデル化しています。この節ではシステム管理のインフラストラクチャについて説明しますが、ここで示される実装の知識は、WebLogic Server ドメインの管理に必要ではありません。

WebLogic ドメインの管理に使用される MBean には、管理 MBean、コンフィグレーション MBean、および実行時 MBean の 3 種類があります。

管理 MBean には、さまざまな管理機能のコンフィグレーション パラメータを定義する属性のセットが含まれています。管理 MBean のすべての属性には、あらかじめデフォルト値が設定されています。管理サーバは起動時に `config.xml` というファイルを読み込み、管理 MBean のデフォルトの属性値を、`config.xml` ファイル内の属性値でオーバーライドします。

`config.xml` ファイルは管理サーバをホストするマシン上にあり、MBean 属性値の永続ストレージを提供します。システム管理ツールを使用して属性を変更するたびに、その値が適切な管理 MBean に格納され、`config.xml` ファイルに書き込まれます。各 WebLogic Server ドメインには、独自の `config.xml` ファイルがあります。

管理サーバの起動時に、`-D` 引数を使用してコマンドラインでコンフィグレーション属性を設定すると、その値は、デフォルトで設定された値または `config.xml` ファイルの値をオーバーライドします。オーバーライドされた値は、管理サーバによって `config.xml` ファイルに保持されます。コマンドライン引数の詳細については、2-18 ページの「`weblogic.Server` コマンドの使用」を参照してください。

コンフィグレーション MBean は管理 MBean のコピーであり、各管理対象サーバでコンフィグレーションに使用します。管理対象サーバは、起動時に管理サーバからすべての管理 MBean のコピーを受け取り、それらをコンフィグレーション MBean としてメモリに格納します。管理対象サーバの起動時にコンフィグレーション属性をオーバーライドした場合、それらの値は管理サーバから受け取った値をオーバーライドしますが、`config.xml` ファイルには書き込まれません。管理対象サーバの起動については、2-34 ページの「管理対象サーバの起動」を参照してください。

実行時 MBean には、アクティブな WebLogic Server インスタンスとアプリケーションの実行時情報で構成される属性のセットが含まれています。実行時 MBean から属性値を取得することにより、WebLogic Server ドメインの実行時の状況をモニタできます。

MBean には、管理機能の実行に使用される操作も含まれています。

このような MBean や JMX API の知識があると、ユーザはカスタマイズされた管理システムを作成することができますが、ほとんどのユーザは、**WebLogic Server** で提供されるシステム管理ツールを使用して管理タスクを実行できます。これらのツールでは JMX API の知識は必要ありません。詳細については、1-8 ページの「システム管理ツール」を参照してください。

管理サーバと管理対象サーバ

各ドメイン内の **WebLogic Server** インスタンスの 1 つは、管理サーバとしてコンフィグレーションされます。管理サーバでは **WebLogic Server** ドメインを一元的に管理できます。ドメイン内のその他の **WebLogic Server** インスタンスはすべて管理対象サーバと呼ばれます。1 つの **WebLogic Server** インスタンスしか含まれないドメインでは、そのサーバが管理サーバおよび管理対象サーバとして機能します。

通常のプロダクション システムでは、アプリケーションを管理対象サーバにのみデプロイすることをお勧めします。これにより、管理サーバを、ドメインのコンフィグレーションおよびモニタ専用にすることができます。

詳細については、2-1 ページの「**WebLogic Server** の起動と停止」を参照してください。

管理サーバのフェイルオーバー

管理サーバのシングルポイント障害を回避するため、管理対象サーバは、管理サーバが存在しなくても常に機能することができます。ただし、管理サーバではドメインを管理およびモニタする必要があります。ドメインの `config.xml` ファイルや他のリソースのバックアップを保持しておく、障害が発生した管理サーバをバックアップの **WebLogic Server** インスタンスに置き換えて、管理サーバの役目を引き継がせることができます。詳細については、2-13 ページの「管理サーバの起動」および「障害が発生したサーバの回復」を参照してください。

管理対象サーバのフェイルオーバー

管理対象サーバは起動時に管理サーバにアクセスして、コンフィグレーション情報を取得します。管理対象サーバは、指定された管理サーバに起動時にアクセスできない場合、管理対象サーバのファイルシステムに格納されているコンフィグレーションファイルおよびその他のファイルを直接読み込むことによってコンフィグレーション情報を取得します。

この方法で起動した管理対象サーバは、「管理対象サーバ独立モード」で実行されます。このモードでは、サーバはキャッシュされたアプリケーションファイルを使用して、サーバに割り当てられたアプリケーションをデプロイします。管理サーバとの通信が回復するまで、管理対象サーバのコンフィグレーションを変更することはできません。詳細については、「障害が発生したサーバの回復」を参照してください。

ドメイン全体の管理ポート

ドメイン内のサーバで使用する管理ポートを有効にできます。管理ポートは省略可能ですが、2つの重要な機能を提供します。

- サーバをスタンバイ状態で起動できる。管理ポートは、スタンバイ状態でもサーバのアクティブ化や管理に利用できます。ただし、サーバの他のネットワーク接続はクライアント接続を受け付けることができません。スタンバイ状態については、「WebLogic Server の起動と停止」を参照してください。
- ドメイン内で管理トラフィックとアプリケーショントラフィックを分離できる。プロダクション環境では、2つのトラフィックを分離すると、同じネットワーク接続上に大量のアプリケーショントラフィックがある状態で重要な管理操作（サーバの起動と停止、サーバのコンフィグレーションの変更、およびアプリケーションのデプロイ）を行う、ということがなくなります。

詳細については、『WebLogic Server ドメイン管理』の「ドメイン全体の管理ポートのコンフィグレーション」を参照してください。

サービス パックと WebLogic Server インスタンス

ドメイン内のすべての WebLogic Server インスタンスでは、同じバージョンの WebLogic Server ソフトウェアを実行する必要があります。管理サーバは、そのドメイン内の管理対象サーバにインストールされているものと同じかそれ以降のサービスパックを適用している必要があります。たとえば、管理対象サーバがサービスパック 1 を適用していないバージョン 7.0 を実行している場合に、管理サーバはバージョン 7.0 サービスパック 1 を実行できます。

システム管理ツール

基底のアーキテクチャとして JMX を使用して、さまざまな管理機能に対応したシステム管理ツールが用意されています。システム管理ツールを使用してドメインを管理するときは、管理サーバが動作している必要があります。以降の節で、それぞれのツールについて説明します。

システム管理ツールのセキュリティ保護

すべてのシステム管理操作は、システム管理ツールへのアクセスに使用したユーザ名に基づいて保護されます。ユーザ（またはユーザが属するグループ）は、4 つのセキュリティ ロールのいずれかのメンバーである必要があります。ロールでは、ユーザに対して、さまざまなシステム管理操作へのアクセスを付与または拒否します。ロールには、Admin、Operator、Deployer、および Monitor があります。ドメイン内の WebLogic Server に対してセキュリティ ポリシーを設定することもできます。詳細については、3-1 ページの「システム管理操作の保護」を参照してください。

システム Administration Console

Administration Console は、管理サーバにホストされる JSP (Java ServerPage) ベースのアプリケーションです。Administration Console には、管理サーバと通信できるローカル ネットワーク上のマシンの Web ブラウザを使用してアクセスでき

ます(管理サーバと同じマシン上で動作するブラウザも含まれます)。
Administration Console を使用すると、複数の **WebLogic Server** インスタンス、クラスタ、およびアプリケーションを含む **WebLogic Server** ドメインを管理できます。以下のような管理機能があります。

- コンフィグレーション
- サーバの起動と停止
- サーバの状態とパフォーマンスのモニタ
- アプリケーションのパフォーマンスのモニタ
- サーバ ログの表示
- **Web** アプリケーション、**EJB**、**J2EE** コネクタ、およびエンタープライズアプリケーションのデプロイメント記述子の編集

Administration Console を使用すると、システム管理者は、**JMX API** や基底の管理アーキテクチャについて理解していなくても、**WebLogic Server** のすべての管理タスクを簡単に実行できます。管理サーバでは、属性の変更内容を管理対象のドメインの `config.xml` ファイルに保持します。

詳細については、以下を参照してください。

- 1-22 ページの「**Administration Console** の起動と使い方」
- **Administration Console** オンライン ヘルプ (オンライン ヘルプには、**Administration Console** で [?] アイコンをクリックしてアクセスすることもできます。)

コマンドライン インタフェース

コマンドライン インタフェースは、**WebLogic Server** ドメインを **Administration Console** で管理するのが実際的でない場合に使用します。たとえば、スクリプトを使用してドメインを管理したい場合、**Administration Console** へのアクセスに **Web** ブラウザを使用できない場合、**GUI** よりもコマンドライン インタフェースの方が作業しやすい場合、などが考えられます。ドメインは、コマンドライン

インタフェースのみでも管理できますが、**Administration Console**などのシステム管理ツールとコマンドライン インタフェースの両方を使用して管理することも可能です。

コマンドライン インタフェースでは `weblogic.Admin` という Java クラスを呼び出します。このクラスの引数を利用すると、**JMX API** について理解していなくても、多くの一般的な管理機能を実行できるようになります。詳細については、以下を参照してください。

- B-6 ページの「**WebLogic Server 管理コマンドのリファレンス**」
- **WebLogic Server AIP リファレンス (Javadoc)** (`weblogic.management` パッケージを参照)

`weblogic.Admin` 管理機能より細かい制御が必要な場合は、コマンドライン インタフェースを使用して **MBean** 属性に対し直接 `set` または `get` 操作を実行できます。この機能には **WebLogic Server MBean** アーキテクチャの知識が必要です。詳細については、以下のリソースを参照してください。

- B-50 ページの「**MBean 管理コマンド リファレンス**」では、コマンドライン インタフェースの使い方について説明しています。
- **WebLogic Server クラスの Javadoc**
 - **コンフィグレーション MBean (WebLogic ドメインのコンフィグレーション用)** については、`weblogic.management.configuration` パッケージを選択します。
 - **実行時 MBean (モニタ用)** については、`weblogic.management.runtime` パッケージを選択します。
- **MBean** のリファレンスは、『**コンフィグレーション リファレンス**』で提供されています。このリファレンスは `config.xml` ファイル内の **MBean** を表す要素と関連しています。

JMX

Sun Microsystems Inc. の **JMX API** および **WebLogic Server MBean** の知識を持つ高度なプログラマは、独自の管理コンポーネントを Java クラスとして記述することができます。

詳細については、以下を参照してください。

- 『WebLogic JMX Service プログラマーズ ガイド』
- WebLogic Server AIP リファレンス (Javadoc) ((weblogic.management パッケージを参照)

コンフィグレーション ウィザード

コンフィグレーション ウィザードを使用すると、新しい WebLogic Server ドメインを作成できます。このツールでは、複数のスタンドアロン サーバ、管理サーバと管理対象サーバ、クラスタ化されたサーバなどのドメイン コンフィグレーションを作成できます。コンフィグレーション ウィザードでは、ドメインの適切なディレクトリ構造、基本的な config.xml ファイル、ドメイン内のサーバの起動に使用できるスクリプトを作成します。

コンフィグレーション ウィザードは、グラフィカル ユーザ インタフェース (GUI) またはテキストベースのコマンドライン環境のいずれを使用しても実行できます。コンフィグレーション ウィザードによって、ユーザ定義のドメイン コンフィグレーション テンプレートを作成することもできます。

詳細については、『WebLogic Server ドメイン管理』の「コンフィグレーション ウィザードを使用した新しいドメインの作成」を参照してください。

Java ユーティリティ

アプリケーションのデプロイメントや DBMS コンフィグレーションのテストのような一般的なタスクを目的としたユーティリティプログラムが提供されています。詳細については、A-1 ページの「WebLogic Java ユーティリティの使い方」を参照してください。

ノード マネージャ

ノード マネージャは WebLogic Server に付属する Java プログラムです。このプログラムを使用すると、リモートの WebLogic Server インスタンスを起動、停止、再起動およびモニタできます。この機能を有効にするには、ドメイン内の物理的な各マシン上でノード マネージャを実行します。

詳細については、「ノード マネージャによるサーバの可用性の管理」を参照してください。

SNMP

WebLogic Server には、Simple Network Management Protocol (SNMP) を使用してエンタープライズ全体の管理システムと通信する機能があります。WebLogic Server SNMP 機能によって WebLogic Server の管理を SNMP に準拠した管理システムに統合し、複雑に分散したシステムのさまざまなソフトウェアとハードウェアのリソースをまとめて表示することができます。

詳細については、以下を参照してください。

- 『SNMP 管理ガイド』
- 『SNMP MIB リファレンス』

ログ

WebLogic Server の多くの操作では、そのアクティビティのログが生成されます。各サーバには、標準の HTTP アクセス ログの他に、独自のログがあります。これらのログ ファイルをさまざまな方法でコンフィグレーションおよび使用して、サーバやアプリケーションの状態とアクティビティをモニタできます。

詳細については、以下を参照してください。

- 4-1 ページの「ログ メッセージを使用した WebLogic Server の管理」
- 6-14 ページの「HTTP アクセス ログの設定」

■ 『WebLogic Server ログイン サービスの使い方』

ドメイン内のすべての WebLogic Server インスタンスから送られるログ メッセージの限定したサブセットが含まれるように、特別なドメイン ログをコンフィグレーションすることもできます。システム管理ツールを使用して、ローカルサーバのどのログ メッセージをドメイン ログに表示するかを変更できます。ドメイン ログは、Administration Console またはテキスト エディタかテキストビューアを使用して表示できます。

詳細については、「ドメイン ログ フィルタ」を参照してください。

config.xml の編集

コンフィグレーションの永続ストアである config.xml ファイルを手動で編集して、WebLogic Server ドメインを管理することもできます。その他のシステム管理ツールでは、コンフィグレーションが config.xml ファイルに自動的に保存されます。このファイルに必要な XML 構文を正しく編集するのは難しいため、この方法によるコンフィグレーションはお勧めしませんが、状況によっては利点があります。

注意： 管理サーバの実行中に config.xml ファイルを編集しないでください。

詳細については、『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。

WebLogic Server ドメインで管理できるリソース

この節では、システム管理ツールを使用して管理できるドメインのリソースについて説明します。

サーバ

サーバという管理概念は、ドメイン内の **WebLogic Server** インスタンスを表します。システム管理ツールを使用して以下のことができます。

- サーバを起動および停止する（リモート マシン上のサーバを起動および停止するには、リモート マシンにノード マネージャがインストールされている必要があります）。詳細については、1-12 ページの「ノード マネージャ」を参照してください。
- サーバの接続（ポート、HTTP 設定、jCOM 設定、およびタイムアウト）をコンフィグレーションする。
- HTTP サーバ機能と仮想ホストをコンフィグレーションする。
- ロギングとログの表示をコンフィグレーションする。
- アプリケーションを特定のサーバにデプロイする。
- JDBC 接続プールや起動クラスなど、サーバでアクティブな **WebLogic** サーバ リソースをコンフィグレーションする。

クラスタ

WebLogic Server クラスタを使用すると、アプリケーションの作業負荷を複数の **WebLogic Server** に分散できます。クラスタによって、パフォーマンスを向上させ、サーバ インスタンスが使用できなくなった場合にフェイルオーバーを提供することができます。たとえば、クラスタには、アプリケーションで使用されるオブジェクトをレプリケートする複数の方法があるため、ハードウェアに障害が発生してもデータが失われることはありません。

アプリケーションに最高のパフォーマンスが提供される方法で作業負荷を分散するために、複数のクラスタを組み合わせることで構築できます。

WebLogic Server の単一のインスタンスにホストされるサービスには、サーバの障害発生時に別のサーバへ移行できるものがあります。システム管理ツールでは、これらの移行を制御できます。

詳細については、『**WebLogic Server** クラスタ ユーザーズ ガイド』を参照してください。

マシン

マシンという管理概念は、WebLogic Server インスタンスをホストする物理的なマシンを表します。WebLogic Server は、マシン名を使用して、HTTP セッションレプリケーションなどのタスクを委託するのに最適なクラスタ内のサーバを決定します。

システム管理ツールを使用して、1 つまたは複数のマシンを定義し、それらのマシン用にノード マネージャをコンフィグレーションしたり、マシンにサーバを割り当てたりできます。UNIX マシンの場合は、UID および GID 情報をコンフィグレーションできます。

詳細については、『WebLogic Server クラスタ ユーザーズ ガイド』を参照してください。

ネットワーク チャネル

ネットワーク チャネルはオプションの機能です。この機能を使用すると、1 つまたは複数の WebLogic Server インスタンスまたはクラスタで、追加のポートをコンフィグレーションできます。ネットワーク チャネルを使用するすべてのサーバとクラスタは、チャネルの基本的なポート コンフィグレーションを継承します。チャネルの詳細チューニングを使用して、個々のサーバに対するチャネルのポート設定をカスタマイズすることもできます。詳細チューニング処理では、ネットワーク アクセス ポイントと呼ばれる追加のネットワーク リソースが作成されます。

詳細については、「ネットワーク リソースのコンフィグレーション」を参照してください。

JDBC

Java Database Connectivity (JDBC) を使用すると、Java プログラムで Oracle、Microsoft SQL Server、Sybase のような一般的な DBMS と対話できます。

システム管理ツールを使用して、**WebLogic Server** とデータベース管理システムとの接続を管理およびモニタできます。通常、接続は接続プールを使用して確立されます。

詳細については、8-1 ページの「**JDBC 接続の管理**」を参照してください。

JMS

Java Message Service (JMS) は、アプリケーション間の通信を実現するエンタープライズ 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。

システム管理ツールを使用して、以下のコンフィグレーション属性を定義できません。

- **JMS** の有効化
- **JMS** サーバの作成
- **JMS** サーバ、接続ファクトリ、送り先 (物理的なキューとトピック)、分散送り先 (クラスタ内の物理的なキューおよびトピック メンバーの集合)、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、ページング ストア、セッション プール、および接続コンシューマの作成またはそれらの値のカスタマイズ
- カスタム **JMS** アプリケーションの設定
- しきい値と割り当ての定義
- 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、フロー制御、分散送り先のロード バランシングなど、必要な **JMS** 機能の有効化

詳細については、9-1 ページの「**JMS の管理**」を参照してください。

WebLogic メッセージングブリッジ

メッセージングブリッジは、2つのメッセージングプロバイダ間でメッセージを転送します。プロバイダは **WebLogic JMS** の別の実装である場合と、サードパーティ **JMS** プロバイダの場合があります。

詳細については、10-1 ページの「**WebLogic メッセージングブリッジの使い方**」を参照してください。

Web サーバと Web コンポーネント

WebLogic Server は高機能な Web サーバとして実行できます。WebLogic Server は、HTML ファイルなどの静的ファイルと、Java サーブレットや JavaServer Pages (JSP) などの動的ファイルの両方を提供します。仮想ホスティングもサポートされています。

WebLogic Server で Web サーバ機能を管理する方法については、6-1 ページの「**WebLogic Server Web コンポーネントのコンフィギュレーション**」を参照してください。

アプリケーション

Administration Console などのアプリケーションデプロイメントツールを使用すると、アプリケーションをデプロイ、管理、更新、およびモニタできます。アプリケーションデプロイメントツールでは、WebLogic Server のクラスタ内のアプリケーションをデプロイおよび更新することもできます。

WebLogic Server 7.0 は、デプロイメント処理をより自在に制御できる新しい 2 フェーズデプロイメントモデルを備えています。詳細については、「**WebLogic Server デプロイメント**」を参照してください。

システム管理ツールを使用して以下のことができます。

- アプリケーションをドメイン内の 1 つまたは複数の WebLogic Server またはクラスタにデプロイする

- アプリケーションの実行時パラメータをコンフィグレーションする
- アプリケーションのパフォーマンスをモニタする
- セキュリティパラメータをコンフィグレーションする
- セキュリティロールまたはセキュリティポリシーに基づいてアプリケーションへのアクセスを保護する。詳細については、「WebLogic リソースの保護の設定」を参照してください。

アプリケーションの形式

アプリケーションは、以下の J2EE アプリケーションの形式を 1 つまたは複数使用してデプロイします。

- Web アプリケーション
- エンタープライズ JavaBeans (EJB)
- エンタープライズ アプリケーション
- J2EE コネクタ
- Web サービス。Web サービスは、Web サービスをコンフィグレーションするための特別なデプロイメント記述子を含む Web アプリケーションとしてデプロイされます。

詳細については、以下を参照してください。

- 「WebLogic Server デプロイメント」
- 「WebLogic Server アプリケーションのパッケージ化」
- 『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』
- 5-1 ページの「アプリケーションのデプロイメント」
- 『WebLogic Server アプリケーションの開発』
- 『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』
- 『WebLogic J2EE コネクタ』
- 『WebLogic Web サービス プログラマーズ ガイド』

- [ポリシーを定義]
- 「WebLogic リソースの保護の設定」

Administration Console を使用したデプロイメント記述子の編集

Administration Console を使用して、J2EE アプリケーションのデプロイメント記述子を編集できます。詳細については、以下を参照してください。

- 「アプリケーション デプロイメント記述子エディタ (EAR)」
- 「リソースアダプタ (コネクタ) デプロイメント記述子エディタ」
- 「EJB デプロイメント記述子エディタ」
- 「Web アプリケーション デプロイメント記述子エディタ (war)」

WebLogic Builder ツールを使用したデプロイメント記述子の編集および作成

デプロイメント記述子を編集するには、Administration Console を使用する以外に、WebLogic Server 配布キットに含まれる、より強力な WebLogic Builder ツールを使用することもできます。WebLogic Builder は、J2EE アプリケーションのアセンブル、デプロイメント記述子の作成および編集、WebLogic Server へのアプリケーションのデプロイメントを行うための、スタンドアロンのグラフィカルツールです。詳細については、『WebLogic Builder Online Help』を参照してください。

起動クラスと停止クラス

起動クラスは、WebLogic Server が起動または再起動されるときに、他のサーバ初期化タスクが完了した後で自動的にロードされて実行される Java プログラムです。停止クラスは、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。

システム管理ツールを使用して、起動クラスと停止クラスを登録および管理できます。

詳細については、2-1 ページの「WebLogic Server の起動と停止」を参照してください。

JNDI

JNDI (Java Naming and Directory Interface) 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などを名前を検索できます。Administration Console を使用して JNDI ツリーを参照できます。

詳細については、以下を参照してください。

- 11-1 ページの「JNDI の管理」
- 『WebLogic JNDI プログラマーズ ガイド』

トランザクション

システム管理ツールを使用して、WebLogic Server Java Transaction API (JTA) をコンフィグレーションおよび有効化します。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、以下のもの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ムアウトと制限
- トランザクション マネージャの動作

詳細については、以下を参照してください。

- 7-1 ページの「トランザクションの管理」
- 『WebLogic JTA プログラマーズ ガイド』

XML

XMLレジストリは WebLogic Server インスタンスの XML リソースをコンフィグレーションおよび管理するための機能です。WebLogic Server の XML リソースには、アプリケーションで XML データの解析に使用するパーサ、アプリケーションで XML データの変換に使用するトランスフォーマ、外部エンティティの解決、および外部エンティティのキャッシングがあります。

詳細については、「WebLogic Server XML の管理」を参照してください。

セキュリティ

セキュリティ機能は WebLogic Server バージョン 7.0 で全面的に改訂されました。新しいセキュリティシステムを使用すると、サードパーティのセキュリティソリューションを組み込んだり、さまざまな一般のセキュリティシステムに対して独自の実装を提供したりできます。独自のセキュリティソリューションを作成して WebLogic Server に実装することもできます。

下位互換性のために、互換性モードで実行すると、WebLogic Server バージョン 6.0 および 6.1 で使用できるセキュリティ機能もサポートされます。

管理ツールを使用して、レルム、ユーザ、グループ、パスワード、ACL、およびその他のセキュリティ機能を定義できます。

詳細については、以下を参照してください。

- 『WebLogic Security の管理』
- 『WebLogic Security の管理』の「互換性セキュリティの使い方」
- Administration Console オンライン ヘルプの「セキュリティ」
- セキュリティのインデックス ページ

WebLogic Tuxedo Connector

WebLogic Tuxedo Connector は WebLogic Server アプリケーションと Tuxedo サービスとの相互運用性を提供します。このコネクタを使用すると、サービス要求に応じて、WebLogic Server クライアントでは Tuxedo サービスを呼び出し、Tuxedo クライアントでは WebLogic Server のエンタープライズ JavaBean (EJB) を呼び出すことができます。

詳細については、「WebLogic Tuxedo Connector」を参照してください。

Jolt

Jolt は、Tuxedo サーバ上で動作する Jolt サービス リスナ (JSL) を経由して BEA Tuxedo サービスへの要求を管理する Java ベースのクライアント API です。

詳細については、「BEA Jolt」を参照してください。

メール

WebLogic Server には Sun Microsystems の JavaMail API バージョン 1.1.3 参照実装が含まれています。

詳細については、「プログラミング トピック」の「WebLogic Server アプリケーションでの JavaMail の使い方」を参照してください。

Administration Console の起動と使い方

この節では、Administration Console の起動と使い方について説明します。

Administration Console がサポートされているブラウザ

Administration Console を起動するには、以下のいずれかの Web ブラウザを使用します。

- Microsoft Internet Explorer バージョン 5 (Windows)
- Microsoft Internet Explorer バージョン 6 (Windows)
- Netscape バージョン 4.7 (Windows または SunOS)
- Netscape バージョン 6 (Windows または SunOS)

上記以外の Web ブラウザを使用すると、機能やフォーマットの面で問題が生じるおそれがあります。

Administration Console の起動

1. WebLogic 管理サーバを起動します。詳細については、2-13 ページの「管理サーバの起動」を参照してください。
2. 上記の Web ブラウザのいずれかを起動し、次の URL を開きます。

```
http://hostname:port/console
```

hostname は管理サーバの DNS 名または IP アドレス、*port* は管理サーバで要求がリスンされるポートのアドレス (デフォルトは 7001) です。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
```

システム管理のための SSL の設定については、「[サーバ] --> [接続] --> [SSL ポート]」を参照してください。

3. ログイン ページが表示されたら、管理サーバを起動するために使用したユーザ名とパスワードを入力します (このユーザ名とパスワードはインストールプロセス中に指定したものです)。または、Administrators、Operators、Deployers、または Monitors のいずれかのセキュリティ グループに属する

ユーザ名を入力します。このグループは **Administration Console** のシステム管理機能に対するさまざまなアクセス レベルを提供します。詳細については、3-1 ページの「システム管理操作の保護」を参照してください。

セキュリティ システムを使用すると、これらのグループにユーザを追加または削除して、コンソールへのアクセスを制御できます。詳細については、3-1 ページの「システム管理操作の保護」を参照してください。

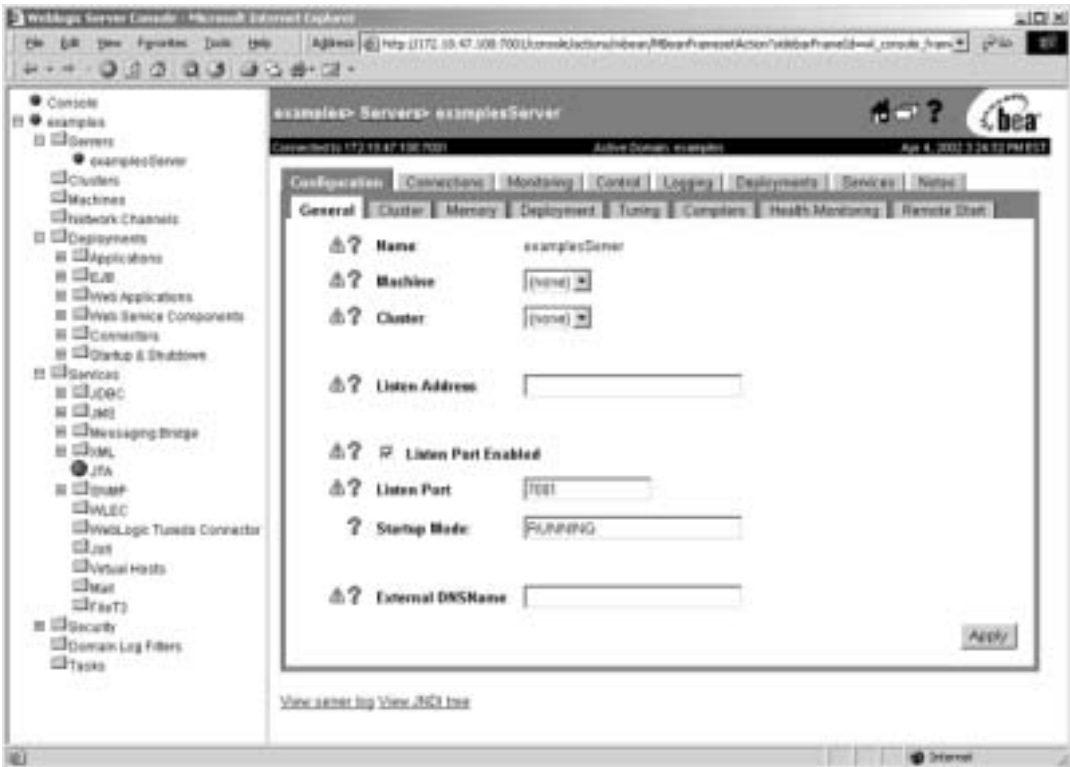
注意： ブラウザが HTTP リクエストをプロキシサーバに送信するようコンフィグレーションしてある場合、管理サーバの HTTP リクエストをプロキシに送信しないように、ブラウザをコンフィグレーションする必要があります。管理サーバがブラウザと同じマシン上にある場合は、localhost または 127.0.0.1 に送信されるリクエストがプロキシに送信されないようにする必要があります。

Administration Console の使い方

この節では、**Administration Console** を使用して **WebLogic Server** ドメインを管理およびモニタする方法について説明します。

Administration Console 内の移動

図 1-2 Administration Console



Administration Console の左ペインには、データのテーブル、コンフィグレーション ページ、モニタ ページ、またはログ ファイルへの移動に使用するナビゲーション ツリーが表示されます。ドメイン ツリーのノードを選択 (左クリック) すると、リソースのデータ テーブル、または選択したリソースのコンフィグレーション ページとモニタ ページを表示できます。ツリーのノードの先頭に正符号が付いている場合、正符号をクリックしてツリーを展開すると、別のリソースにアクセスできます。

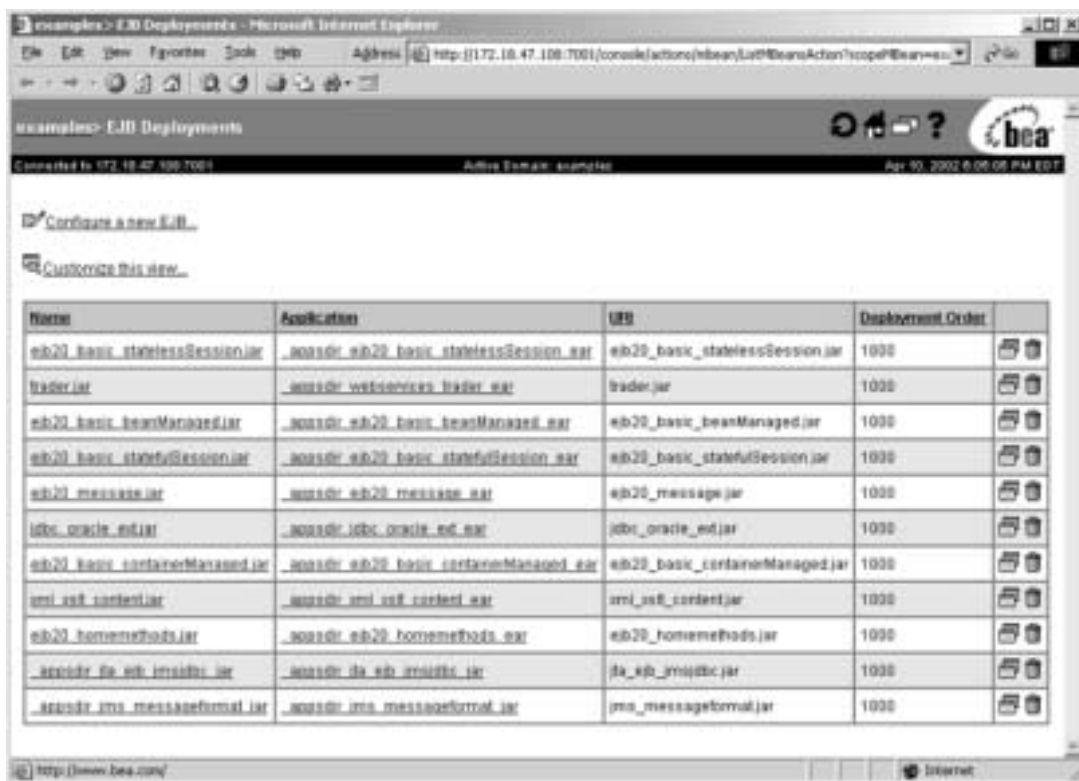
ノードを右クリックして、さまざまな操作を利用することもできます。

1 WebLogic システムの概要

ナビゲーション ツリーでノードを選択すると、右ペインにはコンフィグレーション済みのリソースやオブジェクトのテーブル形式のリスト、またはタブ付きのインタフェースが表示されます。

表示されたデータが、特定の種類のリソースやオブジェクトのデータ テーブルである場合、カラムを追加または削除してテーブルをカスタマイズできます。カラム ヘッドをクリックしてデータ テーブルをソートすることもできます。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。

図 1-3 Administration Console のテーブル ページ




The screenshot shows the Administration Console interface in a browser window. The main content area displays a table with the following columns: Name, Application, URL, and Deployment Order. The table lists various deployed applications, including session beans, message beans, and other services.

Name	Application	URL	Deployment Order
eb20_basic_statelessSession.jar	_appsrcr_eb20_basic_statelessSession_war	eb20_basic_statelessSession.jar	1000
trader.jar	_appsrcr_welcomepages_trader_war	trader.jar	1000
eb20_basic_beanManaged.jar	_appsrcr_eb20_basic_beanManaged_war	eb20_basic_beanManaged.jar	1000
eb20_basic_statefulSession.jar	_appsrcr_eb20_basic_statefulSession_war	eb20_basic_statefulSession.jar	1000
eb20_message.jar	_appsrcr_eb20_message_war	eb20_message.jar	1000
jdbc_oracle_md.jar	_appsrcr_jdbc_oracle_md_war	jdbc_oracle_md.jar	1000
eb20_basic_containerManaged.jar	_appsrcr_eb20_basic_containerManaged_war	eb20_basic_containerManaged.jar	1000
xml_xsl_content.jar	_appsrcr_xml_xsl_content_war	xml_xsl_content.jar	1000
eb20_homeMethods.jar	_appsrcr_eb20_homeMethods_war	eb20_homeMethods.jar	1000
_appsrcr_its_eb_inject.jar	_appsrcr_its_eb_inject.jar	its_eb_inject.jar	1000
_appsrcr_its_messageformat.jar	_appsrcr_its_messageformat.jar	its_messageformat.jar	1000

オブジェクトまたはリソースのコンフィグレーション

オブジェクトまたはリソースをコンフィグレーションするには、その名前をクリックします。右ペインにタブ付きの画面が表示されます。この画面を使用して、リソースやオブジェクトのコンフィグレーション画面またはモニタ画面に移動できます。

コンフィグレーションを編集するには、右ペインに表示されるフィールドの値を変更します。コンフィグレーションを編集したら、[適用] ボタンをクリックして変更を行い、その内容を `config.xml` ファイルに保持します。フィールドに


 アイコンが表示されている場合、そのフィールドの変更を有効にするには、変更の影響を受けるサーバを再起動する必要があります。

Administration Console を使用した複数ドメインの管理

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、アクセスする必要がある管理サーバ上の Administration Console の URL 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。

詳細については、「Administration Console について」を参照してください。

Administration Console を使用したドメインのモニタ



ドメイン リソースをモニタするには、ナビゲーション ツリーでリソースを右クリックして、モニタ用のオプションを選択します。または、リソースに移動して、右ペインの [モニタ] タブを選択します。表示されるデータは、そのリソースの現在の状態を表します。情報を更新するには、画面の右上にある  アイコンをクリックします。アイコンを再びクリックするまでに、データは定期的に更新されます。自動更新が行われているとき、アイコンがアニメーション表示で回転します。デフォルトでは、データは 10 秒ごとに更新されます。[コンソール] ノードを選択して [自動更新間隔] フィールドの値を変更すると、更新間隔を変更できます。

Administration Console のタスクのモニタ

Administration Console のナビゲーションツリーで [タスク] ノードをクリックすると、開始したさまざまな操作の進捗状況をコンソールからモニタできます。

Administration Console を使用したヘルプの表示

コンフィグレーション属性の概要、手順、および情報が含まれるオンラインヘルプは、Administration Console からいつでも利用できます。以下のいずれかのアイコンをクリックすると、オンラインヘルプにアクセスできます。

-  ■ コンソールの右上隅にあるこのアイコンをクリックすると、表示中のコンソール ページについての情報が含まれた別のブラウザ ウィンドウが開きます。このウィンドウから Administration Console の他のトピックを参照することもできます。
-  ■ フィールドの隣に表示されるこのアイコンをクリックすると、そのフィールドについての情報が含まれた小さなブラウザ ウィンドウが開きます。

Web サーバを伴う WebLogic Server の使用

Web サーバ プラグインの 1 つを使用すると、主要な Web サーバから WebLogic Server のインスタンスまたはクラスタにリクエストをプロキシできます。以下の Web サーバ用のプラグインが使用できます。

- Netscape Enterprise Server または iPlanet
- Microsoft Internet Information Server
- Apache

これらのプラグインは Web サーバのネイティブ環境で動作するため、プラグインの管理は、その Web サーバの管理機能を使用して行います。

詳細については、『WebLogic Server における Web サーバプラグインの使い方』を参照してください。

WebLogic Server のインスタンスから別のインスタンスまたはクラスタへリクエストをプロキシするために、特別なサブレットも利用できます。詳細については、以下を参照してください。

- 「別の HTTP サーバへのリクエストのプロキシ」
- 「プロキシプラグインをコンフィグレーションする」

モニタ

システム管理ツールには、WebLogic Server、ドメイン、およびリソースをモニタするための多数の機能が含まれています。ツールを使用して以下の項目をモニタできます。

- サーバの状態とパフォーマンス
 - 実行キュー
 - 接続
 - ソケット
 - スレッド
 - スループット
 - メモリ使用率
- セキュリティ
 - ロックアウトされたユーザ
 - 無効なログイン
 - ログインの試行
- トランザクション
 - コミットされたトランザクション
 - ロールバックされたトランザクション

- JMS 接続と JMS サーバ
- WebLogic メッセージング ブリッジ
- アプリケーション
 - サブレット セッション
 - コネクタ接続プール
 - EJB のパフォーマンス
- JDBC 接続と接続プール

詳細については、「WebLogic Server ドメインのモニタ」を参照してください。

ライセンス

WebLogic Server が機能するためには、有効なライセンスが必要です。

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る必要があります。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。

詳細については、13-1 ページの「WebLogic Server ライセンスの管理」を参照してください。

2 WebLogic Server の起動と停止

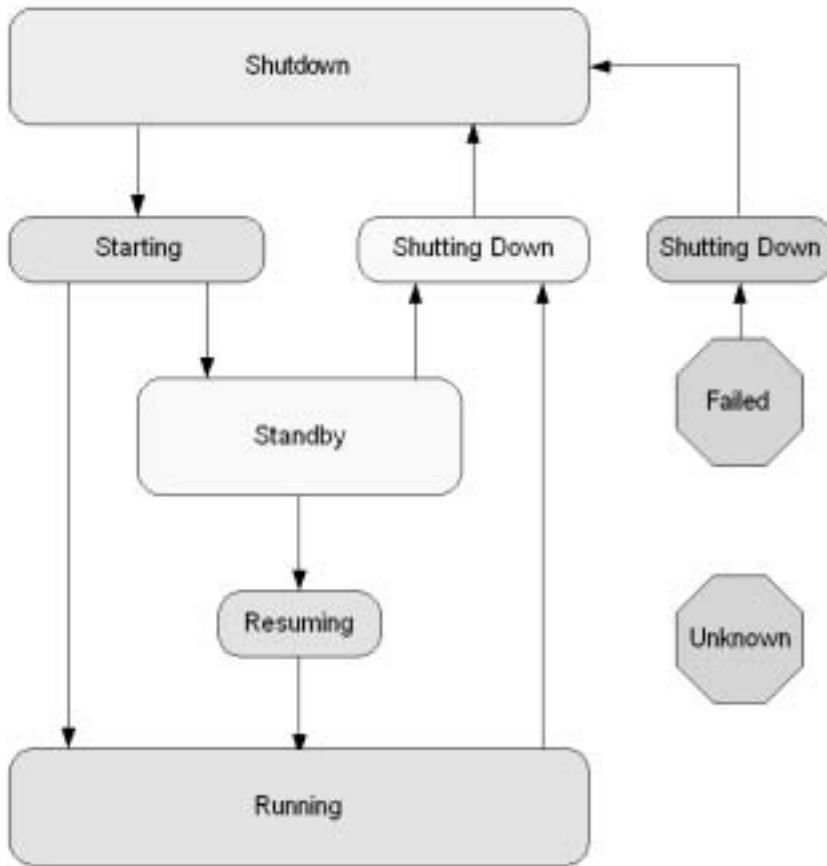
以下の節では、管理サーバと管理対象サーバを起動および停止する手順について説明します。

- サーバのライフサイクル
- サーバを起動するためのユーザ名とパスワードの指定
- 管理サーバの起動
- 管理対象サーバの起動
- WebLogic Server の停止
- 起動クラスと停止クラスのコンフィグレーション
- WebLogic Server インスタンスの Windows サービスとしての設定

サーバのライフサイクル

WebLogic Server は、複数ある状態のうち、常にいずれか1つの状態にあり、それらの状態の間を遷移する時期と方法が定義された規則に従っています。サーバが遷移していく一連の状態のことを、サーバのライフサイクルと呼びます。図 2-1 を参照してください。

図 2-1 サーバのライフサイクル



遷移の一般的なパターンは、以下のとおりです。

1. 停止中 (SHUTTING_DOWN)。この状態のサーバは、コンフィグレーションされていますがアクティブではありません。
2. 起動中 (STARTING)。サーバの起動時には、以下のアクションが行われます。
 - a. コンフィグレーションデータを取得する。

管理サーバは、ドメインのコンフィグレーションファイルからコンフィグレーションデータ (セキュリティ コンフィグレーションデータを含む) を取得します。管理対象サーバはコンフィグレーションデータとセキュ

リティ データを取得するために管理サーバにアクセスします。SSL が設定されている場合、管理対象サーバは、証明書ファイル、キー ファイルなどの独自の SSL 関連ファイルを使用し、残りのコンフィグレーションデータとセキュリティ データを取得するために管理サーバにアクセスします。

- b. カーネル レベルのサービス (ログイン サービス、タイマー サービスなど) を起動する。
- c. 手順 2a で取得したコンフィグレーション データを使用して、サブシステムレベルのサービスを初期化する。以下のようなサービスがあります。

■ セキュリティ サービス	■ JCA コンテナ
■ RMI サービス	■ JDBC コンテナ
■ クラスタ サービス	■ EJB コンテナ
■ IIOP サービス	■ Web コンテナ
■ ネーミング サービス	■ デプロイメント マネージャ
■ RMI ネーミング サービス	■ JMS プロバイダ
■ ファイル サービス	■ リモート管理
	■ トランザクション サービス

- d. 別の管理ポートを使用するようにサーバをコンフィグレーションした場合は、リモート コンフィグレーションおよびモニタを有効にする。管理ポートについては、『WebLogic Server ドメイン管理』の「ドメイン全体の管理ポートのコンフィグレーション」を参照してください。
- e. **WebLogic Server Administration Console** で指定した順序で、適切なコンテナにモジュールをデプロイする。
アプリケーションをデプロイする前にロードするようコンフィグレーションされた起動クラスは、サーバが **JDBC** 接続プール、**Web** アプリケーション、および **EJB** をデプロイする前にロードされ実行されます。
- f. アプリケーションをデプロイした後にロードするようコンフィグレーションされた起動クラスは、サーバが **JDBC** 接続プール、**Web** アプリケーション、および **EJB** をデプロイした後にロードされ実行されます。

3. スタンバイ (STANDBY)。管理ポートをコンフィグレーションした場合にのみ使用できる状態です。サーバを起動し、この状態にするコマンドを実行できます。この状態のサーバは、すべてのサービスおよびアプリケーションが初期化されており、管理コマンドを受け入れたり、クラスタの通信に参加したりできます。外部クライアントからのリクエストにはアクセスできません。

スタンバイ状態の一般的な使用法は、可用性の高い、またはミッションクリティカルな環境において、サーバを「ホット」バックアップとして使用可能な状態にしておくことです。バックアップサーバの使用が必要になった場合には、スタンバイサーバをすぐに再開して、クライアントのリクエストを処理できます。

4. 実行中 (RUNNING)。この状態のサーバは、クライアントにサービスを提供し、クラスタの正規メンバーとして機能できます。
5. 停止中 (SHUTTING_DOWN)。実行中状態またはスタンバイ状態のいずれかから、サーバをこの状態にすることができます。停止状態に遷移するとき、サーバは停止中状態を経由します。

正常な停止リクエストを発行すると、コンフィグレーションしたすべての停止クラスが呼び出されます。サーバを正常に停止できるのは、実行中状態またはスタンバイ状態からのみです。

強制的な停止を発行すると、すべてのアプリケーションおよびサブシステムに対して、すべての作業を中断してすべてのリソースを開放するよう通知されます。強制的な停止では、トランザクションがロールバックされたり、いくつかのクライアントのセッションが失われるおそれがあります。サーバの強制的な停止は、どの状態からでも実行できます。

サーバにはさらに2つの状態があります。

- 障害発生 (FAILED)。サーバが終了するまでの間に1つまたは複数の重要なサービスが機能しなくなると、サーバは障害発生状態になります。障害発生状態から回復する唯一の方法は、サーバを停止することです。重要なサービスが機能しなくなった場合には自動的に再起動するようにサーバを設定できます。自動再起動については、『**WebLogic Server** ドメイン管理』の「サーバの自動状態モニタ」を参照してください。
- 不明 (UNKNOWN)。あるサーバにアクセスできない場合、そのサーバは不明状態にあると見なされます。

サーバのライフサイクルの制御

以下のいずれかのインタフェースを使用して、サーバのライフサイクルを制御できます。

- **Administration Console** では、以下の方法でサーバのライフサイクルを制御できます。
 - [サーバ | コンフィグレーション | 一般] タブの [起動モード] フィールドで、サーバがデフォルトでは実行中状態またはスタンバイ状態のいずれで起動するかを指定する。
 - [サーバ | チューニング] タブの [サーバ ライフサイクル オペレーションのタイムアウト] フィールドで、ライフサイクル オペレーションがタイムアウトするまでの待機秒数を指定する。詳細については、2-6 ページの「ライフサイクル オペレーションのタイムアウト期間」を参照してください。
 - [サーバ | 制御 | 起動 / 停止] タブにある一連のコマンドを使用して、サーバを起動、停止、および再開する。詳細については、**Administration Console** オンライン ヘルプでサーバのタスクを参照してください。
 - `weblogic.Server` 起動コマンドでは、デフォルトの起動状態をオーバーライドする引数を指定できます。
-`Dweblogic.management.startupMode=STANDBY` 引数については、2-21 ページの「よく使用される任意指定の引数」を参照してください。
 - `weblogic.Admin` ユーティリティには、以下のコマンドが用意されています。
 - `START`(ノード マネージャが必要)
 - `STARTINSTANDBY`(ノード マネージャが必要)
 - `RESUME`
 - `SHUTDOWN`
 - `FORCESHUTDOWN`
- さらに、`GETSTATE` コマンドは、サーバの現在の状態を返します。
- `weblogic.Admin` ユーティリティの使用方法については、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」を参照するか、またはコマンドラインで次のコマンドを入力してください。

```
java weblogic.Admin HELP
```

ノードマネージャについては、『WebLogic Server ドメイン管理』の「ノードマネージャによるサーバの可用性の管理」を参照してください。

ライフサイクル オペレーションのタイムアウト期間

ライフサイクル コマンドを実行すると、サーバはサブシステムおよびアプリケーションにそのリクエストを通知し、サブシステムおよびアプリケーションが応答するまで何秒間か待機します。指定した秒数以内に応答がない場合、そのライフサイクル オペレーションはタイムアウトします。タイムアウトした後に行われるアクションは、オペレーションの種類によって異なります。

このタイムアウト期間は、SHUTDOWN オペレーションおよび FORCESHUTDOWN オペレーションにのみ適用されます。オペレーションがコンフィグレーションされた期間内に完了しない場合は、以下のいずれかのアクションが発生します。

- その時点のサーバの状態が停止中状態だった場合や、オペレーションが FORCESHUTDOWN だった場合は、サーバは自動的に停止する。
- それ以外の場合は、タイムアウト条件を記述したメッセージと共に `ServerLifecycleException` が送出される。

[サーバ | チューニング] タブで、デフォルトのタイムアウト期間を変更できます。詳細については、Administration Console オンライン ヘルプの「ライフサイクル オペレーションのタイムアウト設定」を参照してください。

サーバを起動するためのユーザ名とパスワードの指定

デフォルトでは、サーバ プロセスを実行するコマンド シェルでユーザ名とパスワードの入力が求められます。ユーザ名は、サーバの起動を許可されているロールに属していなければなりません。ロールとパーミッションについては、3-1 ページの「システム管理操作の保護」を参照してください。

この節では、以下のタスクについて説明します。

- 初期管理ユーザ名の指定

- ユーザ名とパスワードのプロンプトの回避

初期管理ユーザ名の指定

コンフィグレーション ウィザードでは、ユーザ名とパスワードの入力が求められます。入力したユーザ名は myrealm セキュリティ レルムの初期管理ユーザ名となります。**セキュリティ レルム**は、ユーザを認証したり、ユーザがアクセスできるリソースの種類を指定したりといった、**WebLogic** リソースのセキュリティ関連のサービスを提供するコンポーネント (プロバイダ) の集合です。

WebLogic Server には myrealm セキュリティ レルムがインストールされており、デフォルトではそれが使用されます。

初めて **WebLogic Server** を起動するとき、この初期管理ユーザ名とパスワードを入力します。コンフィグレーション ウィザードを使用しなかった場合は、初期ユーザ名とパスワードの入力が求められます。

Administration Console を使用して、myrealm にユーザを追加できます。

WebLogic Server にインストールされている認証プロバイダ以外の認証プロバイダを使用する場合には、そのプロバイダの管理ツールを使用して管理者特権を持ったユーザを最低でも 1 人作成する必要があります。管理者特権の付与については、3-1 ページの「システム管理操作の保護」を参照してください。

注意： guest ユーザは、**WebLogic Server** バージョン 7.0 ではデフォルトでは提供されなくなりました。guest ユーザを使用するには、互換性モードで実行するか、またはセキュリティ レルムの認証プロバイダのユーザとして guest ユーザを定義する必要があります。互換性モードについては、『**WebLogic Security** の管理』の「互換性セキュリティの使い方」を参照してください。

別のセキュリティ レルムを使用するように **WebLogic Server** をコンフィグレーションできます。異なるセキュリティ レルムを設定した場合は、設定したレルムから 1 つをデフォルトとして指定する必要があります。起動時には **WebLogic Server** はデフォルトのレルムを使用して、入力されたユーザ名を認証します。

ユーザ名とパスワードのプロンプトの回避

ユーザ名とパスワードのプロンプトを回避する場合は、起動 ID ファイルを作成して使用することをお勧めします。起動 ID ファイルにはユーザ名とパスワードが暗号化された形式で格納されます。

この節では、以下の項目について説明します。

- 管理サーバの起動 ID ファイルの作成
- 管理対象サーバの起動 ID ファイルの作成
- 起動 ID ファイルの使用
- 起動後の起動 ID ファイルの削除
- 代替手段: コマンドラインでの ID 情報の入力

管理サーバの起動 ID ファイルの作成

管理サーバの起動 ID ファイルを作成するには、次の手順に従います。

1. 管理サーバを 1 回以上起動し、コマンドラインでユーザ資格を指定します。
管理サーバは最初の起動プロセスの過程でセキュリティ ファイルを生成します。このファイルが適切な場所にないと、サーバは起動 ID ファイルを使用できません。

2. 次の 2 行をテキスト ファイルに入力します。

```
username=username  
password=password
```

ユーザ名とパスワードの値は、デフォルトセキュリティ レルムの認証プロバイダの既存ユーザ アカウントと一致し、さらにサーバを起動するパーミッションを持つロールに属している必要があります。ロールとパーミッションについては、3-1 ページの「システム管理操作の保護」を参照してください。

3. ファイルを保存します。

ファイルを `boot.properties` として保存し、サーバのルート ディレクトリに配置すると、起動時に自動的にこのファイルが使用されます。詳細については、「起動 ID ファイルの使用」を参照してください。

初めてこのファイルを使用してサーバを起動するとき、サーバはファイルを読み込んでから、ユーザ名とパスワードの暗号化バージョンでそのファイルを上書きします。

管理サーバの起動 ID ファイルを作成する別の方法

前の節で説明した手順に従う代わりに `weblogic.Server` クラスをコマンドラインから直接呼び出す場合、**Java** コマンドに以下のオプションを指定することによって起動 ID ファイルを作成できます。

```
-Dweblogic.management.username=username  
-Dweblogic.management.password=password  
-Dweblogic.system.StoreBootIdentity=true
```

これらのオプションを指定すると、サーバインスタンスが指定したユーザ資格で起動され、`boot.properties` というファイルに格納されます。

たとえば、次のコマンドを実行すると、**myAdminServer** という名前の管理サーバが起動し、起動 ID ファイルが作成されます。

```
java -Dweblogic.management.username=username  
-Dweblogic.management.password=password  
-Dweblogic.system.StoreBootIdentity=true  
-Dweblogic.Name=myAdminServer weblogic.Server
```

コマンドラインから直接 `weblogic.Server` クラスを呼び出す方法の詳細については、2-18 ページの「`weblogic.Server` コマンドの使用」を参照してください。

注意： スクリプトを使用して管理サーバを起動する場合は、以下の理由からこの節で説明した方法を使用することはお勧めできません。

- 起動スクリプトに暗号化されていないパスワードを格納する必要がある
- スクリプトを実行するたびに、指定したユーザ資格でサーバが起動されて新しい起動 ID ファイルが作成される

管理対象サーバの起動 ID ファイルの作成

管理対象サーバが、管理サーバと同じルート ディレクトリを使用しているならば、その管理対象サーバを別のユーザ資格に基づいて実行する場合以外は、管理対象サーバのためにさらに起動 ID ファイルを作成する必要はありません。サーバのルート ディレクトリについては、2-31 ページの「サーバのルート ディレクトリ」を参照してください。

また、ノード マネージャを使って管理対象サーバを起動する場合は、起動 ID ファイルを作成する必要はありません。その代わりに、**Administration Console** における管理対象サーバの [リモート スタート] タブでユーザ資格を指定する必要があります。詳細については、「管理対象サーバの起動引数のコンフィグレーション」を参照してください。

管理対象サーバの起動 ID ファイルを作成するには、次の手順に従います。

1. ドメインの管理サーバを起動します。
2. 管理サーバのルート ディレクトリから管理対象サーバのルート ディレクトリへ、`SerializedSystemIni.dat` ファイルをコピーします。
3. 次の 2 行をテキスト ファイルに入力します。

```
username=username  
password=password
```

ユーザ名とパスワードの値は、デフォルトセキュリティ レルムの認証プロバイダの既存ユーザ アカウントと一致し、さらにサーバを起動するパーミッションを持つロールに属している必要があります。ロールとパーミッションについては、3-1 ページの「システム管理操作の保護」を参照してください。

4. ファイルを保存します。
ファイルを `boot.properties` として保存し、サーバのルート ディレクトリに配置すると、起動時に自動的にこのファイルが使用されます。詳細については、「起動 ID ファイルの使用」を参照してください。

起動 ID ファイルの使用

サーバインスタンスは、次のように起動 ID ファイルを使用します。

- サーバのルート ディレクトリに有効な `boot.properties` が格納されている場合には、デフォルトでそのファイルが使用されます。サーバのルート ディ

レクトリについては、2-31 ページの「サーバのルート ディレクトリ」を参照してください。

- 別のファイルを指定する場合や、サーバのルート ディレクトリに起動 ID ファイルを格納しない場合には、サーバの `weblogic.Server` 起動コマンドで次の引数を使用します。

```
-Dweblogic.system.BootIdentityFile=filename
```

`filename` は、有効な起動 ID ファイルの絶対パス名です。

`startWebLogic` スクリプトを使用する場合は、`JAVA_OPTIONS` 変数の値として `-Dweblogic.system.BootIdentityFile` を追加します。次に例を示します。

```
JAVA_OPTIONS=-Dweblogic.system.BootIdentityFile=C:\BEA\user_domains\mydomain\myidentity.prop
```

- サーバ インスタンスで起動 ID ファイルを使用しない場合は、サーバの `weblogic.Server` 起動コマンドに以下のオプションを含めます。

```
-Dweblogic.management.username=username
```

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

これらのオプションにより、サーバ インスタンスは起動 ID ファイルを一切使用しないようになり、サーバが起動 ID ファイルを使用することになるような他の起動オプションはオーバーライドされます。

注意： この方法では、暗号化されていないパスワードを起動スクリプトに格納する必要があるため、スクリプトを使用してサーバ インスタンスを起動する場合にはお勧めできません。この方法は、`weblogic.Server` クラスをコマンドラインから直接呼び出す場合にのみ使用してください。詳細については、2-18 ページの「`weblogic.Server` コマンドの使用」を参照してください。

- サーバが起動 ID ファイルにアクセスできない場合は、コマンド シェルにユーザ名とパスワードのプロンプトが表示され、ログ ファイルにメッセージが書き込まれます。

特定のサーバ インスタンスでは、そのインスタンスが作成した起動 ID ファイルのみを使用してください。`WebLogic Server` は、あるサーバのルート ディレクトリから別のサーバのルート ディレクトリへの起動 ID ファイルのコピーはサポートしていません。

たとえば、ServerA を使用して起動 ID ファイルを生成した場合、ServerA ではその起動 ID ファイルだけを使用します。ServerA の起動 ID ファイルを、ServerB のルートディレクトリへコピーすることはできません。代わりに、上述の手順で ServerB の起動 ID ファイルを作成します。

起動後の起動 ID ファイルの削除

サーバの起動後に起動 ID ファイルを削除する場合は、サーバの `weblogic.Server` 起動コマンドで次の引数を使用します。

```
-Dweblogic.system.RemoveBootIdentity=true
```

この引数は、サーバが起動に使用したファイルのみを削除します。たとえば、`-Dweblogic.system.BootIdentityFile=c:\secure\boot.MyServer` と指定した場合には、サーバのルートディレクトリに `boot.properties` というファイルが格納されていても `boot.MyServer` のみが削除されます。

代替手段：コマンドラインでの ID 情報の入力

対話形式のプロンプトを回避するには、起動 ID ファイルの使用が最も安全で便利な方法です。ただし、起動 ID ファイルを使用する代わりに、`weblogic.Server` 起動コマンドに以下の引数を追加することもできます。

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

これらの引数を両方とも入力すると、対話形式のプロンプトを回避できます。

これらのオプションにより、サーバインスタンスは起動 ID ファイルを一切使用しないようになり、サーバが起動 ID ファイルを使用することになるような他の起動オプションはオーバーライドされます。

サーバを起動するコマンドは長くなる場合があるので、通常は起動コマンドの大部分がスクリプトに組み込まれています。セキュリティに少しでも不安のある環境では、`-Dweblogic.management.password=password` 引数を起動スクリプトに保存しないことをお勧めします。

これらの引数の詳細については、2-18 ページの「`weblogic.Server` コマンドの使用」を参照してください。

管理サーバの起動

WebLogic Server は、Java 仮想マシン (JVM) 内のプロセスとして実行されます。各 JVM でホストできるのは、1 つのサーバ プロセスのみです。サーバを起動するには、一連の引数を使用して JVM を起動します。

ドメインが 1 つの WebLogic Server のみで構成されている場合は、そのサーバが管理サーバになります。ドメインが複数の WebLogic Server で構成されている場合は、管理対象サーバを起動する前に管理サーバを起動する必要があります。

同じドメイン内の管理サーバと管理対象サーバは、すべて同じバージョンの WebLogic Server である必要があります。また、管理サーバのサービスパックのレベルは、管理対象サーバのサービスパックのレベル以上でなければなりません。たとえば、管理対象サーバがリリース 7.0 であれば、管理サーバは 7.0 または 7.0 SP1 になります。ただし、管理対象サーバが SP1 である場合は、管理サーバも SP1 でなければなりません。ドメイン内の各サーバの名前はユニークでなければなりません。

この節では、管理サーバの起動について説明します。管理サーバを起動するには、以下の中から必要なタスクをローカル ホストで行います。

- Windows の [スタート] メニューからの管理サーバの起動
- スクリプトを使用した管理サーバの起動
- `weblogic.Server` コマンドの使用
- デフォルト コンフィグレーションを使用したサーバの起動

詳細については、以下を参照してください。

- 2-47 ページの「WebLogic Server インスタンスの Windows サービスとしての設定」
- Administration Console オンライン ヘルプの「UNIX 上の保護されているポートへのバインディング」

注意： WebLogic Server の起動時に数多くのクラスをロードしようとする、JDK 1.3 によって `OutOfMemory` エラーが送出される場合があります。このエラーは、使用可能なメモリが十分あるような場合でも発生します。WebLogic Server の起動時に `java.lang.OutOfMemory` エラー例外が発生した場合は、次の JVM オプションの値を大きくしてください。

```
java -XX:MaxPermSize=<value>
```

<value> は、キロバイト単位の数字です。

JDK 1.3.1 では、MaxPermSize のデフォルトの値は 64m です (m は MB を表します)。

Windows の [スタート] メニューからの管理サーバの起動

コンフィグレーション ウィザードを使用して、Windows コンピュータ上で単一サーバ、管理対象サーバを持つ管理サーバ、またはクラスタ化された管理対象サーバを持つ管理サーバを作成すると、Windows の [スタート] メニューにドメインをインストールするよう求められます。インストールすることを選択した場合は、次の手順に従って単一サーバまたは管理サーバを起動できます。

Windows のデスクトップから、[スタート | プログラム | BEA WebLogic Platform 7.0 | User Projects | *domain_name* | Start Server] をクリックします。

コマンドウィンドウが開き、スクリプト *domain_name*\startWebLogic.cmd が呼び出されます。このスクリプトについては 2-14 ページの「スクリプトを使用した管理サーバの起動」で説明します。サーバが正常に起動プロセスを完了すると、コマンドウィンドウに次のメッセージが表示されます。

```
<Notice> <WebLogicServer> <000360> <Server started in RUNNING mode>
```

スクリプトを使用した管理サーバの起動

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長くなる場合があり、エラーが発生しやすくなるので、コマンドをスクリプトに組み込むことをお勧めします。

この節では、以下のタスクについて説明します。

- コンフィグレーション ウィザードスクリプトを使用した管理サーバの起動
- 管理サーバを起動する独自のスクリプトの作成
- WebLogic Server でのデフォルト以外の JVM の使用

コンフィグレーション ウィザード スクリプトを使用した管理サーバの起動

コンフィグレーション ウィザードを使用してドメインを作成すると、そのドメインの管理サーバの起動に使用できるスクリプトも作成されます。スクリプトを使用するには、コマンドプロンプトで以下のいずれかのコマンドを入力します。

- `domain_name\startWebLogic.cmd` (Windows)
- `domain_name\startWebLogic.sh` (UNIX および Windows。このスクリプトは Windows 上で MKS および Cygnus BASH UNIX シェル エミュレータをサポート)

`domain_name` は、ドメインを配置したディレクトリです。

このスクリプトは一部のドメイン固有の変数の値を設定してから、マスター起動スクリプト `WL_HOME\server\bin\startWLS.cmd` (UNIX では `startWLS.sh`) を呼び出します。`WL_HOME` は WebLogic Server をインストールした場所です。マスター起動スクリプトは、JVM の場所などの環境変数を設定し、その後 WebLogic Server の引数を使用して JVM を起動します。

管理サーバを起動する独自のスクリプトの作成

Administration Console など別の方法でドメインを作成する場合は、次の手順に従って独自の起動スクリプトを作成できます。

1. `SERVER_NAME` 変数の値を設定します。ドメイン内のすべてのサーバには名前が必要です。次に例を示します。

```
set SERVER_NAME=myserver
```

ドメインの `config.xml` ファイルでは、サーバの名前は `<Server Name=serverName>` として指定されます。`set SERVER_NAME` の値が `config.xml` で指定されているサーバ名になっていることを確認してください。

2. 以下の任意指定の変数の値を設定します。

表 2-1 任意指定の変数

変数	説明
WLS_USER	サーバを起動するユーザをクリアテキストで設定する変数。この変数は使用せず、起動 ID ファイルを使用することを推奨。詳細については、2-8 ページの「ユーザ名とパスワードのプロンプトの回避」を参照。
WLS_PW	サーバを起動するパスワードをクリアテキストで設定する変数。この変数は使用せず、起動 ID ファイルを使用することを推奨。詳細については、2-8 ページの「ユーザ名とパスワードのプロンプトの回避」を参照。
ADMIN_URL	この変数に URL を指定すると、サーバは管理対象サーバとして起動し、指定した URL を使用して管理サーバにアクセスする。 詳細については、1-6 ページの「管理サーバと管理対象サーバ」を参照。
STARTMODE	サーバをプロダクションモードまたは開発モードのいずれで実行するかを指定する。true を指定するとプロダクションモードで、false を指定すると開発モードで実行される。 プロダクションモードまたは開発モードの使用に関する詳細については、2-29 ページの「開発モードとプロダクションモード」を参照。
JAVA_OPTIONS	サーバを実行するための Java コマンドラインオプション。Java コマンドラインオプションは、JAVA_VM および MEM_ARGS が渡された後に JVM に渡される。-Dweblogic.ListenAddress は、ドメインの起動スクリプトから呼び出せる Java オプションの例。コマンドラインオプションの詳細については、2-18 ページの「weblogic.Server コマンドの使用」を参照。 UNIX シェルで複数のオプションをリストする場合は、一連のオプション全体を引用符で囲み、各オプションの間にスペースを挿入する。次に例を示す。 JAVA_OPTIONS="-Dweblogic.attribute=value -Djava.attribute=value"

表 2-1 任意指定の変数

変数	説明
JAVA_VM	<p>仮想マシンを実行するモードを指定するための Java 引数。以下のいずれかのオプションを使用。</p> <ul style="list-style-type: none"> ■ -server ■ -client ■ -hotspot (Windows のみ) <p>これらの操作モードをサポートしない JVM を使用している場合は、これらの引数が JVM に渡されないように、マスタースクリプトを編集する必要があります。詳細については、2-18 ページの「WebLogic Server でのデフォルト以外の JVM の使用」を参照。</p>
MEM_ARGS	<p>Java に渡されたデフォルトのメモリ引数をオーバーライドする変数。マスター起動スクリプトでは、オプションはデフォルトで -Xms200m および -Xmx200m に設定されている。</p>

3. マスター起動スクリプト `WL_HOME\server\bin\startWLS.cmd` (UNIX では `startWLS.sh`) を呼び出します。

マスター起動スクリプトは、JVM の場所などの環境変数を設定し、その後 WebLogic Server の引数を使用して JVM を起動します。WebLogic Server とともにインストールされる JVM を使用していない場合は、マスター起動スクリプトを編集する必要があります。詳細については、2-18 ページの「WebLogic Server でのデフォルト以外の JVM の使用」を参照してください。

4. 起動スクリプトをドメインのルートディレクトリ以外の場所に配置する場合は、JAVA_OPTIONS 変数に次の値を指定する必要があります。

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

`path` は、ドメインのルートディレクトリの場所を指定します。

次に例を示します。

```
JAVA_OPTIONS=-Dweblogic.RootDirectory=c:\serverRoot
```

WebLogic Server でのデフォルト以外の JVM の使用

WebLogic Server とともにインストールされた JVM を使用しない場合は、`JAVA_HOME` 変数にシステム上の JVM の適切な場所が指定されるように、マスター起動スクリプトを編集する必要があります。

詳細については、「WebLogic Platform と共にバンドルされていない JVM を使用する」(<http://edocs.beasys.co.jp/e-docs/platform/docs70/relnotes/relnotes.html>) を参照してください。

weblogic.Server コマンドの使用

`weblogic.Server` は、ローカル ホストで WebLogic Server を起動するコマンドです。既に説明したように起動スクリプトは、このコマンドに一連のオプションを送るラッパーです。このコマンドとそれに付随するオプションを起動スクリプトに組み込むことをお勧めしますが、コマンドラインに直接 `weblogic.Server` コマンドを入力して単純に呼び出すこともできます。

たとえば、Windows 上で `examples` サーバを起動する単純な呼び出しは次のとおりです（このコマンドを `WL_HOME\samples\server\config\examples` ディレクトリから入力する必要があります）。

```
c:\bea\jdk131\bin\java
-hotspot -Xms200m -Xmx200m
-classpath "c:\bea\jdk131\lib\tools.jar;
           c:\bea\weblogic700\server\lib\weblogic_sp.jar;
           c:\bea\weblogic700\server\lib\weblogic.jar;"
-Dweblogic.Name=examplesServer
-Dbea.home="C:\bea"
-Djava.security.policy="c:\bea\weblogic700\server\lib\weblogic.policy"
weblogic.Server
```

ここでは、以下の項目について説明します。

- クラスパスの設定
- `weblogic.Server` コマンドの構文
- 必須の引数
- よく使用される任意指定の引数
- その他の任意指定の引数

- 開発モードとプロダクション モード
- 管理ポートおよび `weblogic.Admin` ユーティリティの起動引数
- サーバのルート ディレクトリ

リモート ホスト上の管理対象サーバの起動については、『WebLogic Server ドメイン管理』の「ノード マネージャによるサーバの可用性の管理」を参照してください。

クラスパスの設定

Java 仮想マシン (JVM) では、`classpath` と呼ばれる設定を使用して、必要不可欠なファイルおよびディレクトリの場所を指定します。

以下のコマンドを使用して、WebLogic Server のクラスパスを設定できます。

```
WL_HOME\server\bin\setWLSEnv.cmd (Windows)
```

```
WL_HOME/server/bin/setWLSEnv.sh (UNIX)
```

`setWLSEnv` を使用する代わりに、起動コマンドで環境変数または `-classpath` 引数を使用できます。いずれの方法でも、WebLogic Server のインスタンスを実行する JVM のクラスパスに以下を含める必要があります。

- `WL_HOME/server/lib/weblogic_sp.jar`

インストールされている WebLogic Server のリリース、サービス パック、パッチによっては、このファイルがシステム上に存在しない場合があります。ファイルがシステム上に存在するかどうかに関係なく、クラスパスに `WL_HOME/server/lib/weblogic_sp.jar` を含めることで更新時の互換性を確保しておくことをお勧めします。このファイルをクラスパスに追加する際は、`weblogic.jar` より前に追加する必要があります。

- `WL_HOME/server/lib/weblogic.jar`

- PointBase (Java だけで作られているデータベース管理システム) の試用版を使用している場合は、続いて以下のファイルを指定する。

```
SAMPLES_HOME/server/eval/pointbase/server/lib/  
pbserver4lev.jar および pbclient4lev.jar
```

`SAMPLES_HOME` は、`WL_HOME/samples`。

- WebLogic Enterprise Connectivity を使用している場合は、以下のファイルを指定する。

`WL_HOME/server/lib/wlepool.jar`

`WL_HOME/server/lib/wleorb.jar`

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

weblogic.Server コマンドの構文

`weblogic.Server` コマンドの構文は、次のとおりです。

```
java RequiredArguments [OptionalArguments] weblogic.Server
```

必須の引数

次の表には、`java` コマンドラインからの **WebLogic Server** の起動に不可欠な引数が示されています。

表 2-2 サーバの起動に不可欠な引数

引数	説明
<code>-Xms</code> および <code>-Xmx</code>	<p>Java ヒープメモリの最小値と最大値を指定する (MB 単位)。</p> <p>たとえば、デフォルトの 200MB 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する。そのためには、<code>java -Xms200m</code> オプションおよび <code>-Xmx200m</code> オプションを使用してサーバを起動できる。</p> <p>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する。</p> <p>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る。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る。</p>
<code>-classpath</code>	<p>このオプションで指定する最低限の内容は、2-19 ページの「クラスパスの設定」で説明されている。</p> <p>注意： <code>classpath</code> がユーザ環境で設定される場合には、このオプションは省略可能。</p>

表 2-2 サーバの起動に不可欠な引数

引数	説明
<code>-Dweblogic.Name=servername</code>	サーバに名前を割り当てる。 サーバ名はドメイン内でユニークである必要がある。たとえば、DomainA というドメインにあるサーバインスタンスに ManagedServer1 という名前を付ける場合、DomainA 内の別のサーバインスタンスに ManagedServer1 と名付けることはできない。
<code>-Dbea.home=bea_home</code>	BEA ホーム ディレクトリの場所を指定する。BEA ホーム ディレクトリには、ライセンス情報などの必要不可欠な情報が格納されている。

よく使用される任意指定の引数

次の表には、よく使用される任意指定の引数が示されています。各引数の説明では、その引数を **Administration Console** やその他の **WebLogic Server** コマンドを使用して設定することもできるかどうか示されています。**MBean (Managed Bean)** の属性を設定する引数はすべて、**MBean** の API を使用して設定することもできます。**Mbean** の属性の設定については、2-28 ページの「その他の任意指定の引数」を参照してください。

表 2-3 よく使用される任意指定の引数

引数	説明
<code>-Dweblogic.RootDirectory=path</code>	サーバのルート ディレクトリを指定する。詳細については、2-31 ページの「サーバのルート ディレクトリ」を参照。

2 WebLogic Server の起動と停止

表 2-3 よく使用される任意指定の引数

引数	説明
<code>-Dweblogic.ConfigFile= file_name</code>	<p>ドメインのコンフィグレーション ファイルを指定する。 <code>file_name</code> の値は、<code>config.dtd</code> に準拠した有効な XML ファイルでなければならない。XML ファイルは、ドメイン の管理サーバのルート ディレクトリに存在する必要がある。 ルート ディレクトリは、カレント ディレクトリか、または <code>-Dweblogic.RootDirectory</code> で指定したディレクトリ。 <code>file_name</code> 値にはパス名を指定できない。たとえば、次の 値は無効となる。</p> <pre>-Dweblogic.ConfigFile=c:\mydir\myfile.xml</pre> <p>代わりに次の引数を使用する。</p> <pre>-Dweblogic.RootDirectory=c:\mydir -Dweblogic.ConfigFile=myfile.xml</pre> <p><code>config.dtd</code> については、『コンフィグレーション リファレンス』を参照。 この値を指定しない場合のデフォルトは、<code>config.xml</code> になる。</p>
<code>-Dweblogic.management. username=username</code>	<p>ユーザ名を指定する。</p> <p>ユーザ名は、サーバを起動するパーミッションを持つロールに属していなければならない。ロールとパーミッションについては、3-1 ページの「システム管理操作の保護」を参照。</p> <p>このオプションにより、サーバインスタンスは起動 ID ファイルを一切使用しないようになり、サーバが起動 ID ファイルを使用することになるような他の起動オプションはオーバーライドされる。詳細については、2-8 ページの「ユーザ名とパスワードのプロンプトの回避」を参照。</p>
<code>-Dweblogic.management. password=password</code>	<p>ユーザ パスワードを指定する。</p> <p>このオプションにより、サーバインスタンスは起動 ID ファイルを一切使用しないようになり、サーバが起動 ID ファイルを使用することになるような他の起動オプションはオーバーライドされる。詳細については、2-8 ページの「ユーザ名とパスワードのプロンプトの回避」を参照。</p>

表 2-3 よく使用される任意指定の引数

引数	説明
-Dweblogic.ListenAddress= <i>host</i>	<p>このサーバのリスンアドレスを指定する。<i>host</i> の値は、サーバの DNS 名または IP アドレスのいずれかでなければならない。</p> <p>このオプションは、ServerMBean の <code>listenAddress</code> 属性の値を設定する。この属性には、Administration Console からアクセスできる ([サーバ コンフィグレーション 一般 リスンアドレス])。</p> <p>リスンアドレスを指定しない場合、マシンの DNS 名または IP アドレスのいずれかが使用される。</p> <p>この引数ではなく Administration Console を使用して、確認済みの DNS 名または IP アドレスを指定することを推奨。詳細については、「ネットワーク リソースのコンフィグレーション」を参照。</p>
-Dweblogic.ListenPort= <i>portnumber</i>	<p>このサーバのプレーンテキスト (非 SSL) リスンポートを指定し、有効にする。</p> <p>この引数は、ServerMBean の <code>listenPort</code> 属性の値を設定する。この属性には、Administration Console からアクセスできる ([サーバ コンフィグレーション 一般 リスンポート])。</p> <p>リスンポートを指定しない場合、デフォルトで 7001 が使用される。</p> <p>詳細については、「ネットワーク リソースのコンフィグレーション」を参照。</p>
-Dweblogic.ssl.ListenPort= <i>portnumber</i>	<p>WebLogic Server が SSL 接続リクエストをリスンするポートを指定し、有効にする。</p> <p>この引数は、SSLBean の <code>listenPort</code> 属性の値を設定する。この属性には、Administration Console からアクセスできる ([サーバ 接続 SSL ポート SSL リスンポート])。</p> <p>リスンポートを指定しない場合、デフォルトで 7002 が使用される。</p> <p>詳細については、「ネットワーク リソースのコンフィグレーション」を参照。</p>

2 WebLogic Server の起動と停止

表 2-3 よく使用される任意指定の引数

引数	説明
<code>-Dweblogic.system. StoreBootIdentity=true</code>	サーバのルート ディレクトリに <code>boot.properties</code> を作成する。このファイルには、サーバの起動に使用したユーザ名とパスワードの暗号化バージョンが格納される。 詳細については、2-8 ページの「ユーザ名とパスワードのプロンプトの回避」を参照。
<code>-Dweblogic.system. BootIdentityFile=<i>filename</i></code>	ユーザ名とパスワードを格納する起動 ID ファイルを指定する。 <i>filename</i> の値は、有効な起動 ID ファイルの絶対パス名でなければならない。次に例を示す。 <code>-Dweblogic.system.BootIdentityFile=C:\BEA\wlserver7.0\user_config\mydomain\myidentity.prop</code> ファイル名を指定しない場合、サーバのルート ディレクトリにある <code>boot.properties</code> が使用される。起動 ID ファイルがない場合は、ユーザ名とパスワードの入力が求められる。
<code>-Dweblogic.system. RemoveBootIdentity=true</code>	サーバの起動後に起動 ID ファイルを削除する。
<code>-Dweblogic.management. pkpassword=<i>pkpassword</i></code>	暗号化されたフラット ファイルからセキュア ソケット レイヤ (SSL) プライベート キーを取得するためのパスワードを指定する。 このオプションは、プライベート キーを暗号化されたフラット ファイルに格納する場合に使用する。
<code>-Dweblogic.security.SSL. trustedCAKeyStore=<i>path</i></code>	SSL を使用する場合、この引数を使用してサーバまたはクライアントで信頼される認証局を指定する。 <i>path</i> の値は、Sun JKS キーストア ファイル (キーおよび証明書のリポジトリを格納) への相対パス名または絶対パス名でなければならない。 この引数を指定しない場合、WebLogic Server またはクライアントは、 <code>JAVA_HOME\jre\lib\security\cacerts</code> で指定されているすべての証明書を信頼する。 プロダクション環境でデモ用認証局を使用することは望ましくない。

表 2-3 よく使用される任意指定の引数

引数	説明
<code>-Dweblogic.security.SSL.ignoreHostnameVerification=true</code>	<p>ホスト名検証を無効にする。</p> <p>WebLogic Server 付属のデモ用デジタル証明書を使用する場合に、この引数を指定する。</p> <p>注意: 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。</p> <p>この引数を指定しない場合、WebLogic Server のホスト名検証では、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が比較される。SubjectDN とホスト名が一致しない場合、SSL 接続は中断される。</p>
<code>-Dweblogic.security.SSL.HostnameVerifier=hostnameverifierimplmentation</code>	<p>カスタム ホスト名検証クラスの名前を指定する。このクラスは <code>weblogic.security.SSL.HostnameVerifier</code> インタフェースを実装する必要がある。</p>
<code>-Dweblogic.security.SSL.sessionCache.size=sessionCacheSize</code> <code>-Dweblogic.security.SSL.sessionCache.ttl=sessionCacheTimeToLive</code>	<p>SSL セッションキャッシングに対するサーバセッションキャッシュのデフォルトのサイズと存続期間を変更する。<code>sessionCacheSize</code> の値はセッションキャッシュ内の項目数を指定し、<code>sessionCacheTimeToLive</code> の値はセッションキャッシュの存続期間 (秒単位) を指定する。</p> <p><code>sessionCache.size</code> の場合、</p> <ul style="list-style-type: none"> ■ 最小値は 1 ■ 最大値は 65537 ■ デフォルト値は 211 <p><code>sessionCache.ttl</code> の場合、</p> <ul style="list-style-type: none"> ■ 最小値は 1 ■ 最大値は <code>Integer.MAX_VALUE</code> ■ デフォルト値は 600

2 WebLogic Server の起動と停止

表 2-3 よく使用される任意指定の引数

引数	説明
<code>-Djava.security.manager</code> <code>-Djava.security.policy=filename</code>	<p>Java 2 セキュリティ マネージャを有効にする。Java 2 セキュリティ マネージャは、ポリシー ファイルで制限されているアクションが、信頼性のないコードによって実行されることを防止する。</p> <p><code>-Djava.security.policy</code> 引数は、Java 2 セキュリティ ポリシーを格納するファイル名を、相対パス名または絶対パス名で指定する。</p> <p>編集して使用できる WebLogic Server のサンプル ポリシー ファイルは、<code>WL_HOME\server\lib\weblogic.policy</code>。詳細については、『WebLogic Security プログラマーズ ガイド』の「weblogic.policy ファイルの修正」を参照。</p>
<code>-Dweblogic.security.anonymousUserName=guest</code>	<p>ユーザ アカウント <code>guest</code> のサポートを有効にする。この引数を使用して WebLogic Server インスタンスを起動する場合は、デフォルトのセキュリティ レベルの認証プロバイダに <code>guest</code> ユーザを追加しておく必要がある。</p> <p>詳細については、『WebLogic リソースのセキュリティ』の「ユーザの作成」を参照。</p>
<code>-Dweblogic.management.startupMode=STANDBY</code>	<p>サーバを起動して、スタンバイ (STANDBY) 状態にする。この起動引数を使用するには、別の管理ポートを使用するようにサーバをコンフィグレーションする必要がある。</p> <p>管理ポートについては、『WebLogic Server ドメイン管理』の「ドメイン全体の管理ポートのコンフィグレーション」を参照。</p> <p>この値は、Administration Console の [サーバ コンフィグレーション 一般] タブで指定された <code>startupMode</code> の値を、現在のセッションに対してのみオーバーライドする。</p> <p>コマンドラインまたは <code>config.xml</code> のいずれかで、この値を指定しない場合、デフォルトでは実行中 (RUNNING) 状態で起動される。</p>

表 2-3 よく使用される任意指定の引数

引数	説明
<code>-Dweblogic.ProductionModeEnabled={true false}</code>	<p>ドメイン内のすべてのサーバがプロダクションモードで起動するかどうかを指定する。このオプションは、管理サーバを起動する場合にのみ適用できる。管理対象サーバはすべて、管理サーバと同じモードで起動する。</p> <p><code>true</code> を指定すると、WebLogic Server は <code>domain_name/applications</code> ディレクトリにあるアプリケーションを自動的にデプロイおよび更新できなくなる。このオプションを指定しない場合、値は <code>false</code> と見なされる。</p> <p>詳細については、2-29 ページの「開発モードとプロダクションモード」を参照。</p>
<code>-Dweblogic.management.discover={true false}</code>	<p>管理サーバに障害が発生し、再起動された後、そのサーバがドメインの管理を回復するかどうかを指定する。</p> <p><code>true</code> を指定すると、管理サーバはその <code>running-managed-servers.xml</code> ファイルを参照する。このファイルには、デプロイ可能なモジュールのデプロイメント状態に関する情報および現在実行中のすべての管理対象サーバのリストが格納されている。この引数に <code>true</code> を指定して管理サーバを起動すると、管理サーバは管理対象サーバと通信し、実行中であることを管理対象サーバに通知する。</p> <p><code>false</code> を指定すると、管理サーバはファイルを参照できず、そのため、ドメイン内で現在アクティブな管理対象サーバと通信できなくなる。</p> <p>警告： <code>false</code> の値は、単一サーバの開発環境でのみ指定すること。<code>false</code> を指定すると、ドメイン内のサーバインスタンスが持つデプロイ済みモジュールのセットに整合性がなくなる可能性がある。</p> <p>このオプションを指定しない場合、値は <code>true</code> と見なされる。</p>

表 2-3 よく使用される任意指定の引数

引数	説明
<code>-Dweblogic.Stdout="filename"</code>	サーバおよび JVM の標準出力ストリームをファイルにリダイレクトする。パス名を、完全修飾で指定するか、WebLogic Server のルートディレクトリからの相対で指定する。 詳細については、4-12 ページの「System.out および System.err のファイルへのリダイレクト」を参照。
<code>-Dweblogic.Stderr="filename"</code>	サーバおよび JVM の標準エラー ストリームをファイルにリダイレクトする。パス名を、完全修飾で指定するか、WebLogic Server のルートディレクトリからの相対で指定する。 詳細については、4-12 ページの「System.out および System.err のファイルへのリダイレクト」を参照。

その他の任意指定の引数

`weblogic.Server` 起動コマンドの引数を使用して、以下の MBean の属性を設定できます。

- `ServerMBean`。次の構文で指定します。
`-Dweblogic.attribute-name=value`
たとえば、`listenPort` 属性の値を設定する場合は、
`-Dweblogic.ListenPort=7010` となります。
- `LogMBean`。次の構文で指定します。
`-Dweblogic.log.attribute-name=value`
たとえば、`FileName` 属性の値を設定する場合は、
`-Dweblogic.log.FileName="C:\logfiles\myServer.log"` となります。
- `SSLMBean`。次の構文で指定します。
`-Dweblogic.ssl.attribute-name=value`
たとえば、`Enable` 属性の値を `true` に設定する場合は、
`-Dweblogic.ssl.Enable="true"` となります。

MBean がセッター メソッドとして公開している属性を設定できます。上記の構文では、`attribute-name` は、MBean のセッター メソッドの名前から `set` プレフィックス除いたものです。

たとえば、`ServerMBean` は以下のセッター メソッドでそのリスン ポート属性を公開します。

- `setListenPort()`

`weblogic.Server` コマンドでリスン ポートの値を設定するには、次の構文を使用します。`-Dweblogic.ListenPort=portnumber`

コマンドライン引数は MBean の現在の設定をオーバーライドします。その設定は、ドメインの `config.xml` ファイルには保持されません。

開発モードとプロダクション モード

WebLogic Server は、開発環境とプロダクション環境の 2 つの異なるモードで実行できます。開発モードは、アプリケーションをテストする場合に使用します。プロダクション環境への準備が整ったら、プロダクション モードで起動されたサーバにアプリケーションをデプロイします。

開発モードでは、WebLogic Server は `domain_name/applications` ディレクトリ (`domain_name` は WebLogic Server ドメインの名前) にあるアプリケーションを自動的にデプロイおよび更新できます。

プロダクション モードでは、自動デプロイメント機能は無効になります。代わりに、WebLogic Server Administration Console または `weblogic.Deployer` ツールを使用する必要があります。デプロイメントの詳細については、『WebLogic Server アプリケーションの開発』の「WebLogic Server デプロイメント」を参照してください。

デフォルトでは、WebLogic Server は開発モードで稼働します。サーバのモードを指定するには、以下のいずれかを実行します。

- `startWebLogic` 起動スクリプトを使用する場合は、スクリプトを編集して、`STARTMODE` 変数を次のように設定する。
 - `STARTMODE = false` 開発モードを有効にする。
 - `STARTMODE = true` プロダクション モードを有効にする。

`startWebLogic` の詳細については、2-14 ページの「スクリプトを使用した管理サーバの起動」を参照してください。

2 WebLogic Server の起動と停止

- コマンドラインに `weblogic.Server` コマンドを直接入力してサーバを起動する場合は、次のように `-Dweblogic.ProductionModeEnabled` オプションを使用する。
 - `-Dweblogic.ProductionModeEnabled=false` 開発モードを有効にする。
 - `-Dweblogic.ProductionModeEnabled=true` プロダクションモードを有効にする。

注意： プロダクションモードまたは開発モードはドメイン全体の設定であり、管理サーバの起動時に指定します。管理対象サーバはすべて、管理サーバと同じモードで実行されます。

管理ポートおよび `weblogic.Admin` ユーティリティの起動引数

管理ポートとは、サーバインスタンスを `STANDBY` 状態で起動する場合、またはドメイン内の管理トラフィックをアプリケーショントラフィックから分離する場合に設定する必要がある個別のポートです。

管理ポートを使用して `weblogic.Admin` ユーティリティからの要求を実行する場合は、以下のことを行う必要があります。

1. 『WebLogic Server ドメイン管理』の「ドメイン全体の管理ポートのコンフィグレーション」で説明するように、ドメイン内の**すべての**サーバインスタンスに対して `SSL` と管理ポートを設定します。
2. すべてのサーバインスタンスに対する `weblogic.Server` コマンドに、以下の起動引数を含めます。

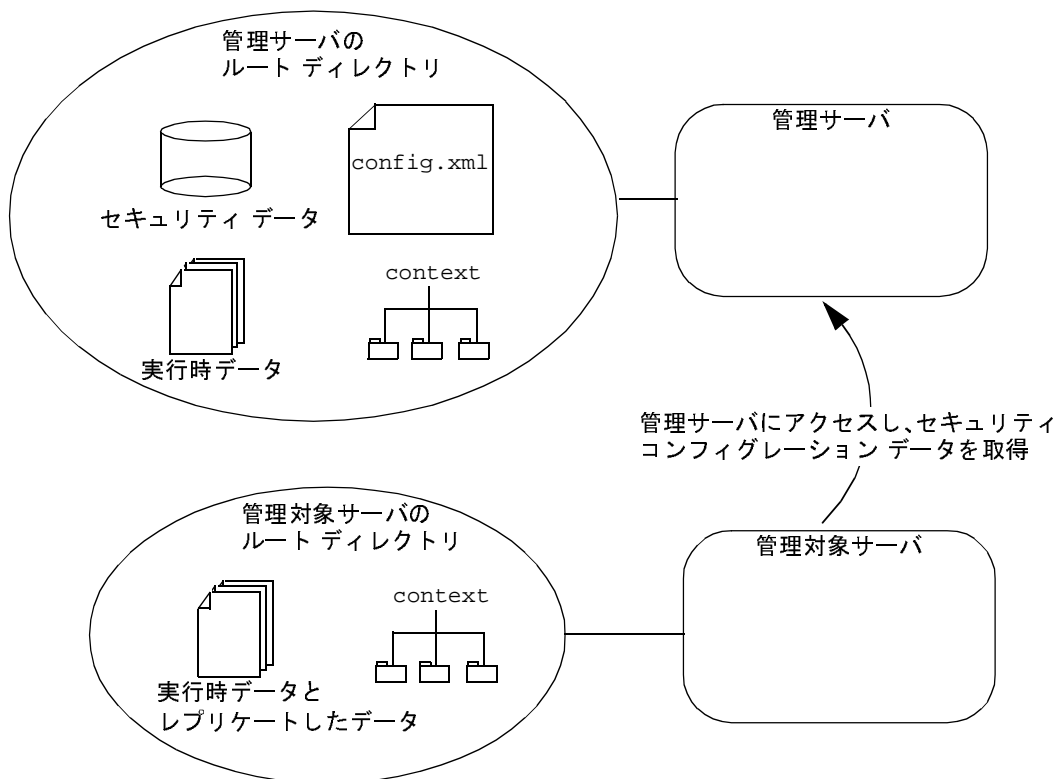
```
-Dweblogic.security.SSL.trustedCAKeystore=path  
-Dweblogic.security.SSL.ignoreHostnameVerification=true
```

サーバのルート ディレクトリ

`WebLogic Server` のすべてのインスタンスは、ルートディレクトリを使用して、実行時データを格納したり、サーバのコンフィグレーション内での相対パス名のコンテキストを提供したりします。

また、管理サーバでは、ドメインのコンフィグレーションデータ (`config.xml` など) やセキュリティリソース (デフォルトの組み込み `LDAP` サーバなど) のリポジトリとして、ルートディレクトリを使用します。一方、管理対象サーバでは、レプリケートした管理データをルートディレクトリに格納します (図 2-2 を参照してください)。

図 2-2 WebLogic Server インスタンスのルート ディレクトリ



WebLogic Server の複数のインスタンスで、同じルート ディレクトリを使用することもできます。ただし、サーバインスタンスでルート ディレクトリを共有する場合、すべての相対ファイル名はユニークでなければなりません。たとえば、2つのサーバが1つのディレクトリを共有している場合、両方のサーバで `.\MyLogFile` を指定すると、一方のサーバインスタンスによってもう一方のサーバインスタンスの `.\MyLogFile` が上書きされます。

管理サーバのルート ディレクトリの判別は以下の手順で行われます。

- サーバの起動コマンドに `-Dweblogic.RootDirectory=path` オプションが含まれている場合は、`path` の値がルート ディレクトリになる。

- `-Dweblogic.RootDirectory=path` が指定されておらず、作業ディレクトリ（起動コマンドを発行したディレクトリ）に `config.xml` ファイルが格納されている場合は、その作業ディレクトリがルートディレクトリになる。
- 上記の2つとも満たされない場合は、`working-directory/config/domain-name` 内で `config.xml` ファイルを検索する。このディレクトリに `config.xml` が格納されている場合は、`working-directory/config/domain-name` がルートディレクトリになります。
- `config.xml` ファイルが見つからない場合は、新たにディレクトリを作成するかどうかを尋ねるメッセージが表示される（2-33 ページの「デフォルト コンフィグレーションを使用したサーバの起動」を参照）。

管理対象サーバのルートディレクトリの判別は以下の手順で行われます。

- サーバの起動コマンドに `-Dweblogic.RootDirectory=path` オプションが含まれている場合は、`path` の値がルートディレクトリになる。
- `-Dweblogic.RootDirectory=path` が指定されていない場合は、作業ディレクトリがルートディレクトリになる。たとえば、`weblogic.Server` コマンドを `c:\config\MyManagedServer` から実行した場合、`c:\config\MyManagedServer` がルートディレクトリになります。

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。

デフォルト コンフィグレーションを使用したサーバの起動

使用している環境で問題が発生し、サーバをデフォルトのコンフィグレーションで起動する場合は、新しい `config.xml` ファイルを生成する方法で WebLogic Server を起動できます。

この新しい `config.xml` ファイルには、コマンドライン引数を使用してデフォルト値をオーバーライドしない限り、デフォルトの設定のみが格納されます。サーバの起動時に入力したユーザ名とパスワードが、デフォルトの管理ユーザとなります。

サーバは新しいドメインの管理サーバとして起動されます。このドメインには他のサーバはなく、また、デプロイメントもサードパーティ製ソリューションもありません。それらは任意の **WebLogic** ドメインに追加するのと同じように追加できます。

WebLogic Server で新しい `config.xml` ファイルを生成するには、`config.xml` ファイルがまだ格納されていないルートディレクトリを使用して管理サーバを起動します。たとえば次のようにします。

1. デフォルトのコンフィグレーション用に新しいディレクトリを作成します。
2. そのディレクトリに移動し、コマンドシェルで次のコマンドを入力します。
`WL_HOME\server\bin\setWLSEnv.cmd` (Windows)
`WL_HOME/server/bin/setWLSEnv.sh` (UNIX)
3. 次のコマンドを入力します。
`java weblogic.Server`
4. 要求されたら、ユーザ名とパスワードを入力します。このユーザがドメインのデフォルトの管理ユーザとなります。
5. 新しいデフォルト コンフィグレーションを作成するよう求められたら、`y` を入力します。

パスワードの再入力が必要とされます。新しいコンフィグレーションでサーバが起動されます。

管理対象サーバの起動

WebLogic Server を管理対象サーバとして実行する前に、以下のことを行っておく必要があります。

- ドメインの管理サーバを起動する。
- 「管理対象サーバのドメインへの追加」で説明されているとおりに、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。

この節では、管理対象サーバのドメインへの追加方法を説明してから、管理対象サーバの起動について説明します。管理対象サーバを起動するには、以下の中から必要なタスクを行います。

- Windows の [スタート] メニューからの管理対象サーバの起動
- スクリプトを使用した管理対象サーバの起動
- コマンドラインからの管理対象サーバの起動
- 管理サーバへの接続のコンフィグレーション
- デフォルトの起動状態の指定
- リモート管理対象サーバの起動
- ドメインまたはクラスタ内のすべての **WebLogic Server** の起動および強制停止

管理サーバが使用できない場合の管理対象サーバの起動については、『**WebLogic Server** ドメイン管理』の「管理サーバにアクセスできない場合の管理対象サーバの起動」を参照してください。

管理対象サーバのドメインへの追加

管理対象サーバをドメインに追加するには、次の手順に従います。

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. サーバが管理サーバとは異なるマシンで実行されている場合は、次の手順に従います。
 - a. **Administration Console** の左ペインで、[マシン] ノードをクリックします。
 - b. 右ペインで [新しい Machine のコンフィグレーション] をクリックします。
 - c. 名前を入力し、[作成] をクリックします。

4. 左ペインで、[サーバ] ノードをクリックします。
5. 右ペインで [新しい Server のコンフィグレーション] をクリックし、次の手順に従います。
 - a. サーバの名前を入力します。

ドメイン内の各サーバの名前はユニークでなければなりません。
 - b. マシンを作成した場合は、この管理対象サーバ用のマシンを選択します。
 - c. [作成] をクリックします。
6. このサーバに管理チャネルを設定する場合は、『WebLogic Server ドメイン管理』の「ドメイン全体の管理ポートのコンフィグレーション」を参照してください。

Windows の [スタート] メニューからの管理対象サーバの起動

コンフィグレーション ウィザードを使用して、Windows コンピュータ上で (管理サーバのコンフィグレーションと共に) 管理対象サーバを作成すると、Windows の [スタート] メニューにドメインをインストールするよう求められます。インストールすることを選択した場合は、次の手順に従って管理対象サーバを起動できます。

Windows のデスクトップから、[スタート | プログラム | BEA WebLogic Platform 7.0 | User Projects | *domain_name* | Start Server] をクリックします。

コマンド ウィンドウが開き、スクリプト

`domain_name\startManagedWebLogic.cmd` が呼び出されます。このスクリプトについては 2-37 ページの「スクリプトを使用した管理対象サーバの起動」で説明します。サーバが正常に起動プロセスを完了すると、コマンド ウィンドウに次のメッセージが表示されます。

```
<Notice> <WebLogicServer> <000360> <Server started in RUNNING mode>
```

スクリプトを使用した管理対象サーバの起動

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長くなる場合があり、エラーが発生しやすくなるので、コマンドをスクリプトに組み込むことをお勧めします。

この節では、以下のタスクについて説明します。

- コンフィグレーション ウィザード スクリプトを使用した管理対象サーバの起動
- 管理対象サーバを起動する独自のスクリプトの作成

WebLogic Server とともにインストールされる **JVM** を使用していない場合は、2-18 ページの「**WebLogic Server** でのデフォルト以外の **JVM** の使用」を参照してください。

コンフィグレーション ウィザード スクリプトを使用した管理対象サーバの起動

コンフィグレーション ウィザードを使用してドメインを作成すると、管理対象サーバの起動に使用できるスクリプトも作成されます。

- `domain_name\startManagedWebLogic.cmd` (Windows)
- `domain_name/startManagedWebLogic.sh` (UNIX および Windows。このスクリプトは Windows 上で MKS および Cygnus BASH UNIX シェル エミュレータをサポート)

`domain_name` は、ドメインを配置したディレクトリです。

管理サーバを起動するスクリプトと同様、`startManagedWebLogic` スクリプトも一部のドメイン固有の変数の値を設定します。ただし、`startManagedWebLogic` では、ドメインの管理サーバのリスン アドレスも指定します。これにより、サーバは管理対象サーバとして実行され、管理サーバからコンフィグレーションを取得できるようになります。

2 WebLogic Server の起動と停止

`startManagedWebLogic` を使用する前に、テキスト エディタでスクリプトを開き、`SERVER_NAME` 変数が、起動する **WebLogic** 管理対象サーバの名前に設定されていることを確認してください。`ADMIN_URL` には、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**) が指定されていることも確認してください。次に例を示します。

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

`startManagedWebLogic` を開いて修正する代わりに、以下のいずれかのコマンドを入力することもできます。

- `domain_name\startWebLogic managed_server_name admin_url`
管理サーバを起動するスクリプトに 2 つのパラメータを渡すことにより、管理対象サーバを起動できます。
- `domain_name\startManagedWebLogic managed_server_name admin_url`
上記の構文は、`startManagedWebLogic` スクリプトの `SERVER_NAME` および `ADMIN_URL` の値をオーバーライドします。

たとえば、次のコマンドは `startWebLogic.cmd` を使用して、ポート **7001** でリスンしている **peach** という管理サーバから **myManagedServer** という管理対象サーバを起動します。

```
c:\user_domains\mydomain\startWebLogic.cmd myManagedServer http://peach:7001
```

`startManagedWebLogic` で指定できる変数と **Java** オプションの詳細については、2-14 ページの「スクリプトを使用した管理サーバの起動」の表 2-1 を参照してください。

管理サーバへの接続のコンフィグレーションに関する詳細については、2-40 ページの「管理サーバへの接続のコンフィグレーション」を参照してください。

サーバが正常に起動プロセスを完了すると、コマンド ウィンドウに次のメッセージが表示されます。

```
<Notice> <WebLogicServer> <000360> <Server started in RUNNING mode>
```

管理対象サーバを起動する独自のスクリプトの作成

Administration Console など別の方法でドメインを作成する場合は、ドメインの管理対象サーバを起動する独自の起動スクリプトを作成できます。独自にスクリプトを作成する手順は、2-15 ページの「管理サーバを起動する独自のスクリプトの作成」で説明されている手順に、次の作業を追加したものになります。

ADMIN_URL 変数の値を設定する必要があります。管理サーバへの接続のコンフィグレーションについては、2-40 ページの「管理サーバへの接続のコンフィグレーション」を参照してください。

サーバが正常に起動プロセスを完了すると、コマンド ウィンドウに次のメッセージが表示されます。

```
<Notice> <WebLogicServer> <000360> <Server started in RUNNING mode>
```

コマンドラインからの管理対象サーバの起動

コマンドラインから WebLogic 管理対象サーバを起動する場合は、管理サーバの起動に使用するコマンドと引数に、管理サーバへの接続をコンフィグレーションする以下のいずれかの引数を追加します。

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

管理サーバへの接続のコンフィグレーションについては、2-40 ページの「管理サーバへの接続のコンフィグレーション」を参照してください。

管理サーバの起動に使用するコマンドと引数については、2-18 ページの「weblogic.Server コマンドの使用」を参照してください。

サーバが正常に起動プロセスを完了すると、コマンド ウィンドウに次のメッセージが表示されます。

```
<Notice> <WebLogicServer> <000360> <Server started in RUNNING mode>
```

管理サーバへの接続のコンフィグレーション

Windows の [スタート] メニュー、スクリプト、`weblogic.Server` コマンド、いずれの方法で管理対象サーバを起動する場合でも、管理サーバの適切なリスンアドレスが指定されていることを確認する必要があります。管理対象サーバは、このアドレスを使用して管理サーバからコンフィグレーションを取得します。

注意： 初めて管理対象サーバを起動するときは、管理サーバにアクセスできなければなりません。その後は、管理サーバが使用できない場合でも起動するよう管理対象サーバをコンフィグレーションできます。詳細については、『WebLogic Server ドメイン管理』の「管理サーバにアクセスできない場合の管理対象サーバの起動」を参照してください。

リスンアドレスは、以下の形式のいずれかを使用して指定できます。

■ `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` は管理サーバのデフォルトの非 SSL 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。

この形式では、管理対象サーバはデフォルトのプロトコル (t3) を使用して管理サーバにアクセスします。デフォルトのプロトコルを変更するには、次の手順に従います。

- a. 管理サーバを起動します。
- b. **Administration Console** の左ペインで、[サーバ] ノードを展開し、管理対象サーバの名前をクリックします。
- c. 右ペインで [接続 | プロトコル] をクリックします。
- d. [デフォルト プロトコル] フィールドで、サーバのデフォルトのプロトコルを指定します。

■ `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` は管理サーバのデフォルトの非 SSL 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。

管理サーバのホスト IP アドレス、名前、およびデフォルトのリスン ポートを確認するには、コマンドシェルで管理サーバを起動します。サーバが正常

に起動プロセスを完了すると、標準出力に次のようなメッセージが他のメッセージと共に出力されます。

```
<Apr 19, 2002 9:24:19 AM EDT> <Notice> <WebLogicServer>
<000355> <Thread "Listen Thread.Default" listening on port
7001, ip address 11.12.13.141>
```

...

```
<Apr 19, 2002 9:24:19 AM EDT> <Notice> <WebLogicServer>
<000331> <Started WebLogic Admin Server "myserver" for domain
"mydomain" running in Development Mode>
```

IP アドレスおよびリスン ポートの値は、**Administration Console** からサーバの [**コンフィグレーション | 一般**] タブにアクセスして変更できます。

- `https://host:port`

管理対象サーバと管理サーバに対してセキュア ソケット レイヤ (SSL) 通信をコンフィグレーションしている場合は、この形式を使用できます。*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 SSL リスン ポートです。

管理ポートを使用するように管理サーバを設定している場合は、*port* には管理ポートを指定する必要があります。

SSL の有効化については、**Administration Console** オンライン ヘルプの「**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。管理ポートの詳細については、『**WebLogic Server** ドメイン管理』の「ドメイン全体の管理ポートのコンフィグレーション」を参照してください。

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。

デフォルトの起動状態の指定

`weblogic.Server` コマンド (またはこのコマンドを実行するスクリプト) で起動する場合に、デフォルトでスタンバイ (STANDBY) 状態で起動するようサーバを設定するには、次の手順に従います (サーバをスタンバイ状態で起動するには、サーバに管理ポートを設定する必要があります)。

1. **Administration Console** の左ペインで [**サーバ**] ノードを展開します。[**サーバ**] ノードの下に、サーバのリストが表示されます。

2. 左ペインで特定のサーバを選択します。
3. [一般] タブの [起動モード] フィールドで、「STANDBY」と入力します。
4. [適用] をクリックして変更を保存します。

リモート管理対象サーバの起動

管理対象サーバのホストマシン上でノードマネージャが実行されている場合は、Administration Console または `weblogic.Admin` ユーティリティを使用してリモートホストから管理対象サーバを起動できます。ノードマネージャは、WebLogic Server 付属のスタンドアロンの Java プログラムであり、リモートの管理対象サーバを起動および停止できます。

Administration Console からのリモートサーバの起動については、Administration Console オンラインヘルプの「サーバの起動」および「STANDBY 状態でのサーバの起動」を参照してください。

`weblogic.Admin` コマンドユーティリティの使用方法については、B-26 ページの「START」および B-28 ページの「STARTINSTANDBY」を参照してください。

ノードマネージャについては、『WebLogic Server ドメイン管理』の「ノードマネージャによるサーバの可用性の管理」を参照してください。

ドメインまたはクラスタ内のすべての WebLogic Server の起動および強制停止

管理対象サーバのホストマシン上でノードマネージャが実行されている場合は、Administration Console を使用して、ドメインまたは特定のクラスタ内のすべての管理対象サーバを起動できます。管理サーバを Administration Console から起動することはできません。

また、Administration Console を使用して、ドメインまたはクラスタ内のすべての WebLogic Server を強制停止することもできます。強制停止コマンドは、管理対象サーバおよび管理サーバに対して強制的な停止を開始します。ノードマネージャは必要ありません。

この節では、以下のタスクについて説明します。

- ドメイン内のすべての管理対象サーバの起動
- クラスタ内のすべての管理対象サーバの起動
- ドメイン内のすべてのサーバの強制停止
- クラスタ内のすべてのサーバの強制停止

ノードマネージャについては、『WebLogic Server ドメイン管理』の「ノードマネージャによるサーバの可用性の管理」を参照してください。

ドメイン内のすべての管理対象サーバの起動

アクティブなドメイン内のすべての管理対象サーバを起動するには、次の手順に従います。

1. ドメインの管理サーバを起動します。
2. ドメイン内のすべてのマシンでノードマネージャを起動します。詳細については、『WebLogic Server ドメイン管理』の「ノードマネージャの起動」を参照してください。
3. Administration Console の左ペインで、アクティブなドメインの名前を右クリックします。
4. **[このドメインを開始 ...]** を選択します。
5. 強制停止の確認を求めるメッセージが表示されたら **[はい]** をクリックします。
Administration Console に、そのドメイン内の各 WebLogic Server の名前を表示したページが表示されます。
6. サーバの起動処理の結果を参照するには、そのサーバの名前をクリックします。

クラスタ内のすべての管理対象サーバの起動

クラスタ内のすべての管理対象サーバを起動するには、次の手順に従います。

1. ドメインの管理サーバを起動します。

2. クラスタ内のすべてのマシンでノード マネージャを起動します。詳細については、『WebLogic Server ドメイン管理』の「ノード マネージャの起動」を参照してください。
3. Administration Console の左ペインで、クラスタの名前を右クリックします。
4. **[このクラスタを開始 ...]** を選択します。
5. 強制停止の確認を求めるメッセージが表示されたら **[はい]** をクリックします。
Administration Console に、そのクラスタ内の各管理対象サーバの起動タスクの状態を表示する **[タスク]** ページが表示されます。
6. 起動タスクの状態の詳細を参照するには、**[タスク]** ページでその起動タスクの名前をクリックしてから **[詳細]** タブをクリックします。

ドメイン内のすべてのサーバの強制停止

ドメイン内のすべてのサーバに対して強制停止を開始するには、次の手順に従います。

1. Administration Console の左ペインで、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 ...]** を選択します。
3. 強制停止の確認を求めるメッセージが表示されたら **[はい]** をクリックします。
管理サーバおよび管理対象サーバは即座にすべての作業を中断し、停止します。管理対象サーバが応答しない場合や、ノード マネージャを使用してサーバを起動した場合は、ノード マネージャによってサーバが強制停止されません。
4. ドメインが強制停止されたことを確認するには、管理サーバを実行しているシェル プロセスで出力を参照します。停止シーケンスが開始された後、プロセスが中止されたことを示す ALERT メッセージが表示されます。

クラスタ内のすべてのサーバの強制停止

クラスタ内のサーバに対して強制停止を開始するには、次の手順に従います。

1. Administration Console の左ペインで、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 ...]** を選択します。
3. 強制停止の確認を求めるメッセージが表示されたら **[はい]** をクリックします。

クラスタ内のすべてのサーバは即座にすべての作業を中断し、停止します。管理対象サーバが応答しない場合や、ノード マネージャを使用してサーバを起動した場合は、ノード マネージャによってサーバが強制停止されます。
4. クラスタが強制停止されたことを確認するには、次のいずれかの手順に従います。
 - 管理サーバがクラスタに含まれていない場合は、左ペインで **[タスク]** ノードをクリックします。**[タスク]** ページでその停止タスクの名前をクリックしてから **[詳細]** タブをクリックします。
 - 管理サーバがクラスタに含まれている場合は、管理サーバを実行しているシェル プロセスで出力を参照します。停止シーケンスが開始された後、プロセスが中止されたことを示す **ALERT** メッセージが表示されます。

WebLogic Server の停止

以下のいずれかの方法で、WebLogic Server を停止できます。

- Administration Console を使用する。
 - 「サーバの停止」
 - 「サーバの強制停止」
- `weblogic.Admin` ユーティリティを使用する。
 - B-24 ページの「SHUTDOWN」
 - B-11 ページの「FORCESHUTDOWN」

正常な停止を開始すると、サーバはサブシステムに作業中のすべてのリクエストを完了するように指示します。サブシステムが作業を完了した後、サーバは停止します。

強制停止を開始すると、サーバはサブシステムに作業中のリクエストを即座に中断するように指示します。管理対象サーバが応答しない場合や、ノード マネージャを使用してサーバを起動した場合は、ノード マネージャによってサーバが強制停止されます。

サーバは、すべてのサブシステムが正常に停止するのを設定されている秒数まで待ちます。時間が経過すると、サーバは以下のいずれかの処理を行います。

- サーバが実行中 (RUNNING) 状態のときにタイムアウトした場合は、サーバは標準出力にメッセージを返す (その後、サーバを停止するには、強制停止コマンドを実行する必要があります)。
- サーバがスタンバイ (STANDBY) 状態または停止中 (SHUTTING_DOWN) 状態のときにタイムアウトした場合は、サーバはすべてのプロセスを強制停止して、停止する。

デフォルトの秒数の変更については、**Administration Console** オンライン ヘルプの「ライフサイクル オペレーションのタイムアウト設定」を参照してください。

起動クラスと停止クラスのコンフィグレーション

起動クラスと停止クラスを使用すると、サーバの起動時や正常な停止時にタスクを実行するよう、**WebLogic Server** をコンフィグレーションできます。起動クラスは、**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。

デフォルトでは、起動クラスは、他のサーバ サブシステムが初期化され、サーバがモジュールをデプロイした後に、自動的にロードされて実行されます。起動クラスに対しては、デフォルトをオーバーライドして、サーバが **JDBC** 接続プール、**Web** アプリケーション、および **EJB** をデプロイする前にロードされ実行されるよう指定することもできます。

停止クラスは、**Administration Console** または `weblogic.admin.shutdown` コマンドで **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る **Java** プログラムです。サーバが起動クラスと停止クラスを呼び出す時期に関する詳細については、2-1 ページの「サーバのライフサイクル」を参照してください。

起動クラスまたは停止クラスを使用するには、それらのクラスをコンフィグレーションし、サーバに割り当てる必要があります。Administration Console でクラスをコンフィグレーションするには、Administration Console オンラインヘルプの「起動クラスと停止クラス」を参照してください。

WebLogic Server インスタンスの Windows サービスとしての設定

Windows ホスト コンピュータの起動時に WebLogic Server インスタンスを自動的に起動する場合は、サーバを Windows サービスとして設定します。

Windows サービスとして設定した各サーバインスタンスに対して、Windows レジストリの HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services にキーが作成されます。レジストリ エントリには、サーバの名前などの起動引数の情報が格納されます。

Windows ホストを起動すると、Windows オペレーティング システムの一部である Windows サービス コントロール マネージャ (SCM) は、Windows レジストリ キーの情報を使用して、weblogic.Server メインクラスを呼び出します。Windows SCM は、管理サーバの起動にノード マネージャを使用するようにはコンフィグレーションできません。したがって、Windows サービスとして実行されるサーバでは、ノード マネージャのモニタ機能および自動再起動機能を使用できません。

以下のタスクにより、Windows サービスとして実行される WebLogic Server インスタンスを設定および管理できます。

- Windows サービスの設定: 主な手順
- 設定の確認
- コントロール パネルを使用したサーバインスタンスの停止と再起動
- Windows サービスとしてのサーバの削除
- Windows サービスとしてのサーバ設定に対する起動資格の変更

Windows サービスの設定：主な手順

Windows サービスを設定するには、次の手順に従います。

1. 以下のいずれかを実行します。
 - ドメイン コンフィグレーション ウィザードでサーバを **Windows** サービスとしてインストールするプロンプトが表示されたら、[はい]を選択します。

ドメイン コンフィグレーション ウィザードの一部のドメイン テンプレートでは、サーバを **Windows** サービスとして設定することを求められます。[はい]を選択すると、ウィザードはサーバを **Windows** サービスとしてインストールします。管理サーバおよび管理対象サーバの作成にウィザードを使用した場合、**Windows** サービスは管理サーバについてのみ作成されます。ウィザードはまた、サーバ固有のスクリプトを作成します。これを修正して、他のサーバを **Windows** サービスとしてインストールするために使用できます。このスクリプトは `domain-name\installService.cmd` という名前を付けられます。`domain-name` は、作成したドメインの名前を表します。
 - サーバ固有の変数の値を設定するスクリプトを作成してから、**WebLogic Server** マスター スクリプトを呼び出します。詳細については、2-49 ページの「サーバ固有のスクリプトの作成」を参照してください。
2. 管理対象サーバを **Windows** サービスとしてインストールする場合、サーバ固有のスクリプトに追加の変数を設定する必要があります。詳細については、2-52 ページの「管理対象サーバのその他の値の設定」を参照してください。
3. 同じコンピュータ上で管理サーバと管理対象サーバの両方を **Windows** サービスとして実行するように設定する場合、管理サーバの起動サイクル完了後にはのみ管理対象サーバが起動するようにマスター スクリプトを修正することができます。詳細については、2-53 ページの「管理サーバより後に管理対象サーバを起動する要求」を参照してください。
4. **Windows** コントロール パネルを使用して **Windows** サービスを停止する際に、サーバインスタンスを正常に停止するには、**Java** クラスを作成して、**Windows SCM** がそのクラスを呼び出すようにマスター スクリプトを修正します。詳細については、2-55 ページの「コントロール パネルからの正常な停止の有効化」を参照してください。

5. サーバ インスタンスが標準出力および標準エラーに出力するメッセージ (スタック トレースおよびスレッド ダンプを含む) を表示できるようにするには、標準出力および標準エラーをファイルにリダイレクトします。詳細については、2-58 ページの「標準出力および標準エラーのファイルへのリダイレクト」を参照してください。
6. WebLogic Server インスタンスで呼び出す Java クラスを他にも作成した場合は、これらをサーバのクラスパスに追加する必要があります。詳細については、クラスのクラスパスへの追加を参照してください。
7. サーバ固有のスクリプトを実行します。詳細については、2-63 ページの「サーバ固有のスクリプトの実行」を参照してください。

サーバ固有のスクリプトの作成

ドメイン コンフィグレーション ウィザードでドメインのサーバ固有のスクリプトが既に作成されていない場合は、自分で作成する必要があります。このスクリプトでは、サーバインスタンス名などサーバ固有の情報を識別する変数値を設定します。その後、スクリプトはマスター スクリプト `WL_HOME\server\bin\installSvc.cmd` を呼び出します。`WL_HOME` は、WebLogic Server をインストールしたディレクトリです。マスター スクリプトは `beasvc` ユーティリティを呼び出します。これにより、Windows レジストリにキーが追加されます。

注意： `beasvc` の詳細を参照するには、コマンド プロンプトでコマンド `WL_HOME\server\bin\beasvc -help` を入力します。`WL_HOME` は、WebLogic Server のインストール先ディレクトリです。

サーバ固有のスクリプトの例は、2-52 ページのコードリスト 2-1 「サーバを Windows サービスとして設定するためのスクリプト例」を参照してください。

サーバ固有のスクリプトを作成するには、次の手順に従います。

1. ドメインの管理サーバのルート ディレクトリ (ドメインの `config.xml` ファイルが格納されているディレクトリ) に、テキスト ファイルを作成します。
2. 以下の**必須** batch コマンドをテキスト ファイルに追加します。コマンドはそれぞれ別の行に記述します。
 - SETLOCAL

2 WebLogic Server の起動と停止

これは、バッチ ファイル内の環境変数のローカライゼーションを開始する batch コマンドです。

- `set DOMAIN_NAME=domain-name`

`domain-name` は、使用している **WebLogic Server** の名前を表します。

- `set USERDOMAIN_HOME=absolute-pathname`

`absolute-pathname` は、管理サーバのルート ディレクトリ (ドメインのコンフィグレーション ファイルが格納されているディレクトリ) の絶対パス名を表します。サーバのルート ディレクトリの詳細については、2-31 ページの「サーバのルート ディレクトリ」を参照してください。

- `set SERVER_NAME=server-name`

`server-name` は、**Windows** サービスとして設定する既存のサーバインスタンスの名前を表します。

3. 以下の**省略可能な** batch コマンドをテキスト ファイルに追加します。各コマンドは別々の行に入れます。

- `set WLS_USER=username`
`set WLS_PW=password`

`username` は、サーバインスタンスを起動する特権を持つ既存ユーザ名です。`password` は、そのユーザのパスワードです。`beasvc` ユーティリティは、ログイン資格を暗号化し、**Windows** レジストリに格納します。

これは、サーバインスタンス起動時にユーザ名とパスワードの入力が求められないようにするための、2つある方法のうちの1つです。この方法の短所は、サーバインスタンスのユーザ名またはパスワードを変更すると、**Windows** サービスを削除して新しいユーザ名とパスワードで新しい**Windows** サービスを設定しなければならないことです。この方法を使用する代わりに、起動 ID ファイルを使用できます。起動 ID ファイルを使うと、**Windows** サービスを修正する必要なしにログイン資格を変更できます。詳細については、2-8 ページの「ユーザ名とパスワードのプロンプトの回避」を参照してください。

- `set STARTMODE=[true]`

`STARTMODE` 変数が `true` に設定されていると、サーバインスタンスはプロダクション モードで起動します。指定されていないか、`false` に設定されている場合は、サーバは開発モードで起動します。開発モードおよびプロダクション モードの詳細については、2-29 ページの「開発モードとプロダクション モード」を参照してください。

- `set JAVA_OPTIONS=java-options`

java-options は、Java 仮想マシン (JVM) に渡す 1 つまたは複数の Java 引数です。複数の引数はスペースで区切ります。WebLogic Server 固有の Java オプションのリストについては、2-20 ページの「weblogic.Server コマンドの構文」を参照してください。使用する JVM は追加のオプションをサポートしており、JVM ベンダによってドキュメント化されています。

- `set JAVA_VM=-JVM-mode`

JVM-mode は、JVM を実行するモードを示すテキスト文字列です。指定する値は、使用している JVM によって変わります。JRockit JVM を使用している場合、デフォルト値は、`-jrockit` です。詳細については、『JRockit User Guide』の「Starting and Configuring the JRockit JVM」を参照してください。

サポートされている JVM の一覧については、http://edocs.beasys.co.jp/e-docs/wls/certifications/certs_700/overview.html の「サポート対象プラットフォームの一覧」を参照してください。

- `set MEM_ARGS=[-XmsNumberm] [-XmxNumberm]`

Number は、メガバイト (MB) による数値です。`-XmsNumberm` 引数では、JVM の最小ヒープ サイズを設定します。`-XmxNumberm` では、最大ヒープ サイズを設定します。デフォルトでは、最小ヒープ サイズは 23 MB、最大ヒープ サイズは 200 MB です。

4. 以下の **必須** コマンドを、スクリプトの最後に追加します。

- `call "WL_HOME\server\bin\installSvc.cmd"`

WL_HOME は WebLogic Server をインストールしたディレクトリの絶対パス名です。このコマンドは、WebLogic Server のマスター スクリプトを呼び出します。

- `ENDLOCAL`

これは、バッチ ファイル内の環境変数のローカライゼーションを終了する `batch` コマンドです。

5. テキスト ファイルの拡張子を `.cmd` にして保存します。デフォルトでは、Windows のコマンドプロンプトで拡張子 `.cmd` と `batch` ファイルが関連付けられています。

管理対象サーバのその他の値の設定

管理対象サーバを Windows サービスとしてインストールする場合は、ドメインの管理サーバの場所を指定する変数を含める必要があります。管理対象サーバは、管理サーバに接続してコンフィグレーションデータを取得する必要があります。

管理対象サーバのその他の値を設定するには、以下の手順に従います。

1. テキスト エディタで、サーバ固有のスクリプトを開きます。

ドメイン コンフィグレーション ウィザードで作成したスクリプトを修正する場合は、`domain-name\installService.cmd` (`domain-name` は作成したドメインの名前) のコピーを作成し、そのコピーを開くことをお勧めします。

2. テキスト ファイルで、SETLOCAL コマンドと call コマンドの間に、以下のコマンドを作成します。

```
set ADMIN_URL=protocol://listen-address:listen-port
```

各要素の説明は次のとおりです。

- `protocol` は、http または https
- `listen-address` は、管理サーバのリスンアドレス
- `listen-port` は、管理サーバのポート

詳細については、2-40 ページの「管理サーバへの接続のコンフィグレーション」を参照してください。

たとえば、コードリスト 2-1 の太字部分を参照してください。

3. サーバ固有のスクリプトへの変更を保存します。

コードリスト 2-1 サーバを Windows サービスとして設定するためのスクリプト例

```
echo off
SETLOCAL

set DOMAIN_NAME=myWLSdomain
set USERDOMAIN_HOME=d:\bea\user_projects\myWLSdomain
set SERVER_NAME=myWLSserver
set STARTMODE=true
set JAVA_OPTIONS=-Dweblogic.Stdout="d:\bea\user_projects\myWLSdomain\stdout.txt"
-Dweblogic.Stderr="d:\bea\user_projects\myWLSdomain\stderr.txt"
```

```
set ADMIN_URL=http://adminserver:7501
set MEM_ARGS=-Xms40m -Xmx250m
call "d:\bea\weblogic700\server\bin\installSvc.cmd"
ENDLOCAL
```

管理サーバより後に管理対象サーバを起動する要求

同じコンピュータ上で管理サーバと管理対象サーバの両方を Windows サービスとして実行するように設定する場合、管理サーバが起動した後にだけ管理対象サーバが起動するように指定することができます。

管理対象サーバが管理サーバの Windows サービスより後に起動するように要求するには、以下の手順に従います。

1. WL_HOME\server\bin\installSvc.cmd マスター スクリプトのバックアップコピーを作成します。
2. すでに管理サーバを Windows サービスとしてインストールしている場合は、そのサービスを削除してください。詳細については、2-66 ページの「Windows サービスとしてのサーバの削除」を参照してください。
3. 管理サーバを Windows サービスとしてインストール (または再インストール) する前に、以下を実行してください。

- a. テキスト エディタで、WL_HOME\server\bin\installSvc.cmd マスター スクリプトを開きます。

このスクリプトの最後のコマンドにより `beasvc` が呼び出されます。これは、Windows レジストリを変更する WebLogic Server ユーティリティです。

- b. `installSvc.cmd` で、`beasvc` ユーティリティを呼び出すコマンドに、以下の引数を追加します。

```
-delay:delay_milliseconds
```

この引数には、サービスの状態が Windows SCM によって `SERVER_START_PENDING` から `STARTED` に変更されるのを待機する時間をミリ秒単位で指定します。

たとえば、管理サーバが起動サイクルを完了してリクエストのリスンを開始するのに 2 分かかる場合は `-delay=120000` と指定します。この場

合、Windows ホスト コンピュータを起動すると、Windows SCM は状態として SERVER_START_PENDING を 2 分間報告した後 STARTED に変更します。

管理サーバの beasvc 呼び出しは、変更後には次のようになります。

```
"%WL_HOME%\server\bin\beasvc" -install
-svcname:"%DOMAIN_NAME%\_%SERVER_NAME%"
-delay:120000
-javahome:"%JAVA_HOME%" -execdir:"%USERDOMAIN_HOME%"
-extrapath:"%WL_HOME%\server\bin" -password:"%WLS_PW%"
-cmdline:%CMDLINE%
```

beasvc の詳細を参照するには、コマンドプロンプトでコマンド `WL_HOME\server\bin\beasvc -help` を入力します。WL_HOME は、WebLogic Server のインストール先ディレクトリです。

4. 管理サーバ Windows サービスをインストールします。
5. **管理対象サーバ**を Windows サービスとしてインストールする前に、以下を実行します。
 - a. テキスト エディタで、WL_HOME\server\bin\installSvc.cmd マスター スクリプトを開きます。
 - b. installSvc.cmd で、beasvc ユーティリティを呼び出すコマンドに、以下の引数を追加します。

```
-depend:Administration-Server-service-name
```

Administration-Server-service-name は、管理サーバ Windows サービスの名前です。サービス名を確認するには、Windows サービスのコントロール パネルを参照します。

このオプションを指定すると、管理サーバ Windows サービスの状態が STARTED になってから、管理対象サーバ Windows サービスが起動されます。

たとえば、管理対象サーバの beasvc 呼び出しは、変更後には次のようになります。

```
"%WL_HOME%\server\bin\beasvc" -install
-svcname:"%DOMAIN_NAME%\_%SERVER_NAME%"
-depend:"myDomain_myAdminServer"
-javahome:"%JAVA_HOME%" -execdir:"%USERDOMAIN_HOME%"
-extrapath:"%WL_HOME%\server\bin" -password:"%WLS_PW%"
-cmdline:%CMDLINE%
```

管理対象サーバ Windows サービスの状態が STARTED と報告されたときにコンフィグレーションしたい場合は、サービスに `-delay:delay_milliseconds` オプションを追加することも可能です。

コントロール パネルからの正常な停止の有効化

デフォルトでは、Windows コントロール パネルを使用してサーバインスタンスを停止すると、Windows サービス コントロール マネージャ (SCM) がサーバの Java 仮想マシン (JVM) を強制停止します。JVM を強制停止すると、サーバは直ちにすべての処理を終了します。セッション データはすべて失われます。サーバが `config.xml` ファイルへの書き込みを行っている間に管理サーバの JVM を強制停止すると、`config.xml` ファイルが破損する可能性があります。

Windows コントロール パネルから正常に停止を実行できるようにするには、次の手順に従います。

1. `weblogic.management.runtime.ServerRuntime.shutdown()` メソッドを呼び出す Java クラスを作成します。

このメソッドにより、サーバは処理中の全作業を完了後に正常に停止されます。そのようなクラスの例は、2-57 ページの「サーバインスタンスを停止する Java クラス」を参照してください。
2. `WL_HOME\server\bin\installSvc.cmd` マスター スクリプトのバックアップ コピーを作成します。
3. テキスト エディタで、`WL_HOME\server\bin\installSvc.cmd` マスター スクリプトを開き、以下の処理を実行します。
 - a. 作成したクラスを `set CLASSPATH` 文に追加します。

たとえば、`c:\myJar` という名前のファイルにクラスをアーカイブした場合、修正後の文は以下のようになります。

```
set
CLASSPATH=%JAVA_HOME%\lib\tools.jar;%WL_HOME%\server\lib\weblogic_sp.jar;%WL_HOME%\server\lib\weblogic.jar;c:\myJar;%CLASSPATH%
```
 - b. スクリプトの最終行に、`beasvc` ユーティリティを呼び出す以下の引数を追加します。

```
-stopclass:javaclass
```

`javaclass` は、作成したクラスの完全なクラスパス名です。この引数により `javaclass` がロードされ、その後 `public void static stop()` メソッドが呼び出されます。

たとえば、2-57 ページの「サービンスタンスを停止する Java クラス」に記載のクラスを `com.myClasses` にパッケージ化する場合、修正後の `beasvc` コマンドは以下のようになります。

修正後の `beasvc` の呼び出しは以下に類似したものとなります。

```
%WL_HOME%\server\bin\beasvc" -install
-svcname: "%DOMAIN_NAME%_%SERVER_NAME%"
-stopclass: com.myClasses.ServerStopper
-javahome: "%JAVA_HOME%" -execdir: "%USERDOMAIN_HOME%"
-extrapath: "%WL_HOME%\server\bin" -password: "%WLS_PW%"
-cmdline: %CMDLINE%
```

`beasvc` の詳細を参照するには、コマンドプロンプトでコマンド `WL_HOME\server\bin\beasvc -help` を入力します。`WL_HOME` は、WebLogic Server のインストール先ディレクトリです。

4. Windows SCM で指定されているデフォルトのタイムアウト値の修正を検討します。

デフォルトでは、Windows 2000 のコントロールパネルを使用して Windows サービスを停止する場合、Windows SCM はサービスの停止を 30 秒間待機してからサービスを強制終了し、システムのイベントログにタイムアウトメッセージを出力します。

`-stopclass` を使用してサーバを正常に停止する場合は、サーバが処理を正常に終了するまでにかかる時間は 30 秒を超える可能性があります。

Windows 2000 でタイムアウト期間をコンフィグレーションするには、次のレジストリ キーに `ServicesPipeTimeout` という名前の `REG_DWORD` レジストリ値を作成します。

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control
```

キー値の単位はミリ秒とします。

この値は、Windows オペレーティング システムの起動中にレジストリから読み込まれ、インストールされているすべてのサービスに影響を及ぼします。

5. WebLogic Server のマスター スクリプトへの変更を保存します。

サーバインスタンスを停止する Java クラス

以下の Java クラスは、サーバインスタンスの停止に **Java Management Extensions (JMX)** を使用します。各サーバは、**JMX Managed Bean (MBean)** を使用して、管理の属性および操作を公開します。そのような **MBean** の 1 つである **ServerRuntime** では、サーバを正常に停止する `shutdown()` メソッドが公開されています。

コードリスト 2-2 に記載のクラスは、管理 **MBeanHome** インタフェースを使用します。これは、ドメイン内のすべてのサーバインスタンスの **ServerRuntime MBean** 操作を取得し、呼び出すことができます。

JMX プログラミングの詳細については、『**WebLogic JMX Service プログラマーズガイド**』を参照してください。

コード リスト 2-2 サーバインスタンスを停止する Java クラス

```
import java.util.Set;
import java.util.Iterator;
import java.rmi.RemoteException;
import javax.naming.*;
import weblogic.jndi.Environment;

import weblogic.management.MBeanHome;
import javax.management.ObjectName;
import weblogic.management.WebLogicMBean;
import weblogic.management.configuration.ServerMBean;
import weblogic.management.runtime.ServerRuntimeMBean;
import weblogic.management.WebLogicObjectName;

public class ServerStopper {
    public static void stop() throws Exception {
        MBeanHome home = null;

        // 管理サーバの url
        String url = "t3://qall3:7001";
        String username = "system";
        String password = "gumby1234";
        ServerRuntimeMBean serverRuntime = null;
        Set mbeanSet = null;
        Iterator mbeanIterator = null;

        try {
            // ContextClassLoader を設定してアサーションを防止
            URL[] urls = { new File("/").toURL() };
            Thread.currentThread().setContextClassLoader(new
                URLClassLoader(urls));
```

```
Environment env = new Environment();
env.setProviderUrl(url);
env.setSecurityPrincipal(username);
env.setSecurityCredentials(password);
Context ctx = env.getInitialContext();
home = (MBeanHome)
    ctx.lookup("weblogic.management.adminhome");
mbeanSet = home.getMBeansByType("ServerRuntime");
mbeanIterator = mbeanSet.iterator();

while(mbeanIterator.hasNext()) {
    serverRuntime = (ServerRuntimeMBean)mbeanIterator.next();
    if(serverRuntime.getState().equals("RUNNING")) {
        serverRuntime.shutdown();
    }
}

} catch (Exception e) {
    e.printStackTrace();
}
}
```

標準出力および標準エラーのファイルへのリダイレクト

WebLogic Server インスタンスを Windows サービスとしてインストールした場合、デフォルトでは、サーバまたはその JVM が標準出力および標準エラーに出力したメッセージを表示することはできません。

これらのメッセージを表示できるようにするには、標準出力および標準エラーをファイルにリダイレクトする必要があります。

1. `WL_HOME\server\bin\installSvc.cmd` マスター スクリプトのバックアップ コピーを作成します。
2. テキスト エディタで、`WL_HOME\server\bin\installSvc.cmd` マスター スクリプトを開きます。
3. `installSvc.cmd` スクリプトの最後のコマンドは `beasvc` ユーティリティを呼び出します。`beasvc` コマンドの最後に、次のコマンド オプションを追加します。

`-ejbJar pathname`

`pathname` は、サーバの標準出力と標準エラーのメッセージを格納するファイルの絶対パスとファイル名です。

修正後の `beasvc` の呼び出しは以下のようになります。

```
%WL_HOME%\server\bin\beasvc" -install
-svcname:"%DOMAIN_NAME%_%SERVER_NAME%"
-javahome:"%JAVA_HOME%" -execdir:"%USERDOMAIN_HOME%"
-extrapath:"%WL_HOME%\server\bin" -password:"%WLS_PW%"
-cmdline:%CMDLINE%
-log:"d:\bea\user_projects\myWLSdomain\myWLSserver-stdout.txt"
```

4. デフォルトでは、Windows サービスは 24 時間おきにメッセージを *pathname-yyyy_mm_dd-hh_mm_ss* という名前のファイルに保管します。新しいメッセージは、前の手順で指定したファイルに収集されます。

デフォルト動作の変更については、2-59 ページの「デフォルトのローテーション基準の変更」を参照してください。

サービスをインストールして Windows ホストを再起動したら、以下のいずれかを実行して標準出力または標準エラーに書き込まれたメッセージを表示します。

- 指定したファイルのコピーを作成し、そのファイルを表示する。Windows ファイル システムでは、現在開いているファイルに書き込みを行うことはできません。
- メッセージをファイルに出力されると同時に表示するには、コマンドプロンプトを開いて DOS コマンド `tail -f stdout-filename` を実行する。

デフォルトのローテーション基準の変更

デフォルトでは、Windows サービスは 24 時間おきにメッセージを *pathname-yyyy_mm_dd-hh_mm_ss* という名前のファイルに保管します。新しいメッセージは、サービスの設定時に指定したファイルに収集されます。

時間間隔を変更したり、時間間隔の代わりにメッセージファイルのサイズに基づいてローテーションが行われるように設定したりすることもできます。

Windows サービスがメッセージ ファイルをローテーションするデフォルト基準を変更するには、次の手順を行います。

1. Windows サービスが動作している場合は、それを停止します。
2. `-log: pathname` 引数で指定したファイルを編集します。ファイルが存在しない場合は、新たに作成します。

たとえば、前節の手順 3. のサンプル コマンドを発行した場合は、`d:\bea\wlserver6.1\config\mydomain\myserver-stdout.txt` という名前のファイルを作成します。

3. 以下のいずれかを実行します。

- サイズに関係なく指定の時間間隔でメッセージファイルがローテーションされるようにするには、以下の文をそれぞれ別の行としてファイルの先頭に追加します (最後の行を入力した後は必ず [Enter] または [Return] を押してください)。

```
# ROTATION_TYPE = TIME
# TIME_START_DATE = date-in-required-format
# TIME_INTERVAL_MINS = number-of-minutes
```

`TIME_START_DATE` は、最初のローテーションがいつ行われるのかを指定します。指定された時刻が既に過ぎている場合、最初のローテーションは `TIME_INTERVAL_MINS` で指定された時間間隔が経過したときに行われます。開始時刻の指定には、`Month Day Year Hour:Minutes:Seconds` という形式を使用する必要があります。

`Month` は、グレゴリアン暦の月を英語で表現したときの最初の 3 文字です。

`Day` は、グレゴリアン暦の 2 桁の日付です。

`Year` は、グレゴリアン暦の 4 桁の年です。

`Hour:Minutes:Seconds` は、24 時間形式の時刻を表します。

`TIME_INTERVAL_MINS` は、Windows サービスがファイルをローテーションする頻度 (分間隔) を指定します。

次に例を示します。

```
# ROTATION_TYPE = TIME
# TIME_START_DATE = Jul 17 2003 05:25:30
# TIME_INTERVAL_MINS = 1440
```

時間間隔が経過すると、Windows サービスはファイルを `pathname-yyyy_mm_dd-hh_mm_ss` として保存します。次に、`pathname` という名前の新しいファイルを作成します。この新しいファイルには最初に指定したすべてのヘッダが含まれており、新しい標準出力と標準エラーのメッセージが収集されます。

`ROTATION_TYPE = TIME` を指定しても、他の行を挿入していない場合、Windows サービスは 24 時間おきにメッセージファイルをローテーションします。

- 指定サイズを超えたときにメッセージファイルがローテーションされるようにするには、以下の文をそれぞれ別の行としてファイルの先頭に追加します (最後の行を入力した後は必ず [Enter] または [Return] を押してください)。

```
# ROTATION_TYPE = SIZE
# SIZE_KB = file-size-in-kilobytes
# SIZE_TRIGGER_INTERVAL_MINS = polling-interval
```

SIZE_KB には、Windows サービスがメッセージを別のファイルに移動するきっかけになる最小のファイル サイズ (KB 単位) を指定します。

SIZE_TRIGGER_INTERVAL_MINS には、Windows サービスがファイル サイズを確認する頻度 (分単位) を指定します。このヘッダを挿入しない場合、Windows サービスは 5 分おきにファイル サイズを確認します。

次に例を示します。

```
# ROTATION_TYPE = SIZE
# SIZE_KB = 1024
# SIZE_TRIGGER_INTERVAL_MINS = 3
```

ファイル サイズを確認してファイルが指定サイズより大きい場合、Windows サービスはそのファイルを *pathname-yyyy_mm_dd-hh_mm_ss* として保存します。次に、*pathname* という名前の新しいファイルを作成します。この新しいファイルには最初に指定したすべてのヘッダが含まれており、新しい標準出力と標準エラーのメッセージが収集されます。

ROTATION_TYPE = SIZE を指定しても、他の行を挿入していない場合、Windows サービスは 5 分おきにメッセージファイルのサイズを確認します。ファイルが 1MB より大きい場合、そのファイルはローテーションされます。

スレッド ダンプの標準出力への出力

標準出力にスレッド ダンプを出力するには、以下のいずれかを実行します。

- `weblogic.Admin THREAD_DUMP` コマンドを使用する。詳細については、B-37 ページの「`THREAD_DUMP`」を参照してください。
- コマンドプロンプトを開き、次のコマンドを入力する。

```
WL_HOME\bin\beasvc -dump -svcname:service-name
```

WL_HOME は WebLogic Server のインストール ディレクトリです。

service-name は、サーバインスタンスを実行している Windows サービスです。

次に例を示します。

```
D:\bea\weblogic70\server\bin\beasvc -dump
-svcname:mydomain_myserver
```

クラスのクラスパスへの追加

クラスパスとは、JVM が呼び出せる Java クラスの場所の宣言です。WebLogic Server のマスター スクリプトを使用してサービンスタンスを Windows サービンスタンスとしてインストールする場合、マスター スクリプトでは、サービンスタンスの実行に必要なすべてのクラスが指定されます。独自の Java クラスを追加して WebLogic Server を拡張する場合は、それらをクラスパスに追加する必要があります。

クラスパスにクラスを追加するには、次の手順に従います。

1. WL_HOME\server\bin\installSvc.cmd マスター スクリプトのバックアップコピーを作成します。
2. テキスト エディタで、WL_HOME\server\bin\installSvc.cmd マスター スクリプトを開きます。
3. set CLASSPATH 文にクラスを追加します。

たとえば、c:\myJar という名前のファイルにクラスをアーカイブした場合、修正後の文は以下のようになります。

```
set
CLASSPATH=%JAVA_HOME%\lib\tools.jar;%WL_HOME%\server\lib\weblogic_sp.jar;%WL_HOME%\server\lib\weblogic.jar;c:\myJar;%CLASSPATH%
```

注意： 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。

この制限に対処するには、次の操作を行う必要があります。

- a. set CLASSPATH コマンドの値を別個のテキスト ファイルに記述し、そのテキスト ファイルを WL_HOME\server\bin ディレクトリに保存します。
- b. WL_HOME\server\bin\installSvc.cmd マスター スクリプトで、set CMDLINE コマンドを探します。
- c. set CMDLINE コマンドの -classpath \"%CLASSPATH%\" オプションを以下のオプションと置き換えます。

```
-classpath @filename
```

filename は、クラスパス値が含まれるファイルの名前です。

次に例を示します。

```
set CMDLINE="%JAVA_VM% %MEM_ARGS% %JAVA_OPTIONS%  
-classpath @myClasspath.txt -Dweblogic.Name=%SERVER_NAME%  
-Dbea.home=\"D:\bea_70sp2\"  
-Dweblogic.management.username=%WLS_USER%  
-Dweblogic.management.server=\"%ADMIN_URL%\"  
-Dweblogic.ProductionModeEnabled=%STARTMODE%  
-Djava.security.policy=\"%WL_HOME%\server\lib\weblogic.polic  
y\" weblogic.Server"
```

4. WebLogic Server のマスター スクリプトへの変更を保存します。

サーバ固有のスクリプトの実行

注意： サーバ固有のスクリプトを実行するには、Windows レジストリを変更する権限のあるユーザ アカウントで Windows コンピュータにログインする必要があります。

Windows サービスをプロダクション環境にインストールした場合は、管理者レベルの権限があるオペレーティング システム ユーザ アカウントではサービスを実行しないことをお勧めします。詳細については、2-64 ページの「サービスを実行するユーザ アカウントの確認」を参照してください。

サーバ固有のスクリプトを実行するには、以下の手順に従います。

1. コマンド プロンプトを開いて、管理サーバのルート ディレクトリに移動します。これは、サーバ固有のスクリプトが格納されているディレクトリです。
2. サーバ固有のスクリプトの名前を入力します。

コマンド プロンプトは、スクリプトを batch ファイルとして実行します。

スクリプトが正常に実行されると、*DOMAIN_NAME_SERVER_NAME* という名前の Windows サービスが作成され、以下のような行が標準出力に出力されます。

```
mydomain_myserver installed.
```

デフォルトでは標準出力は、サーバ固有の batch ファイルを実行するコマンド プロンプトです。

3. *WL_HOME\server\bin\installSvc.cmd* マスター スクリプトを変更した場合は、他のサーバインスタンスの設定にスクリプトを使用できるように、変更箇所を元に戻しておくことを検討してください。

設定の確認

WebLogic Server が Windows サービスとして正常に設定されていることを確認するには、次の手順に従います。

1. コマンド ウィンドウを開き、次のコマンドを入力します。

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

2. ドメインディレクトリのすぐ上のディレクトリに移動します。たとえば、`BEA_HOME\user_domains\mydomain` に設定されていることを確認するには、`BEA_HOME\user_domains` に移動します。

3. 入力するコマンド:

```
beasvc -debug "yourServiceName"
```

たとえば、`beasvc -debug "mydomain_myserver"` です。

正常に設定されている場合は、`beasvc -debug` コマンドによってサーバが起動されます。スクリプトが以下のようなエラーを返す場合は、正しいサービス名を指定したかどうか確認してください。

```
Unable to open Registry Key .....  
System\CurrentControlSet\Services\beasvc  
example_examplesServer\Parameters
```

サービスを実行するユーザ アカウントの確認

プロダクション環境では、アクセス権限が限定された特別なオペレーティングシステム ユーザ アカウントで WebLogic Server Windows サービスを実行する必要があります。たとえば、その OS ユーザによるアクセスを BEA ファイルとドメイン ファイルに限定します。そして、これらのファイルへのアクセスを、このユーザ アカウントにのみ許可します。

WebLogic Server インスタンスが、常に特別な OS ユーザ アカウントで実行されるようにするには、以下の手順に従います。

1. [サービス] コントロール パネルを開きます。

Windows 2000 デスクトップの場合であれば次の手順で操作します。

- a. [スタート] メニューを選択します。
- b. [スタート] メニューで、[設定 | コントロール パネル] を選択します。

- c. コントロールパネルで [管理ツール] フォルダを開きます。
- d. [管理ツール] ウィンドウで、[サービス] コントロールパネルを開きます。
2. [サービス] コントロールパネルで、**WebLogic Server Windows** サービスを右クリックして [プロパティ] を選択します。
3. [プロパティ] ウィンドウで、[ログオン] タグをクリックします。
4. [ログオン:] で [アカウント] を選択し、特別な OS ユーザアカウントのユーザ名とパスワードを入力します。
5. [OK] をクリックします。

コントロールパネルを使用したサーバインスタンスの停止と再起動

Windows サービスとして実行されるようサーバインスタンスを設定したら、コントロールパネルの [サービス] を使用して、サーバを停止および再起動できます。

デフォルトでは、Windows コントロールパネルを使用してサーバインスタンスを停止すると、Windows サービスコントロールマネージャ (SCM) がサーバの Java 仮想マシン (JVM) を強制停止します。JVM を強制停止すると、サーバは直ちにすべての処理を終了します。セッションデータはすべて失われます。サーバが `config.xml` ファイルへの書き込みを行っている間に管理サーバの JVM を強制停止すると、`config.xml` ファイルが破損する可能性があります。Windows のコントロールパネルによって正常な停止を可能にするための詳細については、2-55 ページの「コントロールパネルからの正常な停止の有効化」を参照してください。

Windows サービスとしてインストールされている WebLogic Server インスタンスを停止または再起動するには、次の手順に従います。

1. [スタート | 設定 | コントロールパネル] を選択します。
2. Windows 2000 で、コントロールパネルの [管理ツール] を開きます。次に、コントロールパネルの [サービス] を開きます。

Windows NT では、[コントロール パネル] から直接、[サービス] を選択して開きます。

3. [サービス] ダイアログ ボックスで、作成したサービスを見つけます。デフォルトでは、サービス名は「beasvc」で始まります。
4. サービス名を右クリックして、ショートカット メニューからコマンドを選択します。

Windows サービスとしてのサーバの削除

WebLogic Server インスタンスを実行する Windows サービスを削除するには、beasvc ユーティリティが Windows レジストリから関連のキーを削除するように指定したスクリプトを使用できます。Windows サービスを削除しても、ドメインのコンフィグレーション ファイルに保存されているサーバインスタンスのコンフィグレーションには影響はありません。Windows サービスを削除した後は、WebLogic Server インスタンスを起動スクリプトで、または管理対象サーバの場合はノードマネージャで起動できます。

ドメイン コンフィグレーション ウィザードでドメインのスクリプトが既に作成されていない場合は、自分で作成する必要があります。このスクリプトでは、サーバインスタンス名などサーバ固有の情報を識別する変数値を設定します。その後、スクリプトはマスター アンインストール スクリプト

WL_HOME\server\bin\uninstallSvc.cmd を呼び出します。WL_HOME は、WebLogic Server をインストールしたディレクトリです。マスター スクリプトは beasvc ユーティリティを呼び出します。これにより、Windows レジストリからキーが削除されます。

サーバ固有のアンインストーラ スクリプトの例は、2-67 ページのコード リスト 2-3 「Windows サービスを削除するスクリプト」を参照してください。

WebLogic Server インスタンスを実行する Windows サービスを削除するスクリプトを作成するには、次の手順に従います。

1. ドメインの管理サーバのルート ディレクトリ (ドメインの config.xml ファイルが格納されているディレクトリ) に、テキスト ファイルを作成します。
2. 以下の**必須** batch コマンドをテキスト ファイルに追加します。コマンドはそれぞれ別の行に記述します。
 - SETLOCAL

WebLogic Server インスタンスの Windows サービスとしての設定

これは、バッチ ファイル内の環境変数のローカライゼーションを開始する batch コマンドです。

- `set DOMAIN_NAME=domain-name`

`domain-name` は、使用している WebLogic Server の名前を表します。

- `set SERVER_NAME=server-name`

`server-name` は、Windows サービスとして設定する既存のサービンスタンスの名前を表します。

- `call "WL_HOME\server\bin\uninstallSvc.cmd"`

`WL_HOME` は WebLogic Server をインストールしたディレクトリの絶対パス名です。このコマンドは、WebLogic Server のマスターアンインストールスクリプトを呼び出します。

- `ENDLOCAL`

これは、バッチ ファイル内の環境変数のローカライゼーションを終了する batch コマンドです。

3. テキスト ファイルの拡張子を `.cmd` にして保存します。デフォルトでは、Windows のコマンドプロンプトで拡張子 `.cmd` と batch ファイルが関連付けられています。

4. サーバ固有のスクリプトの名前を入力します。

コマンドプロンプトは、スクリプトを batch ファイルとして実行します。

削除スクリプトは正常に実行されると、標準出力に以下のような行を出力します。

```
mydomain_myserver removed.
```

デフォルトでは、標準出力は、batch ファイルを実行するコマンドプロンプトです。

コード リスト 2-3 Windows サービスを削除するスクリプト

```
echo off
SETLOCAL

set DOMAIN_NAME=myWLSdomain
set SERVER_NAME=myWLSserver
call "D:\bea\weblogic700\server\bin\uninstallSvc.cmd"

ENDLOCAL
```

Windows サービスとしてのサーバ設定に対する起動資格の変更

WebLogic Server インスタンスが別のユーザ資格に基づいて実行されるように Windows サービスを変更するには、以下の処理のいずれかを実行します。

- 起動 ID ファイルからユーザ名とパスワードを取得するように Windows サービスを設定した場合は、既存のファイルを、新しいユーザ名とパスワードを格納した新しいファイルで上書きできます。WebLogic Server のデフォルトのセキュリティ レベル内の既存のユーザ名を指定する必要があります。詳細については、2-8 ページの「管理サーバの起動 ID ファイルの作成」を参照してください。

- Windows レジストリからユーザ名とパスワードを取得するように Windows サービスを設定した場合は、その Windows サービスを削除し、新しいユーザ名とパスワードを使用する新しいサービスを作成しなければなりません。

1. WebLogic Server インスタンスを実行する Windows サービスをアンインストールします。詳細については、2-66 ページの「Windows サービスとしてのサーバの削除」を参照してください。

2. テキスト エディタで、サービスのインストールに使用したスクリプトを開き、`set WLS_USER` コマンドや `set WLS_PW` コマンドのための値として新しいユーザ名やパスワードを入力します。この値は Windows レジストリでは暗号化されます。

3. スクリプトへの変更を保存します。

4. サーバ固有のスクリプトの名前を入力します。

コマンド プロンプトは、スクリプトを batch ファイルとして実行します。

スクリプトが正常に実行されると、`DOMAIN_NAME_SERVER_NAME`

という名前の Windows サービスが作成され、以下のような行が標準出力に出力されます。

```
mydomain_myserver installed.
```

デフォルトでは標準出力は、サーバ固有の batch ファイルを実行するコマンド プロンプトです。

5. (省略可能) スクリプト ファイルからユーザ名とパスワードを削除します。

3 システム管理操作の保護

各人のスキルを活かすために、多くの Web 開発チームでは、システム管理の責任が複数のロールに明確に割り当てられています。コンポーネントをデプロイするパーミッションは 1 人か 2 人のチームメンバーにしか付与しないが、WebLogic Server コンフィグレーションを参照するパーミッションはチームメンバー全員に付与する、というような設定がプロジェクトごとに行われます。

WebLogic Server では、このようなロールベースの開発がサポートされており、システム管理操作に対するアクセス特権が定義されたグローバル ロール (Anonymous、Admin、Deployer、Operator、Monitor) が用意されています。

以下の節では、セキュリティ ロールおよびシステム管理操作について説明します。

- 3-2 ページの「各ロールで使用可能な操作」
- 3-4 ページの「保護されたユーザ インタフェース」
- 3-11 ページの「WebLogic Server の起動および停止に対するパーミッション」

注意： 7.0 より前のリリースで使用されていた、WebLogic Server MBean を保護するためのアクセス制御リスト (ACL) に代わって、現在ではこのようなロールベースのパーミッションが使用されます。

各ロールで使用可能な操作

表 3-1 に、システム管理操作のアクセス特権を決定するために使用される 5 種類のグローバル ロールと、各ロールに付与されるパーミッションを示します。

表 3-1 グローバル ロールとパーミッション

グローバル ロール	パーミッション
Anonymous	<p>すべてのユーザ (everyone グループ) にこのグローバル ロールが付与される。</p> <p>注意: このグローバル ロールは利便性のために用意されており、weblogic.xml および weblogic-ejb-jar.xml デプロイメント記述子で指定できる。</p>
Admin	<ul style="list-style-type: none">■ サーバのコンフィグレーションを参照する。暗号化された値や属性も参照できる。■ サーバのコンフィグレーション全体を変更する。■ エンタープライズアプリケーション、起動クラスと停止クラス、Web アプリケーション、EJB、J2EE コネクタ、および Web サービス モジュールをデプロイする。■ デフォルトでは、サーバを起動、再開、および停止する。
Deployer	<ul style="list-style-type: none">■ サーバのコンフィグレーションを参照する。暗号化された属性は参照できない。■ エンタープライズアプリケーション、起動クラスと停止クラス、Web アプリケーション、EJB、J2EE コネクタ、および Web サービス モジュールをデプロイする。
Operator	<ul style="list-style-type: none">■ サーバ コンフィグレーション (暗号化された属性は除く) を表示する。■ デフォルトでは、サーバを起動、再開、および停止する。

表 3-1 グローバル ロールとパーミッション

グローバル ロール	パーミッション
Monitors	<ul style="list-style-type: none"> ■ サーバのコンフィグレーションを参照する。暗号化された属性は参照できない。 <p>注意： このセキュリティ ロールでは、WebLogic Server Administration Console、weblogic.Admin ユーティリティ、および MBean API への読み込み専用アクセスが可能。</p>

どのロール メンバーシップのユーザも、暗号化される属性を暗号化していない状態で表示することはできません。

注意： WebLogic Server MBean を直接操作している際に、グローバル ロールやその特権に関して表 3-1 より詳しい情報が必要になった場合は、『WebLogic リソースのセキュリティ』の「保護されている MBean の属性および操作」を参照してください。

独自のセキュリティ ロール (グローバルまたはスコープ) を作成し、デフォルトのグローバル ロールに追加することもできます。詳細については、『WebLogic リソースのセキュリティ』の「Administration Console でのセキュリティ ロールの作成方法」を参照してください。

デフォルト グループの関連付け

デフォルトでは、WebLogic Server は 4 つのデフォルト グループに 4 つのグローバル ロールを付与します。これらのグループのいずれかにユーザを追加すると、そのユーザにはグローバル ロールが自動的に付与されます。このデフォルトグループの関連付けを表 3-2 に示します。

表 3-2 デフォルト グループの関連付け

グループ	関連付けられたロール
Administrators	Admin
Deployers	Deployer

表 3-2 デフォルト グループの関連付け

グループ	関連付けられたロール
Operator	Operator
Monitors	Monitors

ユーザの作成とグループへの割り当てについては、『WebLogic リソースのセキュリティ』の「ユーザの作成」および「ユーザのグループへの追加」を参照してください。

保護されたユーザ インタフェース

WebLogic Server Administration Console、`weblogic.Admin` コマンド、および MBean API は、表 3-1 と表 3-2 で説明したデフォルト グローバル ロールとデフォルトグループに基づくデフォルトセキュリティ ポリシーを使用して保護されています。したがって、Administration Console を使用するユーザは、いずれかのデフォルトグループに属しているか、いずれかのグローバル ロールが付与されている必要があります。また、MBean との対話が必要となる管理操作は、『WebLogic リソースのセキュリティ』の「保護されている MBean の属性および操作」で説明する MBean による保護を使用して保護する必要があります。したがって、以下にあげる保護されたパブリック インタフェースとの対話では、両方のセキュリティ方式を満たしていなければなりません。

- WebLogic Server Administration Console—WebLogic Security サービスは、特定のユーザがログインすると、そのユーザに Administration Console へのアクセス権があるかどうかを確認します。ユーザがアクセス権のない操作を呼び出すと、アクセス拒否エラーが表示されます。

このパブリック インタフェースの使用方法については、Administration Console オンライン ヘルプを参照してください。

- `weblogic.Admin` コマンド—WebLogic Security サービスは、特定のユーザがコマンドを呼び出すと、そのユーザにコマンドを実行するパーミッションがあるかどうかを確認します。ユーザがアクセス権のない操作を呼び出すと、WebLogic Server によって `weblogic.management.NoAccessRuntimeException` (プログラムで明示的

に検出可能)が送出されます。この例外はサーバのログファイルに送信されます。例外を標準出力に送信するよう、サーバをコンフィグレーションすることもできます。

このパブリック インタフェースの使用方法については、「WebLogic Server コマンドライン インタフェース リファレンス」の「WebLogic Server 管理コマンドのリファレンス」を参照してください。

注意： `weblogic.Admin` コマンドは、MBean API (後述) との対話を抽象化できる便利なユーティリティです。したがって、`weblogic.Admin` コマンドを使用して実行できるすべての管理タスクは、MBean API を使用して記述することも可能です。

- **MBean API —WebLogic Security** サービスは、特定のユーザが MBean で操作を実行すると、そのユーザに API にアクセスするためのパーミッションがあるかどうかを確認します。ユーザがアクセス権のない操作を呼び出すと、WebLogic Server によって

`weblogic.management.NoAccessRuntimeException` (プログラムで明示的に検出可能)が送出されます。この例外はサーバのログファイルに送信されます。例外を標準出力に送信するよう、サーバをコンフィグレーションすることもできます。

これらの API の使用方法については、『WebLogic JMX サービス プログラマーズ ガイド』を参照してください。

サーバリソースの階層化されたセキュリティ方式

以下の節では、サーバリソースの階層化されたセキュリティ方式について説明します。

- 3-6 ページの「サーバリソースのセキュリティ ポリシー」
- 3-6 ページの「MBean による保護」
- 3-7 ページの「WebLogic Security サービスによる階層化された保護の検証」
- 3-8 ページの「例」

- 3-10 ページの「一貫性のあるセキュリティ方式の維持」

サーバ リソースのセキュリティ ポリシー

他のタイプの WebLogic リソースと同様、サーバ リソースも WebLogic Server Administration Console を使用してセキュリティ ポリシーで保護されています。

より具体的にいえば、すべてのサーバ リソースは、Admin および Operator デフォルトグローバルセキュリティロールに基づくデフォルトセキュリティポリシーを継承します。3-2 ページの「各ロールで使用可能な操作」で説明したように、Admin および Operator グローバルロールには、管理者が Administration Console や `weblogic.Admin` などの管理インタフェースと対話するのに必要な特定の特権が付与されています。これらのデフォルト グローバル ロールは、デフォルト グループに基づいています (3-3 ページの「デフォルト グループの関連付け」を参照してください)。したがって、サーバ リソースにアクセスする必要のある管理者は、Administrators または Operators デフォルト グループのどちらかに属している必要があります。

注意： WebLogic Server では、4つのデフォルト グループに4つのグローバルロールを付与しているため、ユーザをこれらのグループのいずれかに追加するとグローバル ロールも自動的に付与されます。

警告： 制限を強化するためにサーバ リソースのデフォルト セキュリティ ポリシーを変更しないでください。既存のセキュリティ ロールの中には、削除すると WebLogic Server の機能に悪影響を与えるものがあります。ただし、新しいセキュリティ ポリシーを追加するなどして、デフォルト セキュリティ ポリシーをより包括的にすることはできます。

MBean による保護

それぞれのタイプの WebLogic リソース (サーバ リソースを含む) は、`weblogic.security.spi.Resource` インタフェース (サーバ リソースの場合は `weblogic.security.service.ServerResource` クラス) の独自の実装を使用して操作セットをエクスポートします。したがって、`ServerResource` クラスは、3-6 ページの「サーバ リソースのセキュリティ ポリシー」で説明したセキュリティ ポリシーによって実際に保護されたエンティティです。

WebLogic Server では、サーバリソースのコンフィグレーションを複数の MBean を使用してエクスポートします。したがって、ServerResource クラスが保護するアクションは、基底となる MBean の属性および操作に対応します。たとえば、Resource インタフェースの start() メソッドは、ServerRuntime MBean の start 操作に直接マップされています。

サーバリソースのコンフィグレーションをエクスポートする MBean は、4 つのデフォルト グローバル ロールのいずれかを使用して保護されます。この保護は、サーバリソースのセキュリティ ポリシーとは別に行われるもので、現時点では WebLogic Security サービスの変更不能な部分となっています。したがって、サーバリソースを保護するために独自のグローバル ロールを作成することはできませんが、サーバのコンフィグレーションを表示したり変更したりできるのは、いずれかのデフォルト グローバル ロールを付与されたユーザのみです。

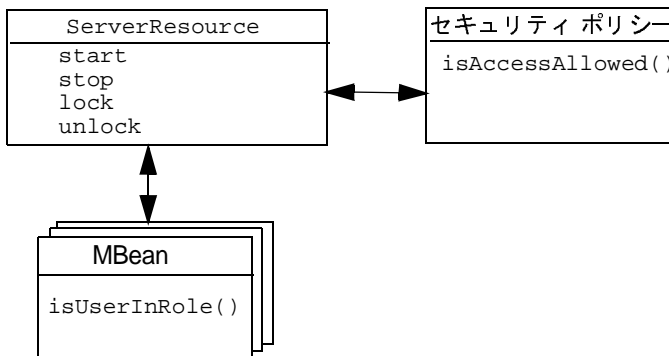
WebLogic Security サービスによる階層化された保護の検証

管理者がサーバリソースとの対話を試行すると、WebLogic Security サービスは以下を行います。

- そのユーザに、MBean の属性の変更や操作の呼び出しが可能なデフォルト グローバル ロールが付与されているかどうかを識別する。
- サーバリソースのデフォルト セキュリティ ポリシーを確認して、そのセキュリティ方式で定義された要件をユーザが満たしているかどうかを検証する。

したがって、要求した処理を正常に実行するには、ユーザが両方のセキュリティ方式の要件を満たしている必要があります。図 3-1 に、サーバリソースのセキュリティ ポリシーが、基底となる MBean でのセキュリティ ロールベースの保護とどのように相互作用するかを示します。

図 3-1 サーバリソースの階層化された保護



MBean による保護によって付与される特権は変更できないため、一貫性のあるセキュリティポリシーを維持する必要があります。詳細については、3-10 ページの「一貫性のあるセキュリティ方式の維持」を参照してください。

例

この例では、サーバリソースが階層化されたセキュリティ方式によってどのように保護されるかを示します。

ユーザ名 JDoe の管理者が、myserver というサーバを起動したいとします。この管理ユーザ (JDoe) は、デフォルトグループ Administrators のメンバーです。デフォルトでは、Admin グローバルセキュリティロールが付与されています。このユーザ対グループ、グループ対セキュリティロールのコンフィグレーションは、前述したように WebLogic Server Administration Console を使用して設定されています。

パート 1 : MBean による保護

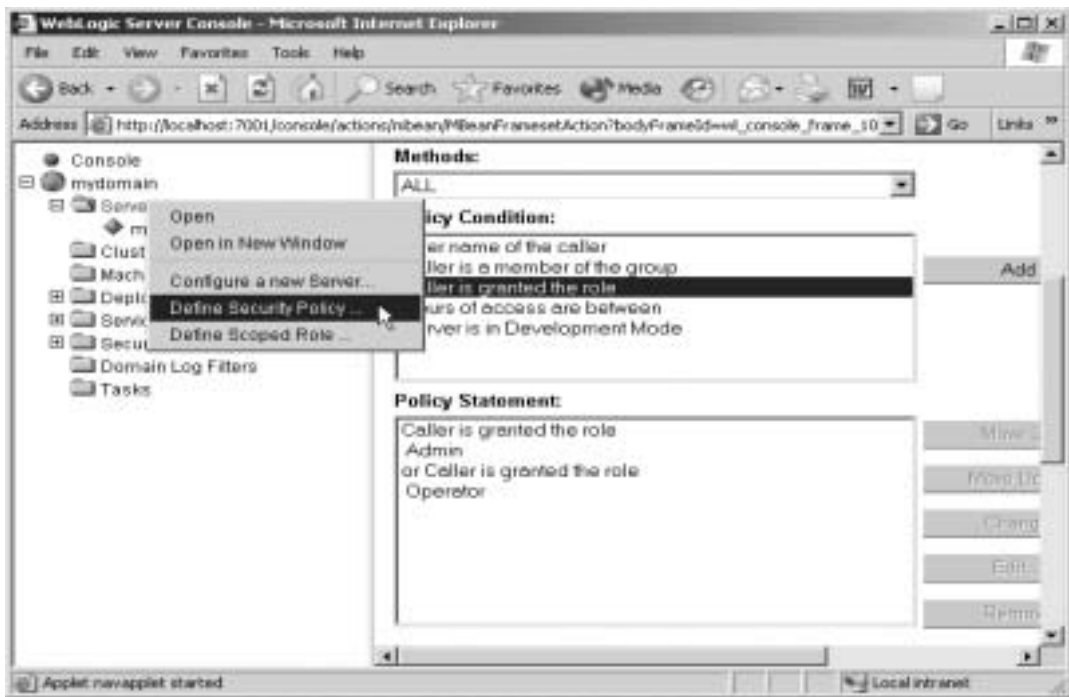
サーバを起動するにはさまざまな MBean との対話が必要であり、しかも MBean による保護は WebLogic Security サービスの変更不能な部分であるため、このような操作を行うユーザには Admin または Operator デフォルトグローバルロールが付与されている必要があります。たとえば、Server MBean および ServerRuntime MBean (start 操作を持つ MBean) へのアクセスは、上記のデフォルトセキュリティロールが付与されたユーザにしか許可されていません。

管理ユーザ JDoe はデフォルトグループ Administrators のメンバーであり、Admin グローバルセキュリティ ロールも付与されているため、サーバリソースの階層化されたセキュリティ方式の1つ目の要件は満たしています。

パート 2: サーバリソースのセキュリティ ポリシー

図 3-2 の [ポリシー文] リスト ボックスに示されているように、myserver のデフォルトセキュリティ ポリシー (表示するには、ナビゲーションツリーのmyserver を右クリックして [ポリシーの定義] を選択) に基づいて、サーバリソースと対話するために必要な Admin または Operator グローバル ロールをユーザに付与できます。管理ユーザ JDoe はデフォルトグループ Administrators のメンバーであり、Admin グローバルセキュリティ ロールも付与されているため、サーバリソースの階層化されたセキュリティ方式の2つ目の要件も満たしています。

図 3-2 myserver サーバのデフォルト セキュリティ ポリシー



注意： 管理ユーザ JDoe は Operators グループのメンバーでもあり、したがって Operator デフォルト グローバル ロールが付与されています。この点からも、階層化されたセキュリティ方式の両方の要件を満たしています。

一貫性のあるセキュリティ方式の維持

WebLogic Server のデフォルトのコンフィグレーションでは、グループ、グローバル ロール、サーバリソースのセキュリティ ポリシー、および MBean による保護が、相互に連携して一貫性のあるセキュリティ方式を確立しています。しかし、変更を行うことで、意図しないアクセス制限が発生する場合があります。デフォルトのセキュリティ設定を変更する場合は、MBean による保護およびサーバリソースのサーバ ポリシーの両方によるユーザの認証を妨げないように注意してください。

たとえば、WebLogic Server Administration Console を使用してユーザを Operator グローバル ロールに追加する際、サーバリソースに対して定義されたセキュリティ ポリシーの Operator グローバル ロールを使用し忘れると、そのユーザは起動および停止シーケンスで使用される MBean を呼び出すことはできません。サーバを起動したり停止したりするサーバリソースの操作はできなくなります。同様に、Administration Console を使用してサーバリソースのセキュリティ ポリシーから Operator グローバル ロールを削除すると、Operator グローバル ロールでは MBean 操作を呼び出すことはできません。これは、デフォルト グローバル ロールの MBean による保護が WebLogic Security サービスの一部であり、現時点では変更不能になっているためです。

MBean による保護をセキュリティ ポリシーと同期させるには、ポリシーの作成時または変更時に以下を考慮してください。

- Admin グローバル ロールには、常にサーバリソースへのアクセスを許可する。
- サーバのセキュリティ ポリシーには、Operator グローバル ロールを使用する。

注意： Operator グローバル ロールやその中にネストされたセキュリティ ロールを使用し忘れた場合は、WebLogic Security サービスにおいて問題が発生するおそれがあります。

- デプロイ可能なリソース (アプリケーション、EJB モジュール、Web アプリケーション モジュール、コネクタ モジュール、起動/停止クラスなど) のセキュリティ ポリシーでは、Deployer グローバル ロールを使用する。

WebLogic Server の起動および停止に対するパーミッション

WebLogic Server で WebLogic Server インスタンス (サーバ) を起動および停止するには、`weblogic.Server` コマンドを使用する方法とノード マネージャを使用する方法があります。`weblogic.Server` コマンドとノード マネージャでは基底のコンポーネントが異なるため、これら 2 つのコマンドでは異なる認可方法が使用されます。

以下の節では、サーバを起動および停止するためのパーミッションについて説明します。

- 3-11 ページの「`weblogic.Server` コマンドを使用したパーミッション」
- 3-12 ページの「ノード マネージャを使用したパーミッション」
- 3-12 ページの「WebLogic Server インスタンスの停止」

weblogic.Server コマンドを使用したパーミッション

`weblogic.Server` コマンドは WebLogic Server インスタンスを起動するためのコマンドで、サーバ リソースのセキュリティ ポリシーによって保護されたメソッドを呼び出します。このコマンドを使用するには、サーバ リソースのセキュリティ ポリシーの要件を満たす必要があります。

`weblogic.Server` 引数の中には MBean の属性を設定するものもあります。ただし、これらの引数はサーバが実行中 (RUNNING) 状態になる前に MBean を変更するので、MBean による保護ではなく、サーバ リソースのセキュリティ ポリシーによって認可されます。たとえば、Operator グローバル ロールに属するユーザは `-Dweblogic.ListenPort` 引数を使用してサーバのデフォルトのリسن ポートを変更できますが、WebLogic Server インスタンスが実行中状態になると、このユーザはリسن ポートの値を変更できません。

`weblogic.Server` の詳細については、2-18 ページの「`weblogic.Server` コマンドの使用」を参照してください。

ノード マネージャを使用したパーミッション

ノード マネージャでは、リモート サーバの起動に **MBean** による保護とサーバリソースのセキュリティ ポリシーの両方を使用します。

リモート **WebLogic Server** インスタンスのホスト マシン上でノード マネージャがコンフィグレーションされている場合、デフォルトでは **Admin** または **Operator** グローバル ロールに属するユーザはノード マネージャを使用してリモート サーバを起動できます。

ノード マネージャについては、『**WebLogic Server** ドメイン管理』の「ノード マネージャによるサーバの可用性の管理」を参照してください。

WebLogic Server インスタンスの停止

WebLogic Server インスタンスの停止においても、**MBean** による保護とサーバリソースのセキュリティ ポリシーの両方が使用されます。ユーザが停止コマンドを発行すると、サーバではまず、**MBean** による保護に基づいてそのユーザが **Admin** または **Operator** グローバル ロールのメンバーかどうかを判別されます。続いて、**MBean** 操作を実行した後、サーバリソースのセキュリティ ポリシーに基づいてそのユーザがサーバを停止する権限を持っているかどうかを判別されません。

4 ログメッセージを使用した WebLogic Server の管理

WebLogic Server では、ログメッセージを使用して、新しいアプリケーションのデプロイメントやサブシステムの障害などのイベントについての情報を記録します。メッセージには、イベントの時刻と日付についての情報や、イベントを開始したユーザの ID が含まれています。

これらのメッセージを表示してソートすることで、問題の検出、障害発生源の特定、およびシステムパフォーマンスの監視ができます。たとえば、どのユーザが特定のアプリケーションをデプロイしたか、どのユーザが特定日のスレッドプール数を変更したか、などを判別できます。また、これらのメッセージをリスンして自動的に応答するクライアントアプリケーションを作成することも可能です。たとえば、障害が発生したサブシステムを示すメッセージをリスンするアプリケーションを作成できます。サブシステムに障害が発生したら、アプリケーションからシステム管理者に電子メールを送信することもできます。

この章の内容は以下のとおりです。

- WebLogic Server のログメッセージ
- 例外とスタックトレース
- WebLogic Server のログファイル
- 標準出力への出力
- コンフィグレーション監査
- 追加のログファイル

メッセージをリスンするアプリケーションの設定については、『WebLogic Server ロギングサービスの使い方』を参照してください。

WebLogic Server のログメッセージ

weblogic.jar 内にコンパイルされたファイルは、WebLogic Server の各サブシステムがステータスとの通信に使用するメッセージのセットです。たとえば、WebLogic Server インスタンスを起動すると、セキュリティサブシステムによって初期化ステータスを報告するメッセージが書き込まれます。

この節では、以下の項目について説明します。

- メッセージの属性
- メッセージの出力

メッセージの属性

次の表で説明するように、すべてのサブシステムのメッセージには一連のフィールド (属性) が含まれています。

表 4-1 メッセージの属性

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。詳細については、4-3 ページの「メッセージの重要度」を参照。
Subsystem	メッセージのソースとなった WebLogic Server のサブシステムを示す。たとえば EJB、RMI、JMS など。
Server Name Machine Name Thread ID Transaction ID	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。

表 4-1 メッセージの属性

属性	説明
User	<p>関連付けられたイベントを実行したユーザ ID。</p> <p>内部コードの一部を実行する場合、WebLogic Server では実行を初期化したユーザの ID を認証してから、特別な Kernel Identity ユーザ ID でコードを実行する。</p> <p>サービインスタンスにデプロイされている EJB などの J2EE モジュールは、モジュールがサーバに渡すユーザ ID を報告する。</p> <p>クライアント JVM 内で生成されるログメッセージには、これらのフィールドは含まれない。</p>
Message ID	<p>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。</p>
Message Text	<p>イベントまたは状況の説明。</p>

メッセージの重要度

WebLogic Server のログメッセージの **Severity** (重要度) 属性は、メッセージで報告されるイベントまたは状況の潜在的な影響を示します。

次の表は、WebLogic Server サブシステムからのログメッセージの重要度レベルを、最も低いレベルから順に示しています。

表 4-2 メッセージの重要度

重要度	意味
INFO	通常の処理を報告する。
WARNING	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	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。

表 4-2 メッセージの重要度

重要度	意味
NOTICE	サーバのモニタにおいて特に重要な INFO レベルまたは WARNING レベルのメッセージ。
CRITICAL	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	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	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态を示す。

WebLogic Server のサブシステムでは、重要度の低いメッセージが数多く生成され、重要度の高いメッセージほど数が少なくなります。たとえば、通常的环境下では、INFO メッセージが多く生成され、EMERGENCY メッセージは生成されません。

アプリケーションで WebLogic ロギング サービスを使用している場合は、DEBUG という重要度レベルも使用できます。WebLogic Server のサブシステムでは、この重要度レベルは使用しません。詳細については、『WebLogic Server ロギング サービスの使い方』の「デバッグメッセージの書き込み」を参照してください。

メッセージの出力

WebLogic Server インスタンスから出力されるメッセージは、先頭行が ##### で始まり、その後メッセージの属性が続きます。各属性は山括弧で囲まれます。

次に、ログメッセージの例を示します。

```
#####<Jun 2, 2002 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。

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。

メッセージにスタック トレースが含まれている場合、スタック トレースはメッセージ属性のリストの後に続きます。

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。

例外とスタック トレース

WebLogic Server のログ メッセージには、例外が送出されたことを示すものがあります。例外とともにスタック トレースが生成された場合、WebLogic Server のログ メッセージにはスタック トレースが含まれます。

スタック トレースをログ ファイルに書き込むかどうかを指定できます。

WebLogic ロギング サービスを使用するアプリケーションがリモート JVM で実行されている場合、例外とスタック トレースは WebLogic ロギング サービスに送信されます。リモートの例外とスタック トレースをログ ファイルに書き込むかどうかを指定するには、**Administration Console** を使用します。

例外とスタック トレースのロギングのコンフィグレーションについては、**Administration Console** オンライン ヘルプの「サーバ ログ ファイルのデバッグ情報のコンフィグレーション」を参照してください。

WebLogic Server のログ ファイル

WebLogic Server では、生成されたメッセージをログ ファイルに書き込むことで保持します。ログ ファイルを表示するには、標準のテキスト エディタまたは **Administration Console** のログ ファイル ビューアを使用します。

注意： ログ ファイルは手作業で編集しないでください。ログ 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 ファイルのローテーションが混乱するおそれがあります。また、編集時にファイルがロックされてしまうと、**WebLogic Server** からは更新できなくなります。

この節では、以下の項目について説明します。

- ローカル ログ ファイルとドメイン ログ ファイル
- ログ ファイルの名前と場所
- ログ ファイル ローテーション
- WebLogic ログ ファイル ビューア

ローカル ログ ファイルとドメイン ログ ファイル

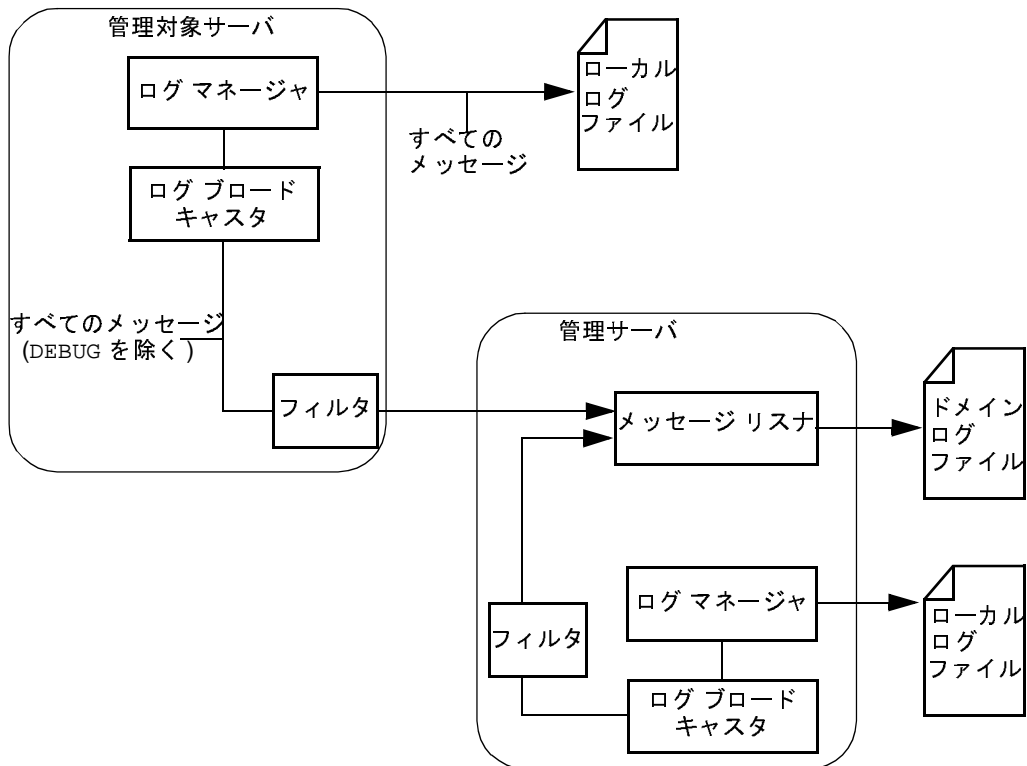
サブシステムおよびアプリケーションからのメッセージは、すべてローカル ホスト マシン上のログ ファイルに書き込まれます。また、メッセージは **Java Management Extension (JMX)** を使用して **JMX** 通知としてブロードキャストされます。ブロードキャストされるのは、以下を除くすべてのメッセージおよびメッセージテキストです。

- 重要度レベルが **DEBUG** のメッセージ
- メッセージに含まれているスタック トレース

WebLogic Server インスタンスを起動すると、管理サーバのメッセージ リスナが自動的にサーバのログ ブロードキャストに登録されます。登録時には、管理サーバがどのメッセージをリスンするかを判別するためのデフォルトフィルタが提供されます。メッセージは、ドメイン全体のログ ファイルに書き込まれます (図 4-1 を参照)。

注意： 管理対象サーバが管理対象サーバ独立 (**MSI**) モードで動作している場合、その管理対象サーバはドメインのログ ファイルに直接書き込みます。「**MSI Mode and the Domain Log File**」を参照してください。

図 4-1 WebLogic Server ロギング サービス



デフォルト フィルタでは、重要度レベルが `ERROR` 以上のメッセージのみ選択されます。こうしたドメイン ログによって、ドメイン全体のステータスがおおまかに把握できます。

すべての **WebLogic Server** インスタンスについて、デフォルト フィルタをオーバーライドし、デフォルトとは異なるメッセージセットをドメイン ログ ファイルに書き込むドメイン ログ フィルタを作成できます。**WebLogic Server** インスタンスのドメイン ログ フィルタの設定については、**Administration Console** オンラインヘルプの「ドメイン ログ フィルタ」を参照してください。

管理サーバが使用できない場合でも、管理対象サーバのローカル ログ ファイルへのメッセージの書き込みは継続されます。ただし、管理サーバが使用できない間にどのメッセージが生成されたかは追跡できません。たとえば、管理サーバが2時間使用できなかった場合、その間に生成されたメッセージは復旧後もドメイン ログには書き込まれません。

ログ ファイルの名前と場所

ローカル サーバのログ ファイル名は、デフォルトでは `./SERVER_NAME/SERVER_NAME.log` となります。`SERVER_NAME` はサーバ名です。パスは、サーバのルート ディレクトリを基準にした相対パスです。

デフォルトのドメイン ログ ファイル名は、`./DOMAIN_NAME.log` です。`DOMAIN_NAME` はドメイン名です。パスは、管理サーバのルート ディレクトリを基準にした相対パスです。

ログ ファイルのローテーション時に日付や時刻をファイル名に含めるには、`java.text.SimpleDateFormat` 変数をログのファイル名に追加します。詳細については、次の節「ログ ファイル ローテーション」を参照してください。

ログ ファイルの名前や位置の変更については、**Administration 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。

- 2-31 ページの「サーバのルート ディレクトリ」
- **Administration Console** オンライン ヘルプの「一般的なログ ファイル設定の指定」
- **Administration Console** オンライン ヘルプの「ドメイン ログ ファイルの名前と場所の指定」

ログ ファイル ローテーション

デフォルトでは、ローカル ログ ファイルとドメイン ログ ファイルのサイズは無限に大きくなります。**WebLogic Server** では、ログ ファイル名が定期的に変更（ローテーション）されるように設定できます。古いメッセージは名前が変更され

る前のログ ファイルに残り、新しいメッセージは新しいログ ファイル内に蓄積されます。ログ ファイルのローテーションは、ログ ファイルのサイズまたは時間間隔に基づいて指定できます。

ログ ファイルのローテーション時に日付や時刻をファイル名に含めるには、`java.text.SimpleDateFormat` 変数をログのファイル名に追加します。各変数はパーセント (%) 記号で囲みます。

たとえば、サーバのローカル ログ ファイル名に次の値を入力したとします。

```
myserver_%yyyy%_%MM%_%dd%_%hh%_%mm%.log
```

この場合、サーバのログ ファイル名は次のようになります。

```
myserver_yyyy_MM_dd_hh_mm.log
```

サーバインスタンスによってログ ファイルがローテーションされると、ローテーションされたファイルの名前には日付が含まれます。たとえば、ローカル ログ ファイルが 2003 年 4 月 2 日の午前 10:05 にローテーションされた場合、古いメッセージを含むログ ファイルの名前は次のようになります。

```
myserver_2003_04_02_10_05.log
```

日付や時刻を含めない場合、ローテーションされるログ ファイルには `filenamennnnn` 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た名前です。たとえば、`myserver.log00007` のようになります。

Administration Console を使用すると、各 **WebLogic Server** インスタンスのローカル ログ ファイルのローテーションの基準を指定できます。また、ドメインメッセージ ログ ファイルのローテーションの基準を指定する場合も **Administration Console** を使用します。

ローテーションするファイルの最大数を指定することもできます。ログ ファイル数が指定した数に達すると、以降のファイルのローテーションでは、もっとも古いログ ファイルから順に上書きされます。

注意： 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。ログ 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おそれがあります。

ローテーションの基準の指定については、**Administration Console** オンラインヘルプの以下の節を参照してください。

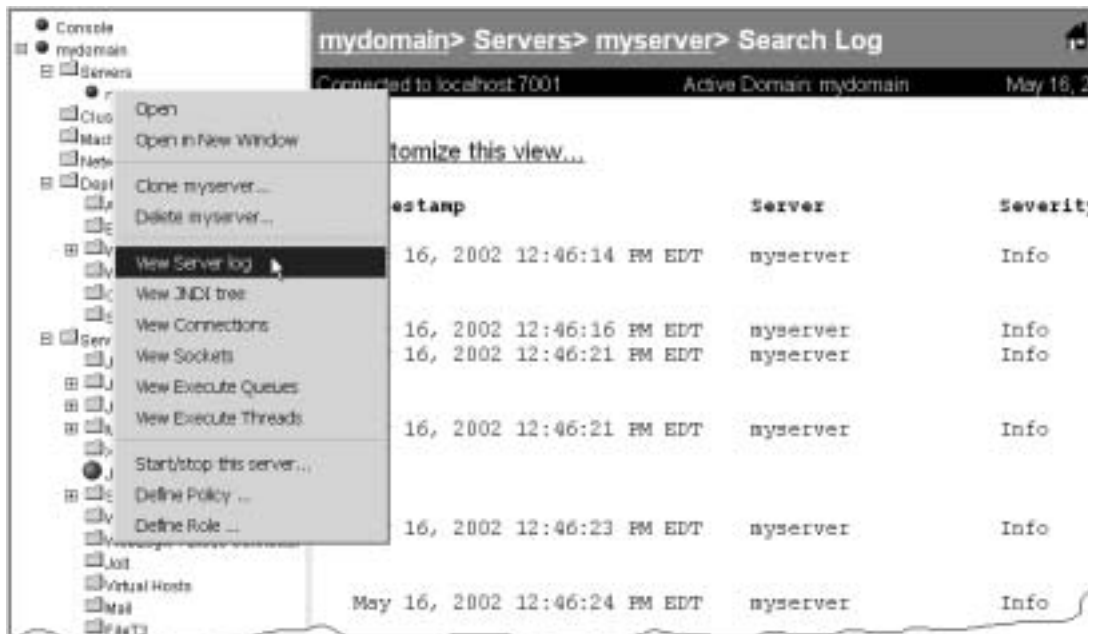
- 「ログ ファイルのローテーションの指定」

- 「ドメイン ログ ファイルのローテーション条件の指定」

WebLogic ログ ファイル ビューア

Administration Console には、ローカル サーバ ログ用のログ ビューアと、ドメイン全体のメッセージ ログ用のログ ビューアが用意されています。2つのビューアは別物ですがよく似ています。ログ ビューアを使用すると、メッセージ内のフィールドをもとにメッセージを検索できます。たとえば、重要度、発生時刻、ユーザ ID、サブシステム、またはメッセージの短い説明に基づいて、メッセージを検索して表示できます。また、記録された順にメッセージを表示したり、過去のログ メッセージを検索したりすることもできます。

図 4-2 Administration Console のログ ビューア



Administration Console でのメッセージ ログの表示、コンフィグレーション、および検索については、Administration 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。

- 「サーバのログの表示」
- 「ドメイン ログの表示」

ログ ファイルはシンプル テキスト ファイルなので、他のアプリケーションを使用して表示することもできます。ログ ファイルの検索については、4-8 ページの「ログ ファイルの名前と場所」を参照してください。

標準出力への出力

メッセージをログ ファイルに書き込むだけでなく、メッセージのサブセットを標準出力へ出力することも可能です。デフォルトでは、重要度が **ERROR** 以上のすべてのメッセージが標準出力に出力されます。ただし、**DEBUG** のメッセージは出力されません。スタック トレースをログ ファイルに出力するようサーバをコンフィグレーションした場合は、スタック トレースも標準出力に出力できます。

ノード マネージャを使用して管理対象サーバを起動した場合、ログ ファイルには標準出力と標準エラー メッセージが書き込まれます。これらのメッセージは、**Administration Console** の [マシン | モニタ] タブで表示できます。

詳細については、以下のトピックを参照してください。

- どの **WebLogic Server** メッセージが標準出力に送信されるかについては、**Administration Console** オンライン ヘルプの「一般的なログ ファイル設定の指定」を参照してください。
- スタック トレースのログ ファイルへの出力については、**Administration Console** オンライン ヘルプの「サーバ ログ ファイルのデバッグ情報のコンフィグレーション」を参照してください。
- ノード マネージャで起動した管理対象サーバでのメッセージの表示については、『**WebLogic Server** ドメイン管理』の「管理対象サーバのログ ファイル」を参照してください。
- **Windows** サービスとして実行するサーバインスタンスの標準出力ストリームの表示については、2-58 ページの「標準出力および標準エラーのファイルへのリダイレクト」を参照してください。

System.out および System.err のファイルへのリダイレクト

サーブレットは、サーバインスタンスが標準出力に出力するコンフィグレーション可能なログメッセージのセットだけでなく `system.out.println` を呼び出すこともでき、WebLogic Server インスタンスを実行する JVM が標準エラーおよび標準出力にメッセージを送信することが可能です。WebLogic Server スクリプトを使用してサーバインスタンスを開始した場合、標準エラーメッセージおよび標準出力メッセージを格納するデフォルトの永続ストレージはありません。

WebLogic Server インスタンスが動作している JVM では、メッセージを標準エラーおよび標準出力に送信することもできます。WebLogic Server スクリプトを使用してサーバインスタンスを開始した場合、標準エラーメッセージおよび標準出力メッセージを格納するデフォルトの永続ストレージはありません。

これらのメッセージを記録しておくには、WebLogic Server 起動スクリプトを編集して以下の Java オプションを指定します。

```
-Dweblogic.Stdout="stdout-filename"  
-Dweblogic.Stderr="stderr-filename"
```

`stdout-filename` は標準出力メッセージを保存するファイルの名前、`stderr-filename` は標準エラーメッセージを保存するファイルの名前です。

これらのファイルの内容を表示するには、テキストエディタを使用するか、DOS `tail` プログラムなどのコマンドプロンプトユーティリティを使用します。これらのファイルは、Administration Console では表示できません。

注意： WebLogic Server では、標準出力へ送信するユーザ名とパスワードの入力を要求するプロンプトが表示されます。`-Dweblogic.Stdout` を使用した場合は、ユーザ名とパスワードの入力が要求されなくなります。このプロンプトを回避するには、2-8 ページの「ユーザ名とパスワードのプロンプトの回避」の説明に従って起動 ID ファイルを使用します。

`weblogic.Server` コマンドでの引数の指定については、2-21 ページの「よく使用される任意指定の引数」を参照してください。

ガベージコレクションコメント

-Dwebllogic.Stdout および -Dwebllogic.Stderr を指定することで、JVM のすべての `java.lang.System.out` および `java.lang.System.err` メッセージがファイルにリダイレクトされますが、ガベージコレクションコメントは `System.out` や `System.err` には出力されません。-verbosegc オプションを指定して JVM を起動した場合は、-Dwebllogic.Stdout または -Dwebllogic.Stderr を指定したかどうかにかかわらず、verbosegc 出力は JVM が実行されているシェルに出力されます。一部の JVM には、ガベージコレクションコメントをファイルに出力するための非標準オプションが用意されています。詳細については、シェルに `java -x` と入力して、JVM の非標準オプションのヘルプを表示してください。

コンフィグレーション監査

管理サーバは、ユーザによってコンフィグレーションが変更されたとき、またはドメイン内のいずれかのリソースで管理操作が実行されたときにログメッセージを送出するようにコンフィグレーションできます。たとえば、ユーザがドメイン内の管理対象サーバで SSL を無効にすると、管理サーバはログメッセージを送出します。そのようなメッセージは、ドメインのコンフィグレーションで行われた変更の証跡となります(コンフィグレーション監査)。

以下の節では、コンフィグレーション監査について説明します。

- 4-14 ページの「コンフィグレーション監査の有効化」
- 4-14 ページの「コンフィグレーション監査メッセージ」

管理サーバは、コンフィグレーション監査メッセージをローカルのログファイルに書き込みます。コンフィグレーション監査メッセージはすべて重大度が Info なので、デフォルトではドメイン全体のメッセージログには書き込まれません。このデフォルトを変更する方法については、Administration Console オンラインヘルプの「ドメイン ログ フィルタ」を参照してください。

管理サーバは、メッセージをローカル ログ ファイルに書き込むだけでなく、コンフィグレーション監査メッセージを JMX 通知としてブロードキャストします。それらのメッセージに応答する JMX リスナおよびフィルタを作成することも可能です。たとえば、無許可のユーザがドメインのコンフィグレーションを変更し

ようとしたことを通知するメッセージが管理サーバから送出された場合に、JMX リスナーおよびフィルタは電子メールを送信できます。『WebLogic JMX Service プログラマーズガイド』の「コンフィグレーション監査メッセージのリスナー」を参照してください。

コンフィグレーション監査の有効化

管理サーバがコンフィグレーション監査メッセージを送出するようにするには、以下のいずれかを行います。

- Administration Console を起動し、コンフィグレーション監査を有効にする。
Administration Console オンライン ヘルプの「Enabling Configuration Auditing」を参照してください。
- 管理サーバを起動するときに、`weblogic.Server` コマンドで次の Java オプションを使用する。
`-Dweblogic.AdministrationMBeanAuditingEnabled=true`
2-18 ページの「weblogic.Server コマンドの使用」を参照してください。
- 管理サーバが起動した後に、`weblogic.Admin` ユーティリティを使用して、`DomainMBean` の `AdministrationMBeanAuditingEnabled` 属性の値を変更する。
たとえば、次のコマンドは `examples` ドメインのコンフィグレーション監査を無効にします。

```
java weblogic.Admin SET  
-mbean examples:Name=examples,Type=Domain  
-property AdministrationMBeanAuditingEnabled true
```

`weblogic.Admin` を使用して Mbean の属性値を変更する方法については、「MBean 管理コマンドリファレンス」を参照してください。

コンフィグレーション監査メッセージ

コンフィグレーション監査メッセージはすべて、重大度が `Info` であり、159900 ~ 159910 の範囲のメッセージ ID で識別されます。

コンフィグレーション監査メッセージでは、管理 Bean (MBean) オブジェクト名を使用してリソースを識別します。MBean オブジェクト名は、操作の呼び出しやリソースの修正に使用するインタフェースが **Administration Console**、コマンドラインユーティリティ、または API のどれであるかに関係なく明確な識別を可能にします。『WebLogic JMX Service プログラマーズ ガイド』の「WebLogic Server MBean の WebLogicObjectName の使用」を参照してください。

表 4-3 に、コンフィグレーション監査メッセージをまとめておきます。

表 4-3 コンフィグレーション監査メッセージの概要

発生したイベント	生成されるメッセージの ID	メッセージのテキスト
認可ユーザがリソースを作成した。	159900	USER <i>username</i> CREATED <i>MBean-name</i> <i>username</i> は、ログインしてリソースを作成した WebLogic Server ユーザを示す。
無認可のユーザがリソースを作成しようとした。	159901	USER <i>username</i> CREATED <i>MBean-name</i> FAILED weblogic.management. NoAccessRuntimeException: <i>exception-text stack-trace</i> <i>username</i> は、無認可の WebLogic Server ユーザを示す。
認可ユーザがリソースを削除した。	159902	USER <i>username</i> REMOVED <i>MBean-name</i> <i>username</i> は、ログインしてリソースを作成した WebLogic Server ユーザを示す。
無認可のユーザがリソースを削除しようとした。	159903	USER <i>username</i> REMOVE <i>MBean-name</i> FAILED weblogic.management. NoAccessRuntimeException: <i>exception-text stack-trace</i> <i>username</i> は、ログインしてリソースを作成した WebLogic Server ユーザを示す。
認可ユーザがリソースのコンフィグレーションを変更した。	159904	USER <i>username</i> MODIFIED <i>MBean-name</i> ATTRIBUTE <i>attribute-name</i> FROM <i>old-value</i> TO <i>new-value</i> <i>username</i> は、ログインしてリソースのコンフィグレーションを変更した WebLogic Server ユーザを示す。

4 ログメッセージを使用した WebLogic Server の管理

表 4-3 コンフィグレーション監査メッセージの概要

発生したイベント	生成されるメッセージの ID	メッセージのテキスト
無認可のユーザがリソースのコンフィグレーションを変更しようとした。	159905	USER <i>username</i> MODIFY <i>MBean-name</i> ATTRIBUTE <i>attribute-name</i> FROM <i>old-value</i> TO <i>new-value</i> FAILED <i>weblogic.management.NoAccessRuntimeException:</i> <i>exception-text stack-trace</i> <i>username</i> は、無認可の WebLogic Server ユーザを示す。
認可ユーザがリソースで操作を呼び出した。 たとえば、アプリケーションをデプロイするか、サーバインスタンスを起動した。	159907	USER <i>username</i> INVOKED ON <i>MBean-name</i> METHOD <i>operation-name</i> PARAMS <i>specified-parameters</i> <i>username</i> は、ログインしてリソースの操作を呼び出した WebLogic Server ユーザを示す。
無認可のユーザがリソースで操作を呼び出そうとした。	159908	USER <i>username</i> INVOKED ON <i>MBean-name</i> METHOD <i>operation-name</i> PARAMS <i>specified-parameters</i> FAILED <i>weblogic.management.NoAccessRuntimeException:</i> <i>exception-text stack-trace</i> <i>username</i> は、無認可の WebLogic Server ユーザを示す。
認可ユーザがコンフィグレーションの監査を有効にした。	159909	USER <i>username</i> , Configuration Auditing is enabled <i>username</i> は、コンフィグレーションの監査を有効にした WebLogic Server ユーザを示す。

表 4-3 コンフィグレーション監査メッセージの概要

発生したイベント	生成されるメッセージの ID	メッセージのテキスト
認可ユーザがコンフィグレーションの監査を無効にした。	159910	<p>USER <i>username</i>, Configuration Auditing is disabled</p> <p><i>username</i> は、コンフィグレーションの監査を無効にした WebLogic Server ユーザを示す。</p>

注意： 認可ユーザがリソースを追加、修正、または削除するたびに、管理サブシステムは ID が 140009 の Info メッセージも生成します。次に例を示します。

```
<Sep 15, 2003 11:54:47 AM EDT> <Info> <Management> <140009>
<Configuration changes for domain saved to the repository.>
```

管理サブシステムは、コンフィグレーション監査が有効かどうかに関係なくこのメッセージを生成します。

このメッセージでは、ドメインのコンフィグレーションが変更されたことが通知されますが、コンフィグレーション監査メッセージが提供する詳細情報は提供されません。また、リソースの操作が呼び出されたときにはこのメッセージは生成されません。

表 4-4 は、コンフィグレーション監査メッセージの追加メッセージ属性のリストです。これらの属性は、すべてのコンフィグレーション監査メッセージで同じ値が指定されます。

表 4-4 共通のメッセージ属性と値

メッセージ属性	属性値
Severity	Info
Subsystem	Configuration Audit
User ID	<p>kernel identity</p> <p>この値は、ユーザがリソースを修正したか、リソースの操作を呼び出したかに関係なく常に kernel identity。</p>

表 4-4 共通のメッセージ属性と値

メッセージ属性	属性値
Server Name	<i>AdminServerName</i> 管理サーバはドメイン内のすべてのリソースのコンフィグレーション データを保持するので、この値は常に管理サーバの名前。
Machine Name	<i>AdminServerHostName</i> 管理サーバはドメイン内のすべてのリソースのコンフィグレーション データを保持するので、この値は常に管理サーバのホスト マシンの名前。
Thread ID	<i>execute-thread</i> この値は、管理サーバ上で動作している実行スレッドの数に依存する。
Timestamp	メッセージが生成された時点の <i>timeStamp</i>

追加のログ ファイル

前の節で説明したログメッセージとログ ファイルは、サーバまたはアプリケーションの操作に影響するイベントおよび状況を伝達するためのものです。

一部のサブシステムには、通常の操作状況でのサブシステムの対話を監査するため、追加のログ ファイルが用意されています。追加のログ ファイルには以下があります。

- HTTP サブシステムでは、すべての HTTP トランザクションのログをテキスト ファイルに記録できます。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。詳細については、6-14 ページの「HTTP アクセス ログの設定」を参照してください。
- JDBC サブシステムでは、JDBC ドライバの登録、SQL 例外の発生など、JDBC 接続に関係するさまざまなイベントのログが記録されます。なお、JDBC に関するイベントの一部（接続の作成や更新、JDBC オブジェクトのコンフィグレーションの変更など）はサーバ ログに書き込まれます。

- **JTA** サブシステムでは、トランザクションの統計値をレポートするためのトランザクション ログを記録できます。詳細については、7-8 ページの「トランザクションのモニタとログ」を参照してください。

5 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
- `weblogic.deploy` ユーティリティ (非推奨) を使用した Web 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

J2EE アプリケーションおよびコンポーネントは、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、J2EE 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 コンポーネントには、EJB アーカイブ (JAR)、Web アプリケーション アーカイブ (WAR)、およびリソース アダプタ アーカイブ (RAR) があります。

J2EE アプリケーションのデプロイメントと **WebLogic Server** のデプロイメントの概要については、「**WebLogic Server デプロイメント**」を参照してください。

Web アプリケーションのデプロイメントについては、『**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。

リソース アダプタのデプロイメントについては、「**リソース アダプタのパッケージ化とデプロイメント**」を参照してください。

EJB のデプロイメントについては、「WebLogic Server コンテナ用の EJB のパッケージ化」を参照してください。

weblogic.deploy ユーティリティ (非推奨) を使用した Web アプリケーションのデプロイメント

WebLogic Server 7.0 では、weblogic.Deployer ユーティリティが新しく登場し、以前の weblogic.deploy ユーティリティは非推奨になりました。古い WebLogic Server 6.x デプロイメントプロトコルを使用するユーティリティと API はすべて非推奨になっています。アプリケーションのデプロイメントでは必ず、WebLogic Server 7.0 の 2 フェーズ デプロイメント ツールおよびユーティリティを使用してください。

weblogic.Deployer については、「weblogic.Deployer ユーティリティ」を参照してください。

weblogic.deploy ユーティリティを使用して Web アプリケーションをデプロイするには、次の手順に従います。

1. WebLogic Server クラスがシステムの CLASSPATH に入り、JDK が利用できるようにローカル環境を設定します。CLASSPATH の設定には、config/mydomain ディレクトリにある setEnv スクリプトを使用できます。
2. 次のコマンドを入力します。

```
% java weblogic.deploy -port port_number -host host_name  
    -user username -component application:target deploy  
    password name application source
```

各値の説明は次のとおりです。

- *port_number* は、WebLogic Server がリクエストをリスンしているポート番号。
- *host_name* は、WebLogic Server のホストマシンの名前。

- *username* は、*-host* 引数で指定したサーバで要求を処理するパーミッションが付与されたユーザ名。デフォルトは、*-host* 引数で指定したサーバの起動に使用するユーザ名。ユーザ名は **WebLogic Server** を起動するためのユーザ アカウント。

システム管理タスクのパーミッションについては、3-1 ページの「システム管理操作の保護」を参照してください。

- *application* は、この **Web** アプリケーションに割り当てる名前。
- *target* は、この **Web** アプリケーションの対象となるサーバ、クラスタ、または仮想ホストの名前。複数の対象を入力するには、カンマでそれらを区切ります。
- *password* はシステム管理用パスワード。
- *name* はシステム管理の名前。
- *source* はデプロイする **.war** ファイルの絶対パス名、または展開ディレクトリ形式の **Web** アプリケーションを含むディレクトリの絶対パス名。

次に例を示します。

```
java weblogic.deploy -port 7001 -host myhost -component
myWebApp:myserver deploy pswd1234 myWebApp d:\myWebApp.war
```

デプロイメントに関するマニュアル

「WebLogic Server デプロイメント」では、**WebLogic Server** のデプロイメントと、デプロイメントのツール、手順、およびベスト プラクティスについて説明しています。

マニュアル	デプロイメントに関するトピック
『WebLogic Builder』	WebLogic Builder を使用して、 J2EE アプリケーションおよびそのコンポーネント用の XML デプロイメント記述子ファイルを編集、生成する方法。

5 アプリケーションのデプロイメント

マニュアル	デプロイメントに関するトピック
『WebLogic Server アプリケーションの開発』	WebLogic Server J2EE アプリケーションのデプロイ方法。
Administration Console オンラインヘルプ	デプロイメント タスクにおける Administration Console の使用方法。
『WebLogic エンタープライズ Java Beans プログラマーズ ガイド』	WebLogic Server EJB のデプロイ方法。
『WebLogic J2EE コネクタ』	WebLogic Server J2EE コネクタのデプロイ方法。
『Web アプリケーションのアセンブルとコンフィギュレーション』	WebLogic Server Web アプリケーションのデプロイ方法。
『WebLogic JSP プログラマーズ ガイド』	JSP からのアプレットのデプロイ方法。
「クラスタのコンフィギュレーションとアプリケーションのデプロイメント」	クラスタ化されたサーバへのデプロイ方法。
「WebLogic Server アプリケーションのパッケージ化」	WebLogic Server アプリケーション コンポーネントのパッケージ方法。

6 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。

- 6-1 ページの「概要」
- 6-2 ページの「HTTP パラメータ」
- 6-4 ページの「リスポートのコンフィグレーション」
- 6-5 ページの「Web アプリケーション」
- 6-7 ページの「仮想ホスティングのコンフィグレーション」
- 6-11 ページの「WebLogic Server による HTTP リクエストの解決方法」
- 6-14 ページの「HTTP アクセス ログの設定」
- 6-24 ページの「POST サービス拒否攻撃の防止」
- 6-25 ページの「HTTP トンネリングのための WebLogic Server の設定」
- 6-27 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	True
[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
[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。 [最大 POST 時間]: -1 [最大 POST サイズ]: -1	整数	30
[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 [接続 | SSL] タブで定義します。

Web アプリケーション

HTTP アプリケーションと Web アプリケーションは、Sun Microsystems のサーブレット仕様 2.3 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。

JSP は、デフォルトではサーバの一時ディレクトリにコンパイルされます。たとえば、サーバが「myserver」で webapp が「mywebapp」である場合、一時ディレクトリは <domain_dir>\myserver\wlnotdelete\appname_mywebapp_4344862 となります。

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』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、『WebLogic Server クラスタ 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、6-7 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)。

WebLogic Server に付属のサンプルドメインでは、デフォルトの Web アプリケーションが既にコンフィグレーションされています。こ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。

仮想ホスティングを使用すると、複数の Web アプリケーション用の画像などの静的ファイルを提供する 1 つのディレクトリを作成することもできます。たとえば、仮想ホストを使用して画像ファイルを提供するには、次のようなマッピングを作成できます。

```
<virtual-directory-mapping>
  <local-path>c:/usr/gifs</local-path>
  <url-pattern>/images/*</url-pattern>
</virtual-directory-mapping>
```

HTTP:// localhost:7001/mywebapp/images/test.gif への要求があると、WebLogic Server 実装は要求された画像を `c:/usr/gifs/images/*` で検索します。

仮想ホストをこのように使用するには、画像ファイルを格納する `images` という名前のディレクトリを作成する必要があります。このディレクトリは「/images/test.gif」などの相対 URI に配置する必要があります。

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ー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、6-11 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. ログインと **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、6-14 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 (省略可能)。既に **WebLogic** クラスタが定義されている必要があります。詳細については、『**WebLogic Server** クラスタ 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. 対象にする **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。

仮想ホスト名が必ず解決されるようにするには、仮想ホスト名を入力した行をサーバ上の `etc/hosts` ファイルに追加する必要があります。

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替の名前を指定できます。詳細については、「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 6-1 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	なし	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	あり	apples Web アプリケー ションの最上位ディレク トリのリスト
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 プレット。 サープレット 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、「サープレッ トのコンフィグレー ション」 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プ 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 プレットのコンフィグ レーション」 を参照。

表 6-1 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/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	あり	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
http://host:port	なし	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	関係なし	エラー 404
myfile.html が apples Web アプリケーションに存在せず、デフォルト サブレットが定義されていない場合。		詳細については、「HTTP エラー応答のカスタマイズ」を参照。

表 6-1 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/	なし	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/	あり	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
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 § 「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。

サーバまたは仮想ホストに対する HTTP ログの設定については、Administration 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。

- 「サーバの HTTP ログ ファイル設定の指定」
- 「仮想ホストの HTTP ログ ファイル設定の指定」

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。アクセス ログ ファイルの名前は、ファイルがいつローテーションされたかを示す日付と時刻が含まれるようにコンフィグレーションできます。時刻をコンフィグレーションしない場合、ローテーションされたファイルの名前には、ローテーションごとに増分される数値が含まれます。HTTP アクセス ログは、定義した Web Server ごとに保存されます。

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。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 の草案仕様に準拠しています。最新バージョンは、「W3C Technical Reports and Publications page」に公開されています。

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] 属性を [Extended] に設定します (6-19 ページの「カスタム フィールド識別子の作成」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トティブによって指定します。ディレクトティブは、実際のログ ファイルに組み込まれます。ディレクトティブは、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 ディレクト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 ファイルが既に存在する場合、そのファイルの先頭には有効なディレクト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 仕様に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 仕様に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 仕様に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せん。

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。6-23 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、次の手順を実行します。

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、6-17 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 フィールドにログイン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、6-21 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。
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、6-16 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、「`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.HttpServletRequest`、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 6-2 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
<code>Object getAttribute(String name);</code>	<code>javax.servlet.ServletRequest</code>
<code>Enumeration getAttributeNames();</code>	<code>javax.servlet.ServletRequest</code>
<code>String getCharacterEncoding();</code>	<code>javax.servlet.ServletRequest</code>
<code>int getResponseContentLength();</code>	<code>javax.servlet.HttpServletResponse.setContentLength()</code> このメソッドは応答のコンテンツ長を取得し、 <code>setContentLength()</code> メソッドと共に設定する。
<code>String getContentType();</code>	<code>javax.servlet.ServletRequest</code>
<code>Locale getLocale();</code>	<code>javax.servlet.ServletRequest</code>
<code>Enumeration getLocales();</code>	<code>javax.servlet.ServletRequest</code>
<code>String getParameter(String name);</code>	<code>javax.servlet.ServletRequest</code>
<code>Enumeration getParameterNames();</code>	<code>javax.servlet.ServletRequest</code>
<code>String[] getParameterValues(String name);</code>	<code>javax.servlet.ServletRequest</code>
<code>String getProtocol();</code>	<code>javax.servlet.ServletRequest</code>
<code>String getRemoteAddr();</code>	<code>javax.servlet.ServletRequest</code>
<code>String getRemoteHost();</code>	<code>javax.servlet.ServletRequest</code>

表 6-2 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
String getScheme();	javax.servlet.HttpServletRequest
String getServerName();	javax.servlet.HttpServletRequest
int getServerPort();	javax.servlet.HttpServletRequest
boolean isSecure();	javax.servlet.HttpServletRequest
String getAuthType();	javax.servlet.http.HttpServletRequest
String getContextPath();	javax.servlet.http.HttpServletRequest
Cookie[] getCookies();	javax.servlet.http.HttpServletRequest
long getDateHeader(String name);	javax.servlet.http.HttpServletRequest
String getHeader(String name);	javax.servlet.http.HttpServletRequest
Enumeration getHeaderNames();	javax.servlet.http.HttpServletRequest
Enumeration getHeaders(String name);	javax.servlet.http.HttpServletRequest
int getIntHeader(String name);	javax.servlet.http.HttpServletRequest
String getMethod();	javax.servlet.http.HttpServletRequest
String getPathInfo();	javax.servlet.http.HttpServletRequest
String getPathTranslated();	javax.servlet.http.HttpServletRequest
String getQueryString();	javax.servlet.http.HttpServletRequest
String getRemoteUser();	javax.servlet.http.HttpServletRequest
String getRequestURI();	javax.servlet.http.HttpServletRequest
String getRequestedSessionId();	javax.servlet.http.HttpServletRequest
String getServletPath();	javax.servlet.http.HttpServletRequest
Principal getUserPrincipal();	javax.servlet.http.HttpServletRequest

表 6-2 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
boolean isRequestedSessionIdFromCookie();	javax.servlet.http.HttpServletRequest
boolean isRequestedSessionIdFromURL();	javax.servlet.http.HttpServletRequest
boolean isRequestedSessionIdFromUrl();	javax.servlet.http.HttpServletRequest
boolean isRequestedSessionIdValid();	javax.servlet.http.HttpServletRequest
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 6-1 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、この種の攻撃を防ぐことができます。3つの属性は、Administration Console 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.
```

[最大 POST サイズ]

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 [接続 | プロトコル] 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。

HTTP トンネリングを使用するには、サーバで HTTP プロトコルと T3 プロトコルの両方がサポートされている必要があります。

[トンネリング 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 タイムアウト]

クライアントがサーバに対して (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 `TransmitFile` 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 を使用して提供され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 を使用して提供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、400K 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>
```

`weblogic.http.nativeIOEnabled` は、`FileServlet` のコンテキストパラメータとしても設定できます。

7 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
- ドメイン間トランザクションに対するドメイ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
- ヒューリスティックな終了の処理
- トランザクションの破棄
- 別のマシンへのサーバの移動
- サーバに障害が発生した後のトランザクションの回復

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、8-1 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、Java Transaction API (JTA) などの **WebLogic Server** 機能をコンフィグレーションするためのツールを利用できます。**Administration Console** を起動するには、http://edocs.beasys.co.jp/e-docs/wls/docs70/adminguide/overview.html#start_admin_console にある『管理者ガイド』の「Administration Console の起動と使い方」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

Administration Console で行う設定は、JTA のコンフィグレーション設定も含め、そのドメインの config.xml ファイルに保持されます。このファイル内のエントリについては、『コンフィグレーション リファレンス』の以下の節を参照してください。

- 「JTA」
- 「JTA MigratableTarget」
- 「JTA RecoveryService」
- 「JDBC TxDataSource」

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 (エンタープライズ JavaBean) 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 プログラマーズ 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

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

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となります。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。

サーバを起動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 (静的コンフィグレーション)、サーバの実行中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。TransactionLogFilePrefix 属性は、サーバの起動前に設定する必要があります。

トランザクション属性を変更するには、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、値を目的に応じて変更するか、また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 TransactionLogFilePrefix 属性 ([トランザクション ログファイルのプレフィックス]) が正しく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、7-8 ページの「トランザクションのモニタとログ」を参照してください。

WebLogic Server で使用できる、有効な値およびデフォルト値などのトランザクション属性の詳細については、Administration Console オンライン ヘルプの「ドメイン」を参照してください。

トランザクション管理用の追加の属性

デフォルトでは、グローバル トランザクションに参加している XA リソースが、WebLogic Server トランザクション マネージャからの XA 呼び出しへの応答に失敗すると、WebLogic Server はそのリソースを不健全で使用できないものとしてフラグを付け、リソース スレッドを保持するために、そのリソースへの以降の呼び出しをブロックします。この障害は、不健全なトランザクションまたは不健全なリソースが原因で発生します (2 つの原因に違いはありません)。どちらの場合でも、リソースは不健全なものとしてマークされます。

この制限を緩和するために、WebLogic Server では表 7-1 に示すコンフィグレーション属性を用意しています。

表 7-1 XA リソースの状態モニタ用コンフィグレーション属性

属性	MBean	定義
EnableResourceHealthMonitoring	weblogic.management.configuration.JDBCConnectionPoolMBean	<p>JDBC 接続プールのリソース状態モニタを有効または無効にする。この属性は、データベース接続に XA JDBC ドライバを使用する接続プールにのみ適用される。XA 非対応 JDBC ドライバが使用される場合は無視される。</p> <p>true に設定すると、リソース状態モニタが有効になる。MaxXACallMillis 属性で指定された期間内に XA リソースが XA 呼び出しへ応答できない場合、WebLogic Server は接続プールを不健全なものとしてマークし、そのリソースへの以降の呼び出しをブロックする。</p> <p>false に設定すると、この機能は無効になる。</p> <p>デフォルト: true</p>

表 7-1 XA リソースの状態モニタ用コンフィグレーション属性

属性	MBean	定義
MaxXACallMillis	weblogic.management.configuration.JTAMBean	XA リソースへの XA 呼び出しで許可される最長期間 (ミリ秒単位) を設定する。この設定はドメイン全体に適用される。 デフォルト: 120000
MaxResourceUnavailableMillis	weblogic.management.configuration.JTAMBean	XA リソースが不健全なものとしてマークされる最長期間 (ミリ秒単位)。この期間を過ぎると、トランザクション マネージャに明示的に再登録されなくても、XA リソースは再び使用可能であると宣言される。この設定はドメイン全体に適用される。 デフォルト: 1800000
MaxResourceRequestOnServer	weblogic.management.configuration.JTAMBean	ドメイン内の各サーバで許可される、リソースへの同時リクエストの最大数。 デフォルト: 50 最小値: 10 最大値: java.lang.Integer.MAX_VALUE

これらの属性は、ドメインがアクティブでないときに、`config.xml` ファイルで直接設定します。これらの属性は **Administration Console** には表示されません。以下の例では、これらの属性を含むコンフィグレーション ファイルの抜粋を示します。

```
...
<JTA
  MaxUniqueNameStatistics="5"
  TimeoutSeconds="300"
  RecoveryThresholdMillis="150000"
  MaxResourceUnavailableMillis="900000"
  MaxResourceRequestOnServer="60"
  MaxXACallMillis="180000"
/>
<JDBCConnectionPool
  Name="XAPool"
  Targets="myserver"
```

```
DriverName="weblogic.qa.tests.transaction.  
testsupport.jdbc.XADataSource"  
InitialCapacity="1"  
MaxCapacity="10"  
CapacityIncrement="2"  
RefreshMinutes="5"  
TestTableName="dual"  
EnableResourceHealthMonitoring="true"  
Properties="user=scott;password=tiger;server=dbserver1"  
</>  
  
<JDBCTxDataSource  
Name="XADataSource"  
Targets="myserver"  
JNDIName="weblogic.jdbc.XADS"  
PoolName="XAPool"  
</>  
  
...
```

ドメイン間トランザクションに対するドメインのコンフィグレーション

トランザクション マネージャで分散トランザクションを管理する場合、トランザクションを準備し、その後コミットまたはロールバックするために、トランザクション マネージャはすべての参加サーバと通信する必要があります。これは、**WebLogic** ドメインが、分散トランザクションにおいて、トランザクション マネージャまたはトランザクション参加コンポーネント（リソース）として機能する場合に当てはまります。以下の節では、ドメイン間トランザクションが可能なようにドメインをコンフィグレーションする方法を説明します。**WebLogic** ドメイン間での相互運用性の詳細については、**Administration Console** オンラインヘルプの「**WebLogic** ドメイン間の信頼関係の有効化」を参照してください。

WebLogic Server 7.0 ドメインのドメイン間トランザクション

複数の WebLogic Server 7.0 ドメインにまたがる（つまり、すべての参加ドメインが WebLogic Server 7.0 で実行される）トランザクションを管理する、またはそのようなトランザクションに参加するには、すべてのドメインのセキュリティ資格を同じ値に設定する必要があります。資格の値を設定するには、各参加ドメインについて、次の手順に従います。

1. いずれかの参加ドメインの **Administration Console** を起動します。
2. 左ペインで、ドメイン名 ([コンソール] のすぐ下) をクリックします。右ペインに、そのドメインの属性を示すタブが表示されます。
3. [セキュリティ] タブをクリックし、次に [詳細設定] タブをクリックします。
4. [生成された資格を有効化] チェック ボックスのチェックを解除して、[適用] をクリックします。
5. [資格] の横の [変更] テキスト リンクをクリックします。
6. [新しい資格] フィールドで新しい資格を入力し、次に [再入力] フィールドに入力して確定します。各ドメインについて同じ資格を入力し、[適用] をクリックします。分散トランザクションに WebLogic Server 6.x ドメインが参加する場合は、WebLogic Server 6.x ドメインの `system` パスワードを使用します。
7. サーバを再起動します。
8. ドメイン間トランザクションに参加する各ドメインについて、1～7 の手順を繰り返します。手順 6 では、各ドメインについて同じ資格を入力します。

WebLogic Server 7.0 および WebLogic Server 6.x ドメインのドメイン間トランザクション

WebLogic Server 7.0 および WebLogic Server 6.x の双方のドメインのサーバを使用するトランザクションを管理するには、次の操作が必要です。

参加するすべての WebLogic Server 6.x ドメインにおいて、次の操作を実行します。

- すべての参加ドメインにおいて、Administration Console の [セキュリティ | ユーザ] タブ上で `system` ユーザのパスワードを同じ値に変更します。
<http://edocs.beasys.co.jp/e-docs/wls61/adminguide/cnfgsec.html#891413> の「システム パスワードの変更」を参照してください。

参加するすべての WebLogic Server 7.0 ドメインにおいて、次の操作を実行します。

- すべてのドメインについて、[ドメイン | セキュリティ | 詳細設定] タブでセキュリティ資格を同じ値に設定します。資格は、参加しているすべての WebLogic Server 6.x ドメインの `system` パスワードに一致する必要があります。手順については、7-7 ページの「WebLogic Server 7.0 ドメインのドメイン間トランザクション」を参照してください。

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、リソースや名前でも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 (トランザクション ログの格納場所) を入力し、[適用] をクリックして属性値を保存します。新しいログ ファイルのプレフィックスは、サーバの再起動後に有効になります。
デフォルトでは、トランザクション ログ ファイルのプレフィックスはサーバの作業ディレクトリです。

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以下の節を参照してください。

- [サーバ] → [モニタ] → [JTA]
- [サーバ] → [ログ] → [JTA]
- [ドメイン] → [コンフィグレーション] → [JTA]

トランザクションのモニタ

WebLogic Console を使用して、進行中のトランザクションをモニタできます。『WebLogic JTA プログラマーズ ガイド』の「トランザクションの統計」で説明されている統計に加えて、以下の情報を表示できます。

- 名前別のトランザクション。ロールバックやアクティブ時間の情報が含まれます。
- リソース別のトランザクション。全トランザクション、コミットされたトランザクション、およびロールバックされたトランザクションの統計が含まれます。
- すべてのアクティブなトランザクション。ステータス、サーバ、リソース、プロパティ、およびトランザクション識別子の情報が含まれます。

トランザクション ログ ファイル

各サーバでは、そのサーバで調整されてコミットされたが、未完了の可能性があるトランザクションの情報を格納するトランザクション ログを備えています。

WebLogic Server では、システムのクラッシュやネットワーク障害からの回復時にトランザクション ログを使用します。トランザクション ログを直接表示することはできません。このファイルはバイナリ フォーマットです。

トランザクション ログは、複数のファイルで構成されています。各ファイルは、ガベージ コレクションの対象となります。つまり、トランザクション ログ ファイル内に必要な記録がなければ、ファイルは削除され、そのディスク領域はファイル システムに戻されます。加えて、以前のログ ファイルが大きくなり過ぎたり、チェックポイントが発生したりした場合には、新しいトランザクション ログ ファイルが作成されます。

警告： 手動でトランザクション ログ ファイルを削除しないでください。トランザクション ログ ファイルを削除すると、データに矛盾が発生する可能性があります。

トランザクション ログ ファイルには、パス名プレフィックス、サーバ名、4 桁の数値サフィックス、およびファイル拡張子で構成されたユニークな名前が付けられます。パス名プレフィックスで、ファイルの格納場所が決まります。**WebLogic Administration Console** を使用して、`TransactionLogFilePrefix` サーバ属性の値を指定できます。`TransactionLogFilePrefix` のデフォルトは、サーバの作業ディレクトリです。

トランザクション ログ ファイルが **RAID** デバイスなどの高可用性ファイル システム上に作成されるように `TransactionLogFilePrefix` を設定する必要があります。クラスタ内のサーバでトランザクション回復サービスの移行機能を利用するには、トランザクション ログをサーバとそのバックアップサーバからアクセス可能な場所、できればデュアルポート **SCSI** ディスクまたはストレージエリア ネットワーク (**SAN**) 上に格納する必要があります。詳細については、7-22 ページの「トランザクション回復サービスの移行準備」を参照してください。

サーバ名が `websvr` で `TransactionLogFilePrefix` が `/usr7/applog1/` に設定された **UNIX** システムでは、次のようなログ ファイルが作成されます。


```
/usr7/applog1/websvr0000.tlog  
/usr7/applog1/websvr0001.tlog  
/usr7/applog1/websvr0002.tlog
```

同様に、TransactionLogFilePrefix が C:\weblogic\logA\ に設定された Windows システムでは、以下のような名前で作成されたログ ファイルが作成されます。

```
C:\weblogic\logA\websvr0000.tlog  
C:\weblogic\logA\websvr0001.tlog  
C:\weblogic\logA\websvr0002.tlog
```

システムで大量のトランザクション ログ ファイルが作成されている場合は、完了していない長時間のトランザクションが複数存在している可能性があります。その原因としては、リソース マネージャの障害や、トランザクションのタイムアウト値が著しく大きいことが考えられます。

トランザクション ログ ファイルが含まれたファイルシステムで領域が不足したり、アクセス不能になったりすると、commit() が SystemException を送出し、トランザクション マネージャによってシステム エラー ログにメッセージが書き込まれます。使用できる領域が増やされるまで、トランザクションはコミットされません。

サーバを別のマシンに移行する場合は、トランザクション ログ ファイルも一緒に移動し、1 つのサーバに関するすべてのログ ファイルをまとめておきます。詳細については、7-16 ページの「別のマシンへのサーバの移動」を参照してください。

トランザクション ログ ファイル書き込みポリシーの設定

トランザクション ログ ファイル書き込みポリシーを選択することで、トランザクション ログ ファイルにエントリを書き込む方法を変更できます。選択できるオプションは以下のとおりです。

- **Cache-Flush**—(デフォルト) トランザクション ログにエントリが書き込まれるたびにオペレーティング システムおよびオンディスク キャッシュをフラッシュする。コミット レコードが永続性のあるストレージに書き込まれるまで、トランザクションをコミットすることはできません。
- **Direct-Write**—トランザクション ログ エントリは、書き込みのたびにオペレーティング システムによって直接ディスクに書き込まれる。このオプションは、Windows、Solaris、および HP-UX プラットフォームで選択できます。

警告： Windows で **Direct-Write** トランザクション ログ ファイル書き込みポリシーを選択すると、トランザクション データはオンディスク キャッシュに残され、すぐにはディスクに書き込まれません。この場合、停電によってオンディスク キャッシュのデータが消失するおそれがあり、トランザクションとしては安全とはいえません。Windows で **Direct-Write** トランザクション ログ ファイル書き込みポリシーを使用する場合にキャッシュ データが消失するのを防ぐには、ディスクの書き込みキャッシュをすべて無効にする (デフォルトでは有効) か、システムのバッテリー バックアップを使用します。

警告： トランザクション ログ ファイル書き込みポリシーは、トランザクションのパフォーマンスに影響します。使用しているシステムでこれらのオプションをテストし、どちらのパフォーマンスがよいかを確認してください。オペレーティング システムおよびそのパラメータ設定にもよりますが、通常は「**Direct-Write**」の方が「**Cache-Flush**」よりもパフォーマンスが良く、Windows、HP-UX、および Solaris で利用できます。Windows システムでは、ディスクへの連続的な書き込みが最適化され、ファイルへの最初の書き込みよりも後続の書き込みのほうが速くなります。

トランザクション ログ ファイル エントリは連続的に書き込まれるため、パフォーマンスが向上する可能性があります。一部の UNIX システムで **Cache-Flush** オプションを選択すると、トランザクション ログ ファイルへの書き込みだけでなく、キャッシュされたすべてのディスク書き込みがフラッシュされるため、トランザクションのパフォーマンスが低下するおそれがあります。

トランザクション ログ ファイル書き込みポリシーを設定するには、次の手順に従います。

1. **Administration Console** の左ペインで、[サーバ] ノードをクリックしてサーバを選択します。
2. 右ペインで [ログ] タブを選択し、続いて [JTA] タブを選択します。
3. [トランザクション ログ書き込みポリシー] で [**Cache-Flush**] (デフォルト) または [**Direct Write**] を選択します。
4. [適用] をクリックして属性設定を保存します。新しいログ ファイルの書き込みポリシーは、サーバの再起動後に有効になります。

ヒューリスティックなログ ファイル

外部のトランザクション マネージャから WebLogic Server にトランザクションをインポートする場合、WebLogic Server トランザクション マネージャはその外部トランザクション マネージャによって調整される XA リソースとして機能します。トランザクションを保持する最長時間の経過後、または WebLogic Server にインポートされたトランザクションに参加している XA リソースがヒューリスティック例外を送出した場合など、滅多にないが破壊力の大きい状況下では、WebLogic Server トランザクション マネージャはヒューリスティックな決定を行います。つまり、WebLogic Server トランザクション マネージャは、外部トランザクション マネージャからの入力がない状態で、トランザクションをコミットするかロールバックするかを決定します。WebLogic Server トランザクション マネージャは、ヒューリスティックな決定を行うと、外部トランザクション マネージャからそのトランザクションの記録を破棄する指示があるまで、その決定の情報をヒューリスティック ログ ファイルに格納します。

ヒューリスティック ログ ファイルはトランザクション ログ ファイルと共に格納されます。.tlog 拡張子の前に .heur が付く、トランザクション ログ ファイルと似たファイル名になります。これらのファイルでは次のフォーマットを使用します。

```
<TLOG_file_prefix>\<server_name><4-digit number>.heur.tlog
```

サーバ名 websvr の UNIX システムでは、次のような名前のヒューリスティック ログ ファイルになります。

```
/usr7/applog1/websvr0000.heur.tlog  
/usr7/applog1/websvr0001.heur.tlog  
/usr7/applog1/websvr0002.heur.tlog
```

同様に、Windows システムでは以下のような名前でヒューリスティック ログ ファイルが作成されます。

```
C:\weblogic\logA\websvr0000.heur.tlog  
C:\weblogic\logA\websvr0001.heur.tlog  
C:\weblogic\logA\websvr0002.heur.tlog
```

ヒューリスティックな終了の処理

ヒューリスティックな終了（ヒューリスティックな決定）は、更新をコミットまたはロールバックする分散トランザクションの終了段階で、リソースが一方だけの決定を行ったときに発生します。これにより、分散されたデータは不確定な状態のままになります。ヒューリスティックな終了の原因としては、ネットワークの障害またはリソースのタイムアウトが考えられます。ヒューリスティックな終了が発生すると、以下のヒューリスティックな結果例外のいずれかが送出されます。

- **HeuristicRollback**— トランザクションに参加する 1 つのリソースが、準備してコミットの決定を待つことに同意しているにもかかわらず、処理のロールバックを自発的に決定した場合。トランザクション マネージャがトランザクションのコミットを決定すると、そのリソースによるヒューリスティックなロールバックの決定は不正になり、トランザクションの他のブランチはコミットされるため、矛盾した結果を招きます。
- **HeuristicCommit**— トランザクションに参加する 1 つのリソースが、準備してコミットの決定を待つことに同意しているにもかかわらず、処理のコミットを自発的に決定した場合。トランザクション マネージャがトランザクションのロールバックを決定すると、そのリソースによるヒューリスティックなコミットの決定は不正になり、トランザクションの他のブランチはロールバックされるため、矛盾した結果を招きます。
- **HeuristicMixed**— トランザクションが、参加リソースの一部がコミットし、一部がロールバックするという混合した結果になったことを、トランザクション マネージャで認識している場合。主に、1 つまたは複数の参加リソースが、ヒューリスティックなロールバックまたはヒューリスティックなコミットを決定したことが原因で発生します。
- **HeuristicHazard**— トランザクションが、参加リソースの一部がコミットし、一部がロールバックするという混合した結果になったことを、トランザクション マネージャで認識している場合。ただし、システムまたはリソースに障害があるため、**HeuristicMixed** の結果が確かに発生したかどうかを確認できない場合。主に、1 つまたは複数の参加リソースが、ヒューリスティックなロールバックまたはヒューリスティックなコミットを決定したことが原因で発生します。

ヒューリスティックな終了が発生すると、サーバログにメッセージが書き込まれます。ヒューリスティックな終了を解決する方法については、データベースベンダが提供するドキュメントを参照してください。

一部のリソース マネージャでは、ヒューリスティックな終了のコンテキスト情報が保存されます。この情報は、リソース マネージャのデータの矛盾を解決するときに便利です。WebLogic Console の JTA パネルで ForgetHeuristics 属性 ([ヒューリスティックを無視]) が true に設定されている場合、この情報はヒューリスティックな終了の後で削除されます。コンテキスト情報を保存するリソース マネージャを使用するときには、ForgetHeuristics 属性を false に設定することをお勧めします。

トランザクションの破棄

指定した時間の経過後に未完了のトランザクションを破棄することもできます。分散トランザクションの 2 フェーズ コミット プロセスでは、トランザクション マネージャはトランザクションに参加するすべてのリソース マネージャを調整します。すべてのリソース マネージャがコミットまたはロールバックを支持すると、トランザクション マネージャはリソース マネージャに、変更に対してコミットまたはロールバックのいずれかの動作を行うよう通知します。2 フェーズ コミット プロセスにおける、この第 2 フェーズでは、トランザクション マネージャはすべてのリソース マネージャでトランザクションの終了が示されるまで、トランザクションを終了しようとし続けます。AbandonTimeoutSeconds 属性を使用すると、トランザクション マネージャがコミット プロトコルの第 2 フェーズでトランザクションの終了を試み続ける最長時間を秒単位で設定できます。デフォルト値は 86400 秒 (24 時間) です。この時間を過ぎると、利用できないリソースや、トランザクションの結果を承認できないリソースが関わるトランザクションでそれ以上の解決は行われません。破棄される前にトランザクションの準備が完了していた場合、トランザクション マネージャは破棄されるトランザクションに代わってトランザクションをロールバックし、ロックを解放してから、サーバ ログにヒューリスティック エラーを書き込みます。

AbandonTimeoutSeconds 属性 ([トランザクションを保持する最長時間]) の設定方法については、Administration Console オンライン ヘルプの「JTA のコンフィグレーション」を参照してください。2 フェーズ コミット プロセスの詳細については、『WebLogic JTA プログラマーズ ガイド』の「分散トランザクションと 2 フェーズ コミット プロトコル」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、そのマシン上で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。新しいシステム上で WebLogic Server を起動すると、サーバはトランザクション ログ ファイルを読み取り、保留中のトランザクションがあれば、それを回復します。新しいマシンでパス名が異なる場合は、TransactionLogFilePrefix 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。TransactionLogFilePrefix の変更方法については、Administration Console オンライン ヘルプの「トランザクション ログ ファイルの場所 (プレフィックス) の指定」を参照してください。

サーバに障害が発生した後のトランザクションの回復

WebLogic Server のトランザクション マネージャは、ユーザによる最低限の介入でシステムのクラッシュから回復するように設計されています。トランザクション マネージャは、クラッシュが何度も発生した後や、クラッシュの回復中であっても、リソース マネージャによって準備されたトランザクション ブランチをコミットまたはロールバックによって解決するために、あらゆる努力を払います。

クラッシュ後の回復を容易にするため、WebLogic Server ではトランザクション回復サービスを提供しています。このサービスでは、システムの起動時にトランザクションの回復が自動的に行われます。トランザクション回復サービスは、サーバのトランザクション ログを所有しています。7-18 ページの「クラッシュ後のトランザクション回復サービスのアクション」で説明するように、トランザクション回復サービスは、起動時にすべてのログ ファイルを解析して未完了のトランザクションを見つけ、そのトランザクションを終了させます。

トランザクション回復サービスは、クラッシュ後にトランザクションの回復を正常に処理するよう設計されているため、サーバがクラッシュした場合は再起動し、トランザクション回復サービスによって未完了のトランザクションを処理することをお勧めします。

サーバがクラッシュして、適切な時間内に再起動できそうにない場合は、ユーザによる処理が必要になることがあります。サーバの障害発生後にトランザクションを回復する手順は、WebLogic Server の環境によって異なります。クラスタ化されていないサーバの場合は、手動でサーバ（およびトランザクション ログ ファイル）を別のシステム（マシン）に移動することでトランザクションを回復できます。詳細については、7-19 ページの「クラスタ化されていないサーバで障害が発生した場合のトランザクションの回復」を参照してください。クラスタ内のサーバの場合は、同じクラスタ内の別のサーバに、トランザクション回復サービスを手動で移行できます。トランザクション回復サービスの移行には、トランザクションを回復するためのトランザクション ログにアクセスできるサーバを選択し、次に Administration Console または WebLogic コマンドライン インタフェースを使用してサービスを移行することが必要です。

注意： クラスタ化されていないサーバの場合は、サーバ全体を新しいシステムに移動することしかできません。クラスタ化されたサーバでは、トランザクション回復サービスを一時的に移行できます。

トランザクション回復サービスの移行の詳細については、7-20 ページの「クラスタ化されたサーバで障害が発生した場合のトランザクションの回復」を参照してください。クラスタの詳細については、『WebLogic Server クラスタ ユーザーズ ガイド』を参照してください。

クラッシュ後のトランザクション回復サービスのアクション

クラッシュ後にサーバを再起動したり、トランザクション回復サービスを別の(バックアップ)サーバに移行したりすると、トランザクション回復サービスは以下の処理を行います。

- 2フェーズコミットの第2フェーズの準備が整っているトランザクションを終了します。

コミットの決定が行われたが2フェーズコミットプロセスの第2フェーズが終了していないトランザクション(トランザクションログに記録されているトランザクション)については、トランザクション回復サービスがコミットプロセスを終了させます。

- 準備されたトランザクションを解決します。

トランザクションマネージャによってリソースマネージャとの準備が整っているトランザクション(2フェーズコミットプロセスの第1フェーズのトランザクション)の場合、トランザクション回復サービスはクラッシュ回復中に、各リソースマネージャに対して `XAResource.recover()` を呼び出す必要があります。次に、`commit()`、`rollback()`、または `forget()` メソッドを呼び出すことで、`recover()` によって返されたすべてのトランザクションIDを最終的に解決します。

- ヒューリスティックな終了を報告します。

リソースマネージャによってヒューリスティックな例外が報告されると、トランザクション回復サービスはそのヒューリスティックな例外をサーバログに記録し、[ヒューリスティックを無視]コンフィグレーション属性が有効になっている場合は、`forget()` を呼び出します。[ヒューリスティックを無視]コンフィグレーション属性が有効になっていない場合は、データベースベンダのマニュアルでヒューリスティックな終了の解決に関する情報を参照してください。詳細については、7-14 ページの「ヒューリスティックな終了の処理」を参照してください。

トランザクション回復サービスには、次のような利点があります。

- 複数のリソースにわたって一貫性を維持します。

トランザクション回復サービスは、一貫した予測しやすい方法でトランザクション回復処理を行います。クラッシュ前にコミットが決定されたものの、コミットがまだ行われておらず、`XAResource.recover()` からトランザクション ID が返されるトランザクションの場合、トランザクション回復サービスは一貫して `XAResource.commit()` を呼び出します。クラッシュ前にコミットが決定しておらず、`XAResource.recover()` からトランザクション ID が返されるトランザクションの場合、トランザクション回復サービスは一貫して `XAResource.rollback()` を呼び出します。トランザクションの回復が一貫性のある予測しやすいものであるため、トランザクション マネージャがクラッシュしても、一部のプランチがコミットされ一部がロールバックされるという混合ヒューリスティックの終了は発生しません。

- トランザクションの解決に固執します。

リソース マネージャがクラッシュすると、トランザクション回復サービスでは準備されている各トランザクションについて `commit()` または `rollback()` を呼び出さなければなりません。`commit()` または `rollback()` の呼び出しは成功するまで続けられます。トランザクション解決の試行は、`AbandonTimeoutSeconds` コンフィグレーション属性を設定することで制限できます。詳細については、7-15 ページの「トランザクションの破棄」を参照してください。

クラスタ化されていないサーバで障害が発生した場合のトランザクションの回復

障害が発生したサーバでトランザクションを回復するには、次の手順に従います。

1. すべてのトランザクション ログ ファイルを、障害が発生したサーバから新しいサーバへ移動させます (新しいサーバで利用できるようにします)。
2. `TransactionLogFilePrefix` 属性に、トランザクション ログ ファイルへのパスを設定します。手順については、**Administration Console** オンライン ヘルプの「トランザクション ログ ファイルの場所 (プレフィックス) の指定」を参照してください。

3. 新しいサーバを起動します。7-18 ページの「クラッシュ後のトランザクション回復サービスのアクション」で説明するように、トランザクション回復サービスはすべてのトランザクション ログ ファイルを検索して未完了のトランザクションを見つけ、そのトランザクションを終了させます。

サーバに障害が発生した後でトランザクション ログを移動する場合は、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、`TransactionLogFilePrefix` 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。

注意： トランザクション回復サービスは、クラッシュ後にトランザクションの回復を正常に処理するように設計されています。サーバがクラッシュした場合は、新しいマシンにサーバを移動するよりも、サーバを再起動し、トランザクション回復サービスによって未完了のトランザクションを処理することをお勧めします。

クラスタ化されたサーバで障害が発生した場合のトランザクションの回復

クラスタ化されたサーバがクラッシュした場合は、**Administration Console** またはコマンドライン インタフェースを使用して、同じクラスタ内でクラッシュしたサーバから別のサーバへ、トランザクション回復サービスを手動で移行できます。次のイベントが発生します。

1. トランザクション ログの所有権がクラッシュしたサーバからバックアップサーバのトランザクション回復サービスに移ります。
2. 7-18 ページの「クラッシュ後のトランザクション回復サービスのアクション」で説明するように、トランザクション回復サービスは障害が発生したサーバのすべてのトランザクション ログ ファイルを検索して未完了のトランザクションを見つけ、そのトランザクションを終了させます。

3. バックアップ サーバのトランザクション回復サービスにより、障害が発生したサーバで未完了のトランザクションがすべて正常に終了すると、サーバは障害が発生したサーバのトランザクション回復サービスに対する所有権を解放し、障害が発生したサーバが再起動時にその所有権を再び要求できるようにします。

Administration Console を使用してトランザクション回復サービスを移行する手順については、**Administration Console** オンラインヘルプの「トランザクション回復サービスの同一クラスタ内サーバへの移行」を参照してください。コマンドライン インタフェースを使用してトランザクション回復サービスを移行する手順については、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」の「MIGRATE」を参照してください。

サーバでは、障害が発生した複数のサーバに対してトランザクション回復を行うことができます。他のサーバのトランザクションを回復している間に、バックアップサーバは自身のトランザクションの処理および回復を続けます。回復中にバックアップサーバで障害が発生した場合は、さらに別のサーバにトランザクション回復サービスを移行できます。この別のサーバで、トランザクション回復は続行されます。また、**Administration Console** またはコマンドライン インタフェースを使用して、トランザクション回復サービスを元の障害が発生したサーバに手動で移行して戻すこともできます。詳細については、**Administration Console** オンラインヘルプの「トランザクション回復サービスの元のサーバへの手動による返還移行」を参照してください。

サーバのトランザクション回復がバックアップサーバで完了すると、バックアップサーバはトランザクション回復サービス（およびトランザクションログ）の所有権を、障害が発生したサーバに解放します。障害が発生したサーバを再起動すると、このサーバはトランザクション回復サービスの所有権を再び要求しようとし、障害が発生したサーバを再起動したときに、バックアップサーバがトランザクション回復処理中だった場合、バックアップサーバはトランザクションの回復を中止し、内部クリーンアップを行い、トランザクション回復サービスの所有権を解放して、障害が発生したサーバがそれを再び要求して正しく起動できるようにします。その後、障害が発生したサーバでは、自身のトランザクション回復を完了します。

障害が発生したサーバのトランザクション回復サービスがまだバックアップサーバで所有されており、障害が発生したサーバを再起動しようとしたときにバックアップサーバが非アクティブだった場合、バックアップサーバがトランザクション回復サービスの所有権を解放できないため、障害が発生したサーバは起動しません。これは、フェイルバック メカニズムが失敗した場合や、バック

アップ サーバが管理サーバと通信できない場合にも当てはまります。トランザクション回復の移行は、Administration Console またはコマンドライン インタフェースを使用して手動で行えます。

トランザクション回復サービスの移行に対する制限

トランザクション回復サービスを移行する場合は、以下の制限があります。

- トランザクション回復サービスを、実行中のサーバからバックアップサーバへ移行することはできません。トランザクション回復サービスを移行する前に、サーバを停止する必要があります。
- バックアップサーバでは、障害が発生したサーバに対する新しいトランザクション処理は受け付けません。終了していないトランザクションを処理するだけです。
- バックアップサーバでは、ヒューリスティック ログ ファイルは処理しません。
- バックアップサーバは、WebLogic Server によって書き込まれたログ記録のみを処理します。WebLogic Tuxedo Connector などのゲートウェイ実装によって書き込まれたログ記録の処理は行いません。

トランザクション回復サービスの移行準備

クラスタ内の障害が発生したサーバから同じクラスタ内の別のサーバ (バックアップサーバ) へトランザクション回復サービスを移行するには、バックアップサーバに、障害が発生したサーバのトランザクション ログ ファイルへのアクセス権が必要です。したがってトランザクション ログ ファイルは、双方の (またはさらに他の) サーバで使用できる永続ストレージに格納する必要があります。ストレージエリア ネットワーク (SAN) デバイスまたはデュアルポート 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をお勧めします。NFS ファイル システムを使ってトランザクション ログ ファイルを保存しないでください。NFS のキャッシング方式により、ディスク上のトランザクション ログ ファイルは常に現在のものとは限りません。NFS デバイスに保存されたトランザクション ログ ファイルを使用して回復を行うと、データが破損するおそれがあります。

トランザクション回復サービスをサーバから移行するときには、実際に移行を行う前に障害が発生している、または障害が発生したサーバを停止する必要があります。元のサーバがまだ実行中の場合は、そこからトランザクション回復サービスを移行することはできません。

トランザクション回復サービスを移行する手順の詳細については、**Administration Console** オンライン ヘルプの「トランザクション回復サービスの同一クラスター内サーバへの移行」を参照してください。

コマンドラインを使用しても、トランザクション回復サービスを移行することができます。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」の「MIGRATE」を参照してください。

8 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。

- 8-1 ページの「JDBC 管理の概要」
- 8-4 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)」
- 8-8 ページの「JDBC 接続プールのセキュリティ」
- 8-10 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」
- 8-20 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」
- 8-42 ページの「Prepared Statement キャッシュのパフォーマンスの向上」

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 Administration 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 WebLogic 管理 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、永続的な接続、つまりサーバを停止して再起動した後も利用できる接続プール、データソース、トランザクション データソース、およびマルチプールを設定します。これらの **JDBC** オブジェクトは、静的オブジェクトと呼ばれます。動的オブジェクト、つまり使用後に削除することが想定されているオブジェクトは、管理コマンドラインまたはアプリケーションコードで作成できます。

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、接続プールおよびデータソース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、『**WebLogic JDBC プログラマーズ 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 コンポーネントと関係して機能し、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、1-22 ページの「Administration Console の起動と使い方」を参照してください。
- JDBC 属性のリストについては、『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。
- 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 プログマーズ 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 jDriver の使い方については、「WebLogic jDriver for Oracle の使い方」または「WebLogic jDriver for Microsoft SQL Server の使い方」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、7-1 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、『WebLogic JTA プログラマーズ ガイド』の「WebLogic Server でのサードパーティ JDBC XA ドライバの使い方」を参照してください。

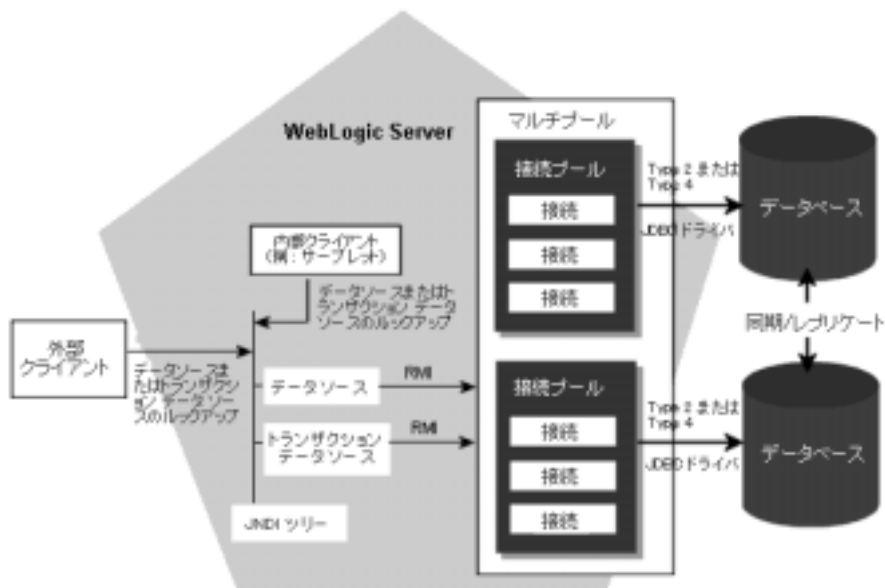
以下のドキュメントは、おもにアプリケーション開発者向けに書かれています。システム管理者は、この章の内容を理解するための補助資料として、必要に応じて以下の情報を参照してください。

- 分散トランザクションについては、『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、『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。

図 8-1 WebLogic Server の JDBC コンポーネント



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 (WebLogic Server の起動時、または対象となるサーバやクラスタに接続プールを割り当てるとき) に作成される JDBC 接続のグループです。接続プールでは、物理的なデータベース接続の作成に、Type 2 または Type 4 の JDBC ドライバを使用します。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戻します。詳細については、『WebLogic JDBC プログラマーズガイド』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 を停止して再起動した後も持続します。動的な接続プール、つまりサーバの実行中に使用および削除することが想定されている接続プールは、コマンドラインを使用して (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 を使用してプログラムで (『WebLogic JDBC プログラマーズ ガイド』の「接続プールの動的作成」を参照) 動的に作成できます。

接続プールの設定は `config.xml` ファイルに保持されます。動的に作成された接続プールの設定も、プログラムによって接続プールが削除されるまではこのファイルに保持されます。`config.xml` ファイル内のエントリについては、『コンフィグレーションリファレンス』の「JDBCConnectionPool」を参照してください。

アプリケーションスコープの JDBC 接続プール

エンタープライズアプリケーションをパッケージ化するとき、`weblogic-application.xml` 補足デプロイメント記述子を含めることができます。この記述子を使用して、アプリケーション スコーピングをコンフィグレーションします。`weblogic-application.xml` ファイルでは、エンタープライズアプリケーションをデプロイするときに作成される JDBC 接続プールをコンフィグレーションできます。

接続プールのインスタンスは、アプリケーションのインスタンスごとに作成されます。つまり、プールのインスタンスは、アプリケーションの対象となっている各ノード上のアプリケーションで作成されます。プールのサイズを検討する場合は、この点に留意してください。

この方法で作成された接続プールは「アプリケーション スコープの接続プール」(アプリケーション スコーププール、アプリケーション ローカルプール、またはローカルプール)と呼ばれ、そのエンタープライズアプリケーションに対してだけスコーピングされます。つまり、各接続プールがエンタープライズアプリケーションごとに分離されます。

アプリケーション スコーピング、およびアプリケーション スコープ リソースの詳細については、以下を参照してください。

- 『WebLogic Server アプリケーションの開発』の「`weblogic-application.xml` デプロイメント記述子の要素」
- 『WebLogic Server アプリケーションの開発』の「エンタープライズアプリケーションのパッケージ化」
- 『WebLogic Server アプリケーションの開発』の「2 フェーズ デプロイメント」

マルチプール

マルチプールとは、接続プールのプールです。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、以下の機能に役立ちます。

- **ロード バランシング** — プールは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、**WebLogic Server** はリストの順序で次の接続プールから接続を選択します。
- **高可用性** — 接続プールは一定の順序でリストされ、その順序で接続プールの切り替えが発生します。つまり、**WebLogic Server** はリスト上の最初の接続プールからデータベース接続を提供します。この接続プールが失敗した場合は 2 番目以降の接続プールからデータベース接続を順番に使用してゆきます。

データ ソース

データ ソース オブジェクトによって、**JDBC** アプリケーションは接続プールから **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、**JNDI** ツリーにバインドし、接続プールまたはマルチプールを指します。アプリケーションは、データ ソースをロックアップして接続を取得します。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 (**Administration Console** の [トランザクション データ ソース])、使用せずに定義することもできます (**Administration Console** の [データ ソース])。分散トランザクションには、[トランザクション データ ソース] を使用します。データ ソースおよびトランザクション データ ソースの使い方の詳細については、8-20 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」を参照してください。

JDBC データ ソース ファクトリ

WebLogic Server では、**JDBC** データ ソース リソースを、リソース ファクトリとして **WebLogic Server JNDI** ツリーにバインドできます。その後、**EJB** デプロイメント記述子のリソース ファクトリ参照を、実行中の **WebLogic Server** の利用可能なリソース ファクトリにマップすると、接続プールから接続を取得できます。

JDBC データ ソース ファクトリの作成と使用方法の詳細については、『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』の「リソース ファクトリ」を参照してください。

JDBC 接続プールのセキュリティ

オプションとして、JDBC 接続プールへのアクセスを制限できます。以前のバージョンの WebLogic Server では、ACL を使って WebLogic リソースを保護していました。WebLogic Server バージョン 7.0 では、WebLogic リソースへの「アクセス権は誰が持つか」という問いに、セキュリティ ポリシーが答えます。セキュリティ ポリシーは、WebLogic リソースとユーザ、グループ、またはロールの間の関連付けを定義するときに作成します。WebLogic リソースは、セキュリティ ポリシーが割り当てられるまでは保護されません。すべての WebLogic Server リソースのセキュリティを設定する手順については、Administration Console オンライン ヘルプの「WebLogic リソースの保護の設定」を参照してください。

互換性モードにおける JDBC 接続プールのセキュリティ

WebLogic Server 7.0 では、下位互換性のためにバージョン 6.1 のセキュリティ モデルを引き続きサポートしています。バージョン 6.1 のセキュリティを使用するには、互換性モードで実行する必要があります。互換性モードでの実行の詳細については、以下のドキュメントを参照してください。

- Administration Console オンライン ヘルプの「セキュリティ互換性モード」
- 『BEA WebLogic Server 7.0 へのアップグレード』の「WebLogic Server 6.x から WebLogic Server 7.0 へのアプリケーションの移植」
- 『BEA WebLogic Server 7.0 へのアップグレード』の「セキュリティのアップグレード」

WebLogic Server 6.1 では、デフォルトのセキュリティ レルムはファイル レルムでした。ファイル レルムは、認証と認可に `fileRealm.properties` ファイルの ACL を使用します。接続プールは、(リソース タイプとして) 接続プール群また

は個々の接続プールに対し ACL を定義するまでは、保護されません。接続プールに対して ACL を定義すると、その ACL に定義されているものだけにアクセスが制限されます。たとえば、fileRealm.properties ファイル内で接続プールの ACL が定義されるまでは、ドメイン内のすべての接続プールに誰でも無制限にアクセスできます。しかし、ファイルに次の行を追加すると、アクセス権は大幅に制限されます。

```
acl.reset.weblogic.jdbc.connectionPool=Administrators
```

この行により、すべての接続プールの管理者にリセット特権が付与され、他のすべてのユーザによる他のアクションはすべて禁止されます。ACL を追加することで、接続プールのファイル レalm保護がアクティブになります。WebLogic Server では、fileRealm.properties で定義された ACL を強制し、ファイル内で明確に付与されたアクセスのみを許可します。接続プールのリセットだけを制限する目的で ACL を追加するのであれば、他のアクションに対する特権を、全員または特定のロールやユーザに、明確に付与する必要があります。次に例を示します。

```
acl.reserve.weblogic.jdbc.connectionPool=everyone
acl.shrink.weblogic.jdbc.connectionPool=everyone
acl.admin.weblogic.jdbc.connectionPool=everyone
```

表 8-1 には、fileRealm.properties で接続プールのセキュリティ保護を行うために使用できる ACL を示します。

表 8-1 ファイル レalmの JDBC ACL

使用する ACL	制限されるアクション
reserve.weblogic.jdbc.connectionPool[.poolname]	接続プール内で接続を予約する。
reset.weblogic.jdbc.connectionPool[.poolname]	割り当てられた接続をすべて停止して再確立することにより、接続プール内で全接続をリセットする。
shrink.weblogic.jdbc.connectionPool[.poolname]	接続プールを元のサイズ(接続数)に縮小する。
admin.weblogic.jdbc.connectionPool[.poolname]	接続プールを有効化、無効化、および停止する。

表 8-1 ファイル レルムの JDBC ACL

使用する ACL	制限されるアクション
<code>admin.weblogic.jdbc.connectionPoolcreate</code>	静的な接続プールを作成する。

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 8-3 を参照してください。

Administration Console で行う JDBC の設定は、接続プール、マルチプール、DataSource、および TxDataSource のコンフィグレーション設定も含め、そのドメインの `config.xml` ファイルに保持されます。このファイル内のエントリについては、『コンフィグレーション リファレンス』の以下の節を参照してください。

- 「JDBCConnectionPool」
- 「JDBCMultiPool」
- 「JDBCDataSource」
- 「JDBCTxDataSource」

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。8-13 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。

まず接続プールおよびオプションで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、そのデータ ソースは特定の 1 つの接続プールまたはマルチプール (「プール」) と関連付けられます。

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。

- 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る。
- クラスタでは、データ ソースおよび関連の接続プールを、クラスタ内の個々のサーバではなく、クラスタに割り当てる。
- 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同じ対象、すなわちサーバまたはクラスタに割り当てる。

クラスタ内の接続プール、データ ソース、およびマルチプールの詳細については、『WebLogic Server クラスタ ユーザーズ ガイド』の「JDBC 接続」を参照してください。実行するタスクの説明については、8-13 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 8-2 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。	<ol style="list-style-type: none"> データソース A を管理対象サーバ 1 に割り当てる。 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。	<ol style="list-style-type: none"> データソース B をクラスター X に割り当てる。 接続プール B をクラスター 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/クラスターの対象に割り当てられる。
3	データソース C と接続プール C を関連付ける。	<ul style="list-style-type: none"> ■ データソース C と接続プール C を管理対象サーバ 1 に割り当てる。 および ■ データソース C をクラスター X に割り当て、接続プール C をクラスター 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。

これらのデータソースと接続プールの組み合わせを複数のサーバまたはクラスターに割り当てることはできますが、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。

動的な接続プールは、(サーバの起動後に) WebLogic API を使用しても (『WebLogic JDBC プログラマーズ ガイド』の「動的接続プールの作成」を参照)、コマンドライン インタフェースを使用しても (8-17 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)、コンフィグレーションできます。また WebLogic Server では、インストールの際に

サンプルのインストールを選択した場合、サーバ サンプル内に動的なデータソースおよび接続プールを作成およびコンフィグレーションするためのサンプルコードが入っています。 `SAMPLES_HOME\server\src\examples\jdbc` を参照してください。ここで `SAMPLES_HOME` は、WebLogic プラットフォーム用のすべてのサンプルおよび例の格納先の最上位ディレクトリです (デフォルトでは `c:\bea\weblogic700\samples`)。

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、以下の手順を実行します。

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. ツリー内で、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、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) のノードを選択します。
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 8-3 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。これらのタスクは異なる順序で実行してもかまいませんが、オブジェクトは関連付けや割り当てを行う前にコンフィグレーションする必要があります。

表 8-3 JDBC のコンフィグレーションタスク

JDBC コンポーネント/タスク	説明
接続プールのコンフィグレーション	右ペインの [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。

表 8-3 JDBC のコンフィグレーション タスク

JDBC コンポーネント / タスク	説明
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。
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 8-2 「関連付けと割り当てのシナリオ」を参照。 また、複数の接続プールをサーバに割り当てる方法については、オンラインヘルプの「サーバへの JDBC 接続プールの割り当て」を参照。
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。
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。
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。
データソース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データソースをサーバまたはクラスタに割り当てる。

表 8-3 JDBC のコンフィグレーションタスク

JDBC コンポーネント/タスク	説明
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。
トランザクションデータソースのサーバへの割り当て	[対象] タブを使用して、コンフィグレーションされたトランザクションデータソースをサーバに割り当てる。

接続プールのコンフィグレーションにおけるデータベースパスワード

接続プールを作成する場合、一般にはデータベースへの接続用として、1つ以上のパスワードを組み込みます。オープン文字列を使用して、XA を有効にする場合は、パスワードを2つ使用できます。パスワードは、名前と値のペアとして[プロパティ]フィールドに入力するか、名前と値をそれぞれ対応するフィールドに入力します。

- **[パスワード]**。このフィールドを使用して、データベースパスワードを設定します。この値によって、物理的なデータベース接続の作成時に2層 JDBC ドライバに渡される、[プロパティ]の定義にある一切の password 値がオーバーライドされます。この値は config.xml ファイル内で (JDBCConnectionPool タグの [パスワード] 属性として格納されて) 暗号化され、Administration Console 上では非表示になります。
- **[オープン文字列のパスワード]**。このフィールドを使用して、WebLogic Server のトランザクションマネージャがデータベース接続を開くときに使用するオープン文字列内にパスワードを設定します。この値によって、[プロパティ]フィールド内のオープン文字列の一部として定義されたすべてのパスワードがオーバーライドされます。この値は config.xml ファイル内で (JDBCConnectionPool タグの XAPassword 属性として格納されて) 暗号化さ

れ、Administration Console 上では非表示になります。実行時に、WebLogic Server はこのフィールドで指定されたオープン文字列を再構築します。[プロパティ] フィールドのオープン文字列は、次のフォーマットにする必要があります。

```
openString=Oracle_XA+Acc=P/userName/+SesTm=177+DB=dbHost+Threads=true+Sqlnet=dvi0+logDir=.
```

userName の後にパスワードが入らないことに注意してください。

接続プールを初めてコンフィグレーションするときに [プロパティ] フィールドにパスワードを指定した場合、次の起動時に WebLogic Server は、パスワードを Properties 文字列から取り除き、その値を暗号化して Password の値として設定します。既に接続プールの [パスワード] 属性に値が設定されている場合は、どの値も変更されません。ただし、[プロパティ] の文字列の値は、[パスワード] 属性の値によってオーバーライドされます。この動作は、オープン文字列の一部として定義するすべてのパスワードに対して同じように適用されます。たとえば、接続プールを初めてコンフィグレーションするときに次のプロパティを指定したとします。

```
user=scott;
password=tiger;
openString=Oracle_XA+Acc=p/scott/tiger+SesTm=177+db=dbHost+Threads=true+Sqlnet=lcs817+logDir=..+dbgFl=0x15;server=dbHost
```

次の WebLogic Server の起動時には、データベースパスワードが [パスワード] 属性に、オープン文字列に含まれているパスワードが [オープン文字列のパスワード] 属性にそれぞれ移され、[プロパティ] フィールドには次の値が残ります。

```
user=scott;
openString=Oracle_XA+Acc=p/scott/+SesTm=177+db=dbHost+Threads=true+Sqlnet=lcs817+logDir=..+dbgFl=0x15;server=dbHost
```

[パスワード] 属性または [オープン文字列のパスワード] 属性の値が確定すると、これらの値によって [プロパティ] 属性のそれぞれの値がオーバーライドされます。これを先の例で説明すると、[パスワード] 属性に暗号化されたデータベースパスワードとして tiger が設定されているため、[プロパティ] 属性内のデータベースパスワードとして tiger2 を指定しても無視されます。データベースパスワードを変更するには、[パスワード] 属性を変更する必要があります。

注意： [パスワード] と [オープン文字列のパスワード] の値は、同じでなくてもかまいません。

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 8-4 動的 JDBC 接続プールの接続の設定

目的とする作業	使用するツール
動的接続プールの作成	<ul style="list-style-type: none">■ コマンドライン —B-42 ページの「CREATE_POOL」、または■ API—『WebLogic JDBC プログラマーズ ガイド』の「WebLogic JDBC のコンフィグレーションと管理」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 プログラマーズ ガイド』の「接続プールの動的作成」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 8-5 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	<p>「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[選択可]に移動)し、新しい対象に割り当てる。</p> <p>複数の接続プールをサーバに割り当てる方法については、オンラインヘルプの「サーバへの JDBC 接続プールの割り当て」を参照。</p>
マルチプールの割り当て先クラスタの変更	<p>「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。</p>
JDBC 接続プールの削除	<p>オンラインヘルプの「JDBC 接続プールの削除」を参照。</p>
マルチプールの削除	<p>オンラインヘルプの「JDBC マルチプールの削除」を参照。</p>
データソースの削除	<p>オンラインヘルプの「JDBC 接続プールの削除」を参照。</p>
接続プールのモニタ	<p>1つの接続プールの接続をモニタする方法については、オンラインヘルプの「JDBC 接続プールのモニタ」を参照。</p> <p>サーバのすべてのアクティブな接続プールをモニタする方法については、オンラインヘルプの「すべてのアクティブな JDBC 接続プールのモニタ」を参照。</p>

表 8-5 JDBC 管理タスク

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	<ol style="list-style-type: none"> 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、各サーバおよびクラスタ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 [選択可] 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 8-6 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

表 8-6 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

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で使用される接続プール、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アプリケーションコードまたはコマンドライン) 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。

トランザクションシナリオのタイプは以下の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
- XA 対応ドライバを使用する分散トランザクション — 2 フェーズコミットを使用する、複数の参加コンポーネントでの分散トランザクション
- XA 非対応ドライバを使用する分散トランザクション — 2 フェーズコミットをエミュレートする、1つのリソースマネージャと1つのデータベースインスタンスでのトランザクション

システム内でトランザクションが処理される方法に応じて、データソースオブジェクト (DataSources および TxDataSources)、接続プール、およびマルチプールをコンフィグレーションします。次の表に、3つのトランザクションシナリオ用にこれらのオブジェクトをコンフィグレーションする方法を示します。

表 8-7 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	<ul style="list-style-type: none"> ■ WebLogic jDriver for Oracle および WebLogic jDriver for Microsoft SQL Server ■ 準拠するサードパーティ ドライバ 	<ul style="list-style-type: none"> ■ WebLogic jDriver for Oracle/ XA ■ 準拠するサードパーティ ドライバ 	<ul style="list-style-type: none"> ■ WebLogic jDriver for Oracle および WebLogic jDriver for Microsoft SQL Server ■ 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を推奨 (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が必須。 複数のリソースが関与している場合は、[非 XA ドライバ用に 2 フェーズコミットをエミュレート] を選択する (enable two-phase commit=true を設定)。8-37 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。

表 8-7 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
マルチプル	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の場合は、XA 対応ドライバ (WebLogic jDriver for Oracle の XA 対応バージョンである WebLogic jDriver for Oracle/XA など) を使用します。

トランザクション データ ソースを使用すべき場合

アプリケーションまたは環境が以下の条件のいずれかに合てはまる場合は、データソースではなくトランザクション データ ソースを使用してください。

- Java Transaction API (JTA) を使用する
- トランザクションの管理に WebLogic Server の EJB コンテナを使用する
- 1 つのトランザクション内に複数のデータベース更新を含む
- トランザクション中にデータベースおよび Java Messaging Service (JMS) など複数のリソースにアクセスする
- 複数のサーバで同一の接続プールを使用する

EJB アーキテクチャでは、データベース処理中の複数の EJB が単一のトランザクションの一部として呼び出されることがよくあります。XA を使用しない場合、すべてのトランザクション参加コンポーネントが同一のデータベース接続を使用しないと、このトランザクションは機能しません。WebLogic Server は、JTS ドライバおよび TxDataSource ([非 XA ドライバ用に 2 フェーズ コミットをエミュレート] を選択) を背後で使用して、JDBC 接続を EJB 間で明示的に受け渡さないでもこの処理が行われるようにします。XA を使用すると (XA 対応ドライバが必要)、2 フェーズ コミットによる分散トランザクションに対して WebLogic Server のトランザクション データ ソースを使用できるので、EJB では

トランザクションの各部分に異なったデータベース接続を使用できます。いずれの場合でも (XA の有無にかかわらず)、トランザクション データ ソースは使用してください。

詳細については、『WebLogic JDBC プログラマーズ ガイド』の「データ ソース」を参照してください。

注意： 同じ接続プールを指す 2 つのトランザクション データ ソースを作成することはできません。1 つのトランザクションで、同じ接続プールを指す 2 つの異なるトランザクション データ ソースを使用していると、2 度目の接続へのアクセス試行時に XA_PROTO エラーが発生します。

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、および WebLogic jDrivers for Microsoft SQL Server 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。

XA を使用する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。

XA を使用しない分散トランザクションをサポートするドライバ

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)。XA 非対応の JDBC ドライバは、1 つしか分散トランザクションに参加できません。8-37 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。

正しい接続数によるサーバのロックアップの回避

アプリケーションで、接続プールから接続を取得しようとしたが使用できる接続がない場合、使用できる接続がないという旨の例外が送出されます。接続プールでは、接続要求のキューは作成されません。このエラーを避けるには、接続プールのサイズを接続要求の最大負荷に対応できるサイズに拡大するようにしてください。

Administration Console で接続プールの最大接続数を設定するには、左ペインのナビゲーションツリーを展開して [サービス | JDBC | 接続プール] ノードを表示し、接続プールを選択します。次に右ペインで、[コンフィグレーション | 接続] タブを選択し、[最大容量] の値を指定します。

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。

- [ドライバクラス名] 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 (プロパティごとに行を分けて入力する)。

WebLogic 2 層 JDBC ドライバの詳細については、使用しているドライバに関する BEA のマニュアル、『WebLogic jDriver for Oracle のインストールと使い方』および『WebLogic jDriver for Microsoft SQL Server 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、『WebLogic JTA プログラマーズ ガイド』の「WebLogic Server でのサードパーティ JDBC XA ドライバの使い方」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 jDriver 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： 以下のコンフィグレーション例では、[パスワード] 属性を使用します。
 [パスワード] 属性の値は、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 ファイルに格納され、そのファイルにクリア テキストの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 8-8 WebLogic jDriver for Oracle : 接続プール コンフィグレーション

属性名	属性値
[一般] タブ	
[名前]	myConnectionPool
[URL]	jdbc:weblogic:oracle
[ドライバクラス名]	weblogic.jdbc.oci.Driver
[プロパティ]	user=scott server=localdb
[パスワード]	tiger (入力時には「*****」と表示され、以後は非表示。この値は [プロパティ] 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)
[接続] タブ	
[初期容量]	1
[最大容量]	5
[増加容量]	1
[縮小間隔]	15
[テスト] タブ	
[テストテーブル名]	dual
[対象] タブ	
[対象]	myserver

次の表では、WebLogic jDriver for Oracle または 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 8-9 データ ソースのコンフィグレーション

属性名	属性値
[コンフィグレーション] タブ	
[名前]	myDataSource
[JNDI 名]	myconnection
[プール名]	myConnectionPool
[Row Prefetch サイズ]	48
[ストリーム チャンク サイズ]	256
[対象] タブ	
[対象]	myserver

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 8-10 WebLogic jDriver for Microsoft SQL Server : 接続プール コンフィグレーション

属性名	属性値
[一般] タブ	
[名前]	myConnectionPool
[URL]	jdbc:weblogic:mssqlserver4
[ドライバクラス名]	weblogic.jdbc.mssqlserver4.Driver

表 8-10 WebLogic jDriver for Microsoft SQL Server : 接続プール コンフィグレーション

属性名	属性値
[プロパティ]	user=sa db=pubs server=myHost:1433 appname=MyApplication hostname=myhostName
[パスワード]	secret (入力時には「*****」と表示され、以後は非表示。この値は [プロパティ] 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)
[接続] タブ	
[初期容量]	1
[最大容量]	5
[増加容量]	1
[縮小間隔]	15
[テスト] タブ	
[テスト テーブル名]	member
[対象] タブ	
[対象]	myserver

次の表では、IBM Informix JDBC Driver 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 8-11 IBM Informix JDBC Driver : 接続プールのコンフィグレーション

属性名	属性値
[一般] タブ	

表 8-11 IBM Informix JDBC Driver : 接続プールのコンフィグレーション

属性名	属性値
[名前]	myConnectionPool
[URL]	jdbc:informix-sqli:ifxserver:1543
[ドライバクラス名]	com.informix.jdbc.IfxDriver
[プロパティ]	informixserver=ifxserver user=informix
[パスワード]	informix(入力時には「*****」と表示され、以後は非表示。この値は [プロパティ] 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)
[接続] タブ	
[初期容量]	3
[最大容量]	10
[増加容量]	1
[ログイン遅延時間]	1
[縮小間隔]	15
[対象] タブ	
[対象]	myserver

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。
- DBMS で XA を使用するために必要な追加の接続プール プロパティについては、8-36 ページの「追加の XA 接続プール プロパティ」を参照してください。

以下の表で、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を示します。

表 8-12 WebLogic jDriver for Oracle/XA : 接続プール コンフィグレーション

属性名	属性値
[一般] タブ	
[名前]	<code>fundsXferAppPool</code>
[URL]	(不要)
[ドライバクラス名]	<code>weblogic.jdbc.oci.xa.XADataSource</code>
[プロパティ]	<code>user=scott</code> <code>server=localdb</code>
[パスワード]	<code>tiger</code> (入力時には「*****」と表示され、以後は非表示。この値は [プロパティ] 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)
[接続] タブ	
[初期容量]	1
[最大容量]	5
[増加容量]	1

表 8-12 WebLogic jDriver for Oracle/XA : 接続プール コンフィグレーション

属性名	属性値
[縮小間隔]	15
[テスト] タブ	
[テスト テーブル名]	dual
[対象] タブ	
[対象]	myserver

以下の表は、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を示します。

表 8-13 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
[コンフィグレーション] タブ	
[名前]	fundsXferDataSource
[JNDI 名]	myapp.fundsXfer
[プール名]	fundsXferAppPool
[対象] タブ	
[対象]	myserver

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 8-14 Oracle Thin Driver : 接続プール コンフィグレーション

属性名	属性値
[一般] タブ	
[名前]	jtaXAPool
[URL]	jdbc:oracle:thin:@server:port:sid
[ドライバクラス名]	oracle.jdbc.xa.client.OracleXADataSource
[プロパティ]	user=scott
[パスワード]	tiger (入力時には「*****」と表示され、以後は非表示。この値は [プロパティ] 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)
[接続] タブ	
[初期容量]	4
[最大容量]	20
[増加容量]	2
[縮小間隔]	15
[テスト] タブ	
[テスト テーブル名]	dual
[対象] タブ	
[対象]	myserver

以下の表は、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を示します。

表 8-15 Oracle Thin Driver : 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
[コンフィグレーション] タブ	
[名前]	jtaXADS
[JNDI 名]	jtaXADS
[プール名]	jtaXAPool
[対象] タブ	
[対象]	myserver

以下の表は、Pointbase JDBC ドライバ を使用して、分散トランザクション用に JDBC 接続プールをコンフィグレーションする場合の例を示します。

注意： PointBase Server は WebLogic Server 配布キットに含まれている all-Java の DMBS 製品で、WebLogic Server の評価のみをサポートしています。カスタムの試用版アプリケーションまたは WebLogic Server に付属するパッケージ化されたサンプル アプリケーションの形式です。PointBase Server を評価以外の開発やプロダクション環境で使用する場合、エンドユーザは別個のライセンスを PointBase から直接入手する必要があります。

表 8-16 Pointbase : 接続プールのコンフィグレーション

属性名	属性値
[一般] タブ	
[名前]	demoXAPool
[URL]	jdbc:pointbase:server://localhost/demo

表 8-16 Pointbase : 接続プールのコンフィグレーション

属性名	属性値
[ドライバクラス名]	com.pointbase.xa.xaDataSource
[プロパティ]	user=public DatabaseName=jdbc:pointbase:server://localhost/demo
[パスワード]	public (入力時には「*****」と表示され、以後は非表示。この値は [プロパティ] 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)
[接続] タブ	
[初期容量]	2
[最大容量]	10
[増加容量]	2
[ローカル トランザクション のサポート]	true
[縮小間隔]	15
[テスト] タブ	
[テスト テーブル名]	users
[対象] タブ	
[対象]	myserver

Pointbase 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 8-17 Pointbase : 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
[コンフィグレーション] タブ	
[名前]	jtaXADS
[JNDI 名]	JTAXADS
[プール名]	demoXAPool
[対象] タブ	
[対象]	myserver

WebLogic jDriver for Oracle/XA のデータ ソース プロパティ

表 8-18 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 (S) か、または JDBC に対する WebLogic Server の拡張 (E) 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 8-18 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き 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き SQLException が送出されます。

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

「**」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 8-18 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 標準 / 拡張	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。	S	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。	S	Y	接続プール名
description	String	このデータ ソースの説明。	S	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。	S	Y	なし
password	String	データベースのパスワード。	S	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。	S	Y	なし
roleName**	String	初期 SQL ロール名。	S	Y	なし
serverName	String	データベース サーバ名。	S	Y*	なし
user	String	ユーザのアカウント名。	S	N*	なし
openString	String	Oracle の XA オープン文字列。	E	Y	なし

表 8-18 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 標準 / 拡張	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、 <code>xa_poolnamedate.trc</code> 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。	E	Y	false

表 8-19 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 8-19 `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 <code>xa_open</code> 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	

注意： 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。

追加の XA 接続プール プロパティ

分散トランザクションで接続プール内の接続を使用する場合、接続プールのプロパティを追加設定して、接続プールがトランザクションのコンテキストにおいて WebLogic Server 内で適切に接続を処理できるようにする必要があります。これらのプロパティは、JDBCConnectionPool タグ内のコンフィグレーション ファ

イル (config.xml) に設定します。デフォルトでは、すべての追加プロパティは `false` に設定されています。 `true` に設定すると、これらのプロパティが有効になります。

多くの場合、これらのプロパティの適切な値は自動的に設定されるので、ユーザが手動で設定する必要はありません。

KeepXAConnTillTxComplete

DBMS によっては、同じ物理データベース接続でトランザクションを開始および終了する必要があります。しかし、ある物理データベース接続で開始した WebLogic Server トランザクションが別のデータベース接続で終了する場合があります。トランザクションが完了するまで接続プールが物理接続を保持し、同じアプリケーション接続を提供するようにするには、

`KeepXAConnTillTxComplete="true"` に設定します。次に例を示します。

```
<JDBCConnectionPool KeepXAConnTillTxComplete="true"
DriverName="com.sybase.jdbc2.jdbc.SybXADataSource"
CapacityIncrement="5" InitialCapacity="10" MaxCapacity="25"
Name="demoXAPool" Password="{3DES}vIF8diu4H0QmdfOipd4dWA=="
Properties="User=dbuser;DatabaseName=dbname;ServerName=server_name_or_IP_address;PortNumber=serverPortNumber;NetworkProtocol=Tds;resourceManagerName=Lrm_name_in_xa_config;resourceManagerType=2" />
```

注意： このプロパティは、DB2 と Sybase との分散トランザクションをサポートするために必要です。

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

XA 非対応のドライバが分散トランザクションの他のリソースに関与できるように JDBC 接続プールをコンフィグレーションする場合は、[非 XA ドライバ用に 2 フェーズコミットをエミュレート] 属性 (JDBCTxDataSource MBean の `EnableTwoPhaseCommit`) を選択します。このパラメータは、XAResource インタフェースをサポートしているリソースでは無視されます。XA 非対応の接続プールは、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 [非 XA ドライバ用に 2 フェーズコミットをエミュレート] 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、WebLogic Server は設定を無視し、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 つを使用する場合は、Administration Console で [非 XA ドライバ用に 2 フェーズコミットをエミュレート] を選択します (`EnableTwoPhaseCommit = true`)。

[非 XA ドライバ用に 2 フェーズコミットをエミュレート] が選択されている (`EnableTwoPhaseCommit` が `true` に設定されている) 場合、XA 非対応の JDBC リソースは `XAResource.prepare()` メソッド呼び出し中に必ず `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ตอบสนอง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ています。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 [非 XA ドライバ用に 2 フェーズコミットをエミュレート] オプション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となります。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。

この XA 非対応の JDBC ドライバを「JTS ドライバ」と呼ぶこともあります。これは、WebLogic Server で XA をサポートするために WebLogic JTS ドライバを内部的に使用しているからです。WebLogic JTS ドライバの詳細については、『WebLogic JDBC プログラマーズ ガイド』の「WebLogic JTS ドライバの使い方」を参照してください。

グローバル トランザクションで XA 非対応のドライバを使用する場合の制限とリスク

WebLogic Server では、XA 非対応の JDBC リソースをグローバル トランザクションに参加させることができますが、こうしたリソースを使用するアプリケーションを設計する際に考慮すべき制限事項があります。XA 非対応のドライバは XA/2PC 規約に準拠しておらず、1 フェーズのコミットおよびロールバック操作しかサポートしていません。このため、WebLogic Server では JTS ドライバを使用して、トランザクションに参加するリソースをトランザクション マネージャで管理できるようにする必要があります。

ヒューリスティックな終了とデータの矛盾

XA 非対応のリソース用に [Emulate Two-Phase Commit] を選択した場合 (enableTwoPhaseCommit = true)、XA 非対応リソース用のトランザクションの準備フェーズは必ず問題なく完了します。したがって、XA 非対応リソースは、実際には 2 フェーズ コミット (2PC) プロトコルには参加しないため、障害が発生しても構いません。準備フェーズの後で XA 非対応リソースに障害が発生した場合は、XA トランザクションの参加者がトランザクションをコミットする間に XA 非対応リソースがトランザクションにロールバックされ、結果としてヒューリスティックな終了とデータの矛盾が発生します。

データの整合性が失われるリスクがあるため、[Emulate Two-Phase Commit] オプションはヒューリスティックな状態を許容できるアプリケーションでのみ使用してください。

保留中のトランザクションは回復不可

XA 非対応ドライバではローカル データベース トランザクションのみを操作するため、外部のトランザクション マネージャに関しては、データベース内のトランザクションが保留状態になることは想定されていません。XA 非対応リソース上で XAResource.recover() が呼び出されると、コミットまたはロールバックする必要のあるトランザクションがあったとしても、常に空の Xid (トランザクション ID) のセットが返されます。したがって、グローバル トランザクションで XA 非対応リソースを使用するアプリケーションでは、システム障害から回復したりデータの整合性を維持したりすることはできません。

マルチ サーバ コンフィグレーションで XA 非対応リソースを使用することによるパフォーマンスの低下

WebLogic Server は、XA 非対応リソースのグローバル トランザクションへの参加をサポートする上で、特定の JDBC に関連付けられたデータベース ローカル トランザクションに依存しています。このため、複数の WebLogic Server インスタンスのグローバル トランザクション コンテキストを持つアプリケーションが同じ JDBC データ ソースにアクセスした場合、JTS ドライバはトランザクション内で最初に確立された接続に JDBC 操作をルーティングします。たとえば、あるアプリケーションがサーバ上のトランザクションを開始して XA 非対応リソースにアクセスし、続いて RMI (remote method invocation) 呼び出しを別のサーバに対して実行して同じ基底の JDBC ドライバを使用するデータ ソースにアクセスした場合、JTS ドライバは別のサーバ上のトランザクションに関連付けられた接続を持つリソースを認識し、1 番目のサーバ上の実際の接続への RMI のリダイレクトを設定します。接続におけるすべての操作は、1 番目のサーバで確立された 1 つの接続を使用して実行されます。この動作により、リモート接続の設定と物理接続に対する RMI 呼び出しが原因でパフォーマンスが低下するおそれがあります。

XA 非対応リソースは 1 つのみ参加可能

WebLogic Server トランザクションマネージャに [Emulate Two-Phase Commit] を選択した XA 非対応リソースを登録すると、そのリソースは XAResource インタフェースを実装するクラスの名前で登録されます。[Emulate Two-Phase Commit] を選択したすべての XA 非対応リソースは XAResource インタフェース用の JTS ドライバを使用するため、グローバル トランザクションに参加するこれらの XA 非対応リソースはすべて同じ名前で登録されます。1 つのグローバル トランザクションで複数の XA 非対応リソースを使用すると、名前の衝突やヒューリスティック エラーが発生します。

XA 非対応接続プールと Tx データ ソースのコンフィグレーション例

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 8-20 WebLogic jDriver for Oracle : 接続プール コンフィグレーション

属性名	属性値
[一般] タブ	
[名前]	fundsxferAppPool
[URL]	jdbc:weblogic:oracle
[ドライバ クラス名]	weblogic.jdbc.oci.Driver
[プロパティ]	user=scott server=localdb
[パスワード]	tiger (入力時には「*****」と表示され、以後は非表示。この値は [プロパティ] 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)
[接続] タブ	
[初期容量]	0
[最大容量]	5
[増加容量]	1
[縮小間隔]	15
[テスト] タブ	
[テスト テーブル名]	dual
[対象] タブ	
[対象]	myserver

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 8-21 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
[コンフィグレーション] タブ	
[名前]	fundsXferDataSource
[JNDI 名]	myapp.fundsXfer
[プール名]	fundsXferAppPool
[非 XA ドライバ用に 2 フェーズ コミットをエミュ レート]	selected (EnableTwoPhaseCommit = true)
[対象] タブ	
[対象]	myserver

Prepared Statement キャッシュのパフォーマンスの向上

WebLogic Server 内に作成した各接続プールで Prepared Statement または XA Prepared Statement のキャッシュ サイズを設定することで、Prepared Statement のキャッシングが有効になります。Prepared Statement のキャッシングを有効にすると、アプリケーションおよび EJB で使用した Prepared Statement と Callable Statement が設定した数だけキャッシュされます。アプリケーションまたは EJB で、キャッシュに格納されている Prepared Statement または Callable Statement のいずれかが呼び出されると、WebLogic Server はキャッシュに格納されている Prepared Statement を再利用します。Statement を再利用すると、データベース内

の文を解析する必要がなくなることでデータベース マシンの CPU 使用率が低減され、現在の文に対するパフォーマンスが向上するとともに、CPU サイクルを他のタスク用に残しておけます。

Prepared Statement は、接続プールではなく接続ごとにキャッシュされます。たとえば、**Prepared Statement** のキャッシュ サイズを **10** に設定すると、**WebLogic Server** はアプリケーションまたは **EJB** がその特定の接続を使用して呼び出した **10** 個の **Prepared Statement** を格納します。

WebLogic Server 7.0 サービス パック 3 では、**Prepared Statement** のキャッシュ機能が変更され、**XA** (トランザクション対応) **JDBC** ドライバを使用する接続プールに対しては、**XA** 非対応の **JDBC** ドライバの代わりにデータベース接続が作成されるようになりました。**JDBC** 接続プール内でデータベース接続の作成に使用する **JDBC** ドライバのタイプに応じて、適切なキャッシュ サイズ属性を設定する必要があります。**XA** 非対応 **JDBC** ドライバを使用する接続プールの場合には **PreparedStatementCacheSize** を、**XA** 対応 **JDBC** ドライバを使用する接続プールの場合には **XAPreparedStatementCacheSize** を設定します。詳細については、次の節「**XA** 非対応の **Prepared Statement** キャッシュ」および 8-44 ページの「**XA** 対応の **Prepared Statement** キャッシュ」を参照してください。

XA 非対応の Prepared Statement キャッシュ

XA 非対応の **Prepared Statement** キャッシュの場合、接続プール内の接続ごとのキャッシュにどの文を格納するかが、固定アルゴリズムに基づいて判別されます。接続で使用された **Prepared Statement** および **Callable Statement** は、**Prepared Statement** キャッシュが一杯になるまでキャッシュされます。キャッシュが一杯になると、それ以降に使用された文はキャッシュされません。

注意： **Statement** キャッシュは、**JMX API** を使用してクリアできます。詳細については、**WebLogic** クラスの **Javadoc** の **clearStatementCache()** メソッドを参照してください。

この **Statement** キャッシュは、**XA** 非対応 **JDBC** ドライバを使用してデータベース接続を作成する接続プールにのみ使用します。このキャッシュ設定は、データベース接続に **XA** 対応 **JDBC** ドライバを使用する接続プールでは無視されます。

XA 非対応 **Prepared Statement** のキャッシュ サイズのデフォルト値は、**5** です。以下の方法で、接続プールの **Prepared Statement** キャッシュのサイズを設定できます。

- Administration Console を使用して、接続プールの Prepared Statement キャッシュ サイズ属性を設定する。Administration Console オンライン ヘルプの「JDBC 接続プールの作成とコンフィグレーション」を参照してください。
- WebLogic 管理 API を使用して PreparedStatementCacheSize 属性を設定する。WebLogic クラスの Javadoc の getPreparedStatementCacheSize() メソッドおよび setPreparedStatementCacheSize(int cacheSize) メソッドを参照してください。
- コンフィグレーション ファイルで属性を直接設定する (Weblogic Server の実行中のみ)。

コンフィグレーション ファイルを使用して接続プール用に XA 非対応の Prepared Statement のキャッシュ サイズを設定するには、サーバを起動する前にエディタで config.xml ファイルを開き、JDBCConnectionPool タグの PreparedStatementCacheSize 属性のエントリを追加します。次に例を示します。

```
<JDBCConnectionPool CapacityIncrement="5"
  DriverName="com.pointbase.jdbc.jdbcUniversalDriver"
  InitialCapacity="5" MaxCapacity="20" Name="demoPool"
  Password="{3DES}ANfMduXgaaGMeS8+CR1xoA=="
  PreparedStatementCacheSize="20" Properties="user=examples"
  RefreshMinutes="0" ShrinkPeriodMinutes="15"
  ShrinkingEnabled="true" Targets="examplesServer"
  TestConnectionsOnRelease="false"
  TestConnectionsOnReserve="false"
  URL="jdbc:pointbase:server://localhost/demo"/>
```

XA 対応の Prepared Statement キャッシュ

XA 対応の Prepared Statement キャッシュの場合、最長時間未使用 (LRU : Least Recently Used) アルゴリズムに基づいて、接続プール内の接続ごとのキャッシュにどの文を格納するかを判別します。接続で使用された Prepared Statement および Callable Statement は、Statement キャッシュが一杯になるまでキャッシュされます。アプリケーションが Connection.prepareStatement() を呼び出すと、Prepared Statement キャッシュにその文が格納されているかどうかをチェックされます。格納されている場合は、キャッシュされている文が返されます (まだ使用されていない場合)。文がキャッシュに格納されておらず、キャッシュが一杯 (キャッシュ内の文の数が XAPreparedStatementCacheSize に等しい状態) になっている場合は、キャッシュ内で最長時間未使用になっている文が識別され、この文が新しい文によって置換されます。

注意： Statement キャッシュは、JMX API を使用してクリアできます。詳細については、WebLogic クラスの Javadoc の `clearStatementCache()` メソッドを参照してください。

XA 対応の Statement キャッシュは、XA 対応 JDBC ドライバを使用してデータベース接続を作成する接続プールにのみ使用します。このキャッシュ設定は、データベース接続に XA 非対応 JDBC ドライバを使用する接続プールでは無視されます。

XA 対応 Prepared Statement のキャッシュ サイズのデフォルト値は、5 です。以下の方法で、接続プールの XA 対応 Prepared Statement キャッシュのサイズを設定できます。

- Administration Console を使用して、接続プールの XA Prepared Statement キャッシュ サイズ属性を設定する。Administration Console オンラインヘルプの「JDBC 接続プールの作成とコンフィグレーション」を参照してください。
- WebLogic 管理 API を使用して `XAPreparedStatementCacheSize` 属性を設定する。WebLogic クラスの Javadoc の `getXAPreparedStatementCacheSize()` メソッドおよび `setXAPreparedStatementCacheSize(int cacheSize)` メソッドを参照してください。
- コンフィグレーション ファイルで属性を直接設定する (Weblogic Server の実行中のみ)。

コンフィグレーション ファイルを使用して接続プール用に XA 対応 Prepared Statement のキャッシュ サイズを設定するには、サーバを起動する前にエディタで `config.xml` ファイルを開き、JDBCConnectionPool タグの `PreparedStatementCacheSize` 属性のエントリを追加します。次に例を示します。

```
<JDBCConnectionPool CapacityIncrement="5"
  DriverName="com.pointbase.xa.xaDataSource"
  InitialCapacity="5" MaxCapacity="20" Name="demoXAPool"
  Password="{3DES}ANfMduXgaaGMeS8+CR1xoA=="
  XAPreparedStatementCacheSize="20"
  Properties="user=examples;
  DatabaseName=jdbc:pointbase:server://localhost/demo"
  RefreshMinutes="0" ShrinkPeriodMinutes="15"
  ShrinkingEnabled="true" Targets="examplesServer"
  TestConnectionsOnRelease="false"
  TestConnectionsOnReserve="false"
  URL="jdbc:pointbase:server://localhost/demo"/>
```

Prepared Statement キャッシュに関する制限

Prepared Statement キャッシュを使用することでパフォーマンスを向上させることができますが、制限があることも考慮に入れる必要があります。Prepared Statement のキャッシュを使用する場合は、以下の制限に留意してください。これらの制限は、XA 対応および XA 非対応の両方の Prepared Statement キャッシュに適用されます。

ここに示す以外にも、Prepared Statement に関連した問題が存在する可能性があります。Prepared Statement に関連するシステムでエラーが見つかった場合は、Prepared Statement キャッシュのサイズを 0 に設定して Prepared Statement のキャッシングをオフにし、問題が Prepared Statement のキャッシングに起因するものかどうかをテストする必要があります。

データベース変更後の Prepared Statement 呼び出しはエラーの原因となる可能性がある

キャッシュ内に格納された Prepared Statement は、Prepared Statement のキャッシュ時に特定のデータベース オブジェクトを参照します。キャッシュに格納された Prepared Statement で参照されるデータベース オブジェクトに対して、何らかの DLL (データ定義言語) 処理を実行すると、それらの文は次に実行したときに失敗します。たとえば、`select * from emp` などの文をキャッシュしてから、`emp` テーブルを削除して再作成した場合、その文が準備されたときに存在した `emp` テーブルはなくなるため、キャッシュされた文を次に実行すると、その文は失敗します。

同様に、Prepared Statement は、キャッシュされた時点でデータベース内のテーブルの各カラムのデータ型にバインドされます。テーブルのカラムを追加、削除、または再配列すると、キャッシュに格納されている Prepared Statement は、再び実行されたときに失敗するおそれがあります。

Prepared Statement における setNull の使用

データベースへの接続に WebLogic jDriver for Oracle を使用する場合、`setNull` バインド変数を使用する Prepared Statement をキャッシュするのであれば、変数を適切なデータ型に設定する必要があります。次の例で示すような汎用のデータ型を使用すると、`null` 以外の値で実行したときにその文は失敗します。

```
java.sql.Types.Long sal=null
.
.
.
if (sal == null)
    setNull(2,int)// これは不正
    else
    setLong(2,sal)
```

代わりに次を使用します。

```
if (sal == null)
    setNull(2,long)// これは適正
    else
    setLong(2,sal)
```

この問題は、WebLogic jDriver for Oracle を使用する場合には常に発生します。他の JDBC ドライバでも生じる可能性があります。

キャッシュ内の Prepared Statement はデータベース カーソルを予約可能

WebLogic Server でキャッシュされた Prepared Statement は、データベースのカーソルをオープンすることができます。キャッシュされた文の数が多すぎると、接続のためのオープン カーソル数の上限を超えてしまうことがあります。接続のためのオープン カーソル数の上限を超えないようにするには、データベース管理システムの上限を変更するか、または接続プールの Prepared Statement キャッシュのサイズを低減します。

Prepared Statement キャッシュの適切なサイズの決定

Prepared Statement キャッシュの適切なサイズ設定を決定するには、開発環境におけるサーバ負荷をエミュレートし、次に Oracle の statspack スクリプトを実行します。スクリプトの出力で、1 秒当たりの解析数を確認します。Prepared Statement キャッシュのサイズを増やすにつれ、1 秒当たりの解析数は少なくなります。1 秒当たりの解析数の減少が止まるまで、Prepared Statement キャッシュのサイズを徐々に増やしていきます。

注意: プロダクション環境で Prepared Statement キャッシュを使用すると決定する前に、その制限について考慮してください。詳細については、8-46 ページの「Prepared Statement キャッシュに関する制限」を参照してください。

起動クラスを使用した XA 非対応の Prepared Statement キャッシュのロード

XA 非対応の Prepared Statement キャッシュを有効に活用し、最大限のパフォーマンスを得るために、Prepared Statement キャッシュに格納する Prepared Statement のそれぞれを呼び出す起動クラスを作成できます。WebLogic Server では、Prepared Statement を使用順にキャッシュし、Prepared Statement キャッシュサイズの上限に到達すると、キャッシングを停止します。キャッシュする Prepared Statement を呼び出すための起動クラスを作成すると、数回しか呼び出されない文ではなく、アプリケーションで再利用される文でキャッシュを埋めることができます。したがって、キャッシュする文の数は最小限にとどめながら、パフォーマンスを最大限向上させることができます。8-46 ページの「Prepared Statement キャッシュに関する制限」で説明したような、問題になる可能性のある Prepared Statement のキャッシングを回避することもできます。

起動クラスが失敗しても、WebLogic Server は後で使用するために、文をロードし、キャッシングします。

各接続には、独自の Prepared Statement キャッシュが割り当てられます。起動クラスを使用して Prepared Statement をキャッシュする場合、接続プールから各接続を取得し、キャッシュする各 Prepared Statement を呼び出すように起動クラスを作成します。

接続要求の増加に応じて接続プールのサイズが拡大するよう設定した場合、新しい接続は使用された Prepared Statement をキャッシュします。起動クラスは、新しい接続用の Prepared Statement をロードできません。接続プールのサイズを縮小できるよう設定した場合、接続プールは縮小間隔の経過後に接続が利用可能であれば接続を閉じます。どの接続を最初に閉じるかを指定する方法はありません。このため、ロードした Prepared Statement キャッシュを持つ接続がそうでない接続の前に閉じる場合があります。

また、サーバ起動時には、起動クラスの実行前に **EJB** がデプロイされます。

CMP エンティティ **Bean** 内の **Prepared Statement** およびデプロイ中に **EJB** が使用する **Prepared Statement** は、起動クラス内の **Prepared Statement** よりも前にキャッシュ内に格納されます。この問題を回避するため、すべての **EJB** およびアプリケーションのデプロイ後に **Prepared Statement** キャッシュをクリアし、その後キャッシュプライミングコードを実行できます。

`weblogic.jdbc.extensions` パッケージ内の `clearStatementCache` メソッドについては、**Javadoc** を参照してください。

XA 対応 **Prepared Statement** キャッシュでは、キャッシュ内の文の置換に最長時間未使用アルゴリズムを使用するため、起動クラスでキャッシュをあらかじめロードしても意味がありません。

9 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。

- 9-1 ページの「JMS と WebLogic Server」
- 9-2 ページの「JMS のコンフィグレーション」
- 9-20 ページの「JMS のモニタ」
- 9-23 ページの「JMS のチューニング」
- 9-49 ページの「分散送り先のコンフィグレーション」
- 9-65 ページの「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 9-1 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 サーバを作成し、JMS サーバのデプロイ先となる WebLogic Server インスタンスまたは移行できる対象に割り当てます。
- JMS サーバ、接続ファクトリ、送り先 (物理的なキューとトピック)、分散送り先 (クラスタ内の物理的なキューおよびトピック メンバーの集合)、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、ページングストア、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、フロー制御、分散送り先へのロード バランシングなど、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。Examples サーバには、サンプル コンフィグレーションとして `examplesJMS` が用意されています。Examples サーバの起動については、『インストール ガイド』の「サンプルサーバ、Pet Store サーバ、および Workshop サンプルサーバの起動」を参照してください。また、9-4 ページの「WebLogic Server の起動と JMS のコンフィグレーション」では、基本的な JMS 実装を手動でコンフィグレーションする手順が説明されています。

『WebLogic JMS プログラマーズ ガイド』の「WebLogic JMS アプリケーションの移植」で説明されているように、WebLogic JMS アプリケーションを WebLogic Server の以前のリリースから移植する場合、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 属性をコンフィグレーションするには、以降の節で説明されている手順に従います。また JMS オブジェクトを作成およびコンフィグレーションするには、Administration Console オンラインヘルプの手順に従います。

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』の「WebLogic JMS アプリケーションの開発」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 コンフィグレーションにおける命名規則

ドメイン内の各サーバには、ドメイン内のすべてのコンフィグレーション オブジェクトの間でユニークな名前を付ける必要があります。ドメイン内では、サーバ、マシン、クラスタ、仮想ホストなど、すべてのリソースタイプにユニークな名前を付けます。また、これらをドメインと同じ名前にはできません。この命名規則は、コンフィグレーション可能なすべての JMS オブジェクト (JMS サーバ、ストア、テンプレート、接続ファクトリ、セッションプール、および接続コンシューマ) にも適用されます。

ただし、以下に示すように、ドメイン内にある別の JMS サーバの JMS キューおよび JMS トピックの送り先は例外です。

- キューの送り先には別の JMS サーバ上の他のキューと同じ名前を、トピックの送り先には別の JMS サーバ上の他のトピックと同じ名前を使用できる。
- キューの送り先にトピックの送り先と同じ名前は使用できず、キューおよびトピックに他のコンフィグレーション可能なオブジェクトと同じ名前を使用することもできない。

WebLogic Server の起動と JMS のコンフィグレーション

以下の節では、WebLogic Server および Administration Console の起動方法を概説し、基本的な JMS 実装をコンフィグレーションする手順について説明します。

デフォルトの WebLogic Server の起動

WebLogic Server のデフォルトのロールは管理サーバです。ドメインが 1 つの WebLogic Server のみで構成されている場合は、そのサーバが管理サーバになります。ドメインが複数の WebLogic Server で構成されている場合は、まず管理サーバを起動してから、管理対象サーバを起動します。

管理サーバの起動の詳細については、2-13 ページの「管理サーバの起動」を参照してください。

Administration Console の起動

Administration Console は、WebLogic Server に対する Web ベースの管理者フロントエンド (管理者クライアントインタフェース) です。サーバの Administration Console にアクセスする前に、サーバを起動する必要があります。

Administration Console を使用した WebLogic Server のコンフィグレーションの詳細については、1-22 ページの「Administration Console の起動と使い方」を参照してください。

基本的な JMS 実装のコンフィグレーション

この節では、Administration Console を使用して基本的な JMS 実装をコンフィグレーションする方法について説明します。

1. 左ペインの [サービス] ノードの下の [JMS] ノードをクリックしてリストを展開します。
2. ディスクベースのファイルに永続メッセージを保存するには、JMS ファイルストアを作成する必要があります。JMS ページング機能を使用する場合は、空きメモリがなくなったときにメッセージ本文を一時的にディスクへスワップするための付加的な「ページング」JMS ストアもコンフィグレーションする必要があります。
 - a. JMS ファイルストアが置かれるファイル システム上のディレクトリを作成します。
 - b. 左ペインの [ストア] ノードをクリックしてから、右ペインの [新しい JMS File Store のコンフィグレーション] リンクをクリックします。
 - c. [一般] タブで、ストアに名前を付け、ディレクトリを指定し、必要に応じて同期書き込みポリシーを選択して、ファイルストアがディスクにデータを書き込む方式を決定します。その後、[作成] をクリックします。
 - d. 同じ手順を繰り返して、ページングストアを作成します。

ストアのコンフィグレーションの詳細については、9-15 ページの「ストアのコンフィグレーション」を参照してください。

3. JDBC でアクセス可能なデータベースに永続メッセージを保存するには、JMS JDBC ストアを作成する必要があります (JDBC 接続プールを作成する必要がある場合は、まず手順 A から D を完了させます。それ以外の場合は、手順 E までスキップできます)。JMS ページング機能を使用する場合は、空きメモリがなくなったときにメッセージ本文を一時的にディスクへスワップするための付加的な「ページング」JMS ファイルストアもコンフィグレーションする必要があります。
 - a. 左ペインの [JDBC] ノードをクリックして展開します。
 - b. 左ペインの [接続プール] ノードをクリックしてから、右ペインの [新しい JDBC Connection Pool のコンフィグレーション] リンクをクリックします。

- c. [**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します。完了したら、各タブの [**適用**] をクリックします。
- d. [**対象**] タブをクリックし、接続プールを **WebLogic Server** インスタンスにデプロイする場合は [**サーバ**] タブを、サーバクラスタにデプロイする場合は [**クラスタ**] タブを選択します。[**選択可**] リストから対象を [**選択済み**] リストに移動させることによって選択し、[**適用**] をクリックします。
- e. [**JMS | ストア**] ノードに戻り、右ペインの [**新しい JMSJDBCStore のコンフィグレーション**] リンクをクリックします。
- f. **JDBC** ストアに名前を付け、接続プールとプレフィックス名を選択します。その後、[**作成**] をクリックします。

JDBC 接続プールのコンフィグレーションの詳細については、8-10 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」を参照してください。

4. 必要に応じて、同じような属性設定で複数の送り先を定義するための **JMS** テンプレートを作成します。**JMS** テンプレートは、一時キューの作成にも必要です。
 - a. 左ペインの [**テンプレート**] ノードをクリックしてから、右ペインの [**新しい JMS Template のコンフィグレーション**] リンクをクリックします。
 - b. [**一般**] タブでテンプレートに名前を付け、[**作成**] をクリックします。
 - c. 必要に応じて、[**しきい値と割当**]、[**オーバーライド**]、および [**再配信**] タブに値を入力します。完了したら、各タブの [**適用**] をクリックします。

JMS テンプレートのコンフィグレーションの詳細については、9-13 ページの「**JMS** テンプレートのコンフィグレーション」を参照してください。

5. **JMS** サーバ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. 左ペインの [**サーバ**] ノードをクリックしてから、右ペインの [**新しい JMSServer のコンフィグレーション**] リンクをクリックします。
 - b. [**一般**] タブでサーバに名前を付け、[**ストア**]、[**ページングストア**]、および [**テンプレート**] のうち、作成したものを選択します。その後、[**作成**] をクリックします。

- c. 必要に応じて、[しきい値と割当] タブに値を入力します。完了したら、[適用] をクリックします。
- d. [対象] タブをクリックし、JMS サーバを WebLogic Server インスタンスにデプロイする場合は [サーバ] タブを、移行可能対象サーバにデプロイする場合は [移行できる対象] タブを選択します。[選択可] リストから対象を [選択済み] リストに移動させることによって選択し、[適用] をクリックします。

JMS サーバのコンフィグレーションの詳細については、9-8 ページの「JMS サーバのコンフィグレーション」を参照してください。

6. JMS 送り先を作成します。JMS 送り先は、キュー (ポイント ツー ポイント) またはトピック (Pub/Sub) です。
 - a. 左ペインの [サーバ] ノードの下で、新しい JMS サーバインスタンスをクリックしてリストを展開し、[送り先] ノードをクリックします。
 - b. 右ペインで [新しい JMSQueue のコンフィグレーション] と [新しい JMSTopic のコンフィグレーション] のいずれかのリンクをクリックします。
 - c. [一般] タブで、送り先に名前と JNDI 名を付けます。必要に応じて他の属性を入力し、[作成] をクリックします。
 - d. 必要に応じて、[しきい値と割当]、[オーバーライド]、および [マルチキャスト] タブに値を入力します。完了したら、各タブの [適用] をクリックします。

送り先のコンフィグレーションの詳細については、9-12 ページの「送り先のコンフィグレーション」を参照してください。

7. 接続ファクトリを作成して JMS クライアントを有効化し、JMS 接続を作成するには、次の手順に従います。
 - a. 左ペインの [接続ファクトリ] ノードをクリックして展開し、右ペインの [新しい JMS Connection Factory のコンフィグレーション] リンクをクリックします。
 - b. [一般] タブで、接続ファクトリに名前と JNDI 名を付けます。必要に応じて他の属性を入力し、[作成] をクリックします。
 - c. 必要に応じて、[トランザクション] および [フロー制御] タブに値を入力します。完了したら、各タブの [適用] をクリックします。

- d. [対象] タブをクリックし、接続ファクトリを **WebLogic Server** インスタンスにデプロイする場合は [サーバ] タブを、サーバ クラスタにデプロイする場合は [クラスタ] タブを選択します。[選択可] リストから対象を [選択済み] リストに移動させることによって選択し、[適用] をクリックします。

接続ファクトリのコンフィグレーションの詳細については、9-10 ページの「接続ファクトリのコンフィグレーション」を参照してください。

8. 必要に応じて、[送り先キー] ノードを使用して特定の送り先のソート順を定義します。詳細については、9-14 ページの「送り先キーのコンフィグレーション」を参照してください。
9. 必要に応じて、[分散送り先] ノードを使用し、物理的な送り先をサーバ クラスタ内に設定された論理的な分散送り先の一部にします。詳細については、9-49 ページの「分散送り先のコンフィグレーション」を参照してください。
10. 必要に応じて、アプリケーションでメッセージを並行処理するための **JMS** セッションプール、およびサーバセッションを取得してメッセージを処理する接続コンシューマ (キューまたはトピック) を作成します。詳細については、9-19 ページの「セッションプールのコンフィグレーション」および 9-20 ページの「接続コンシューマのコンフィグレーション」を参照してください。

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 [サーバ] ノードを使用して、以下を定義します。

- 以下の一般的な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 ストアを推奨)。**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。

注意：JMS サーバの名前は、ドメイン内でユニークでなければなりません。詳細については、9-3 ページの「JMS コンフィグレーションにおける命名規則」を参照してください。

- 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）、およびバイト ページングまたはメッセージ ページングを有効にするかどうか。
- JMS サーバのデプロイ先となる WebLogic Server インスタンスまたは移行できる対象を定義する。
 - サーバ – 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、単一の WebLogic Server インスタンスまたは移行できる対象（次項を参照）にデプロイされます。一方、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。

- 移行できる対象 – 移行できる対象は、JMS など「かならず 1 回」のサービスのホストとなり得る、クラスタ内の複数の WebLogic Server インスタンスを定義します。移行できる対象のサーバが起動すると、JMS サーバもクラスタ内のユーザ指定のサーバ上で起動します。ただし、JMS サーバとその送り先はすべて、WebLogic Server の障害発生に応答して、またはスケジューリングされた移行やシステムの保守のため、クラスタ内の別のサーバに移行することができます。JMS の移行できる対象のコンフィグレーションの詳細については、『WebLogic JMS プログラマーズ ガイド』の「WebLogic JMS の管理」を参照してください。

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。WebLogic JMS には、あらかじめコンフィグレーションされたデフォルトの接続ファクトリ

`weblogic.jms.ConnectionFactory` が用意されています。この接続ファクトリは、サーバごとに記述できます。詳細については、Administration Console オンラインヘルプの「[サーバ]-->[サービス]-->[JMS]」を参照してください。もしくは、1つまたは複数の接続ファクトリを定義およびコンフィグレーションし、あらかじめ定義された属性を使用して、よりアプリケーションに適合する接続を作成できます。ただし、各接続ファクトリにユニークな名前が付けられている場合にかぎり、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。

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するには、各サーバインスタンスに対してデフォルトの接続ファクトリを有効にするか、1つまたは複数の接続ファクトリをコンフィグレーションしてクラスタ内の1つまたは複数のサーバインスタンスに割り当てます。これにより、各接続ファクトリを複数の WebLogic Server にデプロイすることが可能となります。JMS のクラスタ化のコンフィグレーションの詳細については、『WebLogic JMS プログラマーズガイド』の「WebLogic JMS の管理」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。

- 以下の一般的な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のクライアント識別子 (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』の「WebLogic JMS アプリケーションの開発」を参照してください。
 - 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、および配信モード)。

- 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ーラン 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。
- 分散送り先の場合は、接続ファクトリで作成された匿名でないプロデューサが、呼び出しごとにロード バランシングの対象となるかどうか。
- 分散送り先の場合、コンシューマまたはプロデューサのロード バランシングを分散送り先で行うときにサーバ アフィニティを使用するかどうか。

注意： JMS 接続ファクトリの名前は、ドメイン内でユニークでなければなりません。詳細については、9-3 ページの「JMS コンフィグレーションにおける命名規則」を参照してください。

- 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、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、およびサーバサイド トランザクションが有効かどうか。
- フロー制御属性 (メッセージプロデューサによってメッセージフローを調整できる)。具体的には、プロデューサはフローを最小値から最大値までの範囲内に制限する属性を受信します。プロデューサは、条件が悪くなると最小値側に移動し、改善されると最大値側に移動します。
- WebLogic Server インスタンスまたはサーバクラスタの対象を定義する。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。
 - サーバ — 接続ファクトリのデプロイ先となる単一の WebLogic Server インスタンスを対象として定義します。
 - クラスタ — 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サポートするための、接続ファクトリのデプロイ先となるクラスタを対象として定義します。

デフォルトの接続ファクトリ `weblogic.jms.ConnectionFactory` では、すべてのコンフィグレーション属性がデフォルト値に設定されています。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 **WebLogic Server** を限定できない。特定の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、適切なサーバ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、**JMS** サーバのキュー (ポイント ツー ポイント) かトピック (Pub/Sub) が識別されます。**JMS** サーバを定義したら、各 **JMS** サーバに 1 つのまたは複数の送り先をコンフィグレーションします。

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 (9-13 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)。

注意: また、クラスタ内の単一の分散送り先のメンバーとして、複数の物理的送り先をコンフィグレーションすることもできます。したがって、クラスタ内で 1 つのインスタンスがダウンした場合に、同じ送り先の他のインスタンスで **JMS** プロデューサおよびコンシューマにサービスを提供できます。詳細については、9-49 ページの「分散送り先のコンフィグレーション」を参照してください。

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。

- 以下の一般的なコンフィグレーション属性。
 - 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

注意： JMS 送り先の名前は、ドメイン内でユニークでなければなりません。詳細については、9-3 ページの「JMS コンフィグレーションにおける命名規則」を参照してください。

- 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)、およびバイト 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 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ない (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可能。

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る。
- 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る。
- [名前] 属性は、送り先によって継承されない。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。

注意：JMS テンプレートの名前は、ドメイン内でユニークでなければなりません。詳細については、9-3 ページの「JMS コンフィグレーションにおける命名規則」を参照してください。

- [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ない。
- [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ない。

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 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。

- **JMS 永続ストレージ** の名前
- **JMS ファイル ストア** の場合 — メッセージが保存される場所へのパスを指定します。
- **JMS 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を指定します。

注意： **JMS** ストアの名前は、ドメイン内でユニークでなければなりません。詳細については、9-3 ページの「**JMS コンフィグレーションにおける命名規則**」を参照してください。

警告： トランザクション (**XA**) 接続プールを、**JDBC データベース ストア** で使用するようにコンフィグレーションすることはできません。詳細については、9-17 ページの「**JMS JDBC トランザクション**」を参照してください。

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ください。その後、サーバを再起動してください。ヒープ

サイズの設定の詳細については、『BEA WebLogic Server パフォーマンス チューニング ガイド』の「WebLogic Server アプリケーションのチューニング」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 JDBC ストア について

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 JMS では、以下のデータベースに対応する一部のドライバが検出されます。

- Pointbase
- Microsoft SQL (MSSQL) Server
- Oracle
- Sybase
- Cloudscape
- Informix
- IBM DB2
- Times Ten

weblogic.jar ファイル内の weblogic/jms/ddl ディレクトリには、これらのデータベースに対応する JMS DDL ファイルが含まれています。JMS DDL ファイルは実際にはテキスト ファイルであり、JMS データベース テーブルを作成する SQL コマンドを含んでいます。異なるデータベースを使用するには、これらの .ddl ファイルのいずれかをコピーして編集するだけです。

注意： WebLogic Server 配布キットに用意されている JMS サンプルは、PointBase 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、PointBase 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。

既存の JMS JDBC ストアが破損している場合、`utils.Schema` ユーティリティを使用して再生成できます。詳細については、『WebLogic JMS プログラマーズ ガイド』の「JDBC データベース ユーティリティ」を参照してください。

JMS JDBC トランザクション

JMS JDBC ストアで使用するように、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。JMS では非 XAResource ドライバを使用する JDBC 接続プールを使用しなければなりません (XA ドライバまたは JTS ドライバは使用できません)。JMS は JDBC ドライバの上で XA をサポートします。

これは、WebLogic JMS には独自のリソース マネージャがあるためです。つまり、JMS 自身が XAResource を実装し、(メッセージがデータベースに格納されている場合でも) データベースに依存しないでトランザクションを処理します。従って、JMS とデータベースを使用する場合は (そのデータベースが JMS メッセージが格納されているデータベースと同じでも)、常に 2 フェーズコミット トランザクションになります。WebLogic JMS でのトランザクションの使い方については、『WebLogic JMS プログラマーズ ガイド』の「WebLogic JMS によるトランザクションの使い方」を参照してください。

パフォーマンスの点から見ると、データベース操作に使用される JDBC 接続プールが JMS キューと同じ WebLogic Server 上にある場合、パフォーマンスが向上する可能性があります。トランザクションは依然として 2 フェーズですが、より小さいネットワーク オーバーヘッドで処理されるためです。また、JMS JDBC ストアではなく JMS ファイル ストアを使用することによっても、パフォーマンスが向上する場合があります。

JMS JDBC セキュリティ

必要に応じて、JDBC 接続プールのリソースを制限することもできます。以前のバージョンの WebLogic Server では、ACL を使って WebLogic リソースを保護していました。WebLogic Server バージョン 7.0 では、WebLogic リソースへの「アクセス権は誰が持つか」という問いに、セキュリティ ポリシーが答えます。セキュリティ ポリシーは、WebLogic リソースとユーザ、グループ、またはロールの間の関連付けを定義するときに作成します。WebLogic リソースは、セキュリティ ポリシーが割り当てられるまでは保護されません。すべての WebLogic

Server リソースに対してセキュリティを設定する方法に関する詳細については、Administration Console オンライン ヘルプの「WebLogic リソースの保護の設定」を参照してください。

JMS ストア テーブルのプレフィックス

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、次のようにプロダクション 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。

JMS JDBC ストア向けの JDBC 接続プールの推奨設定

WebLogic Server の堅牢な JDBC 接続プールは、データベースに障害が発生した場合、復旧後に自動的に再接続します。このため、WebLogic Server を再起動する必要はありません。この機能を活用し、JMS JDBC ストアをより堅牢にするには、JMS JDBC ストアに関連付けられている JDBC 接続プールの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 サーバを定義したら、必要に応じて各 JMS サーバに 1 つの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します。

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。セッションプールの名前は、ドメイン内でユニークでなければなりません。詳細については、9-3 ページの「JMS コンフィグレーションにおける命名規則」を参照してください。
- 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、サーバセッションを取り出し、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したら、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します。

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。接続コンシューマの名前は、ドメイン内でユニークでなければなりません。詳細については、9-3 ページの「JMS コンフィグレーションにおける命名規則」を参照してください。
- 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。セレクタの定義については、『WebLogic JMS プログラマーズ ガイド』の「WebLogic JMS アプリケーションの開発」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手順と、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

アクティブな JMS サーバ、送り先、およびセッション プールの実行時情報を表示するには、次の手順に従います。

1. Administration Console を起動します。
2. 左ペインの [サービス] の下にある [JMS] ノードをクリックし、JMS サーバ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. 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、以下のテキスト リンクを表示します。
 - [すべてのアクティブ JMS サーバのモニタ]
 - [すべてのアクティブな JMS の送り先のモニタ]
 - [すべての JMS Session Pool Runtimes のモニタ]
6. 適切なテキスト リンクをクリックして、JMS オブジェクトのモニタ データを表示します。

注意： 分散送り先をモニタする場合、そのトピックまたはキュー メンバーに対するプロキシ トピック メンバーまたはシステム サブスクリプションが表示されることがあります。詳細については、9-22 ページの「分散送り先のシステム サブスクリプションとプロキシ トピック メンバーのモニタ」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. 9-21 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。[恒久サブスクライバ] 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 [恒久サブスクライバ] のアイコン (または実際の数) をクリックします。

分散送り先のシステム サブスクリプションとプロキシトピックメンバーのモニタ

WebLogic Server 7.0 の特定の JMS コンフィグレーションでは、分散送り先が、トピック メンバーまたはキュー メンバー間に、「プロキシトピック メンバー」と「システム サブスクリプション」を自動的に作成することがあります。その場合、分散送り先のメンバーをモニタすると、システム サブスクリプションとプロキシトピック メンバーが MBean の統計に現れ、Administration Console に表示されます。また、分散送り先メンバーの恒久サブスクリプション名とコンシューマ数にも現れます。

システム サブスクリプションとプロキシトピック メンバーの動作について以下に示します。

- 分散トピック プロキシメンバー — WebLogic Server インスタンスで JMS 接続ファクトリがコンフィグレーションされているが、リモートの分散トピックに対応するローカルの分散トピック メンバーをホストするようにコンフィグレーションされていない場合、その WebLogic Server インスタンスは、リモートの分散トピックに対応するローカルのプロキシトピック メンバーを自動的に作成してホストすることがあります。これは、その分散トピックに対する最初の非恒久サブスクリプションが、サーバの接続ファクト

リで作成されるときに発生します。動的に作成されるプロキシトピック メンバーは、動的に作成される JMS サーバ内に置かれます。手動でコンフィグレーションされる各分散トピック メンバーは、動的に作成されるプロキシトピック メンバーごとにシステム サブスクリプションを作成します。次に、非恒久コンシューマがプロキシトピック メンバー上に作成されます。

- 分散トピック システム サブスクリプション — システム サブスクリプションは、コンフィグレーション済みの分散送り先メンバー間で、メッセージを転送するために使用されます。たとえば、1つの分散トピックの中に n 個のメンバーがある場合、各メンバーは少なくとも $n-1$ 個のシステム サブスクライバを持っています。さらに、各プロキシトピック メンバーに対して、各分散トピック メンバー上に 1つのシステム サブスクリプションがあります。
- 分散キュー システム サブスクリプション — 分散キューの [転送の遅延] 属性を、(デフォルト値の -1 秒を変更して)有効にしている分散キュー メンバーも、システム サブスクリプションを作成することがあります。システム サブスクライバは、コンシューマを持たないキュー メンバーから、コンシューマを持つキュー メンバーへメッセージを転送するために使用されます。
- 恒久システム サブスクリプション — 分散トピック メンバーに対して JMS ファイルストアまたは JDBC ストアがコンフィグレーションされている場合、恒久サブスクライバとしてシステム サブスクリプションが作成されます。システム サブスクリプションは、Administration Console に名前が表示されます。

JMS のチューニング

以下の節では、WebLogic JMS で利用できる管理パフォーマンス チューニング機能を実装することによって、アプリケーションを最大限に活用する方法について説明します。

- 永続ストレージ
- メッセージ ページングのコンフィグレーション
- メッセージのフロー制御の確立
- 分散送り先のチューニング

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。

JMS ファイルストアの同期書き込みポリシーのコンフィグレーション

WebLogic 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。その代わりに、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、OS 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： ファイルストアが非永続的なメッセージのディスクへのページングのために排他的に使用されている場合、同期書き込みポリシーは無視されません。

表 9-1 は、WebLogic Server 上で実行されているすべての JMS ファイルストアの同期書き込みポリシーをコンフィグレーションする際のオプションについて説明しています。

表 9-1 同期書き込みポリシーの属性

属性	説明
Disabled	ファイルストアの書き込みにおいて、オペレーティングシステムのキャッシュとファイルシステムのオンディスクキャッシュの両方を使用できる。もっとも高速なポリシーだが、信頼性はもっとも劣る。他のポリシーに比べて 100 倍以上高速だが、停電やオペレーティングシステムの障害によってメッセージが消失したり重複したりするおそれがある。

表 9-1 同期書き込みポリシーの属性

属性	説明
Cache-Flush	デフォルトのポリシー。すべての書き込みがディスクにフラッシュされるまでトランザクションは完了しない。このポリシーは信頼性とスケーラビリティが高く、同時に処理できるユーザ数も増える。

表 9-1 同期書き込みポリシーの属性

属性	説明
Direct-Write	<p>ファイルストアの書き込みが、直接ディスクに書き込まれる。このポリシーは、Sun Solaris および Windows システムでサポートされている。サポートされていないプラットフォームで Direct-Write ポリシーを設定した場合、ファイルストアは自動的に Cache-Flush ポリシーを代わりに使用する。</p> <p>Direct-Write ポリシーの信頼性とパフォーマンスは、直接書き込みに関するプラットフォームのオンディスク キャッシュの用法によって異なる。たとえば、UNIX システムでは直接書き込みにオンディスク キャッシュを使用しないが、Windows システムでは通常使用する。このポリシーでオンディスク キャッシュ (利用可能な場合) を使用する際の良い点と悪い点は以下のとおり。</p> <ul style="list-style-type: none"> ■ オンディスク キャッシュを有効にすると、Cache-Flush ポリシーに比べて 2 ~ 5 倍高速になる。ただし、非常に高いスケーラビリティが要求される場合は、多少遅くなるおそれがある。 ■ オンディスク キャッシュを無効にすると、1 対多の場合は Cache-Flush ポリシーより速いが、それ以外の場合は Cache-Flush ポリシーより遅くなる。 ■ Direct-Write ポリシーでは、オンディスク キャッシュを有効にするとスケーラビリティが向上し、無効にすると向上しない (Sun Solaris では直接書き込みのオンディスク キャッシュを有効にできないことに注意)。

警告： **Sun Solaris** とは異なり、**Windows** 上で **Direct-Write** ポリシーを使用する場合、トランザクション データはオンディスク キャッシュに残され、す

ぐにはディスクに書き込まれません。この場合、停電などによってオンディスク キャッシュのデータが消えてメッセージが消失したり重複したりするおそれがあり、トランザクションとしては安全とはいえません。**Windows** で **Direct-Write** の信頼性を高めるには、ディスクの書き込みキャッシュをすべて無効にするか、バッテリー バックアップのあるキャッシュを使用します。

Windows でハードディスクのオンディスク キャッシュを無効にするには、[スタート | 設定 | コントロール パネル | システム] を選択し、[ハードウェア] タブで [デバイス マネージャ] ボタンをクリックして [ディスク ドライブ] アイコンをダブルクリックします。次に、ドライブの名前をダブルクリックし、[ディスクのプロパティ] タブの [書き込みキャッシュを有効にする] チェック ボックスのチェックをはずします。ただし、この値を変更できないファイル システムもあります (たとえば、信頼性のあるキャッシュを備えた RAID システム)。

ポリシー設定の比較

以下の表では、信頼性、パフォーマンス、およびスケーラビリティに関して、同期書き込みポリシーを比較しています。以下のキーポイントを使用して、同期書き込みポリシーの設定に基づいて予想される結果を解釈してください。

- **ディスク キャッシュ (オン / オフ)** : オンディスクの書き込みキャッシュが有効または無効のどちらか。
- **1-m** のパフォーマンス : 非常に少ないクライアント。
- **m-m** のパフォーマンス : 多数のクライアント。
- **M-M** のパフォーマンス : 多数の同時クライアント。
- **信頼性 (低 / 高)** : 「かならず 1 回」の (トランザクション) メッセージには高い信頼性が必要です。

表 9-2 相対的なパフォーマンス (同じカラム内で比較)

ポリシー	ディスク キャッシュ	1-m のパ フォーマ ンス	m-m のパ フォーマ ンス	M-M のパ フォーマンス	信頼性
Disabled	オン	****	****	****	低 (OS のキャッシュによる)
	オフ	****	****	****	低
Cache-Flush	オン	*	**	***	高
	オフ	*	**	***	高
Direct-Write	オン	**	***	**	中 (信頼性の高いディスク キャッシュの場合は高)
	オフ	**	*	*	高

表 9-3 相対的なスケーラビリティ (同じ行内で比較)

ポリシー	ディスク キャッシュ	1-m のパ フォーマ ンス	m-m のパ フォーマ ンス	M-M のパ フォーマンス
Disabled	オン	****	****	****
	オフ	****	****	****
Cache-Flush	オン	*	**	***
	オフ	*	**	***
Direct-Write	オン	*	**	***
	オフ	*	*	*

メッセージ ページングのコンフィグレーション

メッセージ ページング機能を使用すると、メッセージ負荷のピーク時に仮想メモリを解放できます。この機能は、大きなメッセージ空間を必要とするアプリケーションには非常に有効です。

JMS メッセージ ページングでは、永続メッセージおよび非永続メッセージの両方においてメモリを節約できます。永続メッセージのデータはメモリにキャッシュされますが、それでもメモリを節約する効果はあります。ページングされた永続メッセージは、通常のバッキング ストア (ファイルまたはデータベース) に書き込まれます。ページングされた非永続メッセージは、**JMS** サーバのメッセージ ページング ストアに書き込まれます。これらは別々にコンフィグレーションできます。

メッセージをページから取り出しても、これに使用されていたすべてのメモリが解放されるわけではありません。メッセージのヘッダとプロパティはメモリに残され、検索、ソート、およびフィルタリングに使用されます。

注意： トランザクションセッションで送信されるメッセージは、セッションがコミットされた後のみページングできます。その前には、メッセージはメモリにのみ保持されます。したがって、**Java** 仮想マシン (**JVM**) のヒープ サイズは、コミットされるまですべてのアクティブセッションからのクライアント負荷の予想される最大量に対応できるように適切に調整する必要があります。ヒープ サイズのチューニングの詳細については、『**BEA WebLogic Server** パフォーマンスチューニングガイド』の「**Java** 仮想マシン (**JVM**) のチューニング」を参照してください。

ページングのコンフィグレーション

ページングがコンフィグレーションされていないか有効になっていない場合は、すべてのメッセージ (永続的なメッセージを含む) がメモリに保持されます。**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 $2^{63}-1$)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

ページングストアは、JMS サーバごとに必要です。各 JMS サーバおよびその送り先への非永続メッセージは、この専用のページストアを使用してページングされます。JMS JDBC ストアではなく、JMS ファイル ストアを使用することをお勧めします。JDBC ストアは、特にメリットがない上、性能も劣るためです。

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 [ページング ストア] 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。[適用] をクリックして変更を保存します。
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、9-30 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. [しきい値と割当] タブで、バイト ページングをコンフィグレーションします。
 - [バイト ページングを有効化] チェック ボックスを選択します。
 - [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。
5. [しきい値と割当] タブで、メッセージ ページングをコンフィグレーションします。
 - [メッセージ ページングを有効化] チェック ボックスを選択します。
 - [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。
- 6. [適用] をクリックして、新しいバイト ページング値とメッセージ ページング値を保存します。
- 7. 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 ～ 6 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。
- 8. **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、9-32 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 9-33 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] タブで、バイト ページングをコンフィグレーションします。
 - [バイト ページングを有効化] チェック ボックスを選択します。
 - [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。
5. [しきい値と割当] タブで、メッセージ ページングをコンフィグレーションします。
 - [メッセージ ページングを有効化] チェック ボックスを選択します。
 - [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。
 6. [適用] をクリックして、新しいバイト ページング値とメッセージ ページング値を保存します。
 7. **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 ~ 6 を繰り返します。
 8. 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] で、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] タブで、バイト ページングをコンフィグレーションします。
 - [バイト ページングを有効化] チェック ボックスを選択します。
 - [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。
5. [しきい値と割当] タブで、メッセージ ページングをコンフィグレーションします。
 - [メッセージ ページングを有効化] チェック ボックスを選択します。
 - [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。
 6. [適用] をクリックして、新しいバイト ページング値とメッセージ ページング値を保存します。
 7. 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 ~ 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。

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、9-34 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 9-2 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] で、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に関連付けられた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] タブで、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 [バイト ページングを有効化] または [メッセージ ページングを有効化] 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、[バイト ページングを有効化] または [メッセージ ページングを有効化] 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、[バイト ページングを有効化] または [メッセージ ページングを有効化] リスト ボックスで [True] を選択します。
5. [適用] をクリックして、新しいバイト ページング値とメッセージ ページング値を保存します。
6. 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 ~ 5 を繰り返します。
7. 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 9-4 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「JMS サーバ」を参照してください。

表 9-4 JMS サーバの属性

属性	説明
[バイト ページングを有効化]	<ul style="list-style-type: none"> ■ [バイト ページングを有効化] 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。 ■ [バイト ページングを有効化] 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、[バイト ページングを有効化] 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。
[メッセージ ページングを有効化]	<ul style="list-style-type: none"> ■ [メッセージ ページングを有効化] 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。 ■ [メッセージ ページングを有効化] 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 -1 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、[メッセージ ページングを有効化] 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。

表 9-4 JMS サーバの属性

属性	説明
[ページング ストア]	<p>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。</p> <p>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。</p>

JMS テンプレートのページング属性

表 9-5 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 9-5 JMS テンプレートの属性

属性	説明
[バイト ページングを有効化]	<ul style="list-style-type: none"><li data-bbox="758 331 1260 509">■ [バイト ページングを有効化] 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。<li data-bbox="758 532 1260 818">■ [バイト ページングを有効化] 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。<li data-bbox="758 841 1260 964">■ JMS テンプレート Mbean に値が定義されていない場合、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 9-5 JMS テンプレートの属性

属性	説明
[メッセージ ページングを有効化]	<ul style="list-style-type: none"> <li data-bbox="672 280 1192 470">■ [メッセージ ページングを有効化] チェックボックスを選択しない場合 (False)、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。 <li data-bbox="672 487 1192 771">■ [メッセージ ページングを有効化] 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。 <li data-bbox="672 787 1192 925">■ JMS テンプレート Mbean に値が定義されていない場合、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 9-6 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 9-6 JMS の送り先の属性

属性	説明
[バイト ページングを有効化]	<ul style="list-style-type: none"> ■ [バイト ページングを有効化] 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。 ■ [バイト ページングを有効化] 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に関して有効になる。 ■ [バイト ページングを有効化] 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)。送り先の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 と同じになる。
[メッセージ ページングを有効化]	<ul style="list-style-type: none"> ■ [メッセージ ページングを有効化] 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。 ■ [メッセージ ページングを有効化] 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に関して有効になる。 ■ [メッセージ ページングを有効化] 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)。送り先の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 と同じになる。

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 9-7 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 の送り先」、および「JMS テンプレート」を参照してください。

表 9-7 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 -1 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ない。
- 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 -1 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ない。

- 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 [バイト ページングを有効化] [メッセージ ページングを有効化] 属性を **True** に設定する必要がある。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。

メッセージのフロー制御の確立

WebLogic JMS のフロー制御機能を使うと、メッセージプロデューサが過負荷になっていると判断したときに、**JMS** サーバまたは送り先で、メッセージプロデューサの処理速度を低下させることができます。具体的には、**JMS** サーバや送り先で、指定のバイト数またはメッセージ数のしきい値を超えた場合に対処機能が作動し、メッセージフロー (1 秒当たりのメッセージ数) を制限するようプロデューサに指示が出されます。

プロデューサでは、**JMS** 接続ファクトリを通じてプロデューサ用にコンフィグレーションされた一連のフロー制御属性に基づき、プロダクション率を制限します。プロデューサは、指定した最大フローのメッセージ数から開始して、所定の間隔ごとに (たとえば、60 秒間、10 秒ごとに)、サーバや送り先がまだ対処機能を作動させたままであるかどうかを評価します。各間隔ごとに、サーバや送り先で対処機能が作動したままであれば、プロデューサは、所定のフロー最小値まで、プロダクション率を低下させます。

プロデューサで処理速度が低下するにつれ、しきい値条件は徐々に補正されます。これはサーバや送り先の対処機能が解除されるまで行われます。この時点で、プロデューサはプロダクション率を上げることができるようになりますが、必ずしも最大限まで上げる必要はありません。実際には、メッセージフローは、所定の最大フローに達してフロー制御の対象でなくなる時点までは、(サーバや送り先が対処機能を解除していても) 引き続き制御されます。

フロー制御のコンフィグレーション

プロデューサは、セッションから一連のフロー制御属性を受信します。セッションは接続から、接続は接続ファクトリから、この属性を受信します。**JMS** 接続ファクトリでのフロー制御属性のコンフィグレーションは、**Administration Console** を通じて行います。

これらの属性により、プロデューサはメッセージフローを調整できます。具体的には、プロデューサはフローを最小値から最大値までの範囲内に制限する属性を受信します。プロデューサは、条件が悪くなると最小値側に移動し、改善されると最大値側に移動します。最小値側および最大値側への移行は、移行率を指定する2つの追加属性によって定義されます。また、最小値側および最大値側への移行の必要性は、コンフィグレーションされた間隔ごとに評価されます。

接続ファクトリでのメッセージフロー制御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. [JMS] ノードを展開します。
2. [接続ファクトリ] ノードをクリックします。右ペインに [JMS 接続ファクトリ] テーブルが表示され、ドメインで定義された接続ファクトリがすべて示されます。
3. メッセージフロー制御を確立する接続ファクトリをクリックします。右ペインにダイアログが表示され、接続ファクトリの変更に関連するタブが示されます。
4. [フロー制御] タブで、次の表で説明する属性を定義します。

表 9-8 フロー制御属性

属性	説明
[フロー制御を有効化]	プロデューサが JMS サーバによってフロー制御可能かどうかを決定する。

表 9-8 フロー制御属性

属性	説明
[最大フロー]	<p>しきい値の条件に達したプロデューサに対する秒当たりの最大メッセージ数。</p> <p>しきい値の条件に達したときにプロデューサがフローを制御している場合、そのプロデューサの初期フロー制限が FlowMaximum に設定される。しきい値の条件に達したときにプロデューサが既にフローを制御している場合 (フロー制限は FlowMaximum 未満)、プロデューサは次にフローが評価されるまで現在のフロー制限で処理を継続する。</p> <p>いったん、しきい値条件への抵触を回避してからは、プロデューサはフロー限度を無視できなくなる。このフロー制限が FlowMaximum 未満の場合、プロデューサはフローが評価されるたびにそのフローを徐々に FlowMaximum まで増やす必要がある。プロデューサが FlowMaximum に達すると、そのフロー制限を無視し、そのフローを制限せずに送信できる。</p>
[最小フロー]	<p>しきい値の条件に達したプロデューサに対する秒当たりの最小メッセージ数。これは、プロデューサのフロー制御の下限値。つまり、WebLogic JMS は、フロー制限が FlowMinimum に達したプロデューサの処理速度はそれ以上落とさない。</p>
[フロー間隔]	<p>プロデューサが FlowMaximum のメッセージ数から FlowMinimum に、またはその反対にフローを調整するときの調整期間 (単位は秒)。</p>

表 9-8 フロー制御属性

属性	説明
[フロー ステップ]	<p data-bbox="584 284 1192 511">プロデューサがフローを [最小フロー] のメッセージ数から [最大フロー] のメッセージ数に、または [最大フロー] のメッセージ数から [最小フロー] のメッセージ数に、調整する場合に使用されるステップ数。フロー間隔の調整期間は、フローステップ数に分割される (たとえば 60 秒を 6 ステップで割ると 1 ステップ 10 秒となる)。</p> <p data-bbox="584 519 1192 673">また、FlowMaximum と FlowMinimum の差をステップに分割することにより、移動 (調整率) が計算される。各フロー ステップでは、次のように現在の条件に基づいて必要に応じてフローが上方または下方に調整される。</p> <ul data-bbox="584 690 1192 876" style="list-style-type: none"> <li data-bbox="584 690 1192 787">■ 下方移動 (減衰) は指定した期間 (フロー間隔) および指定したステップ数に対し幾何級数的 (たとえば、100、50、25、12.5)。 <li data-bbox="584 803 1192 876">■ 上方移動は線形。差は単純にステップ数で除算される。

5. [適用] をクリックして、新しい属性値を保存します。

他の接続ファクトリ属性の詳細、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 接続ファクトリ**」を参照してください。

フロー制御のしきい値

バイト数やメッセージ数のしきい値のコンフィグレーションに使用される属性は、**JMS** サーバおよび **JMS** 送り先の一部として定義されます。表 9-9 では、これらのしきい値がどのように **JMS** サーバや **JMS** 送り先に対するフロー制御を開始および停止するのを示します。

表 9-9 フロー制御しきい値属性

属性	説明
[最大バイトしきい値]または[最大メッセージしきい値]	バイト数やメッセージ数がこのしきい値を超えると、 JMS サーバまたは JMS 送り先で対処機能が作動し、プロデューサに対してメッセージフローを制限するよう指示が行われる。
[最小バイトしきい値]または[最小メッセージしきい値]	バイト数やメッセージ数がこのしきい値を下回る場合、 JMS サーバまたは JMS 送り先で対処機能が解除され、プロデューサに対してメッセージフローを増やし始めるよう指示が行われる。 フロー制御は、メッセージの最大フローを下回っているプロデューサについては、引き続き有効である。プロデューサは、最大フローに到達してフロー制御が行われなくなるまで、プロデュース率を上げ続けることができる。

JMS サーバおよび **JMS** 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」、および「**JMS** の送り先」を参照してください。

分散送り先のチューニング

以降の節では、**JMS** 接続ファクトリに対する属性をコンフィグレーションすることで分散送り先をチューニングする方法について説明します。

分散送り先のコンフィグレーションの詳細については、9-49 ページの「分散送り先のコンフィグレーション」を参照してください。

分散送り先のメッセージロード バランシングのコンフィグレーション

[**JMS Connection Factory** | コンフィグレーション | 一般] タブの [ロードバランシングを有効化] 属性では、接続ファクトリを通じて作成された匿名でないプロデューサが、呼び出しごとにロードバランシングされるかどうかを定義します。

分散送り先を使用して、複数の物理的送り先にまたがってプロデューサおよびコンシューマを分散またはバランシングするが、メッセージが生成されるたびにロードバランシングの決定を行いたくないアプリケーションでは、[ロードバランスを有効化] 属性をオフにできます。

分散送り先でメッセージング負荷が公正に分散されるようにするために、プロデューサで使用される最初の物理的送り先(キューまたはトピック)は必ず、分散送り先メンバーの中から無作為に選択されます。

接続ファクトリでのロードバランシング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. [JMS] ノードを展開します。
2. [接続ファクトリ] ノードをクリックします。右ペインに [JMS 接続ファクトリ] テーブルが表示され、ドメインで定義された接続ファクトリがすべて示されます。
3. メッセージのロードバランシングを確立する接続ファクトリをクリックします。右ペインにダイアログが表示され、接続ファクトリの変更に関連するタブが示されます。
4. [一般] タブで [ロードバランスを有効化] 属性を定義します。

- [ロードバランスを有効化] = *True*

`Queue.sender.send()` メソッドの場合、匿名でないプロデューサは呼び出しのたびに分散キューメンバーでロードバランシングされます。

`TopicPublish.publish()` メソッドの場合、匿名でないプロデューサはロードバランス有効化の設定に関係なく呼び出しのたびに常に同じ物理的トピックに固定されます。

- [ロードバランスを有効化] = *False*

プロデューサは、失敗するまで同じ物理的送り先に対して生成を行います。失敗すると、新しい物理的送り先が選択されます。

5. [適用] をクリックして変更を保存します。

注意： 実装によっては、[サーバアフィニティを有効化] 属性の設定が、分散送り先のロードバランシング設定に影響することがあります。詳細については、『WebLogic JMS プログラマーズ ガイド』の「[サーバアフィニティを有効化] 属性を使用した場合の分散送り先のロードバランシングへの影響」を参照してください。

匿名プロデューサ (作成時に送り先を指定しないプロデューサ) は、送り先を切り替えるたびにロード バランシングされます。同じ送り先を使用し続けると、匿名でないプロデューサと同じ法則 (前述) が適用されます。

分散送り先のメンバー間でメッセージのロード バランシングがどのように行われるのかについては、『WebLogic JMS プログラマーズガイド』の「分散送り先におけるメッセージのロード バランシング」を参照してください。

分散送り先のサーバ アフィニティのコンフィグレーション

[JMS Connection Factory | コンフィグレーション | 一般] タブの [サーバ アフィニティを有効化] 属性では、分散送り先の複数の物理的送り先にわたってコンシューマまたはプロデューサをロード バランシングしている WebLogic Server が、同じ WebLogic Server 上で実行されている他の物理的送り先にまたがるロード バランシングを最初に試みるかどうかを定義します。

注意: [サーバ アフィニティを有効化] 属性は、キュー ブラウザに影響を与えません。したがって、分散キューで作成されたキュー ブラウザは、サーバ アフィニティが有効な場合でもリモートの分散キュー メンバーに固定することができます。

接続ファクトリでサーバ アフィニティを無効化するには、次の手順に従います。

1. [JMS] ノードを展開します。
2. [接続ファクトリ] ノードをクリックします。右ペインに [JMS 接続ファクトリ] テーブルが表示され、ドメインで定義された接続ファクトリがすべて示されます。
3. サーバ アフィニティを無効化する対象となる接続ファクトリをクリックします。右ペインにダイアログが表示され、接続ファクトリの変更に関連するタブが示されます。
4. [一般] タブで [サーバ アフィニティを有効化] 属性を定義します。
 - [サーバ アフィニティを有効化] = *True*
分散送り先セット内の複数の物理的送り先の間でコンシューマまたはプロデューサのロード バランシングを行っている WebLogic Server インスタンスは、まず、同じ WebLogic Server で実行されている他の物理的送り先の間でロード バランシングを試みます。

- [サーバアフィニティを有効化]=*False*
WebLogic Server インスタンスは、分散送り先セットの複数の物理的送り先の間でコンシューマまたはプロデューサをロードバランシングし、同じ WebLogic Server 上で実行されている他の物理的送り先は無視します。

5. [適用]をクリックして変更を保存します。

[サーバアフィニティを有効化]の設定が分散送り先メンバー間でのロードバランシングにどのように影響するのかについては、『WebLogic JMS プログラマーズガイド』の「[サーバアフィニティを有効化]属性を使用した場合の分散送り先のロードバランシングへの影響」を参照してください。

分散送り先のコンフィグレーション

分散送り先とは、1つの JNDI 名で呼び出される物理的送り先(キューまたはトピック)のセットのことです。したがって、セットのメンバーが実際にはクラスタ内の複数のサーバに分散し、各メンバーが別々の JMS サーバに属していても、そのセットはクライアント側からは1つの論理的送り先として認識されます。

複数の物理的キューおよびトピックを分散送り先のメンバーとしてコンフィグレーションできるようにすることで、WebLogic JMS はクラスタ内の JMS 送り先の高可用性とロードバランシングをサポートします。たとえば、サーバの障害などによって使用できない送り先メンバーがある場合は、セット内の他のメンバーにトラフィックがリダイレクトされます。

アプリケーションで分散送り先を使用する手順については、『WebLogic JMS プログラマーズガイド』の「WebLogic JMS アプリケーションの開発」を参照してください。

分散送り先のコンフィグレーション手順

分散 JMS 送り先は、Administration Console の [JMS | 分散送り先] ノードでコンフィグレーションできます。コンフィグレーションプロセスがわかりやすいように、以下のシナリオに対応する手順に分けて説明しています。

- 物理的送り先を持たない **WebLogic JMS** の新しい実装、または、コンフィグレーション済みの送り先を分散送り先の一部にする必要のない **WebLogic JMS** の既存のコンフィグレーション。
 - 分散トピックの作成とメンバーの自動作成
 - 分散キューの作成とメンバーの自動作成
- 以前にコンフィグレーションした送り先を分散送り先セットのメンバーとして追加する必要がある **WebLogic JMS** の既存の実装。
 - 分散トピックの作成および既存の物理的トピックのメンバーとしての手動追加
 - 分散キューの作成および既存の物理キューのメンバーとしての手動追加

注意： 分散送り先のコンフィグレーションをチューニングするためのデフォルトの [ロードバランスを有効化] および [サーバアフィニティを有効化] 属性は、**Administration Console** を使って **JMS 接続ファクトリ** で変更できます。詳細については、9-46 ページの「分散送り先のメッセージロードバランシングのコンフィグレーション」と 9-48 ページの「分散送り先のサーバアフィニティのコンフィグレーション」を参照してください。

分散送り先が作成されると、その分散送り先メンバーのデフォルト属性値を使用して対応する **JMS** テンプレートが自動的に作成されます。新しいテンプレートは、[**JMS テンプレート**] ノードに分散送り先と同じ名前が表示されます。分散送り先メンバーのしきい値や割当などの属性は、このテンプレートを使用してリセットできます。

分散トピックの作成とメンバーの自動作成

分散トピックをコンフィグレーションし、**WebLogic Server** クラスタの一部である **JMS** サーバ (高可用性のため)、またはクラスタの一部ではない単一の **WebLogic Server** インスタンス上の **JMS** サーバにトピック メンバーを自動的に作成するには、以下の手順に従います。

1. [**JMS | 分散送り先**] ノードを展開します。
2. 右ペインで [新しい **Distributed Topic** のコンフィグレーション] リンクをクリックします。[コンフィグレーション] ダイアログに、新しい分散トピックのコンフィグレーションに関連するタブが表示されます。
3. 次の表に従って、[一般] タブの属性を定義します。

表 9-10 [一般] タブの分散トピック属性

属性	説明
[名前]	WebLogic Server ドメイン内で分散トピックをユニークに識別する。
[JNDI名]	分散トピックを JNDI ネームスペース内にバインドするために使用される名前を入力する。アプリケーションは、JNDI 名を使用して分散トピックをルックアップする。 JNDI 名を持たない分散トピックは、分散送り先の名前を <code>javax.jms.TopicSession.createTopic()</code> に渡すことで参照できる。
[ロードバランス ポリシー]	プロデューサがどのようにメッセージを分散トピックのメンバー間に分散するかを定義する。有効値は、9-46 ページの「分散送り先のメッセージロードバランシングのコンフィグレーション」に定義される [ラウンドロビン] と [ランダム]。

- [作成] をクリックして分散トピックを作成します。
- [しきい値と割当] タブで、全分散トピック メンバーの以下の属性を定義します。
 - 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 ページングやメッセージ ページングが分散トピックで有効化されているかどうか。

これらの属性の詳細については、Administration Console オンライン ヘルプの「[JMS トピック] --> [コンフィグレーション] --> [しきい値と割当]」を参照してください。
- [適用] をクリックして、新しい属性値を保存します。
- [自動デプロイ] タブで、分散トピック メンバーを自動作成する WebLogic Server インスタンスを指定します。

8. [選択されたサーバ (および **JMS** サーバ) にメンバを作成する ...] テキストリンクをクリックします。以下のオプションのいずれかを選択するよう要求する自動デプロイ ダイアログが表示されます。
 - 分散トピックの対象とするクラスタを選択して [次へ] をクリックする。
または
 - 個別のサーバまたはクラスタ内のサーバを選択できるように、[なし] オプションをそのまま使用してこのダイアログを無視する (この場合は、手順 10 に進む) 。
9. クラスタを選択した場合は、以下の手順を実行して、クラスタ内の **WebLogic Server** インスタンスを選択します。
 - a. クラスタのメンバーで、まだ分散トピックのホストではない **WebLogic Server** サーバが、すべてリストされ、デフォルトで選択されます。サーバインスタンスが分散トピックのホストにならないようにするには、該当するチェック ボックスのチェックを解除します。
 - b. [次へ] をクリックして次のダイアログに進みます。
 - c. 分散トピック メンバーを作成するために選択した **WebLogic Server** で利用可能な **JMS** サーバを選択するには、手順 11 に進みます。
10. 手順 8 の [クラスタ] ダイアログで [なし] を選択した場合は、ドメイン内の単一の **WebLogic Server** インスタンスを選択します。
 - a. リスト ボックスから、分散トピック メンバーを作成する個々のサーバを選択します。
 - b. [次へ] をクリックして次のダイアログに進みます。
11. 選択した **WebLogic Server** インスタンス上にデプロイされており、まだ分散トピックのホストではない **JMS** サーバが、すべてリストされ、デフォルトで選択されます。**JMS** サーバが分散トピック メンバーのホストにならないようにするには、該当するチェック ボックスのチェックを解除します。

選択した **JMS** サーバに既存の分散トピック メンバーがない場合は、各 **JMS** サーバに新しい **JMSTopic** が 1 つ作成され、分散トピックのメンバーとして追加されます。
12. [次へ] をクリックして、最後の [自動デプロイ] ダイアログに進みます。
13. [適用] をクリックして、[自動デプロイ] での選択を保存します。

14. [コンフィグレーション | メンバ] タブをクリックして、新しい分散トピック用に自動作成されたトピック メンバーを表示します。
15. [JMS | テンプレート] ノードを展開して、分散トピックと同じ名前で自動的に作成された JMS テンプレートを表示します。

分散トピックの作成および既存の物理的トピックのメンバーとしての手動追加

以前にコンフィグレーションした送り先を分散送り先セットのメンバーとして追加する必要がある WebLogic JMS の既存の実装で、分散トピックをコンフィグレーションして既存の物理トピックをメンバーとして手動で追加するには、以下の手順に従います。

1. [JMS | 分散送り先] ノードを展開します。
2. 右ペインで [新しい Distributed Topic のコンフィグレーション] リンクをクリックします。[コンフィグレーション] ダイアログに、新しい分散トピックのコンフィグレーションに関連するタブが表示されます。
3. 次の表に従って、[一般] タブの属性を定義します。

表 9-11 [一般] タブの分散トピック属性

属性	説明
[名前]	WebLogic Server ドメイン内で分散トピックをユニークに識別する。
[JNDI 名]	分散トピックを JNDI ネームスペース内にバインドするために使用される名前を入力する。アプリケーションは、JNDI 名を使用して分散トピックをルックアップする。 JNDI 名を持たない分散トピックは、分散送り先の名前を <code>javax.jms.TopicSession.createTopic()</code> に渡すことで参照できる。

表 9-11 [一般] タブの分散トピック属性

属性	説明
[ロード バランス ポリシー]	プロデューサがどのようにメッセージを分散トピックのメンバー間に分散するかを定義する。有効値は、9-46 ページの「分散送り先のメッセージ ロード バランシングのコンフィグレーション」に定義される [ラウンドロビン] と [ランダム]。

- [作成] をクリックして分散トピックを作成します。
- [しきい値と割当] タブで、全分散トピック メンバーの以下の属性を定義します。
 - 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 ページングやメッセージ ページングが分散トピックで有効化されているかどうか。

分散トピック メンバーの基底の物理トピックに、既にしきい値と割当がコンフィグレーションされた JMS テンプレートが備わっている場合、これらの属性はそのトピック メンバーには適用されません。これらの属性の詳細については、Administration Console オンライン ヘルプの「[JMS トピック] --> [コンフィグレーション] --> [しきい値と割当]」を参照してください。

- [適用] をクリックして、新しい属性値を保存します。

注意： WebLogic Server クラスタの一部である JMS サーバ (高可用性のため)、またはクラスタの一部ではない単一の WebLogic Server インスタンス上の JMS サーバにトピック メンバーを自動的に作成する場合は、9-50 ページの「分散トピックの作成とメンバーの自動作成」を参照してください。

- [コンフィグレーション | メンバ] タブで、既存の物理トピックのための分散トピック メンバーを作成します。
- 右ペインで [新しい Distributed Topic Member のコンフィグレーション] リンクをクリックします。[コンフィグレーション] ダイアログに、新しい分散トピック メンバーのコンフィグレーションに関連するタブが表示されます。
- 次の表に従って、[一般] タブの属性を定義します。

表 9-12 [一般] タブの分散トピック メンバー属性

属性	説明
[名前]	WebLogic Server ドメイン内で分散トピック メンバーをユニークに識別する。
[JMS トピック]	分散トピック メンバーと関連する基底の物理トピックを選択する。
[重み]	トピック メンバーの重み (メッセージ負荷を処理する能力の尺度) を、分散送り先における他のトピック メンバーとの比較で定義する。 ランダム分散ロード バランシング アルゴリズムは、物理的送り先に割り当てられた重みを使用して、一連の物理的送り先の重み付けされた分散を計算する。詳細については、『WebLogic JMS プログラマーズ ガイド』の「WebLogic JMS アプリケーションの開発」を参照。

- [作成] をクリックして新しい分散トピック メンバーを作成します。新しいメンバーが [分散トピック] テーブルに追加されます。
- 必要に応じて手順 8 ～ 10 を繰り返し、トピック メンバーを分散トピックに追加し続けます。
- [JMS | テンプレート] ノードを展開して、分散トピックと同じ名前でも自動的に作成された JMS テンプレートを表示します。

分散キューの作成とメンバーの自動作成

分散キューをコンフィグレーションし、WebLogic Server クラスタの一部である JMS サーバ (高可用性のため)、またはクラスタの一部ではない単一の WebLogic Server インスタンス上の JMS サーバにキュー メンバーを自動的に作成するには、以下の手順に従います。

- [JMS | 分散送り先] ノードを展開します。
- 右ペインで [新しい Distributed Queue のコンフィグレーション] リンクをクリックします。[コンフィグレーション] ダイアログに、新しい分散キューのコンフィグレーションに関連するタブが表示されます。

3. 次の表に従って、[一般] タブの属性を定義します。

表 9-13 [一般] タブの分散キュー属性

属性	説明
[名前]	WebLogic Server ドメイン内で分散キューをユニークに識別する。
[JNDI 名]	分散キューを JNDI ネームスペース内にバインドするために使用される名前を入力する。アプリケーションは、JNDI 名を使用して分散キューをルックアップする。 JNDI 名を持たない分散キューは、分散送り先名を <code>javax.jms.QueueSession.createQueue()</code> に渡すことによって参照できる。
[ロード バランス ポリシー]	プロデューサがどのようにメッセージを分散キューのメンバー間に分散するかを定義する。有効値は、9-46 ページの「分散送り先のメッセージ ロード バランシングのコンフィグレーション」に定義される [ラウンドロビン] と [ランダム]。
[転送の遅延]	メッセージを持つがコンシューマを持たない分散キュー メンバーが、コンシューマを持つ他のキューにメッセージを転送するまでに待機する時間を秒単位で定義する。

4. [作成] をクリックして分散キューを作成します。

5. [しきい値と割当] タブで、すべての分散キュー メンバーの以下の属性を定義します。

- 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 ページングやメッセージ ページングが分散キューで有効化されているかどうか。

これらの属性の詳細については、Administration Console オンライン ヘルプの「[JMS キュー] --> [コンフィグレーション] --> [しきい値と割当]」を参照してください。

6. [適用] をクリックして、新しい属性値を保存します。

7. [自動デプロイ] タブで、分散キュー メンバーを自動作成する **WebLogic Server** インスタンスを指定します。
8. [選択されたサーバ (および **JMS** サーバ) にメンバを作成する ...] テキストリンクをクリックします。以下のオプションのいずれかを選択するよう要求するダイアログが表示されます。
 - 分散キューの対象とするクラスタを選択して [次へ] をクリックする。
または
 - クラスタ内にないサーバを個別に選択できるように、[なし] オプションをそのまま使用してこのダイアログを無視する (この場合は、手順 10 に進む) 。
9. クラスタを選択した場合は、以下の手順を実行して、クラスタ内の **WebLogic Server** インスタンスを選択します。
 - a. クラスタのメンバーで、まだ分散キューのホストではないサーバが、すべてリストされ、デフォルトで選択されます。サーバが分散キューのホストにならないようにするには、該当するチェック ボックスのチェックを解除します。
 - b. [次へ] をクリックして次のダイアログに進みます。
 - c. 分散キュー メンバーを作成するために選択した **WebLogic Server** で利用可能な **JMS** サーバを選択するには、手順 11 に進みます。
10. 手順 8 の [クラスタ] ダイアログで [なし] を選択した場合は、ドメイン内の単一の **WebLogic Server** インスタンスを選択します。
 - a. リスト ボックスから、分散キュー メンバーを作成する個々のサーバを選択します。
 - b. [次へ] をクリックして次のダイアログに進みます。
11. 選択した **WebLogic Server** 上にデプロイされており、まだ分散キューのホストではない **JMS** サーバが、すべてリストされ、デフォルトで選択されます。**JMS** サーバが分散キュー メンバーのホストにならないようにするには、該当するチェック ボックスのチェックを解除します。

選択した **JMS** サーバに既存の分散キュー メンバーがない場合は、各 **JMS** サーバに新しい **JMS** キュー が 1 つ作成され、分散キューのメンバーとして追加されます。

12. [次へ] をクリックして、最後の [自動デプロイ] ダイアログに進みます。
13. [適用] をクリックして、[自動デプロイ] での選択を保存します。
14. [コンフィグレーション | メンバ] タブをクリックして、新しい分散キュー用に自動作成されたキュー メンバーを表示します。
15. [JMS | テンプレート] ノードを展開して、分散キューと同じ名前で自動的に作成された JMS テンプレートを表示します。

分散キューの作成および既存の物理キューのメンバーとしての手動追加

以前にコンフィグレーションした送り先を分散送り先セットのメンバーとして追加する必要がある WebLogic JMS の既存の実装で、分散キューをコンフィグレーションして既存の物理キューをメンバーとして手動で追加するには、以下の手順に従います。

1. [JMS | 分散送り先] ノードを展開します。
2. 右ペインで [新しい Distributed Queue のコンフィグレーション] リンクをクリックします。[コンフィグレーション] ダイアログに、新しい分散キューのコンフィグレーションに関連するタブが表示されます。
3. 次の表に従って、[一般] タブの属性を定義します。

表 9-14 [一般] タブの分散キュー属性

属性	説明
[名前]	WebLogic Server ドメイン内で分散キューをユニークに識別する。
[JNDI 名]	分散キューを JNDI ネームスペース内にバインドするために使用される名前を入力する。アプリケーションは、JNDI 名を使用して分散キューをルックアップする。 JNDI 名を持たない分散キューは、分散送り先名を <code>javax.jms.QueueSession.createQueue()</code> に渡すことによって参照できる。

表 9-14 [一般] タブの分散キュー属性

属性	説明
[ロードバランス ポリシー]	プロデューサがどのようにメッセージを分散キューのメンバー間に分散するかを定義する。有効値は、9-46 ページの「分散送り先のメッセージロード バランシングのコンフィグレーション」に定義される [ラウンドロビン] と [ランダム]。
[転送の遅延]	メッセージを持つがコンシューマを持たない分散キューメンバーが、コンシューマを持つ他のキューにメッセージを転送するまでに待機する時間を秒単位で定義する。

- [作成] をクリックして分散キューを作成します。
- [しきい値と割当] タブで、すべての分散キューメンバーの以下の属性を定義します。
 - 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 ページングやメッセージ ページングが分散キューで有効化されているかどうか。

分散キューメンバーの基底の物理キューに、既にしきい値と割当がコンフィグレーションされた JMS テンプレートが備わっている場合、これらの属性はそのキューメンバーには適用されません。これらの属性の詳細については、Administration Console オンラインヘルプの「[JMS キュー] --> [コンフィグレーション] --> [しきい値と割当]」を参照してください。

- [適用] をクリックして、新しい属性値を保存します。

注意： WebLogic Server クラスタの一部である JMS サーバ (高可用性のため)、またはクラスタの一部ではない単一の WebLogic Server インスタンス上の JMS サーバにキューメンバーを自動的に作成する場合は、9-55 ページの「分散キューの作成とメンバーの自動作成」を参照してください。
- [コンフィグレーション | メンバ] タブをクリックして、分散キューのキューメンバーを定義します。

8. 右ペインで [新しい Distributed Queue Member 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。[コンフィグレーション] ダイアログに、新しい分散キューのコンフィグレーションに関連するタブが表示されます。
9. 次の表に従って、[一般] タブの属性を定義します。

表 9-15 [一般] タブの分散キュー メンバー属性

属性	説明
[名前]	WebLogic Server ドメイン内で分散キュー メンバーをユニークに識別する。
[JMS キュー]	分散キュー メンバーと関連する基底の物理キューを選択する。
[重み]	<p>キュー メンバーの重み (メッセージ負荷を処理する能力の尺度) を、分散送り先における他のキュー メンバーとの比較で定義する。</p> <p>ランダム分散ロード バランシング アルゴリズムは、物理的送り先に割り当てられた重みを使用して、一連の物理的送り先の重み付けされた分散を計算する。詳細については、『WebLogic JMS プログラマーズ ガイド』の「WebLogic JMS アプリケーションの開発」を参照してください。</p>

10. [作成] をクリックして新しい分散キュー メンバーを作成します。新しいメンバーが [分散キュー] テーブルに追加されます。
11. 手順 8 ~ 10 を繰り返して、分散キューにメンバーを追加し続けます。
12. [JMS | テンプレート] ノードを展開して、分散キューと同じ名前でも自動的に作成された JMS テンプレートを表示します。

JMS 分散キュー メンバーの作成

分散キューのメンバーとして既存の物理的キューを追加するには、次の手順に従います。

1. [JMS | 分散送り先] ノードを展開します。[分散送り先] テーブルが右ペインに表示され、すべての分散キューおよびトピックが表示されます。
2. メンバーを追加する分散キューをクリックします。[分散キュー] テーブルに、その分散キューに属しているすべての分散キュー メンバーが表示されます。
3. [新しい Distributed Queue Member 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。ダイアログに、新しい分散キュー メンバーをコンフィグレーションするための [コンフィグレーション] タブが表示されます。
4. 分散キューのコンフィグレーション属性を定義します。
 - WebLogic Server ドメイン内で分散キュー メンバーをユニークに識別する。
 - 分散キュー メンバーと関連する基底の物理キューを選択する。
 - キュー メンバーの重み (メッセージ負荷を処理する能力の尺度) を、分散送り先における他のキュー メンバーとの比較で定義する。分散送り先のロード バランシングの詳細については、『WebLogic JMS プログラマーズガイド』の「WebLogic JMS アプリケーションの開発」を参照してください。

分散キュー メンバーの属性の詳細については、Administration Console オンライン ヘルプの「[分散キュー メンバ] --> [コンフィグレーション]」を参照してください。
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分散キュー メンバーを作成します。右ペインの [分散キュー メンバ] テーブルに、新しいメンバーが追加されます。
6. [適用] をクリックして、変更を保存します。

JMS 分散キュー メンバーの削除

分散キューを削除し、必要に応じてメンバーの基底の物理キューも削除するには、次の手順に従います。

注意： 分散キュー全体を削除する必要がある場合は、9-64 ページの「分散送り先の削除」の指示に従ってください。

1. [JMS | 分散送り先] ノードを展開します。[分散送り先] テーブルが右ペインに表示され、すべての分散キューおよびトピックが表示されます。
2. メンバーを削除する分散キューをクリックします。[分散キュー] テーブルに、その分散キューに属しているすべての分散キュー メンバーが表示されます。
3. 削除する分散キュー メンバーの行にある [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ダイアログが表示されます。
4. 基底の物理的キューも一緒に削除する場合は、[Also Delete] チェック ボックスを選択します。
5. [削除] をクリックして、分散キュー メンバー (および選択した場合は基底の物理キュー) を削除します。
6. [分散キュー] テーブルが右ペインに再表示されます。分散キュー メンバーが、[分散キュー] テーブルから削除されます。

JMS 分散トピック メンバーの作成

分散キューのメンバーとして既存の物理的キューを追加するには、次の手順に従います。

1. [JMS | 分散送り先] ノードを展開します。[分散送り先] テーブルが右ペインに表示され、すべての分散キューおよびトピックが表示されます。
2. メンバーを追加する分散トピックをクリックします。[分散トピック] テーブルに、その分散トピックに属しているすべての分散トピック メンバーが表示されます。
3. [新しい Distributed Topic Member 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。ダイアログに、新しい分散トピック メンバーをコンフィグレーションするための [コンフィグレーション] タブが表示されます。
4. 分散トピックの全般的なコンフィグレーション属性を定義します。
 - WebLogic Server ドメイン内で分散トピック メンバーをユニークに識別する。
 - 分散トピック メンバーと関連する基底の物理トピックを選択する。

- トピック メンバーの重み (メッセージ負荷を処理する能力の尺度) を、分散送り先における他のトピック メンバーとの比較で定義する。分散送り先のロード バランシングの詳細については、『WebLogic JMS プログラマーズ ガイド』の「WebLogic JMS アプリケーションの開発」を参照してください。

分散トピック メンバーの属性の詳細については、Administration Console オンライン ヘルプの「[分散トピック メンバ] --> [コンフィグレーション]」を参照してください。

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分散トピック メンバーを作成します。右ペインの [分散トピック] テーブルに、新しいメンバーが追加されます。
6. [適用] をクリックして、変更を保存します。

JMS 分散トピック メンバーの削除

分散トピックを削除し、必要に応じてメンバーの基底の物理トピックも削除するには、次の手順に従います。

注意： 分散トピック全体を削除する必要がある場合は、9-64 ページの「分散送り先の削除」の指示に従ってください。

1. [JMS] ノードを展開します。
2. [分散送り先] ノードを展開します。[分散送り先] テーブルが右ペインに表示され、すべての分散キューおよびトピックが表示されます。
3. メンバーを削除する分散トピックをクリックします。[分散トピック] テーブルに、その分散トピックに属しているすべての分散トピック メンバーが表示されます。
4. 削除する分散トピック メンバーの行にある [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ダイアログが表示されます。
5. 基底の物理的キューも一緒に削除する場合は、[Also Delete] チェック ボックスを選択します。

6. [削除] をクリックして、分散トピック メンバー（および選択した場合は基底の物理トピック）を削除します。
7. [分散トピック] テーブルが右ペインに再表示されます。分散トピック メンバーが、[分散トピック] テーブルから削除されます。

分散送り先の削除

分散送り先全体を削除する場合は、次の手順で削除する必要があります。

1. 以下の節の説明に従って、分散キューまたは分散トピックのすべてのメンバーを削除します。
 - 9-61 ページの「JMS 分散キュー メンバーの削除」
 - 9-63 ページの「JMS 分散トピック メンバーの削除」
2. [JMS | 分散送り先] ノードを展開し、削除する分散送り先の隣のごみ箱アイコンをクリックして分散送り先それ自体を削除します。

注意： 分散送り先は、そのメンバーがすべて適切に削除されている場合のみ削除できます。
3. 分散送り先と関連付けられている JMS テンプレートを削除できます。ただし、そのテンプレートが他の JMS サーバまたは送り先で使用されていないようにしてください。JMS テンプレートを削除するには、[JMS | テンプレート] ノードを展開してから、削除する JMS テンプレートの行にあるごみ箱アイコンをクリックします。

分散送り先のモニタ

分散送り先をモニタする場合、そのトピックまたはキュー メンバーに対して自動的に作成されるプロキシトピック メンバーまたはシステム サブスクリプションを参照できます。詳細については、9-22 ページの「分散送り先のシステム サブスクリプションとプロキシトピック メンバーのモニタ」を参照してください。

WebLogic Server の障害からの回復

以降の節では、サーバで障害が発生した場合に **JMS** アプリケーションを正常に終了させる方法、およびサーバで障害が発生した後に **JMS** データを移行する方法について説明します。

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 インスタンス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。	<code>JMSException</code> 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。	<code>ConsumerClosedException</code> がセッション例外リスナに配信される。失われた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。

新しいサーバへの **JMS** データの移行

WebLogic JMS では、WebLogic Server のコアに実装される移行フレームワークを使用します。これにより、WebLogic JMS は移行要求に正しく応答できるようになり、WebLogic JMS サーバのオンラインとオフラインの切り替えが順序立って行われるようになります。これには、スケジューリングされた移行だけでなく、WebLogic Server の障害への対応としての移行も含まれます。

いったん、正しくコンフィグレーションされると、JMS サーバとそのすべての送り先メンバーは、クラスタ内の別の WebLogic Server に移行できます。

新しくサーバを起動し、次の表に記載されたタスクのうち 1 つ以上を実行することで、障害が発生した WebLogic Server から JMS データを回復できます。

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	<ul style="list-style-type: none"> ■ 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	<p>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。</p>
トランザクション	<p><code><servername>*.tlog</code> 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。</p> <p>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。</p> <p>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可能性がある。</p> <p>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。</p>

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。

新しい WebLogic Server の起動については、2-1 ページの「WebLogic Server の起動と停止」を参照してください。障害が発生したサーバの回復については、『WebLogic Server ドメイン管理』の「障害が発生したサーバの回復」を参照してください。

移行できる対象の詳細については、『WebLogic Server クラスタ ユーザーズ ガイド』の「移行可能サービスをデプロイ、活性化、および移行する」を参照してください。

10 WebLogic メッセージングブリッジの使い方

以下の節では、WebLogic メッセージングブリッジをコンフィグレーションおよび管理する方法について説明します。

- 10-1 ページの「メッセージングブリッジとは」
- 10-2 ページの「メッセージングブリッジのコンフィグレーション タスク」
- 10-20 ページの「メッセージングブリッジを使用する WebLogic Server のさまざまなリリースおよびドメインとの相互運用」
- 10-26 ページの「メッセージングブリッジを使用するサードパーティのメッセージングプロバイダへのアクセス」
- 10-27 ページの「メッセージングブリッジの管理」

メッセージングブリッジとは

WebLogic メッセージングブリッジを使用すると、2つのメッセージング製品間の転送メカニズムをコンフィグレーションできます。これにより、別々に実装した WebLogic JMS 間や、WebLogic JMS と他のメッセージング製品間の相互運用性が提供されます。WebLogic メッセージングブリッジを使用すると、以下のようなメッセージングアプリケーションを統合できます。

- WebLogic Server のリリースが異なる WebLogic JMS の2つの実装
- 別々の WebLogic ドメインにある WebLogic JMS の実装
- WebLogic JMS とサードパーティの JMS 製品 (MQSeries など)
- WebLogic JMS と JMS 以外のメッセージング製品 (WebLogic Server では提供されない専用のアダプタを使用)

メッセージングブリッジは、ブリッジングされている2つの送り先で構成されます。1つはメッセージの受信元になるソース送り先、もう1つはメッセージの転送先になる対象送り先です。WebLogic JMS とサードパーティの JMS 製品では、ソース送り先および対象送り先との通信に WebLogic Server が提供するリソースアダプタを使用します。JMS 以外のメッセージング製品でソース送り先または対象送り先にアクセスするには、サードパーティ OEM ベンダからカスタムアダプタを入手するか、BEA プロフェッショナル サービスにお問い合わせください。

ソースブリッジ送り先および対象ブリッジ送り先にできるのは、キューおよびトピックです。また、メッセージフィルタ、トランザクションセマンティクス、接続の再試行ポリシーだけでなく、QOS (サービスの品質) も指定できます。いったんコンフィグレーションしたメッセージングブリッジは、Administration Console で容易に管理できます。Administration Console を使用すると、必要に応じてブリッジトラフィックを一時的にサスペンドしたり、実行スレッドプールのサイズを実装に合わせて調整したり、コンフィグレーションしたすべてのブリッジの状態をモニタしたりできます。

メッセージングブリッジのコンフィグレーションタスク

メッセージングブリッジをデプロイできるようになるには、先に必要なコンポーネントをコンフィグレーションする必要があります。

- 10-3 ページの「ブリッジのリソースアダプタについて」
- 10-6 ページの「ブリッジのリソースアダプタのデプロイ」
- 10-7 ページの「ソースと対象のブリッジ送り先のコンフィグレーション」
- 10-13 ページの「メッセージングブリッジインスタンスのコンフィグレーション」

ブリッジのリソースアダプタについて

メッセージングブリッジでは、コンフィグレーションされたソース JMS 送り先および対象 JMS 送り先と通信するためにリソースアダプタを使用します。ブリッジがソース JMS 送り先および対象 JMS 送り先と通信できるようにするには、サポートされているアダプタをそれぞれの送り先に関連付ける必要があります。アダプタの JNDI 名は、アダプタのデプロイメント記述子の一部としてコンフィグレーションされます。

注意： WebLogic JMS には、JMS 以外のメッセージング製品にアクセスするために「一般ブリッジ送り先」というフレームワークが用意されていますが、JMS 以外のメッセージング製品をサポートするアダプタは用意されていません。したがって、サードパーティ OEM ベンダからカスタムアダプタを入手し、そのマニュアルに従ってアダプタをコンフィグレーションする必要があります。カスタムアダプタの入手方法については、BEA プロフェッショナル サービスまでお問い合わせください。

10 WebLogic メッセージングブリッジの使い方

サポートされているアダプタは、`WL_HOME\server\lib` ディレクトリにあります。次の表では、これらのアダプタについて説明します。

表 10-1 メッセージングブリッジアダプタと JNDI 名

アダプタ	JNDI 名	説明
<code>jms-xa-adp.rar</code>	<code>eis.jms.WLSConnectionFactoryJNDIXA</code>	<p>XAResource によるトランザクションセマンティクスを提供する。必要な QOS が [かならず 1 回] の場合に使用する。受信されたメッセージをエンベロープして、ユーザトランザクション (XA/JTA) 内に送信する。このアダプタを使用するには、以下の要件を満たすこと。</p> <ul style="list-style-type: none">■ ブリッジングされている WebLogic Server のいずれかの実装がリリース 6.1 以降である。■ ソースおよび対象の JMS 接続ファクトリが、XAConnectionFactory を使用してコンフィグレーションされている。 <p>注意： このアダプタをデプロイする前に 10-20 ページの「メッセージングブリッジを使用する WebLogic Server のさまざまなリリースおよびドメインとの相互運用」を参照し、特定のトランザクションのコンフィグレーション要件およびガイドラインを確認すること。</p>

表 10-1 メッセージングブリッジアダプタと JNDI 名

アダプタ	JNDI 名	説明
jms-notran-adp.rar	eis.jms.WLSConnection FactoryJNDINoTX	<p>トランザクションセマンティクスは提供されない。必要な QOS が [最大 1 回] または [重複可] の場合に使用する。要求された QOS が [最大 1 回] の場合、アダプタでは AUTO_ACKNOWLEDGE モードを使用する。要求された QOS が [重複可] の場合は、CLIENT_ACKNOWLEDGE を使用する。</p> <p>注意: 非トランザクションセッションで使用する確認応答モードの詳細については、『WebLogic JMS プログラマーズ ガイド』の「WebLogic JMS の基礎」を参照。</p>
jms-notran-adp51.rar	eis.jms.WLS51Connection FactoryJNDINoTX	<p>ソース送り先または対象送り先が WebLogic Server 5.1 の場合に相互運用を実現する。このアダプタでは、トランザクションセマンティクスは提供されない。そのため、サポートされる QOS は [最大 1 回] または [重複可] に限定される。要求された QOS が [最大 1 回] の場合、アダプタでは AUTO_ACKNOWLEDGE モードを使用する。要求された QOS が [重複可] の場合は、CLIENT_ACKNOWLEDGE を使用する。</p>

Administration Console でソースブリッジ送り先および対象ブリッジ送り先をコンフィギュレーションする際は、適切なアダプタを JNDI 名で指定します。

ブリッジのリソースアダプタのデプロイ

メッセージングブリッジのコンポーネントをコンフィグレーションする前に、以下の方法のいずれかを使用して、メッセージングブリッジをホストする WebLogic Server ドメインに適切なリソースアダプタをデプロイします。

- Administration Console を使用する場合 — アダプタをデプロイするドメインを選択して [デプロイメント | アプリケーション] オプションを選択し、10-4 ページの表 10-1 「メッセージングブリッジアダプタと JNDI 名」の定義に従って適切な RAR アダプタファイルを選択します。
 - jms-xa-adp.rar
 - jms-notran-adp.rar
 - jms-notran-adp51.rar
- 自動デプロイメント機能を使用する場合 — この方法は、管理サーバ上のアプリケーションをすばやくデプロイする場合に使用します。アダプタを管理サーバのローカルの `\applications` ディレクトリにコピーすることにより、サーバが既に稼動していても自動的にデプロイされます。これ以外の方法では、WebLogic Server を再起動したときにアダプタがデプロイされます。自動デプロイメントは、単一サーバでの開発環境でアプリケーションをテストする場合にのみ使用します。プロダクションモードでの使用はお勧めできません。

注意： WebLogic Server リリース 7.0 およびリリース 5.1 の間で相互運用するようにメッセージングブリッジをコンフィグレーションする場合は、リリース 5.1 のリソースアダプタ (`jms-notran-adp51.rar`) と非トランザクションアダプタ (`jms-notran-adp.rar`) は、メッセージングブリッジを実行している 7.0 のドメイン上にデプロイする必要があります。

Administration Console でのデプロイメントの詳しい手順、および自動デプロイメント機能の使い方の詳細については、『WebLogic Server アプリケーションの開発』の「WebLogic Server デプロイメント」を参照してください。

ソースと対象のブリッジ送り先のコンフィグレーション

メッセージングブリッジは、実際にブリッジ送り先にマップされる2つの送り先を接続します。1つはメッセージの受信元になるソース送り先、もう1つはメッセージの送信先になる対象送り先です。ブリッジングするメッセージング製品に応じて、2種類のブリッジ送り先があります。

- **JMS** ブリッジ送り先 – **JMS** メッセージング製品 (WebLogic JMS 実装またはサードパーティの **JMS** プロバイダ) の場合、メッセージングブリッジにマップする実際のソース **JMS** 送り先および対象 **JMS** 送り先ごとに `JMSBridgeDestination` インスタンスをコンフィグレーションする必要があります。
- 一般ブリッジ送り先 – **JMS** 以外のメッセージング製品の場合、メッセージングブリッジにマップする実際のソース送り先および対象送り先ごとに汎用の `BridgeDestination` インスタンスをコンフィグレーションする必要があります。

この節の手順を実行する前に、10-20 ページの「メッセージングブリッジを使用する WebLogic Server のさまざまなリリースおよびドメインとの相互運用」を参照して特定のコンフィグレーションの要件およびガイドラインを確認してください。

JMS ブリッジ送り先のコンフィグレーション

`JMSBridgeDestination` インスタンスでは、WebLogic ドメイン内の実際の **JMS** キューまたはトピックの送り先のユニークな名前、指定した送り先との通信に使用するアダプタの名前、アダプタに渡すプロパティ情報 (接続 URL、接続ファクトリ **JNDI** 名など) を定義します。また、必要に応じてユーザ名とパスワードを定義します。

実際のソース **JMS** 送り先および対象 **JMS** 送り先の各々をメッセージングブリッジにマップするには、`JMSBridgeDestination` インスタンスをコンフィグレーションする必要があります。したがって、ソース (対象) **JMS** ブリッジ送り先の属性を定義したら、この手順を繰り返して対象 (ソース) **JMS** ブリッジ送り先を

コンフィグレーションします。ソースおよび対象の **JMS** ブリッジ送り先は、10-13 ページの「メッセージングブリッジインスタンスのコンフィグレーション」で指定します。

JMS ブリッジ送り先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** で、[メッセージングブリッジ] ノードをクリックします。
2. [**JMS** ブリッジ送り先] ノードをクリックして、右ペインの [ブリッジ送り先] タブを開きます。
3. 右ペインの [新しい **JMS Bridge Destination** のコンフィグレーション] リンクをクリックします。[コンフィグレーション] ダイアログに、新しい **JMS** ブリッジ送り先のコンフィグレーションに関連するタブが表示されます。
4. [コンフィグレーション] タブで属性を定義します。

次の表では、[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 10-2 JMS ブリッジ送り先の [コンフィグレーション] タブの属性

属性	説明
[名前]	ブリッジにマップする実際の JMS 送り先の JMS ブリッジ送り先名。この名前は、 WebLogic ドメイン全体にわたってユニークなものでなければならない。 たとえば、 WebLogic Server リリース 6.1 および 7.0 の間でブリッジングする場合、ソース送り先については、デフォルトのブリッジ送り先名を「70to61SourceDestination」に変更する。次に、対応する対象送り先を作成するときには、「0to61TargetDestination」という名前にする。ブリッジ送り先のコンフィグレーションが完了すると、これらの名前が [ブリッジ 一般] タブの [ソース送り先] および [対象送り先] 属性のリストに表示される。
[JNDI アダプタ名]	ブリッジ送り先との通信に使用されるアダプタの JNDI 名。どのアダプタ名を入力すべきかについては、10-4 ページの「メッセージングブリッジアダプタと JNDI 名」を参照。

表 10-2 JMS ブリッジ送り先の [コンフィグレーション] タブの属性

属性	説明
[アダプタ クラスパス]	<p>接続する送り先がバージョン 6.0 以前の WebLogic Server で動作している場合、古い WebLogic Server 実装のクラスの場所を示す CLASSPATH をブリッジ送り先に指定する必要がある。</p> <p>サードパーティの JMS プロバイダに接続する場合は、WebLogic Server の CLASSPATH でプロバイダの CLASSPATH をブリッジ送り先に指定する必要がある。</p>
[接続 URL]	<p>接続ファクトリおよび送り先のルックアップに使用する JNDI プロバイダの URL。</p>
[初期コンテキスト ファクトリ]	<p>JNDI コンテキストの取得に使用されるファクトリ。</p>
[接続ファクトリ JNDI 名]	<p>JMS ブリッジ送り先にマップされる実際の JMS 送り先に対する接続を作成するために使用される JMS 接続ファクトリ。</p> <p>注意: QOS として [かならず 1 回] を指定するには、接続ファクトリが XA 接続ファクトリでなければならない。接続ファクトリおよび QOS 要件の詳細については、10-14 ページの「メッセージングブリッジの [一般] タブの属性」を参照。</p>
[送り先 JNDI 名]	<p>JMS ブリッジ送り先にマップされている実際の JMS 送り先の JNDI 名。</p>
[送り先タイプ]	<p>キューまたはトピックの送り先タイプを選択する。</p>
[ユーザ名] および [ユーザ パスワード]	<p>メッセージングブリッジでブリッジアダプタに付与するユーザ名およびパスワード。</p> <p>注意: 指定の送り先に対して行う操作はすべて、このユーザ名およびパスワードを使用して行う。したがって、メッセージングブリッジを機能させるには、ソース送り先および対象送り先の [ユーザ名] および [ユーザ パスワード] に、基の JMS 送り先にアクセスするためのパーミッションがなければならない。</p>

5. [作成] をクリックして、JMS ブリッジ送り先を作成します。
6. ソース (対象) JMS ブリッジ送り先の属性を定義したら、この手順を繰り返して対象 (ソース) JMS ブリッジ送り先をコンフィグレーションします。

一般ブリッジ送り先のコンフィグレーション

一般的な `BridgeDestination` インスタンスでは、WebLogic ドメイン内の実際のキューまたはトピックの送り先のユニークな名前、指定した送り先との通信に使用するリソースアダプタの名前、アダプタに渡すプロパティのリストを定義します。また、必要に応じてユーザ名とパスワードを定義します。

注意： WebLogic JMS には、JMS 以外のメッセージング製品にアクセスするために「一般ブリッジ送り先」というフレームワークが用意されていますが、JMS 以外のメッセージング製品をサポートするアダプタは用意されていません。したがって、サードパーティ OEM ベンダからカスタムアダプタを入手し、そのマニュアルに従ってアダプタをコンフィグレーションする必要があります。カスタムアダプタの入手方法については、BEA プロフェッショナル サービスまでお問い合わせください。

実際のソース送り先および対象送り先の各々をメッセージングブリッジにマップするには、`BridgeDestination` インスタンスをコンフィグレーションする必要があります。ソースおよび対象の一般的なブリッジ送り先は、10-13 ページの「メッセージングブリッジインスタンスのコンフィグレーション」で指定します。

一般ブリッジ送り先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** で、[メッセージングブリッジ] ノードをクリックします。
2. [一般ブリッジ送り先] ノードをクリックして、右ペインの [ブリッジ送り先] タブを開きます。
3. 右ペインで [新しい **General Bridge Destination** のコンフィグレーション] リンクをクリックします。[コンフィグレーション] ダイアログに、新しい一般ブリッジ送り先のコンフィグレーションに関連するタブが表示されます。
4. [コンフィグレーション] タブで属性を定義します。

次の表では、[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 10-3 一般ブリッジ送り先の [コンフィグレーション] タブの属性

属性	説明
[名前]	<p>ブリッジにマップする実際の送り先のブリッジ送り先名。この名前は、WebLogic ドメイン全体にわたってユニークなものでなければならない。</p> <p>たとえば、WebLogic Server リリース 6.1 および 7.0 の間でブリッジングする場合、ソース送り先については、デフォルトのブリッジ送り先名を「70to61SourceDestination」に変更する。次に、対応する対象送り先を作成するときには、「Oto61TargetDestination」という名前にする。ブリッジ送り先のコンフィグレーションが完了すると、これらの名前が [ブリッジ 一般] タブの [ソース送り先] および [対象送り先] 属性のリストに表示される。</p>
[JNDI アダプタ名]	<p>ブリッジ送り先との通信に使用するアダプタの JNDI 名をブリッジ送り先に指定する必要がある。</p> <p>WebLogic Server では、JMS 以外のメッセージング製品用のアダプタは提供していない。サードパーティ OEM ベンダから入手したカスタム アダプタを使用する必要がある。入手方法については、BEA プロフェッショナル サービスでも情報を提供している。</p>
[アダプタ クラスパス]	<p>ブリッジ送り先の CLASSPATH を定義する。この属性は、バージョン 6.0 以前の WebLogic Server で動作している送り先への接続に主に使用する。</p> <p>サードパーティ製品に接続する場合は、WebLogic Server の CLASSPATH にプロバイダの CLASSPATH を指定する必要がある。</p>

表 10-3 一般ブリッジ送り先の [コンフィグレーション] タブの属性

属性	説明
[プロパティ]	<p>ブリッジ送り先に定義するプロパティをすべて指定する。各プロパティは、セミコロンで区切らなければならない (たとえば、<code>DestinationJNDIName=myTopic;DestinationType=topic;</code>)。</p> <p>JMS 以外のメッセージング製品の場合は、サードパーティ OEM ベンダからカスタム アダプタを入手し、そのマニュアルに従ってプロパティをコンフィグレーションする必要がある。</p> <p>以下のプロパティは、すべての JMS 実装で必要となる。</p> <p><code>ConnectionURL=</code> 送り先への接続の確立に使用される URL。</p> <p><code>InitialContextFactory=</code> JNDI コンテキストの取得に使用するファクトリ。</p> <p><code>ConnectionFactoryJNDIName=</code> JMS ブリッジ送り先にマップされる実際の JMS 送り先に対する接続を作成するために使用される JMS 接続ファクトリ。</p> <p><code>DestinationJNDIName=</code> JMS ブリッジ送り先にマップされる実際の JMS 送り先の JNDI 名。</p> <p><code>DestinationType=</code> キューまたはトピック。</p>
[ユーザ名] および [ユーザ パスワード]	<p>メッセージングブリッジでブリッジアダプタに付与するユーザ名およびパスワード。</p> <p>注意： 指定の送り先に対して行う操作はすべて、このユーザ名およびパスワードを使用して行う。したがって、メッセージングブリッジを機能させるには、ソースブリッジ送り先および対象ブリッジ送り先の [ユーザ名] および [ユーザ パスワード] に、基のソース送り先および対象送り先にアクセスするためのパーミッションがないなければならない。</p>

5. [作成] をクリックして、一般ブリッジ送り先を作成します。

6. ソース(対象)の一般ブリッジ送り先の属性を定義したら、この手順を繰り返して対象(ソース)の一般ブリッジ送り先をコンフィグレーションします。

メッセージングブリッジインスタンスのコンフィグレーション

メッセージングブリッジインスタンスは、コンフィグレーションされたソースブリッジ送り先および対象ブリッジ送り先と通信します。ソース送り先から対象送り先(それぞれ、別の WebLogic JMS 実装、サードパーティの JMS プロバイダ、または別の JMS 以外のメッセージング製品)へのマッピングごとに、Administration Console で MessagingBridge インスタンスをコンフィグレーションする必要があります。各 MessagingBridge インスタンスでは、マッピングのソース送り先と対象送り先、メッセージのフィルタリングセクタ、QOS、トランザクションセマンティクス、および各種の再接続パラメータを定義します。

この節の手順を実行する前に、10-20 ページの「メッセージングブリッジを使用する WebLogic Server のさまざまなリリースおよびドメインとの相互運用」または 10-26 ページの「メッセージングブリッジを使用するサードパーティのメッセージングプロバイダへのアクセス」を参照して特定のコンフィグレーションの要件およびガイドラインを確認してください。

メッセージングブリッジ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 で、[メッセージングブリッジ] ノードをクリックします。
2. [ブリッジ] ノードをクリックして、右ペインの [ブリッジ] タブを開きます。
3. 右ペインの [新しい Messaging Bridge のコンフィグレーション] リンクをクリックします。[コンフィグレーション] ダイアログに、新しいメッセージングブリッジのコンフィグレーションに関連するタブが表示されます。
4. [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 10-4 メッセージングブリッジの [一般] タブの属性

属性	説明
[名前]	WebLogic ドメイン全体でユニークなメッセージングブリッジの名前を入力する。
[ソース送り先]	メッセージングブリッジでメッセージの受信元になるソース送り先を選択する。たとえば、JMS メッセージングブリッジの場合は、[JMS ブリッジ送り先 コンフィグレーション] タブで、あらかじめ作成しておいた「JMS Source Bridge Destination」を選択する。
[対象送り先]	メッセージングブリッジでメッセージの送信先になる対象送り先を選択する。たとえば、JMS メッセージングブリッジの場合は、[JMS ブリッジ送り先 コンフィグレーション] タブで、あらかじめ作成しておいた「JMS Target Bridge Destination」を選択する。
[セレクト]	メッセージングブリッジで送信されたメッセージをフィルタ処理できるようにする。選択条件に一致するメッセージのみが、メッセージングブリッジを経由して送信される。キューの場合、選択条件に一致しないメッセージは後に残され、キュー内で蓄積される。トピックの場合、接続条件に一致しないメッセージは破棄される。 セレクトを使ったメッセージのフィルタ処理の詳細については、『WebLogic JMS プログラマーズガイド』の「WebLogic JMS アプリケーションの開発」を参照。

表 10-4 メッセージングブリッジの[一般]タブの属性

属性	説明
<p>[サービスの品質] (QOS)</p>	<p>メッセージングブリッジでのメッセージの転送で保証する QOS を選択する。有効なサービスの品質は次のとおり。</p> <p>[かならず 1 回]— 各メッセージはかならず 1 回送信される。これは、最高のサービスの品質である。この QOS を使用するには、以下の条件を満たす必要がある。</p> <ul style="list-style-type: none"> ■ WebLogic Server の実装がリリース 6.1 以降である。 ■ ソースおよび対象の JMS 接続ファクトリが、XAConnectionFactory を使用してコンフィギュレーションされている。 ■ ソース送り先と対象送り先の両方において、トランザクション jms-xa-adp.rar アダプタが [JNDI アダプタ名] 属性で「eis.jms.WLSConnectionFactoryJNDIXA」としてデプロイおよび識別されている。 <p>[最大 1 回]— 各メッセージは最大で 1 回送信される。一部のメッセージは、対象送り先に配信されない可能性がある。</p> <p>[重複可]— 各メッセージは最低 1 回送信される。重複メッセージが対象送り先に配信されるおそれがある。</p>
<p>[QOS デグラデーション]</p>	<p>このオプションを選択すると、要求された QOS が利用できない場合にメッセージングブリッジの QOS が自動的に低下する。その場合、メッセージは WebLogic の起動ウィンドウまたはログファイルに配信される。このオプションを選択せず (false)、要求された QOS をメッセージングブリッジが満たすことができない場合、エラーが発生し、メッセージングブリッジは起動しなくなる。</p>

表 10-4 メッセージングブリッジの [一般] タブの属性

属性	説明
[最大待機時間 (秒)]	非同期モードで動作しているブリッジの場合、接続状態のチェックが完了するまでメッセージングブリッジが待機する最大時間 (秒単位) を表す。同期モードで動作しているブリッジの場合、関与するトランザクションがないときに、メッセージングブリッジが受信呼び出しをブロックできる時間を表す。
[非同期モードを有効化]	メッセージングブリッジの非同期モードの有効 / 無効を定義する。非同期モード (true) で動作するメッセージングブリッジは、ソース送り先によって駆動される。メッセージングブリッジはメッセージをリスンし、到着したものを転送する。値を false に設定すると、ソースが非同期受信をサポートしていても、ブリッジは同期モードで機能する。 注意： QOS が [かならず 1 回] のメッセージングブリッジを非同期モードで機能させるには、ソース送り先が <code>MDBTransaction</code> インタフェース (<code>weblogic.jms.extensions</code> の Javadoc を参照) をサポートしていなければならない。ソース送り先が <code>MDBTransaction</code> をサポートしていないことが検出されると、ブリッジは自動的に同期モードに切り替わる。 <code>MDBTransaction</code> 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』の「メッセージ駆動型 Bean の設計」を参照。

表 10-4 メッセージングブリッジの[一般]タブの属性

属性	説明
[永続性を有効化]	<p>この属性は、JMS トピックまたは JMS トピックと同様の特性を備えた送り先のみ設定できる。永続性を有効にすることで、メッセージングブリッジによってソース送り先の恒久サブスクリプションが作成される。これにより、ブリッジが実行中でないときでも、送信されたメッセージをソース JMS 実装で保存できるようになる。保存されたメッセージは、ブリッジの再起動時に対象送り先に転送される。この属性を選択しないと、ブリッジが実行中でない間にソース JMS トピックに送信されたメッセージは、対象送り先に転送できなくなる。</p> <p>注意： ブリッジを永続的にオフライン状態にする必要がある場合は、そのブリッジを使用する恒久サブスクリプションをすべて削除しなければならない。恒久サブスクライバの削除の詳細については、『WebLogic JMS プログラマーズ ガイド』の「恒久サブスクリプションの削除」を参照。</p>
[起動する]	<p>メッセージングブリッジのコンフィグレーション時およびサーバ再起動時の初期状態を示す。またこのフィールドは、メッセージングブリッジを動的に起動および停止するのにも使用できる。ブリッジを停止するには、チェックボックスのチェックを解除する。逆に、チェックボックスを再び選択すると、ブリッジが再起動する。</p> <p>注意： メッセージングブリッジの起動を妨げるコンフィグレーション上の問題がなければ、このフィールドはメッセージングブリッジの予想される実行時状態を示す。ドメイン内でコンフィグレーションされているすべてのメッセージングブリッジのモニタについては、10-28 ページの「メッセージングブリッジのモニタ」を参照。</p>

5. [作成] をクリックして、メッセージングブリッジを作成します。

10 WebLogic メッセージングブリッジの使い方

6. [接続を再試行] タブをクリックし、次の表に従ってブリッジの再接続の間隔を定義します。

メッセージングブリッジのソース送り先と対象送り先は、常に利用可能とはかぎりません。これを前提に、メッセージングブリッジでは、一定の間隔で送り先に再接続できるようにしておく必要があります。以下の属性では、接続の再試行の間隔を設定します。

表 10-5 メッセージングブリッジの [接続を再試行] タブの属性

属性	説明
[最小遅延 (秒)]	接続を再試行する間の最小遅延 (秒単位)。メッセージングブリッジの起動時に送り先に接続できない場合、または接続に失敗してメッセージングブリッジが最初に再接続を試行する場合に、この属性で指定した秒数の間隔で接続が再試行される。
[増加遅延 (秒)]	接続を再試行する間に増加される遅延 (秒単位)。ブリッジが再接続に失敗するたびに、この属性で指定した秒数が遅延に追加され、次の再接続が試行される。
[最大遅延 (秒)]	接続を再試行する間の最大遅延 (秒単位)。再接続を試行するたびに、[増加遅延] で指定した秒数が追加されて遅延するが、この属性で指定した値より長く遅延することはない。

7. [適用] をクリックして、新しい属性値を保存します。
8. [トランザクション] タブをクリックし、次の表に従ってメッセージングブリッジのトランザクション属性を定義します。

表 10-6 メッセージングブリッジの [トランザクション] タブの属性

属性	説明
[トランザクション タイムアウト]	各トランザクションがタイムアウトになるまで、トランザクション マネージャが待機する秒数を定義する。トランザクション タイムアウトは、ブリッジのサービスの品質に 2 フェーズ トランザクションが必要な場合に指定する。

表 10-6 メッセージングブリッジの [トランザクション] タブの属性

属性	説明
[バッチ サイズ]	1回のトランザクションにつきメッセージングブリッジで転送するメッセージ数を定義する。バッチサイズは、同期モードで機能し、サービスの品質に2フェーズトランザクションが必要なブリッジにのみ適用する。
[バッチ間隔 (ミリ秒)]	[バッチ サイズ] で指定したメッセージ数に達したかどうかに関係なく、1回のトランザクションでメッセージをまとめて送信するまで、ブリッジが待機する最大時間 (ミリ秒単位) を定義する。デフォルト値の -1 は、トランザクションが完了する前に、メッセージ数がバッチサイズに達するまで、ブリッジが待機することを示す。 バッチ間隔は、同期モードで機能し、サービスの品質に2フェーズトランザクションが必要なブリッジにのみ適用する。

9. [適用] をクリックして、新しい属性値を保存します。
10. [対象] タブをクリックし、次の表に従ってメッセージングブリッジに関連付ける WebLogic Server インスタンスを割り当てます。

表 10-7 メッセージングブリッジの [対象] タブの属性

属性	説明
[移行できる対象]	メッセージングブリッジがデプロイされる WebLogic Server の移行可能な対象を定義する。WebLogic Server を起動した際に、初期状態でメッセージングブリッジを使用できるのはユーザ指定のサーバのみである。ただし、それ以降は Administration Console またはコマンドラインツールを使用して、移行可能な対象にある別のサーバにブリッジを移行することができる。 詳細については、『WebLogic Server クラスタ ユーザーズガイド』の「固定サービスの移行」を参照。

表 10-7 メッセージングブリッジの [対象] タブの属性

属性	説明
[クラスタ]	メッセージングブリッジがデプロイされる WebLogic Server クラスタを定義する。メッセージングブリッジは、選択したクラスタ内のすべてのサーバで使用できる。
[サーバ]	メッセージングブリッジがデプロイされる WebLogic Server を定義する。メッセージングブリッジは、選択したすべての WebLogic Server で使用できる。

11. [適用] をクリックして、新しい属性値を保存します。

メッセージングブリッジを使用しての WebLogic Server のさまざまなリリースおよびドメインとの相互運用

以下の相互運用性のガイドラインは、メッセージングブリッジを使用して、WebLogic Server のさまざまなリリースや他の WebLogic Server ドメインにおける JMS 送り先にアクセスする場合に適用されます。

- 10-21 ページの「WebLogic Server およびドメインの名前を付ける際のガイドライン」
- 10-21 ページの「WebLogic ドメインのセキュリティ相互運用性の有効化」
- 10-22 ページの「メッセージングブリッジを使用してのリリース 6.1 以降のドメインにおける送り先へのアクセス」
- 10-24 ページの「メッセージングブリッジを使用してのリリース 6.0 のドメインにおける送り先へのアクセス」
- 10-25 ページの「メッセージングブリッジを使用してのリリース 5.1 のドメインにおける送り先へのアクセス」

注意: メッセージングブリッジを使用して、リリースの異なる WebLogic Server を実行する 2 つのドメイン間で通信を行う場合、メッセージングブリッジを、WebLogic Server の最新リリースを使用するドメイン上で実行するようにコンフィグレーションすることを最良の方法として推奨します。

WebLogic Server およびドメインの名前を付ける際のガイドライン

複数のドメインが関与している場合、すべての WebLogic Server デプロイメントに、ユニークな名前を付けるというルールが適用されます。したがって、以下のことを確認してください。

- WebLogic Server インスタンスおよびドメイン名がユニークであること。
- WebLogic JMS サーバ名がドメイン間にわたってユニークであること。
- クラスタ内のサーバを対象とするすべての JMS 接続ファクトリの名前がユニークであること。
- 永続メッセージに対して JMS ファイルストアが使用されている場合は、その JMS ファイルストア名はドメイン間にわたってユニークなものとする必要がある。

WebLogic ドメインのセキュリティ相互運用性の有効化

メッセージングブリッジ間で送信される 2 フェーズトランザクションの処理に「かならず 1 回」の QOS (サービスの品質) が必要である場合以外は、ブリッジを 7.0 またはそれ以降のリリースの 2 つのドメイン間で相互運用するための特別なセキュリティコンフィグレーション要件はありません。

ただし、リリース 7.0 のドメイン上で実行されているブリッジが、6.1 またはそれ以降のリリースの 2 つのドメイン間で (「かならず 1 回」の QOS を使用して) トランザクションメッセージを処理しなければならない場合には、以下の手順に従う必要があります。

1. WebLogic Server 7.0 での相互運用性のセキュリティを次のようにコンフィグレーションします。
 - a. ドメイン ノード (examples など) を展開します。
 - b. [セキュリティ | 詳細設定] タブを選択します。
 - c. 必要に応じて、[生成された資格を有効化] チェック ボックスのチェックを解除します。
 - d. [資格 : 変更 ...] 属性をクリックして、[資格を変更する] ウィンドウを開きます。
 - e. [新しい資格] フィールドで、ドメインのパスワードを指定します。このパスワードは、相互運用するドメインで使用しているパスワードと一致している必要があります。
 - f. パスワードを [再入力] フィールドに再入力して確定します。
 - g. [適用] をクリックします。
2. リリース 6.1 のドメインと相互運用する場合、リリース 7.0 のドメインの資格のパスワードは、6.1 ドメイン用にコンフィグレーションした「システム」ユーザパスワードに完全に一致している必要があります。また、「システム」ユーザが 7.0 ドメインの管理者グループのメンバーであることを確認してください。

注意： バージョン 7.0 ドメインの相互運用性のセキュリティについては、『WebLogic Security の管理』の「WebLogic ドメイン間の信頼関係の有効化」を参照してください。相互運用性のセキュリティの詳細については、『WebLogic Security の管理』の「互換性セキュリティの使い方」を参照してください。

メッセージングブリッジを使用してのリリース 6.1 以降のドメインにおける送り先へのアクセス

リリース 6.1 以降の 2 つのドメイン間での「かならず 1 回」のトランザクションメッセージ通信を提供するためにリリース 7.0 ドメイン上でメッセージングブリッジをコンフィグレーションする場合は、以下のガイドラインを使用します。

注意： 2 フェーズ トランザクションのための「かならず 1 回」のサービスの品質は、リリース 6.1 以降でしかサポートされません。

- 永続メッセージに対して JMS ファイル ストアが使用されている場合は、その JMS ファイル ストア名は WebLogic ドメイン間にわたってユニークなものとする必要があります (10-21 ページの「WebLogic Server およびドメインの名前を付ける際のガイドライン」を参照)。
- ドメイン間のセキュリティの相互運用性が正しくコンフィグレーションされていることを確認します (10-21 ページの「WebLogic ドメインのセキュリティ相互運用性の有効化」を参照)。
- XA 接続ファクトリが両方のドメインで有効になっていることを確認します。有効にするには、[サービス | JMS | 接続ファクトリ | コンフィグレーション | トランザクション] タブの [XA コネクション ファクトリを有効化] チェック ボックスを選択します。
- 7.0 ブリッジドメイン上のトランザクション リソースアダプタ `jms-xa-adp.rar` をデプロイします (10-6 ページの「ブリッジのリソースアダプタのデプロイ」を参照)。
- JMS ブリッジ送り先をコンフィグレーションするには、ソース送り先と対象送り先の双方に対して以下の操作を行います (10-7 ページの「JMS ブリッジ送り先のコンフィグレーション」を参照)。
 - [JNDI アダプタ名] フィールドで、トランザクションアダプタの JNDI 名 `eis.jms.WLSConnectionFactoryJNDIXA` を指定する。
 - [アダプタ クラスパス] フィールドには、何も入力しないこと。
- [メッセージングブリッジ | コンフィグレーション | 一般] タブで、[かならず 1 回] のサービスの品質を選択します (10-13 ページの「メッセージングブリッジインスタンスのコンフィグレーション」を参照)。

メッセージングブリッジを使用するのリリース 6.0 のドメインにおける送り先へのアクセス

WebLogic Server 7.0 とリリース 6.0 のドメインの間の相互運用が必要なメッセージングブリッジをコンフィグレーションする際は、ブリッジが実行されるリリース 7.0 のドメインで次の項目のコンフィグレーションを行う必要があります。

注意： WebLogic Server 6.0 では、「かならず 1 回」の QOS はサポートされていません。ブリッジの QOS オプションの詳細については、10-14 ページの「メッセージングブリッジの [一般] タブの属性」を参照してください。

- 7.0 ブリッジドメイン上のトランザクションリソースアダプタ `jms-notran-adp.rar` をデプロイします (10-6 ページの「ブリッジのリソースアダプタのデプロイ」を参照)。
- JMS ソース送り先および JMS 対象送り先をコンフィグレーションする際には、以下の処理を実行します (10-7 ページの「JMS ブリッジ送り先のコンフィグレーション」を参照)。

[JNDI アダプタ名] フィールド：

- ソース送り先および対象送り先について、非トランザクションアダプタの JNDI 名を `eis.jms.WLSConnectionFactoryJNDINoTX` と指定する。

[アダプタ クラスパス] フィールド：

- 7.0 の送り先については、フィールドを空白のままにする。
- 6.0 の送り先については、WebLogic Server 6.0 リリースのクラスの場所を示す。

たとえば、WebLogic Server 6.0 GA が `WL60_HOME` というディレクトリにインストールされている場合、6.0 の JMS ブリッジ送り先について、[アダプタ クラスパス] を次のように設定します。

```
WL60_HOME\lib\weblogic60.jar
```

- [メッセージングブリッジ | コンフィグレーション | 一般] タブで、[最大 1 回] または [重複可] のサービスの品質を選択します (10-13 ページの「メッセージングブリッジインスタンスのコンフィグレーション」を参照)。

メッセージングブリッジを使用するリリース 5.1 のドメインにおける送り先へのアクセス

WebLogic Server 7.0 とリリース 5.1 の間の相互運用が必要なメッセージングブリッジをコンフィグレーションする際は、ブリッジが実行されるリリース 7.0 のドメインで次のコンフィグレーションを行う必要があります。

注意： WebLogic Server 5.1 では、「かならず 1 回」の QOS はサポートされていません。ブリッジの QOS オプションの詳細については、10-14 ページの「メッセージングブリッジの [一般] タブの属性」を参照してください。

- `WL_HOME\lib` ディレクトリにある `jms51-interop.jar` ファイルは、WebLogic Server 7.0 実装の `CLASSPATH` で指定します。
- リリース 5.1 のリソースアダプタ (`jms-notran-adp51.rar`) および非トランザクションアダプタ (`jms-notran-adp.rar`) は、7.0 のブリッジドメイン上にデプロイする必要があります (10-6 ページの「ブリッジのリソースアダプタのデプロイ」を参照)。
- JMS ソース送り先および JMS 対象送り先をコンフィグレーションする際には、以下の処理を実行します (10-7 ページの「JMS ブリッジ送り先のコンフィグレーション」を参照)。

[JNDI アダプタ名] フィールド:

- 7.0 の送り先について、非トランザクションアダプタの JNDI 名を `eis.jms.WLSConnectionFactoryJNDINoTX` と指定する。
- 5.1 の送り先について、5.1 のアダプタの JNDI 名を `eis.jms.WLS51ConnectionFactoryJNDINoTX` と指定する。

[アダプタ クラスパス] フィールド:

- 7.0 の送り先については、フィールドを空白のままにする。
- 5.1 の送り先については、WebLogic Server 5.1 リリースのクラスの場合を、7.0 リリースの `jms51-interop.jar` ファイルの場所と共に示す。

たとえば、WebLogic Server 5.1 GA が `WL51_HOME` というディレクトリにインストールされ、WebLogic Server 7.0 リリースが `WL71_HOME` にインストールされている場合、5.1 については [アダプタ クラスパス] を次のように設定します。

```
WL51_HOME\classes;WL51_HOME\lib\weblogicaux.jar;  
WL70_HOME\server\lib\jms51-interop.jar
```

注意： 実装で 5.1 Service Pack を使用している場合は、対応する *sp.jar* ファイルも [アダプタ クラスパス] フィールドに追加する必要があります。

- [メッセージングブリッジ | コンフィグレーション | 一般] タブで、[最大 1 回] または [重複可] のサービスの品質を選択します (10-13 ページの「メッセージングブリッジインスタンスのコンフィグレーション」を参照)。

メッセージングブリッジを使用してのサードパーティのメッセージングプロバイダへのアクセス

サードパーティメッセージングプロバイダとの相互運用が必要なメッセージングブリッジをコンフィグレーションする際は、次のようにコンフィグレーションする必要があります。

- WebLogic Server を起動する前に、次のコンフィグレーションを行います。
 - プロバイダの CLASSPATH を WebLogic Server の CLASSPATH で指定します。
 - プロバイダのクライアント側のライブラリに必要なネイティブコードの PATH を、WebLogic Server システムの PATH に追加します (この変数は、使用しているオペレーティングシステムによって異なります)。
- ブリッジングするサードパーティメッセージング製品の JMSBridgeDestination インスタンスに、ベンダ固有の情報を次の属性で指定します。
 - [接続 URL]
 - [初期コンテキストファクトリ]
 - [接続ファクトリ JNDI 名]
 - [送り先 JNDI 名]

注意：メッセージングブリッジでは、ソース送り先と対象送り先が同じリソースマネージャにあるとき（つまり、ブリッジがリソースマネージャのXAリソースを使用してグローバルトランザクションを転送しているとき）は、「かならず1回」のサービスの品質は提供できません。たとえば、MQ Seriesを使用しているときは、ソース送り先および対象送り先に同じキューマネージャを使用することはできません。

JMSブリッジ送り先のその他の属性のコンフィグレーションの詳細については、10-7 ページの「JMSブリッジ送り先のコンフィグレーション」を参照してください。

メッセージングブリッジの管理

作成済みおよび実行中のメッセージングブリッジは、Administration Console を使用して管理できます。

- メッセージングブリッジの停止と再起動
- メッセージングブリッジのモニタ
- 実行スレッドプールサイズのコンフィグレーション

メッセージングブリッジの停止と再起動

アクティブなメッセージングブリッジを一時的にサスペンドして再起動するには、次の手順に従います。

1. [メッセージングブリッジ] ノードを展開し、[ブリッジ] ノードを展開します。
2. 停止するメッセージングブリッジインスタンスを選択します。
3. [コンフィグレーション | 一般] タブで [起動する] チェックボックスのチェックをはずすと、ブリッジが停止します。
4. ブリッジを再起動するには、もう一度 [起動する] チェックボックスを選択します。

メッセージングブリッジのモニタ

Administration Console を使用して、ドメイン内のすべてのメッセージングブリッジの状態をモニタできます。

1. [サーバ] ノードを展開し、メッセージングブリッジ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選択します。
2. 選択したサーバインスタンスと関連付けられているタブを示すダイアログが右ペインに表示されます。
3. [サービス] タブを選択します。
4. [ブリッジ] タブを選択します。
5. [すべてのメッセージブリッジランタイムのモニタ] テキストリンクをクリックすると、モニタデータが表示されます。
6. 選択したサーバのすべてのメッセージングブリッジインスタンスとその状態(稼働しているかどうか)を示すテーブルが表示されます。

実行スレッドプールサイズのコンフィグレーション

メッセージングブリッジのデフォルトの実行スレッドプールサイズは、Administration Console を使用してコンフィグレーションできます。たとえば、デフォルトのサイズを増やすことにより、WebLogic Server のデフォルトスレッドプールでの競合を減らすことができます。値に -1 を入力した場合、このスレッドプールは無効になり、WebLogic Server のデフォルトスレッドプールがメッセージングブリッジで使用されます。

1. [サーバ] ノードを展開します。
2. メッセージングブリッジ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選択します。選択したサーバインスタンスと関連付けられているタブを示すダイアログが右ペインに表示されます。
3. [サービス] タブを選択します。

4. [ブリッジ]タブを選択します。
5. [メッセージングブリッジスレッドのプールサイズ]フィールドに新しい値を入力します。
6. [適用]をクリックして変更を保存します。

実行スレッドのチューニングの詳細については、『BEA WebLogic Server パフォーマンスチューニングガイド』の「WebLogic Server アプリケーションのチューニング」を参照してください。

11 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。

- 11-1 ページの「JNDI 管理の概要」
- 11-2 ページの「JNDI ツリーの表示」
- 11-2 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、『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 11-1 「JNDI ツリーのオブジェクト」を参照してください。

表 11-1 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	EJB デプロイメント記述子エディタ
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
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

12 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 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 コネクタの概要
- デプロイメント用のリソースアダプタ (コネクタ) のコンフィグレーション
- リソースアダプタ (コネクタ) のデプロイメント
- デプロイされたリソースアダプタ (コネクタ) の表示
- デプロイされたリソースアダプタ (コネクタ) のアンデプロイメント
- デプロイされたリソースアダプタ (コネクタ) の更新
- 接続のモニタ
- 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 コネクタの詳細については、『WebLogic J2EE コネクタ』を参照してください。

WebLogic J2EE コネクタの概要

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。

デプロイメント用のリソース アダプタ (コネクタ) のコンフィグレーション

WebLogic Server Administration Console を使用してコネクタ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. WebLogic Server の Administration Console を起動します。
2. 作業を行うドメインを選択します。
3. Console の左ペインで [デプロイメント] を選択します。

4. **Console** の左ペインでコネクタを選択すると、デプロイ済みのコネクタがすべて右ペインの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] オプションを選択します。
6. コンフィグレーションするアーカイブ ファイル (**RAR**) を検索します。
注意： 展開されたアプリケーションまたはコンポーネント ディレクトリもコンフィグレーションできます。ただし、**WebLogic Server** では、指定したディレクトリ以下の階層で検索されたすべてのコンポーネントがデプロイされます。
7. ディレクトリまたはファイルの左側をクリックして選択し、次の手順に進みます。
8. 選択可能なサーバの中から対象サーバを選択します。
9. 表示されたフィールドにコネクタの名前を入力します。
10. [コンフィグレーションとデプロイ] をクリックします。[デプロイ] パネルが表示されます。このパネルには、コネクタのデプロイメント状況およびデプロイメント アクティビティが表示されます。
11. 選択可能なタブで次の情報を入力します。
 - [コンフィグレーション]— ステージング モードを編集し、デプロイ順を入力します。
 - [対象]— [選択可] リストから [選択済み] リストにサーバを移動して、コンフィグレーションされているコネクタの対象サーバを指定します。
 - [デプロイ]— すべての対象または選択した対象にコネクタをデプロイするか、またはすべての対象または選択した対象からコネクタをアンデプロイします。
 - [モニタ]— コネクタのセッション モニタを有効にします。
 - [メモ]— コネクタに関連するメモを入力します。

接続プロファイルを表示するためのコネクタのコンフィグレーション

Administration Console では、コネクタのコール スタック (接続プロファイル) だけでなく、リークされた接続およびアイドル接続のコール スタックも表示することができます。この情報を Administration Console で表示できるようにコンフィグレーションするには、次の操作を行います。

1. コネクタをデプロイした後、Administration Console の左ペインでコネクタを右クリックして、[コネクタ記述子の編集] を選択します。
2. 左ペインで [weblogic-connection-factory-dd] を展開し、[map-config-property] をクリックします。
3. 右ペインで [新しい map-config-property のコンフィグレーション] をクリックします。
4. [新しい map-config-property のコンフィグレーション] の [description] ボックスに、「Connection Profiling」などの説明を入力します。
5. [map-config-property-name] ボックスに、connection-profiling-enabled を入力します。
6. [map-config-property-value] ボックスに true を入力します。
7. [適用] をクリックして、変更を保存します。

プロファイル情報の表示の詳細については、12-7 ページの「接続のモニタ」を参照してください。

リソース アダプタ (コネクタ) のデプロイメント

WebLogic Server Administration Console を使用してコネクタをデプロイするには、次の操作を行います。

1. 左ペインの [デプロイメント] ノードを展開します。
2. [コネクタ] ノードを右クリックします。
3. [新しい Application のコンフィグレーション] を選択します。
4. コンフィグレーションするアーカイブ ファイル (RAR) を検索します。
注意: 展開されたアプリケーションまたはコンポーネント ディレクトリもコンフィグレーションできます。ただし、WebLogic Server では、指定したディレクトリ以下の階層で検索されたすべてのコンポーネントがデプロイされます。
5. ディレクトリまたはファイルの左側をクリックして選択し、次の手順に進みます。
6. 選択可能なサーバの中から対象サーバを選択します。
7. 表示されたフィールドにコネクタの名前を入力します。
8. [コンフィグレーションとデプロイ] をクリックします。[デプロイ] パネルが表示されます。このパネルには、コネクタのデプロイメント状況およびデプロイメント アクティビティが表示されます。
9. [デプロイ] ボタンをクリックして、すべての対象または選択した対象にコネクタをデプロイするか、またはすべての対象または選択した対象からコネクタをアンデプロイします。
10. Web ブラウザでリソースにアクセスしてコネクタをテストします。リソースへのアクセスは、次のように構成された URL で行います。

`http://myServer:myPort/myConnector/resource`

各値の説明は次のとおりです。

- myServer は、WebLogic Server のホスト マシンの名前
- myPort は、WebLogic Server がリクエストをリスンするポートの番号
- myConnector は、コネクタのアーカイブ ファイルの名前 (たとえば、myConnector.rar)、またはコネクタが入っているディレクトリの名前
- resource は、JSP、HTTP サーブレット、HTML ページなどのリソースの名前

デプロイされたリソース アダプタ (コネクタ) 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. Console で [デプロイメント] をクリックします。
2. [コネクタ] オプションをクリックします。
3. デプロイされたコネクタのリストが Console のテーブルに表示されます。

デプロイされたリソース アダプタ (コネクタ) 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. Console で [デプロイメント] をクリックします。
2. [コネクタ] オプションをクリックします。
3. 表示されたテーブルでアンデプロイするコネクタの名前をクリックします。
4. [適用] をクリックします。

デプロイされたリソース アダプタ (コネクタ) の更新

デプロイされたコネクタを更新するには、次の操作を行います。

1. **Console** で [デプロイメント] をクリックします。
2. [コネクタ] オプションをクリックします。
3. 表示されたテーブルで更新するコネクタの名前をクリックします。
4. [デプロイ] タブでデプロイメント状況を更新します。
5. [適用] をクリックします。

接続のモニタ

BEA J2EE コネクタアーキテクチャでは、**WebLogic Server Console** にモニタ機能を備えています。この機能では、検出されたリークを表示することができ、スタックをルックアップしてリークの原因となったアプリケーションを特定することができます。**Console** の [削除] ボタンをクリックすると、識別済みのリークされた接続を動的にクローズできます。接続を削除するオプションは、接続が指定のアイドル時間を超え、削除しても安全である（つまり、接続がトランザクションに関与していない）場合にのみ使用できます。

各接続が割り当てられているコールスタックを接続プールで格納するかどうかは、`weblogic-ra.xml` ファイルの `connection-profiling-enabled` 要素で指定します。この要素を `true` に設定した場合、**Console** でアクティブな接続の情報を表示できます。また、リークされた接続およびアイドル接続のスタックを表示したり、接続をクローズできなかったコンポーネントをデバッグしたりすることもできます。

はじめに

Console を使用してモニタ ツールにアクセスするには、2つの方法があります。

方法 1

1. **Console** の左ペインで [デプロイメント | コネクタ] を選択して、コネクタのリストを表示します。

2. コネクタを右クリックし、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。

選択したコネクタに関する接続プールの接続情報が右ペインに表示されます。

方法 2

1. **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。コネクタのテーブルが表示されます。
2. [名前] カラムで、モニタするコネクタをクリックします。
3. [モニタ] タブで [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。

選択したコネクタに関する接続プールの接続情報が右ペインに表示されます。

リークされた接続の表示

Console の [接続リーク プロファイル] カラムには、リークされた接続に関するプロファイル情報を表示することができます。このカラムと [Leaked Connections Detected] カラムを混同しないでください。後者のカラムには、リークされた接続数だけが表示されます。

この 2 つのカラムの本質的な違いは、[接続リーク プロファイル] カラムが、`weblogic-ra.xml` ファイルで設定する `connection-profiling-enabled` によって制御される点です。デフォルトでは、これは `false` に設定されています。したがって、[接続リーク プロファイル] カラムは通常、ゼロ (無効) になっています。ただし、[Leaked Connections Detected] カラムは常に有効になっているので、リークされた接続数は常に表示されます。

Console を使用してリークされた接続を表示するには、2 つの方法があります。

方法 1

1. **Console** の左ペインで [デプロイメント | コネクタ] を選択して、コネクタのリストを表示します。

2. コネクタを右クリックし、ポップアップメニューから [リークされた接続を表示] を選択します。

選択したコネクタに関する接続プールの接続情報が右ペインに表示されません。

3. [接続リークプロファイル] カラムで、選択したコネクタに関するリークされた接続の数をクリックします。

リークされた接続の情報が、右ペインに表示されます。

方法 2

1. **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。

コネクタのテーブルが表示されます。

2. [名前] カラムで、モニタするコネクタの名前をクリックします。

3. [モニタ] タブで [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。

選択したコネクタに関する接続プールの接続情報が右ペインに表示されません。

4. [接続リークプロファイル] カラムで、選択したコネクタに関するリークされた接続の数をクリックします。

リークされた接続の情報が、右ペインに表示されます。

アイドル接続の表示

Console の [アイドル接続プロファイル] カラムには、アイドル接続に関するプロファイル情報を表示することができます。このカラムと [Idle Connections Detected] カラムを混同しないでください。後者のカラムには、アイドル接続数だけが表示されます。

この 2 つのカラムの本質的な違いは、[アイドル接続プロファイル] カラムが、`weblogic-ra.xml` ファイルで設定する `connection-profiling-enabled` によって制御される点です。デフォルトでは、これは `false` に設定されています。したがって、[アイドル接続プロファイル] カラムは通常、ゼロ (無効) になっています。ただし、[Idle Connections Detected] カラムは常に有効になっているので、アイドル接続数は常に表示されます。

Console を使用してアイドル接続を表示するには、2つの方法があります。

方法 1

1. Console の左ペインで [デプロイメント | コネクタ] を選択して、コネクタのリストを表示します。
2. コネクタを右クリックし、ポップアップ メニューから [アイドル接続を表示] を選択します。

選択したコネクタに関する接続プールの接続情報が右ペインに表示されます。

3. [アイドル接続プロファイル] カラムで、選択したコネクタに関するアイドル接続の数をクリックします。

アイドル接続の情報が、右ペインに表示されます。

方法 2

1. 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。
コネクタのテーブルが表示されます。
2. [名前] カラムで、モニタするコネクタの名前をクリックします。
3. [モニタ] タブで [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。

選択したコネクタに関する接続プールの接続情報が右ペインに表示されます。

4. [アイドル接続プロファイル] カラムで、選択したコネクタに関するアイドル接続の数をクリックします。

アイドル接続の情報が、右ペインに表示されます。

接続の削除

現在、この機能は WebLogic Server Administration Console に実装されていません。

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**』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。
 - [connector] 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-connection-factory-dd] 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。

13 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 60 日です。すぐに WebLogic Server の使用を開始できます。60 日間の評価期間を過ぎても WebLogic Server を使用する場合は、WebLogic Server を使用するサーバ (マシン) 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お問い合わせ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 5 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。
- 60 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. **JDK (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=.\jdk131\bin;%PATH%` (Windows システム)
- `set PATH=./jdk131/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
- CertGen
- Conversion
- der2pem
- dbping
- Deployer
- EJBGen
- getProperty
- ImportPrivateKey
- logToZip
- MulticastTest
- myip
- pem2der
- Schema
- showLicenses

- system
- 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
<i>URL</i>	アプレットの URL
<i>filename</i>	.jar/.cab アーカイブの送り先であるローカルファイル名

CertGen

CertGen ユーティリティは、プロダクション環境用ではなくデモまたはテスト目的専用の証明書を生成します。

構文

```
$ java utils.CertGen password certfile keyfile [export]
```

引数	定義
<i>password</i>	プライベート キーのパスワードを定義する。
<i>certfile</i>	生成された証明書ファイルをコピーするディレクトリを定義する。
<i>keyfile</i>	生成されたプライベート キー ファイルをコピーするディレクトリを定義する。
<i>export</i>	デフォルトでは、CertGen ユーティリティは国内向けレベルの証明書を生成する。ツールで国外向けレベルの証明書を生成する場合は、[export] オプションを指定する。

例

証明書を生成するには、以下の操作を行います。

1. CertGen ツールを実行しているディレクトリに、次のファイルをコピーします。
 - WL_HOME/server/lib/CertgenCA.der—WebLogic Server によって信頼された認証局の証明書
 - WL_HOME/server/lib/CertGenCAKey.der—WebLogic Server によって信頼された認証局のプライベート キー
2. 次のコマンドを入力すると、testcert という名前の証明書ファイルと testkey という名前のプライベート キー ファイルが生成されます。

```
$ java utils.CertGen mykeypass testcert testkey  
Creating Domestic Key Strength - 1024
```

```
Encoding
.....
.....
.....
Created Private Key files - testkey.der and testkey.pem
Encoding
.....
.....
.....
Created Certificate files - testcert.der and testcert.pem
.....
```

ClientDeployer

J2EE EAR ファイルからクライアントサイド JAR ファイルを展開して、デプロイ可能な JAR ファイルを作成するには、`weblogic.ClientDeployer` を使用します。`weblogic.ClientDeployer` クラスは、Java コマンドラインで次の構文を使用して実行します。

```
java weblogic.ClientDeployer ear-file client
```

ear-file 引数は、1 つまたは複数のクライアントアプリケーション JAR ファイルが格納されている展開されたディレクトリか、または拡張子 `.ear` を持つ Java アーカイブ ファイルです。

次に例を示します。

```
java weblogic.ClientDeployer app.ear myclient
```

ここで、`app.ear` は、`myclient.jar` にパッケージ化された J2EE クライアントを格納する EAR ファイルです。

EAR ファイルからクライアントサイドの JAR ファイルが展開されたら、`weblogic.j2eeclient.Main` ユーティリティを使用してクライアントサイドアプリケーションをブートストラップし、次のように **WebLogic Server** インスタンスを示すようにします。

```
java weblogic.j2eeclient.Main clientjar URL [application args]
```

次に例を示します。

```
java weblogic.j2eeclient.Main helloWorld.jar
t3://localhost:7001 Greetings
```

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
<i>derFile</i>	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。
<i>headerFile</i>	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。 <ul style="list-style-type: none">■ "-----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。
<i>footerFile</i>	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。 <ul style="list-style-type: none">■ "-----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。ドライバのインストール方法の詳細については、「WebLogic jDrivers」を参照してください。

構文

```
$ java -Dbea.home=license_location utils.dbping DBMS user password DB
```

引数	定義
<i>license_location</i>	WebLogic Server ライセンス (<i>license.bea</i>) が格納されているディレクトリ (たとえば、 <i>d:\beaHome\</i>)。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。
<i>DBMS</i>	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 Oracle Thin Driver : ORACLE_THIN Sybase JConnect ドライバ : JCONNECT Sybase JConnect 5.5 (JDBC 2.0) ドライバ : JCONN2
<i>user</i>	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。
<i>password</i>	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
<i>DB</i>	<p>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。</p> <p>WebLogic jDriver for Microsoft SQL Server : <i>DBNAME@HOST:PORT</i></p> <p>WebLogic jDriver for Oracle : <i>DBNAME</i></p> <p>Oracle Thin Driver : <i>HOST:PORT:DBNAME</i></p> <p>Sybase JConnect ドライバ : JCONNECT: <i>HOST:PORT/DBNAME</i></p> <p>Sybase JConnect ドライバ : JCONN2: <i>HOST:PORT/DBNAME</i></p> <p>各値の説明は次のとおり。</p> <ul style="list-style-type: none">■ <i>HOST</i> は、DBMS のホスト マシンの名前■ <i>PORT</i> は、DBMS が接続をリスンするデータベースホストのポート■ <i>DBNAME</i> は、DBMS のデータベースの名前 (Oracle の場合は、<i>tnsnames.ora</i> ファイルで定義された DBMS の名前)

例

```
$ C:\bea\weblogic700b\samples\server\config\examples>java
utils.dbping ORACLE_THIN scott tiger lcdbsoll:1561:lcs901

**** Success!!! ****

You can connect to the database in your app using:

java.util.Properties props = new java.util.Properties();
props.put("user", "scott");
props.put("password", "tiger");

java.sql.Driver d =
(java.sql.Driver)Class.forName("oracle.jdbc.driver.OracleD
river").newInstance();
java.sql.Connection conn =
d.connect("jdbc:oracle:thin:@lcdbsoll:1561:lcs901",
```

```
props);  
  
// 特にサーバサイド クラスでは、こちらのモードの方が適している。これは、  
// DriverManager 呼び出しでクラスの同期を回避できるためである。ただし、  
// 接続済みの状態でも、サーバ内の他の JDBC ドライバのボトルネックになる  
// 可能性がある。これは、すべての JDBC ドライバが DriverManager.println()  
// を使用して、  
// 情報および例外のログを記録し、その呼び出しでもクラスの同期が取られるため  
// ある  
  
// 接続を繰り返す場合、1 つのドライバ インスタンスを再利用できる  
  
**** or ****  
  
Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();  
java.sql.Connection conn =  
Driver.connect("jdbc:oracle:thin:@lcdbso11:1561:lcs901", "scott",  
"tiger");  
  
**** or ****  
  
java.util.Properties props = new java.util.Properties();  
props.put("user", "scott");  
props.put("password", "tiger");  
Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();  
java.sql.Connection conn =  
Driver.connect("jdbc:oracle:thin:@lcdbso11:1561:lcs901", props);
```

Deployer

`weblogic.Deployer` は、J2EE アプリケーションおよびコンポーネントを **WebLogic Server** にデプロイします。詳細については、「デプロイメント ツール および手順」を参照してください。

`weblogic.Deployer` ユーティリティは、**WebLogic Server 7.0** で新たに導入され、非推奨になっている従来の `weblogic.deploy` ユーティリティに代わるものです。非推奨の `weblogic.deploy` ユーティリティの詳細については、『**管理者ガイド**』の「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。

構文

```
% java weblogic.Deployer [options]
[-activate|-deactivate|-remove|-cancel|-list] [files]
```

アクション (以下のいずれかを選択)

アクション	説明
activate	<code>-name</code> で指定したアプリケーションを、 <code>-targets</code> で指定したサーバにデプロイまたは再デプロイする。
cancel	<code>-id</code> で示されているタスクの取り消しを試行する。
deactivate	対象サーバ上のアプリケーションを非アクティブ化する。非アクティブ化では、デプロイされたコンポーネントがサスペンドされる。ステージングされたデータは、再びアクティブ化されることを前提に、元の場所に格納されたままとなる。このコマンドは、2 フェーズ デプロイメント プロトコルでのみ機能する。
delete_files	ファイルリストに指定されたファイルを削除し、アプリケーションをアクティブ化されたままとする。このコマンドは、アーカイブされていないアプリケーションでのみ有効。対象サーバを指定する必要がある。
deploy	<code>-activate</code> の便利なエイリアス。
examples	ツールの使用例を表示する。

アクション	説明
help	ヘルプ メッセージを出力する。
list	-id で特定されるタスクのステータスを示す。
remove	アプリケーションおよびステージングされたすべてのデータを、対象サーバから物理的に削除する。コンポーネントは非アクティブ化され、対象がアプリケーション コンフィグレーションから削除される。アプリケーションを完全に削除すると、関連付けられている MBean もシステム コンフィグレーションから削除される。このコマンドは、2 フェーズ デプロイメント モデルでのみ機能する。
undeploy	-unprepare に便利なエイリアス。
unprepare	対象サーバ上の -name で示されているアプリケーションのクラスを非アクティブ化してアンロードする。ステージングされたアプリケーション ファイルは、編集またはすぐに再ロードできる状態で残す。
upload	指定したソース ファイルを管理サーバに転送する。このオプションは、リモート システムを使用しているときに、リモートシステムに常駐するアプリケーションをデプロイする場合に使用する。アプリケーション ファイルは、指定された対象サーバに配布される前に WebLogic Server 管理サーバにアップロードされる。
version	バージョン情報を出力する。

オプション

オプション	説明
adminurl	管理サーバの URL (https://<server>:<port>)、デフォルトは http://localhost:7001。
debug	出力ログのデバッグ メッセージをオンにする。

オプション	説明
<code>external_stage</code>	アプリケーション Mbean の作成時に <code>stagingMethod</code> 属性を設定する。これにより、アプリケーションはステージングされないが、アプリケーションを準備する際にステージングパスの値が使用される。
<code>id</code>	タスク識別子 <code>-id</code> は、デプロイメント タスクのユニークな識別子。 <code>-id</code> は、 <code>-activate</code> 、 <code>-deactivate</code> 、 <code>-remove</code> の各コマンドで指定でき、後で <code>-cancel</code> または <code>-list</code> の引数として使用する。 <code>-id</code> を、既存の他のデプロイメントタスクと重複させることはできない。 <code>-id</code> を指定しなかった場合は、システムによって自動生成される。
<code>name</code>	アプリケーションの <code>-name</code> には、デプロイされるアプリケーションの名前を指定する。この名前は、既存ないしコンフィグレーション済みのアプリケーション名、または新しいコンフィグレーション作成時に使用する名前でも可。
<code>nostage</code>	<code>ApplicationMBean</code> の <code>no-staging</code> 属性を設定することにより、そのアプリケーションにはステージングが不要であることを示す。このアプリケーションは、対象サーバの <code>Path</code> 属性で指定された場所に既に存在するものとみなされる。
<code>nowait</code>	アクションが開始されると、ツールがタスク ID を出力して終了する。これは複数のタスクを開始し、 <code>-list</code> アクションを使用して後でモニタするために用いる。
<code>password</code>	パスワードをコマンドラインで指定する。パスワードを指定しないと、パスワードの入力が求められる。
<code>remote</code>	<code>weblogic.Deployer</code> が管理サーバと同じマシンで動作していないこと、およびソースパスはリモートサーバ上のパスを表すため変更せずに渡す必要があることを示す。

オプション	説明
source	アーカイブまたはディレクトリ。デプロイするファイルまたはディレクトリの場所を指定する。このオプションは、アプリケーションパスの設定に使用する。source オプションはルートディレクトリまたはデプロイされるアーカイブを参照する必要がある。アップロードを使用すると、ソースパスはカレントディレクトリに対する相対パスになる。アップロードを使用しない場合は、管理サーバのルートディレクトリ、すなわち config.xml ファイルのあるディレクトリに対する相対パスとなる。
stage	アプリケーションの作成時に stagingMethod 属性を設定する。これにより、アプリケーションは常にステージングされる。この値は対象サーバの stagingMethod 属性をオーバーライドする。
targets	<server1>,<component>@<serverN>。サーバ名やクラスタ名のカンマ区切りリストを表示する。各対象は、J2EE コンポーネント名で修飾できる。これによりアーカイブの各種コンポーネントをさまざまなサーバにデプロイできる。既にデプロイされているアプリケーションに対して指定された場合、このリストが既存の対象に追加される。既存の対象が再度指定された場合、アプリケーションはそれらの既存の対象に再デプロイされると同時に、新しい対象にもデプロイされる。
timeout	秒数。デプロイメントタスクの完了までの最長時間(単位:秒)を指定する。時間が経過すると、weblogic.Deployer がデプロイメントの現在のステータスを出力して終了する。
user	ユーザ名。
verbose	進行状況を示す追加メッセージを表示する。

例

以下に、weblogic.Deployer コマンドの例を示します。

- 新しいアプリケーションのデプロイ

- アプリケーションの再デプロイ
- アプリケーションの一部の再デプロイ
- アプリケーションの非アクティブ化
- アプリケーションのアンデプロイ
- デプロイメント タスクの取り消し
- すべてのデプロイメント タスクの表示

新しいアプリケーションのデプロイ

```
java weblogic.Deployer -adminurl http://admin:7001 -name app  
-source /myapp/app.ear -targets server1,server2 -activate
```

アプリケーションの再デプロイ

```
java weblogic.Deployer -adminurl http://admin:7001 -name app  
-activate
```

アプリケーションの一部の再デプロイ

```
java weblogic.Deployer -adminurl http://admin:7001 -name appname  
-targets server1,server2 -activate jsps/*.jsp
```

アプリケーションの非アクティブ化

```
java weblogic.Deployer -adminurl http://admin:7001 -name app  
-targets server1 -deactivate
```

アプリケーションのアンデプロイ

```
java weblogic.Deployer -adminurl http://admin:7001 -name app  
-targets server -remove -id tag
```

デプロイメント タスクの取り消し

```
java weblogic.Deployer -adminurl http://admin:7001 -cancel -id  
tag
```

すべてのデプロイメント タスクの表示

```
java weblogic.Deployer -adminurl http://admin:7001 -list
```

EJBGen

EJBGen は、エンタープライズ JavaBean 2.0 コード ジェネレータです。Bean クラス ファイルに javadoc タグでコメントを記述して、EJBGen でリモート クラス とホーム クラス、および EJB アプリケーションのデプロイメント記述子ファイル を生成することができます。これにより、編集および管理する必要のある EJB ファイルを 1 つに減らすことができます。

BEA WebLogic 7.0 のサンプルをインストールしている場合、EJBGen を使用したサンプル アプリケーションについては、`SAMPLES_HOME\server\src\examples\ejb20\ejbgen\` を参照してください。

このツールの完全なマニュアルについては、「WebLogic Server EJB のユーティリティ」の「EJBGen」を参照してください。

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:\javall\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:\
```

ImportPrivateKey

ImportPrivateKey ユーティリティを使用すると、プライベート キーストア ファイルにプライベート キーをロードできます。

構文

```
$ java utils.ImportPrivateKey keystore keystorepass alias keypass  
certfile keyfile
```

引数	定義
<i>keystore</i>	キーストア ファイルの名前を定義する。キーストアが存在しない場合は、新しいキーストアが作成される。
<i>keystorepass</i>	キーストア ファイルを開くためのパスワードを定義する。
<i>alias</i>	キーストア内の証明書およびキーのルックアップに使用する名前を定義する。
<i>keypass</i>	キーストアにあるプライベート キー ファイルのロック解除、およびプライベート キーの保護に使用するパスワードを定義する。
<i>certfile</i>	プライベート キーに関連付けられている証明書の名前。
<i>keyfile</i>	保護されたプライベート キーを保持するファイルの名前。

例

次の操作を行うには、以下の手順に従います。

- CertGen ユーティリティで証明書およびプライベート キーを生成する
 - ImportPrivateKey ユーティリティでキーストアを作成してプライベート キーを格納する
1. WL_HOME/server/lib/CertGenCA.der ファイル、および WL_HOME/server/lib/CertGenCAkey.der ファイルを作業ディレクトリにコピーします。

2. `utils.CertGen` を使用して、証明書およびプライベート キーを生成します。
「`CertGen` ツールの使い方」を参照してください。

```
java utils.CertGen mykeypass testcert testkey
Creating Domestic Key Strength - 1024

Encoding
.....
.....
.....
Created Private Key files - testkey.der and testkey.pem
Encoding
.....
.....
.....
Created Certificate files - testcert.der and testcert.pem
.....
```

3. 証明書を `DER` 形式から `PEM` 形式に変換します。

```
D:\bea2\weblogic700\samples\server\src>java utils.der2pem
CertGenCA.der
Encoding
.....
.....
```

4. 証明書と認証局 (CA) を連結します。

```
D:\bea2\weblogic700\samples\server\src>type testcert.pem
CertGenCA.pem >> newcerts.pem
```

5. `mykeystore` という名前のキーストアを新たに作成し、`testkey.pem` ファイルにあるプライベート キーをロードします。

```
D:\bea2\weblogic700\samples\server\src>java utils.ImportPrivateKey
mykeystore mypasswd mykey mykeypass newcerts.pem testkey.pem
Keystore file not found, creating it
```

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
<i>logfile</i>	必須。ログ ファイルの完全修飾パス名。
<i>codebase</i>	必須。アプレットの CODEBASE、または CODEBASE がいない場合は ""。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。
<i>zipfile</i>	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。

A WebLogic Java ユーティリティの使い方

```
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 クラスタ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い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 クラスタ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 クラスタ 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
<code>-a address</code>	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)。
<code>-p portnumber</code>	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)。
<code>-t timeout</code>	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 <code>stdout</code> に出力される。
<code>-s send</code>	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、 <code>stdout</code> 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
<i>pemFile</i>	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 プログラマーズ 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
<i>driverURL</i>	必須。JDBC ドライバの URL。
<i>driverClass</i>	必須。JDBC ドライバクラスのパス名。
<i>-u username</i>	省略可能。有効なユーザ名。
<i>-p password</i>	省略可能。ユーザの有効なパスワード。
<i>-verbose</i>	省略可能。SQL 文とデータベースのメッセージを出力する。
<i>SQLfile</i>	必須。SQL 文を記述したテキストファイル。

例

次のコードでは、examples.utils パッケージの Schema コマンドラインを示します。

```
D:\bea\weblogic700\samples\server\src>java utils.Schema
"jdbc:pointbase:server://localhost/demo"
"com.pointbase.jdbc.jdbcUniversalDriver" -u "examples"
-p "examples" examples/utils/ddl/demo.ddl
```

utils.Schema will use these parameters:

```
url: jdbc:pointbase:server://localhost/demo
driver: com.pointbase.jdbc.jdbcUniversalDriver
dbserver: null
user: examples
password: examples
SQL file: examples/utils/ddl/demo.ddl
```

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 -Dbea.home=license_location utils.showLicenses
```

引数	説明
<i>license_location</i>	license.bea ファイルがあるディレクトリの完全修飾名。

例

```
$ java -Dbea.home=d:\bea utils.showLicense
```

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
```

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
<i>inputFile</i>	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。
<i>zipFileToCreate</i>	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。 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:\javal17\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 #xxxxxx
```

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 の管理コマンドの使い方」
- B-6 ページの「WebLogic Server 管理コマンドのリファレンス」
- B-40 ページの「WebLogic Server 接続プール管理コマンドリファレンス」
- B-50 ページの「MBean 管理コマンドリファレンス」
- B-64 ページの「weblogic.Admin コマンドを使用したユーザとグループの管理」

J2EE モジュールを WebLogic Server インタフェースにデプロイするには、`weblogic.Deployer` コマンドラインユーティリティを使用します。『WebLogic Server アプリケーションの開発』の「デプロイメント ツールおよび手順」を参照してください。

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

weblogic.Admin ユーティリティを使用するには、以下のように環境を設定しておく必要があります。

1. 『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。
2. WebLogic Server クラスを CLASSPATH 環境変数に追加します。2-19 ページの「クラスパスの設定」を参照してください。
3. 管理トラフィック用に予約されているリスン ポートを weblogic.Admin ユーティリティで使いたい場合は、ドメイン全体の管理ポートをコンフィグレーションする必要があります。詳細については、「ドメイン全体の管理ポートのコンフィグレーション」を参照してください。

WebLogic Server の管理コマンドの使い方

この節では、WebLogic Server の管理コマンドを使用するための構文と必須引数を示します。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] } |
    { [-userconfigfile config-file] [-userkeyfile admin-key] }
  ]
  COMMAND arguments
```

接続とユーザ資格の引数

ほとんどの `weblogic.Admin` コマンドでは呼び出しの際に、WebLogic Server インスタンスに接続するため、およびコマンド呼び出しのパーミッションを持つ WebLogic Server ユーザのユーザ資格を指定するために表 13-1 の引数を指定します。

表 13-1 接続とユーザ資格の引数

引数	定義
<code>-url URL</code>	<p>次のいずれかを指定する。</p> <ul style="list-style-type: none"> ■ ドメインの管理サーバのリスンアドレス。この URL では管理サーバのセキュリティ コンテキスト内でコマンドが実行されるため、ほとんどの場合、この URL の使用を推奨。 ■ コマンドの対象である WebLogic Server のリスンアドレス。管理サーバにアクセスできない場合、および管理対象サーバを指定する場合に、この URL を使用する。 <p>形式は、<code>hostname:port</code>。デフォルトは <code>localhost:7001</code>。</p> <p>注意： 管理ポートを指定する場合は、『WebLogic Server ドメイン管理』の「ドメイン全体の管理ポートのコンフィグレーション」で説明しているように、システム管理者がドメイン内のすべてのサーバインスタンスに対して管理ポートを設定していることを確認すること。</p>
<code>-username username</code>	<p>指定したコマンドを呼び出すパーミッションを持つユーザ名。</p> <p>この引数を指定しない場合、<code>weblogic.Admin</code> はユーザ コンフィグレーションファイルとキーファイルを使用する。</p> <p>システム管理タスクのパーミッションについては、3-1 ページの「システム管理操作の保護」を参照。</p>
<code>-password password</code>	<p><code>username</code> に関連付けられたパスワード。</p> <p><code>-username username</code> を指定しても、<code>-password</code> 引数を指定しない場合は、<code>weblogic.Admin</code> がパスワードを要求する。</p> <p><code>WL_HOME\server\bin</code> が <code>PATH</code> 環境変数で指定されている場合、<code>weblogic.Admin</code> ではパスワードが標準出力にエコーされるのを防ぐために WebLogic Server ライブラリを使用する。環境変数の設定については、2-19 ページの「クラスパスの設定」を参照。</p>

表 13-1 接続とユーザ資格の引数

引数	定義
<code>-userconfigfile config-file</code>	<p>暗号化されたユーザ名とパスワードが格納されたユーザ コンフィグレーション ファイルの名前と位置を指定する。暗号化されたユーザ名には、指定したコマンドを実行するパーミッションが必要。</p> <p><code>-userconfigfile config-file</code> を指定しない場合、<code>weblogic.Admin</code> はデフォルトのパス名でユーザ コンフィグレーション ファイルを検索する (B-30 ページの「STOREUSERCONFIG」を参照)。</p>
<code>-userkeyfile admin-key</code>	<p>指定したユーザ コンフィグレーション ファイルと関連付けられたキー ファイルの名前と位置を指定する。</p> <p>ユーザ コンフィグレーション ファイルを作成する際に、STOREUSERCONFIG コマンドはキー ファイルを使用してユーザ名とパスワードを暗号化する。ユーザ コンフィグレーション ファイルを暗号化したキー ファイルのみが、ユーザ名とパスワードを解読できる。</p> <p><code>-userkeyfile admin-key</code> を指定しない場合、<code>weblogic.Admin</code> はデフォルトのパス名でキー ファイルを検索する (B-30 ページの「STOREUSERCONFIG」を参照)。</p>

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。

ユーザ資格引数の概要

表 13-1 では、`weblogic.Admin` ユーティリティがサーバインスタンスにユーザ名とパスワードを渡すために提供する代替手段が説明されています。

セキュリティが最優先ではない開発環境では、コマンドラインで直接、またはスクリプトで `weblogic.Admin` ユーティリティを呼び出す際に `-username` 引数と `-password` 引数を使用できます。これらの引数を使用した場合は、ユーザ名とパスワードは暗号化されません。値をスクリプトに格納した場合は、そのスクリプトを読み出すことができる誰もがそのユーザ資格を使用できることになります。

セキュリティが最優先の環境では、ユーザ コンフィグレーション ファイルとキー ファイルを作成します。ユーザ コンフィグレーション ファイルには、1つのキー ファイルだけで解読できる暗号化されたユーザ資格を格納します。スクリプトに `-userconfigfile config-file` 引数と `-userkeyfile admin-key` 引数を使用することにより、スクリプトの読み込み権限を持つユーザにプレーンテキストのユーザ資格を晒さずに済みます。ユーザ コンフィグレーションとキーファイルの作成については、B-30 ページの「STOREUSERCONFIG」を参照してください。

次のリストは、weblogic.Admin ユーザ資格引数の優先順位を簡単に表したものです。

- `-username username -password password` を指定すると、ユーティリティは暗号化されていない値を `-url` 引数で指定されたサーバインスタンスに渡します。

これらの引数は、`{ -userconfigfile config-file -userkeyfile admin-key }` 引数に優先します。`{ -username username -password password }` と `{ -userconfigfile config-file -userkeyfile admin-key }` の両方を指定すると、weblogic.Admin ユーティリティは `{ -username username -password password }` 引数を使用し、ユーザ コンフィグレーション ファイルとキー ファイルの引数を無視します。

- `-username username` を指定すると、ユーティリティはパスワードを要求します。その後に、暗号化されていない値を `-url` 引数で指定されたサーバインスタンスに渡します。

この引数も、`{ -userconfigfile config-file -userkeyfile admin-key }` 引数に優先します。

- `{ -userconfigfile config-file -userkeyfile admin-key }` を指定し、`{ -username username [-password password] }` を指定しない場合、ユーティリティは `config-file` にある暗号化された値を `-url` 引数で指定されたサーバインスタンスに渡します。

- `{ -username username [-password password] }` と `{ -userconfigfile config-file -userkeyfile admin-key }` のどちらも指定しない場合、ユーティリティはデフォルトのパス名でユーザ コンフィグレーション ファイルとキー ファイルを検索します。デフォルト パス名は、JVM とオペレーティング システムによって異なります。B-32 ページの「デフォルト パス名のコンフィグレーション」を参照してください。

ユーザ資格の指定例

次のコマンドでは、ユーザ名 **weblogic** とパスワード **weblogic** をコマンドラインで直接指定します。

```
java weblogic.Admin -username weblogic -password weblogic COMMAND
```

次のコマンドでは、デフォルトのパス名にあるユーザ コンフィグレーション ファイルとキー ファイルを使用します。

```
java weblogic.Admin COMMAND
```

B-32 ページの「デフォルト パス名のコンフィグレーション」を参照してください。

次のコマンドでは、`c:\wlUser1-WebLogicConfig.properties` というユーザ コンフィグレーション ファイルと `e:\secure\myKey` というキー ファイルを使用します。

```
java -userconfigfile c:\wlUser1-WebLogicConfig.properties  
-userkeyfile e:\secure\myKey COMMAND
```

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-40 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。

表 B-1 WebLogic Server 管理コマンドの概要

タスク	コマンド	説明
WebLogic Server の停止の取り消し	CANCEL_SHUTDOWN	(非推奨)。URL で指定された WebLogic Server の SHUTDOWN コマンドを取り消す。

表 B-1 WebLogic Server 管理コマンドの概要 (続く)

タスク	コマンド	説明
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。
WebLogic Server の強制停止	FORCESHUTDOWN	WebLogic Server プロセスを即座に終了する。
サーバの現在の状態の確認	GETSTATE	WebLogic Server の現在の状態を返す。
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方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。
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。
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。
WebLogic Server のロック	LOCK	(非推奨)。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。
サービスの移行	MIGRATE	クラスタ内の対象サーバに JMS サービスまたは JTA サービスを移行する。
WebLogic Server リスナーポートの検証	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 WebLogic Server コマンドライン インタフェース リファレンス

表 B-1 WebLogic Server 管理コマンドの概要 (続く)

タスク	コマンド	説明
サーバの STANDBY 状態から RUNNING 状態への移動	RESUME	外部クライアントからのリクエストをサーバが受け取ることができるようにする。
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。
WebLogic Server の 停止	SHUTDOWN	WebLogic Server を停止する。
リモート WebLogic 管理対象サーバの 起動	START	コンフィグレーションされたノード マネージャを使用して、RUNNING 状態で管理対象サーバを起動する。
リモート WebLogic 管理対象サーバの 起動および STANDBY 状態への 移動	STARTINSTANDBY	コンフィグレーションされたノード マネージャを使用して 管理対象サーバを起動し、STANDBY 状態にする。
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。
WebLogic Server の ロック解除	UNLOCK	(非推奨)。LOCK 操作の後に WebLogic Server のロックを解除する。
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。

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

(非推奨)。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、この遅延時間内に停止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、停止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-24 ページの「SHUTDOWN」と B-38 ページの「UNLOCK」を参照してください。

このコマンドが非推奨になったのは、SHUTDOWN コマンドでの遅延の指定が非推奨になったためです。SHUTDOWN コマンドで遅延を指定する代わりに、属性を設定してサーバの停止を制御できるようになりました。詳細については、Administration Console オンラインヘルプの「ライフサイクルオペレーションのタイムアウト設定」を参照してください。

構文

```
java weblogic.Admin [ 接続とユーザ資格の引数 ] CANCEL_SHUTDOWN
```

例

次の例では、ユーザ名が weblogic、パスワードが weblogic 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 の停止の取り消しを要求します。

```
java weblogic.Admin -url t3://localhost:7001 -username weblogic  
-password weblogic 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 [ 接続とユーザ資格の引数 ] CONNECT count
```

引数	定義
<i>count</i>	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
```

FORCESHUTDOWN

サーバプロセスを即座に終了します。

このコマンドを発行すると、すべてのアプリケーションおよびサブシステムに対して、すべての作業を中断してすべてのリソースを開放するよう通知されます。強制的な停止では、トランザクションがロールバックされたり、いくつかのクライアントのセッションが失われるおそれがあります。サーバの強制的な停止は、どの状態からでも実行できます。

このコマンドを管理対象サーバに実行した場合、管理対象サーバは応答しなくなります。ノードマネージャでサーバを起動した場合、ノードマネージャはサーバプロセスを強制停止します。

このタスクの **Administration Console** での実行については、**Administration Console** オンラインヘルプの「サーバの強制停止」を参照してください。

このコマンドは、ライフサイクルオペレーションのタイムアウト期間の影響を受けます。詳細については、『管理者ガイド』の「ライフサイクルオペレーションのタイムアウト期間」、および **Administration Console** オンラインヘルプの「ライフサイクルオペレーションのタイムアウト設定」を参照してください。

構文

```
java weblogic.Admin [ 接続とユーザ資格の引数 ] FORCESHUTDOWN
[ targetserver ]
```

引数	定義
<code>targetserver</code>	省略可能。再開するサーバの名前。値を指定しない場合、 <code>-url</code> 引数で指定したサーバが停止される。

例

次の例では、ユーザ名が `adminuser`、パスワードが `gumby1234` のユーザが、管理サーバで `MyServer` という名前のサーバを強制的に停止します。

```
java weblogic.Admin -url myAdminServer:7001 -username adminuser
-password gumby1234 FORCESHUTDOWN MyServer
```

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

次の例では、管理サーバが使用できない場合に、同じユーザが管理対象サーバのホストマシンにログオンして、次のコマンドを発行します。

```
java weblogic.Admin -url localhost:7001 -username adminuser  
-password gumby1234 FORCESHUTDOWN
```

GETSTATE

サーバの現在の状態を返します。

サーバの状態の詳細については、「サーバのライフサイクル」を参照してください。

構文

```
java weblogic.Admin [ 接続とユーザ資格の引数 ] GETSTATE targetserver
```

引数	定義
<i>targetserver</i>	省略可能。再開するサーバの名前。値を指定しない場合、 <code>-url</code> 引数で指定したサーバの状態が返される。

例

次の例では、ユーザ名が `adminuser`、パスワードが `gumby1234` のユーザが、管理サーバで `MyServer` という名前のサーバの状態の確認を試行します。

```
java weblogic.Admin -url myAdminServer:7001 -username adminuser  
-password gumby1234 GETSTATE MyServer
```

HELP

すべての WebLogic Server コマンドの構文と使用方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 [ 接続とユーザ資格の引数 ] LICENSES
```

例

次の例では、デフォルトのユーザ名 (installadministrator) とパスワード (installadministrator) 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  
installadministrator  
-password installadministrator 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 [ 接続とユーザ資格の引数 ] LIST context
```

引数	定義
<i>context</i>	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 -url localhost:7001 -username adminuser  
-password gumby1234 LIST weblogic.ejb
```

LOCK

(非推奨)。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。

LOCK コマンドを使用する代わりに、STANDBY 状態でサーバを起動します。この状態では、サーバインスタンスはドメイン全体の管理ポート上の管理要求にのみ応答します。詳細については、B-28 ページの「STARTINSTANDBY」を参照してください。

構文

```
java weblogic.Admin [ 接続とユーザ資格の引数 ] LOCK "string_message"
```

引数	定義
<i>"string_message"</i>	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。

MIGRATE

サーバクラスタ内の対象サーバに JMS サービスまたは JTA トランザクション回復サービスを移行します。

構文

JMS リソースを移行するには

```
java weblogic.Admin [ 接続とユーザ資格の引数 ]
MIGRATE -migratabletarget "serverName (migratable)"
        -destination serverName [-sourcedown] [-destinationdown]
```

JTA リソースを移行するには

```
java weblogic.Admin [-url URL]
        -username username [-password password]
MIGRATE -jta -migratabletarget serverName
        -destination serverName [-sourcedown] [-destinationdown]
```

引数	定義
{-url [protocol://]listen-address:listen-port}	WebLogic 管理サーバのリスンアドレスとリスンポートを指定する。セキュアリスンポートを指定する場合には、セキュアなプロトコルも指定する必要がある。この値を指定しない場合は、t3://localhost:7001 を指定したものと見なされる。
-jta	移行が JTA サービスの移行であることを指定する。この引数を指定しない場合、MIGRATE コマンドは JMS サービスを移行する。
-migratabletarget	サービスの移行元となるサーバの名前を指定する。サーバ名を指定するための構文は、移行するサービスの種類によって異なる。 <ul style="list-style-type: none"> ■ JMS の場合、"servername (migratable)" と指定する。例えば、"myserver (migratable)"。 ■ JTA の場合、servername と指定する。例えば、myserver。
-destination	サービスの移行先となるサーバの名前を指定する。

引数	定義
<code>-sourcedown</code>	ソース サーバの停止を指定する。このスイッチは、慎重に使用する必要がある。ソース サーバが実際には停止しておらず、単にネットワークの問題で利用できなくなっている場合、サービスはソースサーバから削除されないまま、送り先サーバでアクティブ化される。その結果、同じサービスが2つ同時に実行されることになる。このため、トランザクション ログまたは JMS メッセージに障害が発生する可能性がある。
<code>-destinationdown</code>	送り先サーバの停止を指定する。動作していないサーバに移行した JMS サービスは失われる。動作していないサーバに JTA トランザクション回復サービスを移行した場合、対象サーバは起動時に回復サービスを実行すると仮定する。

例

次の例では、JMS サービスが `myserver2` から `myserver3` に移行されます。

```
java weblogic.Admin -url AdminHost:7001 -username adminuser
  -password gumby1234 MIGRATE
  -migratabletarget "myserver2 (migratable)"
  -destination myserver3
```

次の例では、JTA トランザクション回復サービスが `myserver2` から `myserver3` に移行されます。

```
java weblogic.Admin -url AdminHost:7001 -username adminuser
  -password gumby1234 MIGRATE -jta
  -migratabletarget myserver2 -destination myserver3 -sourcedown
```

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 [ 接続とユーザ資格の引数 ] PING [round_trips]
[message_length]
```

引数	定義
<i>round_trips</i>	省略可能。ping の数。
<i>message_length</i>	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
```

RESUME

サーバを STANDBY 状態から RUNNING 状態に移動します。

このタスクの Administration Console での実行については、Administration Console オンライン ヘルプの「サーバの再開」を参照してください。サーバの状態の詳細については、「サーバのライフサイクル」を参照してください。

構文

```
java weblogic.Admin [ 接続とユーザ資格の引数 ] RESUME [ targetserver ]
```

引数	定義
<i>targetserver</i>	省略可能。再開するサーバの名前。値を指定しない場合、-url 引数で指定したサーバが再開される。

例

次の例では、ユーザ名が `adminuser`、パスワードが `gumby1234` のユーザが、管理サーバで `MyServer` という名前のサーバの再開を試行します。

```
java weblogic.Admin -url myAdminServer:7001 -username adminuser  
-password gumby1234 RESUME MyServer
```

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 [ 接続とユーザ資格の引数 ] SERVERLOG  
[[starttime]][endtime]]
```

引数	定義
<i>starttime</i>	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 <i>yyyy/mm/dd</i> 。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。" <i>yyyy/mm/dd hh:mm</i> "
<i>endtime</i>	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 <i>yyyy/mm/dd</i> 。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。" <i>yyyy/mm/dd hh:mm</i> "

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 -username adminuser  
-password gumby1234 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

指定した **WebLogic Server** を停止します。

このコマンドを発行すると、コンフィグレーションしたすべての停止クラスが呼び出されます。つづいて、すべてのアプリケーションおよびサブシステムに対し、外部クライアントからの新しい要求の受信を停止し、現在の作業をすべて完了するよう通知されます。サーバを正常に停止できるのは、**RUNNING** 状態または **STANDBY** 状態からのみです。

リリース **6.x** では、停止プロセスを開始するまでの待機秒数を指定するオプションがこのコマンドに導入されましたが、現在では非推奨のオプションになっています。この非推奨オプションをサポートするには、**SHUTDOWN** コマンドの直後にあるフィールドで指定する値が秒を表すことを明確にする必要があります。サーバの名前がすべて数字で構成されている場合、このコマンドを使用して正常に停止することはできません。このような場合は、**Administration Console** を使用する必要があります。詳細については、**Administration Console** オンラインヘルプの「サーバの停止」を参照してください。

SHUTDOWN コマンドで遅延を指定する代わりに、属性を設定してサーバの停止を制御できるようになりました。詳細については、『管理者ガイド』の「ライフサイクル オペレーションのタイムアウト期間」、および **Administration Console** オンラインヘルプの「ライフサイクル オペレーションのタイムアウト設定」を参照してください。

構文

```
java weblogic.Admin [ 接続とユーザ資格の引数 ] SHUTDOWN [targetserver]
( 非推奨 ) java weblogic.Admin [-url URL] -username username
                [-password password] SHUTDOWN
                [seconds ] [ "lockMessage" ]
```

引数	定義
<i>targetserver</i>	再開するサーバの名前。 値を指定しない場合、 <code>-url</code> 引数で指定したサーバが停止される。
<i>seconds</i>	(非推奨)。省略可能。このコマンドの実行時からサーバの停止までの経過秒数。

引数	定義
" <i>lockMessage</i> "	(非推奨)。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

次の例では、ユーザ名が `adminuser`、パスワードが `gumby1234` のユーザが、管理サーバで `MyServer` という名前のサーバを停止します。

```
java weblogic.Admin -url MyAdminServer:7001 -username adminuser  
-password gumby1234 SHUTDOWN MyServer
```

START

ノード マネージャを使用してリモート管理対象サーバを起動します。

ノード マネージャを使用してリモート管理対象サーバを起動します。

このコマンドでは以下の環境が必要です。

- ドメインの管理サーバを起動している。
- ノード マネージャは、管理対象サーバのホスト マシン上で動作している。
- 管理対象サーバの起動項目とノード マネージャを、「ノード マネージャによるサーバの可用性の管理」で説明されているとおりに設定している。

このタスクの **Administration Console** での実行については、**Administration Console** オンライン ヘルプの「サーバの起動」を参照してください。

注意： **Administration Console** の [サーバ | 一般] タブの [起動モード] フィールドでは、サーバが起動する状態を指定できます。ただし、この設定が適用されるのは、`weblogic.Server` コマンドを使用してローカル ホストからサーバを起動する場合のみです。したがって、ノード マネージャ (`weblogic.Admin START` コマンド) では、指定した値は適用されません。たとえば、[起動モード] の値に `STANDBY` を指定しても、`weblogic.Admin START` コマンドを発行すれば、サーバは `RUNNING` 状態で起動します。

構文

```
java weblogic.Admin [ 接続とユーザ資格の引数 ]  
START targetserver
```

引数	定義
<code>-url URL</code>	ドメインの管理サーバのリスン アドレスを指定しなければならない。 デフォルトは <code>localhost:7001</code> 。
<code>targetserver</code>	<code>RUNNING</code> 状態で起動する管理対象サーバの名前。

例

次の例では、ユーザ名が `adminuser`、パスワードが `gumby1234` のユーザが、管理サーバで **MyServer** という名前のサーバの起動を試行します。

```
java weblogic.Admin -url myAdminServer:7001 -username adminuser  
-password gumby1234 START MyServer
```

STARTINSTANDBY

ノード マネージャを使用してリモート管理対象サーバを起動し、その状態を STANDBY にします。この状態では、外部クライアントからのリクエストはサーバにアクセスできなくなります。

このコマンドでは以下の環境が必要です。

- ドメインの管理サーバを起動している。
- ノード マネージャは、管理対象サーバのホスト マシン上で動作している。
- 管理対象サーバの起動項目とノード マネージャを、「ノード マネージャによるサーバの可用性の管理」で説明されているとおりに設定している。
- ドメインは、「ドメイン全体の管理ポートのコンフィグレーション」で説明されているように、ドメイン全体の管理ポートを使用するようにコンフィグレーションされている。

注意： Administration Console の [サーバ | 一般] タブの [起動モード] フィールドでは、サーバが起動する状態を指定できます。ただし、この設定が適用されるのは、weblogic.Server コマンドを使用してローカル ホストからサーバを起動する場合のみです。したがって、ノード マネージャ (weblogic.Admin STARTINSTANDBY コマンド) では、指定した値は適用されません。たとえば、[起動モード] の値に RUNNING を指定しても、weblogic.Admin STARTINSTANDBY コマンドを発行すれば、サーバは STANDBY 状態で起動します。

このタスクの Administration Console での実行については、Administration Console オンライン ヘルプの「STANDBY 状態でのサーバの起動」を参照してください。サーバの状態の詳細については、「サーバのライフサイクル」を参照してください。

構文

```
java weblogic.Admin [ 接続とユーザ資格の引数 ] STARTINSTANDBY  
[ targetserver ]
```

引数	定義
<code>-url</code>	ドメインの管理サーバを指定しなければならない。 デフォルトは <code>localhost:7001</code> 。
<code>targetserver</code>	省略可能。STANDBY 状態で起動する WebLogic Server の名前。 値を指定しない場合、 <code>-url</code> 引数で指定したサーバが起動される。

例

次の例では、ユーザ名が `adminuser`、パスワードが `gumby1234` のユーザが、管理サーバで **MyServer** という名前のサーバの起動を試行します。

```
java weblogic.Admin -url myAdminServer:7001 -username adminuser  
-password gumby1234 STARTINSTANDBY MyServer
```

STOREUSERCONFIG

ユーザ コンフィグレーション ファイルと、関連するキー ファイルを作成します。ユーザ コンフィグレーション ファイルには、暗号化されたユーザ名とパスワードを格納します。キー ファイルには、ユーザ名とパスワードの暗号化と解読に使用する秘密鍵を格納します。

他の `weblogic.Admin` コマンドまたは `weblogic.Deployer` コマンドを使用する際には、暗号化されていないユーザ名とパスワードをコマンドラインで入力したり、暗号化されていない資格をスクリプトに挿入したりする代わりにユーザ コンフィグレーション ファイルとキー ファイルを指定できます。B-4 ページの「ユーザ資格引数の概要」を参照してください。

元々ユーザ名とパスワードを暗号化したキー ファイルだけが、値を解読できます。キー ファイルがなくなった場合は、ユーザ コンフィグレーション ファイルとキー ファイルの新しいペアを作成しなければなりません。

警告： 必ず、認可されたユーザのみがキー ファイルにアクセスできるようにする必要があります。有効なユーザ コンフィグレーション ファイルとキー ファイルにアクセスしたユーザは誰でも、暗号化されているユーザ名の権限を得ることになります。キー ファイルへのアクセスを保護するために、**WebLogic Server** 管理者などの認可されたユーザのみに読み書きアクセスを許可するディレクトリにキー ファイルを格納することができます。あるいは、フロッピー ディスクや CD などのリムーバブル メディアにキー ファイルを書き込み、使用時以外は引き出しにしまって鍵をかけておくこともできます。

他の `weblogic.Admin` コマンドと違って、`STOREUSERCONFIG` コマンドは **WebLogic Server** インスタンスに接続しません。データの暗号化とファイルの作成は、`STOREUSERCONFIG` コマンドが実行された **JVM** で行われます。**WebLogic Server** インスタンスに接続しないので、このコマンドではユーザ名とパスワードが有効な **WebLogic Server** の資格であることを検証できません。

構文

```
java weblogic.Admin
  -username username [-password password]
  [ -userconfigfile config-file ] [ -userkeyfile keyfile ]
STOREUSERCONFIG
```

引数	定義
<code>-userconfigfile</code> <code>config-file</code>	<p>STOREUSERCONFIG コマンドがユーザ コンフィグレーション ファイルを作成するファイルパス名を指定する。パス名は絶対パス名でも、コマンドを入力したディレクトリとの相対パス名でもかまわない。</p> <p>指定のパス名にファイルが既にある場合は、新しく暗号化されたユーザ名とパスワードが格納されている新しいファイルでそのファイルが上書きされる。</p> <p>このオプションを指定しない場合、STOREUSERCONFIG は次のように動作する。</p> <ul style="list-style-type: none">■ ユーザ コンフィグレーション ファイルを作成するディレクトリを決めるために、JVM のユーザ ホーム ディレクトリを使用する。デフォルト値は、SDK とオペレーティング システムの種類によって異なる。B-32 ページの「デフォルトパス名のコンフィグレーション」を参照。■ ファイル名を決めるために、オペレーティング システムのユーザ名を文字列 <code>-WebLogicConfig.properties</code> の先頭に付ける。たとえば、<code>username-WebLogicConfig.properties</code> のようになる。Java のオプションを使用すると、別のユーザ名を指定できる。B-32 ページの「デフォルトパス名のコンフィグレーション」を参照。

引数	定義 (続く)
<code>-userkeyfile keyfile</code>	<p>STOREUSERCONFIG コマンドがキー ファイルを作成するファイル パス名を指定する。パス名は絶対パス名でも、コマンドを入力したディレクトリとの相対パス名でもかまわない。</p> <p>指定のパス名に既にファイルがある場合、STOREUSERCONFIG はその既存のキー ファイルを使用して新しいユーザ コンフィグレーション ファイルを暗号化する。</p> <p>このオプションを指定しない場合、STOREUSERCONFIG は次のように動作する。</p> <ul style="list-style-type: none"> ■ キー ファイルを作成するディレクトリを決めるために、JVM のユーザ ホーム ディレクトリを使用する。デフォルト値は、SDK とオペレーティング システムの種類によって異なる。B-32 ページの「デフォルト パス名のコンフィグレーション」を参照。 ■ ファイル名を決めるために、オペレーティング システムのユーザ名を文字列 <code>-WebLogicKey.properties</code> の先頭に付ける。たとえば、<code>username-WebLogicKey.properties</code> のようになる。Java のオプションを使用すると、別のユーザ名を指定できる。B-32 ページの「デフォルト パス名のコンフィグレーション」を参照。
<code>-username username [-password password]</code>	<p>暗号化するユーザ名とパスワードを指定する。</p> <p>STOREUSERCONFIG コマンドは、ユーザ名とパスワードが有効な WebLogic Server のユーザ資格であることを検証しない。</p> <p><code>-password password</code> 引数を省略すると、STOREUSERCONFIG はパスワードの入力を要求する。</p>

デフォルト パス名のコンフィグレーション

ユーザ コンフィグレーション ファイルとキー ファイルを作成および使用する場所を指定しない場合、`weblogic.Admin` と `weblogic.Deployer` の各ユーティリティは以下のデフォルト値を使用します。

- `user-home-directory\username-WebLogicConfig.properties`
- `user-home-directory\username-WebLogicKey.properties`

`user-home-directory` は JVM で決められたオペレーティングシステム ユーザアカウントのホームディレクトリ、`username` はオペレーティングシステムのユーザ名です。

ホームディレクトリの値は、SDK とオペレーティングシステムの種類によって異なります。たとえば UNIX 上では、ホームディレクトリは通常「`~username`」です。Windows の場合、ホームディレクトリは通常「`C:\Documents and Settings\username`」です。

以下の Java オプションを使用すると、`user-home-directory` と `username` の値を指定できます。

- `-Duser.home=pathname` は `user-home-directory` の値を指定する
- `-Duser.name=username` は `username` の値を指定する

たとえば次のコマンドでは、ユーザホームディレクトリは `c:\myHome`、ユーザ名は `wlAdmin` としてコンフィグレーションされます。このコマンドは、以下のユーザコンフィグレーションファイルとユーザキーファイルを検索します。

```
c:\myHome\wlAdmin-WebLogicConfig.properties
c:\myHome\wlAdmin-WebLogicKey.properties
```

```
java -Duser.home=c:\myHome -Duser.name=wlAdmin
weblogic.Admin COMMAND
```

ユーザコンフィグレーションファイルとキーファイルの作成

ユーザコンフィグレーションファイルとキーファイルを作成するには、次の手順を行います。

1. `-username username` および `-password password` 引数を使用して、暗号化するユーザ名とパスワードを指定します。
2. 以下のいずれかの手段で、ユーザコンフィグレーションファイルとキーファイルの名前と位置を指定します。
 - `-userconfigfile config-file` 引数と `-userkeyfile key-file` 引数を使用する

```
java weblogic.Admin -username username -password password
-userconfigfile config-file -userkeyfile key-file
STOREUSERCONFIG
```

- デフォルトの動作を使用して、
`user-home-directory\username-WebLogicConfig.properties` および
`user-home-directory\username-WebLogicKey.properties` という名
前のファイルを作成する

```
java weblogic.Admin -username username -password password  
STOREUSERCONFIG
```

- `-Duser.home=directory` と `-Duser.name=username` の Java オプション
を使用して、`directory\username-WebLogicConfig.properties` およ
び `directory\username-WebLogicKey.properties` という ファイルを
作成する

```
java -Duser.home=directory -Duser.name=username  
weblogic.Admin -username username -password password  
STOREUSERCONFIG
```

ユーザ コンフィグレーション ファイルとキー ファイルの名前と位置は、2つの
ファイルをペアとして使用する限り作成後に変更できます。

複数のユーザ コンフィグレーション ファイルに 1 つのキー ファイルを使用する

1 つのキー ファイルで複数のユーザ コンフィグレーション ファイルを暗号化する
には、次の手順を行います。

1. 最初のユーザ コンフィグレーション ファイルとキー ファイルのペアを作成
します。

たとえば、次のコマンドを入力します。

```
java weblogic.Admin -username username -password password  
-userconfigfile c:\AdminConfig -userkeyfile e:\myKeyFile  
STOREUSERCONFIG
```

2. 追加のユーザ コンフィグレーション ファイルを作成するときに、既存の
キー ファイルを指定します。

たとえば、次のコマンドを入力します。

```
java weblogic.Admin -username username -password password  
-userconfigfile c:\anotherConfigFile -userkeyfile e:\myKeyFile  
STOREUSERCONFIG
```

例

次の例では、UNIX オペレーティング システムに joe としてログインしたユーザが、ユーザ名 wlAdmin およびパスワード wlPass を暗号化します。

```
java weblogic.Admin -username wlAdmin -password wlPass  
STOREUSERCONFIG
```

このコマンドは、~joe/joe-WebLogicKey.properties という名前のキー ファイルが存在するかどうかを確認します。そのようなファイルがない場合は、y を選択してキー ファイルの作成を確認するようユーザに要求します。コマンドが正常に実行されると、以下の 2 つのファイルが作成されます。

```
~joe\joe-WebLogicConfig.properties  
~joe\joe-WebLogicKey.properties
```

ファイル joe-WebLogicConfig.properties には、文字列 wlAdmin および wlPass の暗号化バージョンが格納されます。

~joe\joe-WebLogicConfig.properties ファイルを使用するコマンドはすべて、~joe\joe-WebLogicKey.properties キー ファイルを指定する必要があります。

次の例では、システム管理者であるユーザ joe が、pat というオペレーティング システム アカウントのユーザ コンフィグレーション ファイルを作成しようとしています。利便性を考慮して、joe は pat のホーム ディレクトリにユーザ コンフィグレーション ファイルを作成したいと考えています。そうすることで、pat が呼び出す weblogic.Admin コマンドの構文が簡単になります。セキュリティを向上させるために、joe の組織で使用するキー ファイルは 1 つだけであり、そのファイルはリムーバブルのハードディスクに保存されています。

e:\myKeyFile という名前のキー ファイルで暗号化および解読されるユーザ コンフィグレーション ファイルを pat のホーム ディレクトリに作成するには、次のコマンドを使用します。

```
java -Duser.name=pat -Duser.home="C:\Documents and Settings\pat"  
weblogic.Admin -username wlOperatorPat -password wlOperator1  
-userkeyfile e:\myKeyFile  
STOREUSERCONFIG
```

pat のアカウントにログインしたユーザは、次の構文を使用して weblogic.Admin コマンドを呼び出すことができます。

```
java weblogic.Admin -userkeyfile e:\myKeyFile COMMAND
```

ユーザ コンフィグレーション ファイルとキー ファイルの使い方については、
B-4 ページの「ユーザ資格引数の概要」を参照してください。

THREAD_DUMP

現時点で特定のサーバインスタンス用に実行されている WebLogic Server スレッドのスナップショットを出力します。スナップショットは標準出力に出力されません。

注意： THREAD_DUMP コマンドは、Sun JVM と JRockit でのみサポートされています。

構文

```
java weblogic.Admin [ 接続とユーザ資格の引数 ] THREAD_DUMP
```

例

次の例では、ManagedHost というホストで実行されているサーバインスタンスから標準出力にスレッド ダンプを出力します。

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic  
-password weblogic THREAD_DUMP
```

コマンドが正常に実行されると、コマンド自体によって以下が返されます。

```
Thread Dump is available in the command window that is running the  
server.
```

サーバインスタンスは、スレッド ダンプを標準出力に出力します。標準出力は、デフォルトではそのサーバインスタンスが実行されているシェル (コマンドプロンプト) になります。

UNLOCK

(非推奨)。LOCK 操作の後で WebLogic Server のロックを解除します。

このコマンドは、LOCK コマンドが非推奨になったため非推奨となりました。LOCK および UNLOCK の代わりに、STARTINSTANDY および RESUME を使用してください。詳細については、B-21 ページの「RESUME」を参照してください。

構文

```
java weblogic.Admin [ 接続とユーザ資格の引数 ] UNLOCK
```

引数	定義
<i>username</i>	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ります。
<i>password</i>	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ります。

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す。

構文

```
java weblogic.Admin [ 接続とユーザ資格の引数 ] 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  
installadministrator  
-password installadministrator VERSION
```

注意： この例では、引数 `username` と `password` の両方にデフォルト値の `installadministrator` 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、『WebLogic JDBC プログラマーズ ガイド』および『管理者ガイド』の「JDBC 接続の管理」を参照してください。

表 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-42 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。
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。

表 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なことがある。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往々にしてある。

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、『WebLogic JDBC プログラマーズ 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 [ 接続とユーザ資格の引数 ] 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 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を示す。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は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までにかかる遅延時間 (秒数)。この遅延は、最初にプールが作成されるときにも、プールの生存期間中に物理データベース接続が作成されるときにも発生する。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、最初にプールが作成されるときにも、プールの生存期間中に物理データベース接続が作成されるときにも発生する。

例

次の例では、名前が `weblogic`、パスワードが `weblogic` 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 weblogic  
-password weblogic CREATE_POOL MyPool
```

```
java weblogic.Admin -url t3://forest:7901 -username weblogic  
-password weblogic 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する。

構文

```
java weblogic.Admin [ 接続とユーザ資格の引数 ] DESTROY_POOL poolName
[ true/false ]
```

引数	定義
<i>poolName</i>	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。

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 [ 接続とユーザ資格の引数 ] 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
-passwd 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 [ 接続とユーザ資格の引数 ] ENABLE_POOL poolName
```

引数	定義
<i>poolName</i>	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、動的接続プールが既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 [ 接続とユーザ資格の引数 ] EXISTS_POOL poolName
```

引数	定義
<i>poolName</i>	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 管理ユーザのユーザ名と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 [ 接続とユーザ資格の引数 ] RESET_POOL poolName
```

引数	定義
<i>URL</i>	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)。
<i>username</i>	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。
<i>password</i>	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。
<i>poolName</i>	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)。

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 インスタンスの「demoPool」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 -url t3://xyz.com:7001 -username system  
-password gumby RESET_POOL demoPool
```

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。

表 B-3 MBean 管理コマンドの概要

タスク	コマンド	説明
管理 MBean の作成	CREATE	管理 MBean 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。ローカル コンフィグレーション MBean を作成するために使用しないことを推奨。
MBean の削除	DELETE	MBean を削除する。成功した場合は、OK を stdout に返す。
MBean プロパティ (属性) の表示	GET	MBean のプロパティを表示する。
MBean 操作の呼び出し	INVOKE	MBean が基底のリソースに公開する管理操作を呼び出す。
管理 MBean またはローカル コンフィグレーション MBean のプロパティ値を設定する	SET	指定した MBean の指定したプロパティ値を設定する。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。

MBean タイプの指定

すべての MBean 管理コマンドでは `-type` 引数を指定できます。この引数を使用すると、コマンドは、指定するタイプのインスタンスであるすべての MBean を操作します。MBean のタイプとは、MBean がそのインスタンスとなるインタフェースクラスのことです。すべての WebLogic Server MBean は、`weblogic.management.configuration` パッケージまたは

`weblogic.management.runtime` パッケージで定義されているいずれかのインタフェースクラスのインスタンスです。コンフィグレーション MBean の場合は、タイプによって、インスタンスが管理 MBean またはローカル コンフィグレーション MBean のどちらであるかもわかります。すべての WebLogic Server MBean インタフェースクラスの完全なリストについては、`weblogic.management.configuration` または `weblogic.management.runtime` パッケージの WebLogic Server Javadoc を参照してください。

-type 引数に指定する値を決定するには、次の手順に従います。

1. MBean のインタフェースクラスを見つけて、クラス名から MBean サフィックスを削除する。
`weblogic.management.runtime.JDBCConnectionPoolRuntimeMBean` のインスタンスである MBean の場合は、`JDBCConnectionPoolRuntime` を使用します。
2. ローカル コンフィグレーション MBean の場合は、名前に `Config` を追加する。たとえば、
`weblogic.management.configuration.JDBCConnectionPoolMBean` インタフェースクラスのインスタンスであるローカル コンフィグレーション MBean の場合は、`JDBCConnectionPoolConfig` を使用します。対応する管理 MBean には、`JDBCConnectionPool` を使用します。

サーバの指定

すべての MBean 管理コマンドには `-url` 引数があります。この引数は、MBean をホストする WebLogic Server インスタンスの指定に使用します。

管理 MBean を操作するには、`-url` 引数を使用して管理サーバを指定する必要があります。ローカル コンフィグレーション MBean または実行時 MBean を操作するには、`-url` 引数を使用して、MBean をホストする WebLogic Server インスタンスを指定する必要があります。

以下の条件の**すべて**に当てはまる場合は、`weblogic.Admin` を使用して、ドメイン全体の中から任意の MBean にアクセスできます。

- 管理サーバがポート 7001 でリスンしている。
- 管理サーバから `weblogic.Admin` コマンドを発行する。

- `-url` 引数を省略する。

たとえば、上記の条件にすべて当てはまる場合、以下のコマンドを発行すると、ドメイン内のすべてのサーバの状態を判別できます。

```
java weblogic.Admin -username weblogic -password weblogic GET -type  
ServerRuntime -property State
```

CREATE

WebLogic Server 管理 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。ローカル コンフィグレーション MBean を作成するために使用しないことをお勧めします。

このコマンドを使用して管理 MBean インスタンスを作成すると、WebLogic Server はデフォルト値を備えた MBean を取得し、MBean のコンフィグレーションをドメインの config.xml ファイルに格納します。WebLogic Server は、基底の管理対象リソースをホストするサーバインスタンスを再起動するまでは、対応するローカル コンフィグレーション MBean のレプリカを作成しません。たとえば、MyServer という管理対象サーバにある JDBC 接続プールを管理するために JDBCConnectionPool 管理 MBean を作成する場合、作成した JDBCConnectionPool 管理 MBean のローカルのレプリカを作成するには、MyServer を再起動する必要があります。MBean のレプリケーションとライフサイクルの詳細については、『WebLogic JMX Service プログラマーズ ガイド』の「管理対象リソースをコンフィグレーションするための MBean」を参照してください。

構文

```
java weblogic.Admin [ 接続とユーザ資格の引数 ] 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
URL	コマンドの対象である WebLogic Server インスタンス。詳細については、B-51 ページの「サーバの指定」を参照。
name	作成している MBean を呼び出すときの名前を指定する。
mbean_type	作成する MBean のタイプ。詳細については、B-50 ページの「MBean タイプの指定」を参照。

引数	定義
<i>mbean_name</i>	MBean の完全修飾オブジェクト名を <code>WebLogicObjectName</code> 形式で指定する。次に例を示す。 <code>"domain:Type=type,Name=name"</code> 詳細については、 <code>WebLogicObjectName</code> の Javadoc を参照。
<i>domain_name</i>	MBean インスタンスを作成するドメインの名前。 <i>domain_name</i> を指定しない場合、コマンドでは、対象のサーバが属するドメインを想定する。

例

次の例では、`-name` および `-type` 引数を使用して、`MyPool` という名前の `JDBCConnectionPool` 管理 **MBean** を管理サーバ上に作成します。

```
java weblogic.Admin -url AdminHost:7001 -username weblogic  
-password weblogic CREATE -name MyPool -type JDBCConnectionPool
```

コマンドが正常に実行されると、標準出力に次のよう出力されます。

Ok

次の例では、`-mbean` 引数および `-WebLogicObjectName` 規約を使用して、`MyPool` という名前の `JDBCConnectionPool` 管理 **MBean** を管理サーバ上に作成します。

```
java weblogic.Admin -url AdminHost:7001 -username weblogic  
-password weblogic  
CREATE -mbean "MyDomain:Type=JDBCConnectionPool,Name=MyPool"
```


DELETE

MBean を削除します。管理 MBean を削除すると、WebLogic Server は、ドメインの `config.xml` ファイルから対応するエントリを削除します。成功した場合は、OK を stdout に返します。

注意： 管理 MBean を削除しても、対応するコンフィグレーション MBean はサーバインスタンスを再起動するまで削除されません。

構文

```
java weblogic.Admin [ 接続とユーザ資格の引数 ]
{-type mbean_type|-mbean mbean_name}
```

引数	定義
<i>URL</i>	コマンドの対象である WebLogic Server インスタンス。詳細については、B-51 ページの「サーバの指定」を参照。
<i>mbean_type</i>	指定したタイプのすべての MBean を削除する。詳細については、B-50 ページの「MBean タイプの指定」を参照。
<i>mbean_name</i>	MBean の完全修飾オブジェクト名を WebLogicObjectName 形式で指定する。次に例を示す。 <code>"domain:Type=type,Name=name"</code> 詳細については、WebLogicObjectName の Javadoc を参照。

例

次の例では、MyPool という名前の JDBCConnectionPool 管理 MBean を削除します。

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
  -password weblogic DELETE -mbean
  MyDomain:Name=MyPool,Type=JDBCConnectionPool
```

コマンドが正常に実行されると、標準出力に次のように出力されます。

```
Ok
```

次の例では、MyServer というサーバインスタンスにある、MyPool という名前の JDBCConnectionPool ローカル コンフィグレーション MBean を削除します。

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

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
  -password weblogic DELETE -mbean
  MyDomain:Location=MyServer,Name=MyPool,
  Type=JDBCConnectionPoolConfig
```

次の例では、ドメイン内のすべてのサーバインスタンスにある、すべての JDBCConnectionPool ローカル コンフィグレーション MBean を削除します。

```
java weblogic.Admin -adminurl AdminHost:7001 -username weblogic
  -password weblogic DELETE -type JDBCConnectionPoolConfig
```

次の例では、ManagedHost という名前のコンピュータで実行されている管理対象サーバにある、すべての JDBCConnectionPool ローカル コンフィグレーション MBean を削除します。

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
  -password weblogic DELETE -type JDBCConnectionPoolConfig
```

GET

MBean プロパティ (属性) と JMX オブジェクト名 (WebLogicObjectName 形式) を表示します。

このコマンドの出力は、次のとおりです。

```
{MBeanName object-name {property1 value} {property2 value} . . .}
{MBeanName object-name {property1 value} {property2 value} . . .}
. . .
```

プロパティと値は名前と値のペアとして表され、各ペアは中括弧で囲まれています。この形式によって、スクリプトが出力を解析しやすくなります。

-pretty を指定した場合、プロパティと値のペアが 1 行に 1 組ずつ表示され、ペアを区切るための中括弧は使用されません。

```
MBeanName: object-name
property1: value
property2: value
.
.
MBeanName: object-name
property1: value
attribute2: value
```

構文

```
java weblogic.Admin [ 接続とユーザ資格の引数 ]
GET [-pretty] {-type mbean_type|-mbean mbean_name}
    [-property property1] [-property property2]...
```

引数	定義
URL	コマンドの対象である WebLogic Server インスタンス。詳細については、 B-51 ページの「サーバの指定」を参照。
mbean_type	指定したタイプのすべての MBean に関する情報を返す。詳細については、 B-50 ページの「MBean タイプの指定」を参照。
mbean_name	MBean の完全修飾オブジェクト名を WebLogicObjectName 形式で指定する。 " <i>domain:Type=type,Location:location,Name=name</i> " 詳細については、WebLogicObjectName の Javadoc を参照。

引数	定義
<code>pretty</code>	プロパティと値のペアを 1 行に 1 組ずつ配置する。
<code>property</code>	一覧表示される MBean プロパティ (属性) の名前。
	注意: この引数を使用してプロパティを指定しない場合、すべてのプロパティが表示される。

例

次の例では、MyPool という接続プールにある JDBCConnectionPool 管理 MBean のすべてのプロパティを表示します。管理 MBean から情報を取得するには、コマンドを管理サーバに接続する必要があります。

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
  -password weblogic GET -pretty -mbean
  MyDomain:Name=MyPool,Type=JDBCConnectionPool
```

コマンドが正常に実行されると、次のような出力 (一部を抜粋) が返されます。

```
-----
MBeanName: "MyDomain:Name=MyPool,Type=JDBCConnectionPool"
  ACLName:
  CachingDisabled:true
  CapacityIncrement: 1
  ConnLeakProfilingEnabled: false
  ConnectionCreationRetryFrequencySeconds: 0
  ConnectionReserveTimeoutSeconds: 10
...

```

次の例では、ドメイン内のすべてのサーバにある JDBCConnectionPoolRuntime MBean のすべてのインスタンスを表示します。

```
java weblogic.Admin -adminurl AdminHost:7001 -username weblogic
  -password weblogic GET -pretty -type JDBCConnectionPoolRuntime
```

次の例では、ポート ManagedHost:8001 をリスンするサーバインスタンスにデプロイされているすべての JDBCConnectionPoolRuntime MBean のすべてのインスタンスを表示します。

```
java weblogic.Admin -url ManagedHost:8001 -username weblogic
  -password weblogic GET -pretty -type JDBCConnectionPoolRuntime
```

INVOKE

1 つまたは複数の MBean の管理操作を呼び出します。WebLogic Server MBean の場合、通常このコマンドを使用して、ほとんどの WebLogic Server MBean が備えている `getAttribute` と `setAttribute` 以外の操作を呼び出します。

構文

```
java weblogic.Admin [ 接続とユーザ資格の引数 ] INVOKE { -type
mbean_type | -mbean mbean_name } -method
methodname [ argument . . . ]
```

引数	定義
<i>URL</i>	コマンドの対象である WebLogic Server インスタンス。詳細については、B-51 ページの「サーバの指定」を参照。
<i>mbean_type</i>	特定のタイプのすべての MBean の操作を呼び出す。詳細については、B-50 ページの「MBean タイプの指定」を参照。
<i>mbean_name</i>	MBean の完全修飾オブジェクト名を <code>WebLogicObjectName</code> 形式で指定する。 "domain:Type=type,Location=location,Name=name" 詳細については、 <code>WebLogicObjectName</code> の Javadoc を参照。
<i>methodname</i>	呼び出すメソッドの名前を指定する。
<i>argument</i>	メソッド呼び出しに渡される引数。 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。 " <i>String1</i> ; <i>String2</i> ; . . . "

例

次の例では、`JDBCConnectionPoolRuntime` MBean の `enable` メソッドを呼び出して、JDBC 接続プールを有効にします。

```
java weblogic.Admin -url AdminHost:7001 -username weblogic
-password weblogic INVOKE
-mbean MyDomain:Location=MyServer,Name=MyPool,
ServerRuntime=MyServer,Type=JDBCConnectionPoolRuntime
-method enable
```

コマンドが正常に実行されると以下が返されます。

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

```
{MBeanName="MyDomain:Location=MyServer,Name=MyPool,ServerRuntime=MyServer,Type=JDBCConnectionPoolRuntime"}
```

Ok

次の例では、すべての `JDBCConnectionPoolRuntime` **MBean** の `enable` メソッドを呼び出して、ドメイン内のすべての **JDBC** 接続プールを有効にします。

```
java weblogic.Admin -adminurl AdminHost:7001 -username weblogic  
-password weblogic INVOKE  
-type JDBCConnectionPoolRuntime -method enable
```

SET

指定されたプロパティ (属性) 値を、コンフィグレーション MBean に対して設定します。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。

このコマンドを管理 MBean に対して使用する場合、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せん。

構文

```
java weblogic.Admin [ 接続とユーザ資格の引数 ]
SET {-type mbean_type|-mbean mbean_name}
    -property property1 property1_value
    [-property property2 property2_value] . . .
```

引数	定義
<i>URL</i>	コマンドの対象である WebLogic Server インスタンス。詳細については、B-51 ページの「サーバの指定」を参照。
<i>mbean_type</i>	特定のタイプのすべての MBean のプロパティを設定する。詳細については、B-50 ページの「MBean タイプの指定」を参照。
<i>mbean_name</i>	MBean の完全修飾オブジェクト名を WebLogicObjectName 形式で指定する。次に例を示す。 "domain:Type=type,Name=name" 詳細については、WebLogicObjectName の Javadoc を参照。
<i>property</i>	設定するプロパティの名前を指定する。

引数	定義
<code>property _value</code>	<p>設定する値。</p> <ul style="list-style-type: none">プロパティによっては、WebLogic Server MBean を指定しなければならない場合がある。その場合は、MBean の完全修飾オブジェクト名を <code>WebLogicObjectName</code> 形式で指定する。次に例を示す。 <code>"domain:Type=type,Name=name"</code> 詳細については、<code>WebLogicObjectName</code> の Javadoc を参照。プロパティ値が MBean 配列の場合は、各 MBean オブジェクト名をセミコロンで区切り、プロパティ値リスト全体を二重引用符で囲む。 <code>"domain:Name=name,Type=type;domain:Name=name,Type=type"</code>プロパティ値が文字列配列の場合は、各文字列をセミコロンで区切り、プロパティ値リスト全体を二重引用符で囲む。 <code>"String1:String2;. . ."</code>プロパティ値が文字列または文字列配列の場合は、以下のいずれかを指定して <code>null</code> 値を設定できる。 <code>-property property-name ""</code> <code>-property property-name</code> たとえば、<code>-property ListenAddress ""</code> と指定しても <code>-property ListenAddress</code> と指定しても、リスンアドレスは <code>null</code> に設定される。プロパティ値にスペースが含まれる場合は、その値を二重引用符で囲む。 <code>"-Da=1 -Db=3"</code> 次に例を示す。 <code>SET -type ServerStart -property Arguments "-Da=1 -Db=3"</code>JDBC 接続プールのプロパティを設定する場合、引数は以下の形式で渡さなければならない。 <code>"user:username;password:password;server:servername"</code>

例

次の例では、`MyServer` というサーバにある `ServerMBean` のローカル コンフィグレーション インスタンスの `StdoutSeverityLevel` プロパティを `64` に設定します。


```
java weblogic.Admin -url http://ManagedHost:8001
  -username weblogic -password weblogic
  SET -mbean
  MyDomain:Location=MyServer,Name=MyServer,Type=ServerConfig
  -property StdoutSeverityLevel 64
```

コマンドが正常に実行されると、サービンスタンスによって次のようなログメッセージが書き込まれます。

```
<Sep 16, 2001 12:11:27 PM EDT> <Info> <Logging> <000000> <Log
messages of every severity will be displayed in the shell console.>
```

コマンドにより、標準出力に ok と出力されます。

次の例では、現在のドメインにある ServerMBean のすべての管理インスタンスの StdoutSeverityLevel プロパティを 64 に設定します。

```
java weblogic.Admin -url http://AdminHost:7001
  -username weblogic -password weblogic
  SET -type Server -property StdoutSeverityLevel 64
```

例 : JDBC 接続プールの割り当て

次の例では、MyPool という JDBC 接続プールを MS1 というサーバに割り当てます。

1. MS1 WebLogic Server インスタンスのオブジェクト名を識別するには、次のコマンドを入力します。

```
java weblogic.Admin -url http://AdminHost:7001
  -username weblogic -password weblogic
  GET -type Server
```

-url 引数で管理サーバが指定されているため、ドメイン内のすべてのサーバインスタンスの Server 管理 MBean が返されます。出力には、MS1 を表す ServerMBean のオブジェクト名が含まれます。

```
MBeanName: "examples:Name=MS1,Type=Server"
  AcceptBacklog: 50
  AdministrationPort: 0
  AutoKillIfFailed: false
.
.
.
```

2. 割り当てる JDBC 接続プールのオブジェクト名を識別するには、次のコマンドを入力します。

```
java weblogic.Admin -url http://AdminHost:7001
  -username weblogic -password weblogic
  GET -type JDBCConnectionPool
```

ドメイン内のすべての JDBC 接続プールの JDBCConnectionPool 管理 MBean が返されます。出力には、MyPool 接続プールのオブジェクト名が含まれます。

```
MBeanName: "examples:Name=MyPool,Type=JDBCConnectionPool"
  ACLName:
  CachingDisabled:true
  CapacityIncrement: 1
.
.
.
```

3. MyPool を MS1 に割り当てるには、次のコマンドを入力します。

```
java weblogic.Admin -url http://AdminHost:7001
  -username weblogic -password weblogic
  SET -mbean "examples:Name=MyPool,Type=JDBCConnectionPool"
  -property Targets "examples:Name=MS1,Type=Server"
```

このタスクを実行するには管理 MBean (JDBCConnectionPoolMBean) を変更しなければならないため、-url オプションにはドメインの管理サーバを指定する必要があります。詳細については、『WebLogic JMX Service プログラマーズ ガイド』の「管理対象リソースをコンフィグレーションするための MBean」を参照してください。

weblogic.Admin コマンドを使用したユーザとグループの管理

WebLogic Security サービスでは、**認証プロバイダ**はユーザまたはシステム プロセスの身元を証明するソフトウェア コンポーネントです。認証プロバイダでは、ID 情報を記憶したり、転送したり、その情報が必要な場合にシステムのさまざまなコンポーネントで利用できるようにしたりします。セキュリティ レルムは、

さまざまな種類の認証プロパティを使用して異なるユーザおよびグループのセットを管理できます。詳細については、『WebLogic Security サービスの開発』の「認証プロバイダ」を参照してください。

weblogic.Admin ユーティリティを使用すると、次に示すタイプの認証プロバイダの操作を呼び出すことができます。

- **WebLogic 認証プロバイダ**
weblogic.management.security.authentication.AuthenticatorMBean。
デフォルトでは、セキュリティ レルムはこの認証プロバイダを使用してユーザとグループを管理します。
- weblogic.management.security.authentication.LDAPAuthenticatorMBean を拡張した LDAP 認証プロバイダ。
- weblogic.security.spi.AuthenticationProvider と任意 Authentication SSPI MBean を拡張したカスタム認証プロバイダ。
詳細については、『WebLogic Security サービスの開発』の「SSPI MBean クリック リファレンス」を参照してください。

以下の節では、weblogic.Admin ユーティリティを使用してユーザとグループを管理するための基本タスクについて説明します。

- B-66 ページの「AuthenticationProvider MBean のオブジェクト名の検索」
- B-66 ページの「ユーザの作成」
- B-67 ページの「グループへのユーザの追加」
- B-67 ページの「ユーザがグループのメンバーであるかどうかの検証」
- B-68 ページの「ユーザが属するグループのリスト」
- B-69 ページの「LDAP サーバでのグループ メンバーシップ検索の制限」

AuthenticationProvider と任意 MBean がサポートするその他のタスクについては、weblogic.management.security.authentication パッケージの Javadoc を参照してください。

AuthenticationProvider MBean のオブジェクト名の検索

認証プロバイダの操作を呼び出すには、そのプロバイダの `AuthenticationProviderMBean` のオブジェクト名を指定する必要があります。

WebLogic 認証プロバイダのオブジェクト名は次のとおりです。

`Security:Name=realmNameDefaultAuthenticator`

ここで、`realmName` はセキュリティ レalmの名前です。たとえば、`Security:Name=myrealmDefaultAuthenticator` です。

LDAP 認証プロバイダのオブジェクト名は次のとおりです。

`Security:Name=realmNameLDAPAuthenticator`

独自の認証プロバイダを作成する場合、次のような規約に従うことをお勧めします。

`Security:Name=realmNameAuthenticatorName`

`Administration Console` を使用して認証プロバイダをレalmに追加する場合、`AuthenticationProviderMBean` 名は上記の推奨規約に従います。

ユーザの作成

ユーザを作成するには、`UserEditorMBean.createUser` メソッドを呼び出します。このメソッドは、セキュリティ レalmの `AuthenticationProvider MBean` によって拡張されます。`createUser` メソッドの Javadoc を参照してください。

このメソッドでは、次の3つの入力パラメータが必要です。

`username password user-description`

パラメータはスペースで区切ります。パラメータにスペースが含まれる場合、そのパラメータを引用符で囲みます。

次の例では、WebLogic 認証プロバイダの `createUser` が呼び出されます。

```
java weblogic.Admin -adminurl localhost:8001 -username weblogic
-password weblogic invoke -mbean
Security:Name=myrealmDefaultAuthenticator
-method createUser my-user1 mypassword "my user"
```

コマンドが成功すると、標準出力に ok が出力されます。

グループへのユーザの追加

ユーザをグループに追加するには、`GroupEditorMBean.addMemberToGroup` メソッドを呼び出します。このメソッドは、セキュリティ レルムの `AuthenticationProvider MBean` によって拡張されます。`addMemberToGroup` メソッドの Javadoc を参照してください。

このメソッドでは、次の 2 つの入力パラメータが必要です。

```
groupname username
```

次の例では、デフォルト認証プロバイダの `addMemberToGroup` が呼び出されます。

```
java weblogic.Admin -adminurl localhost:8001 -username weblogic
-password weblogic invoke -mbean
Security:Name=myrealmDefaultAuthenticator
-method addMemberToGroup Administrators my-user1
```

コマンドが成功すると、標準出力に ok が出力されます。

ユーザがグループのメンバーであるかどうかの検証

ユーザがグループのメンバーであるかどうかを検証するには、`GroupEditorMBean.isMember` メソッドを呼び出します。このメソッドは、セキュリティ レルムの `AuthenticationProvider MBean` によって拡張されます。`isMember` メソッドの Javadoc を参照してください。

このメソッドでは、次の 3 つの入力パラメータが必要です。

```
groupname username boolean
```

`boolean` は、子グループ内を検索するかどうかを指定します。`true` を指定した場合、指定したグループまたはその子グループにメンバーが属していれば `true` が返されます。

次の例では、デフォルト認証プロバイダの `isMember` が呼び出されます。

```
java weblogic.Admin -adminurl localhost:8001 -username weblogic
-passwd weblogic invoke -mbean
Security:Name=myrealmDefaultAuthenticator
-method isMember Administrators weblogic true
```

ユーザがグループのメンバーである場合、標準出力に `true` が出力されます。

ユーザが属するグループのリスト

ユーザまたはグループの含まれるグループのリストを表示するには、`MemberGroupListerMBean.listMemberGroups` メソッドを呼び出します。このメソッドは、セキュリティ レルムの `AuthenticationProvider MBean` によって拡張されます。`listMemberGroups` メソッドの Javadoc を参照してください。

このメソッドでは、次の 1 つの入力パラメータが必要です。

memberUserOrGroupName

memberUserOrGroupName は、既存のユーザまたはグループの名前を指定します。

次の例では、**WebLogic** 認証プロバイダの `listMemberGroups` が呼び出されま

```
java weblogic.Admin -adminurl localhost:8001 -username weblogic
-passwd weblogic invoke -mbean
Security:Name=myrealmDefaultAuthenticator
-method listMemberGroups my-user1
```

このメソッドは、名前を参照するカーソルを返します。

`weblogic.management.utils.NameLister.haveCurrent`、`getCurrentName`、および `advance` メソッドが返されたリスト内で繰り返し実行され、現在のカーソル位置が指し示す名前を検索します。

`weblogic.management.utils.NameLister` インタフェースの Javadoc を参照してください。

LDAP サーバでのグループメンバーシップ検索の制限

グループメンバーシップの検索を無制限に繰り返し行くと、非常に多くの時間がかかり、パフォーマンス上のボトルネックとなってしまいます。グループメンバーシップ検索の限度をコントロールするには、**WebLogic** 認証プロバイダ、**LDAP** 認証プロバイダ、または **AuthenticationProvider** インタフェースを実装する他の認証プロバイダで `GroupMembershipSearching` の値を設定します。

グループ検索を制限するには、**WebLogic** 認証プロバイダまたは **LDAP** 認証プロバイダで `GroupMembershipSearching` の値を `limited` に設定します。`GroupMembershipSearching` 属性では、以下の値を設定できます。

`unlimited` `limited`

`limited` は、ネストされているグループ階層の 1 レベルで検索を行うのか複数レベルで検索を行うのかを指定します。制限付きの検索を指定する場合は、`MaxGroupMembershipSearchLevel` 属性を指定する必要があります。デフォルトは、無制限の検索です。

`MaxGroupMembershipSearchLevel` 属性は、検索するグループメンバーシップのレベル数を指定します。有効な値は 0 と正の整数で、0 の場合は直接のグループメンバーシップだけで検索が行われます。正の整数は、検索するレベル数を指定します。

たとえばグループ A でメンバーシップを検索する場合、0 を指定するとグループ A の直接のメンバーのみが検索されます。グループ B がグループ A のメンバーである場合、グループ B のメンバーは検索の対象になりません。この属性が 1 に設定されている場合、グループ A でのメンバーシップの検索ではグループ A の直接のメンバーと、グループ A の直接のメンバーであるグループのメンバーが返されます。グループ B がグループ A のメンバーである場合は、グループ B のメンバーも検索の対象になります。ただし、グループ C がグループ B のメンバーである場合、グループ C のメンバーは検索の対象になりません。

`GroupMembershipSearching` および `MaxGroupMembershipSearchLevel` の Javadoc を参照してください。

次の例では、**WebLogic** 認証プロバイダの `GroupMembershipSearching` プロパティが取得されます。

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

```
java weblogic.Admin -username xxxx -password yyyy -url  
t3://localhost:7001 get -mbean  
Security:Name=myrealmDefaultAuthenticator -pretty -property  
GroupMembershipSearching -commotype
```

C Ant タスクを使用した WebLogic Server ドメインのコンフィグレーション

以下の節では、WebLogic Ant タスクを使用した WebLogic Server インスタンスの起動と停止の方法、および WebLogic Server ドメインのコンフィグレーション方法を説明します。

- C-1 ページ「Ant タスクを使用したドメインのコンフィグレーションと起動の概要」
- C-2 ページ「wlserver Ant タスクを使用したサーバの起動とドメインの作成」
- C-7 ページ「wlconfig Ant タスクを使用した WebLogic Server ドメインのコンフィグレーション」

Ant タスクを使用したドメインのコンフィグレーションと起動の概要

WebLogic Server は、開発環境で一般的なコンフィグレーション作業を実行する手助けとなる一連のタスクを提供します。それらのコンフィグレーションタスクでは、WebLogic Server インスタンスの起動と停止、および WebLogic Server ドメインの作成とコンフィグレーションを行うことができます。

注意： WebLogic Server Ant タスクは、1.5 より前の Ant バージョンとは互換性がありません。また、WebLogic Server に含まれていない Ant のバージョンを使用する場合は、以降の節の説明に従って build.xml ファイルで適切なタスク定義を指定する必要があります。

他の WebLogic Ant タスクと組み合わせると、カスタム ドメインのあるアプリケーションをデモンストレーションまたはテストする優れた構築スクリプトを作成できます。たとえば、1つの Ant 構築スクリプトで以下のことができます。

- wlservlet Ant タスクを使用して新しい単一サーバ ドメインを作成し、管理サーバを起動する
- wlconfig Ant タスクを使用し、新しいドメインを必要なアプリケーションリソースでコンフィグレーションする
- wldeploy Ant タスクを使用してアプリケーションをデプロイする
- コンパイル済みクライアント アプリケーションを自動的に起動して、製品の機能をデモンストレーションまたはテストする

以降の節では、コンフィグレーション Ant タスク wlservlet および wlconfig の使い方を説明します。wldeploy タスクの詳細については、『WebLogic Server アプリケーションの開発』の「wldeploy Ant タスク」を参照してください。

wlservlet Ant タスクを使用したサーバの起動とドメインの作成

wlservlet Ant タスクの機能

wlservlet Ant タスクを使用すると、WebLogic Server インスタンスの起動、再起動、停止、または接続を行うことができます。そのサーバ インスタンスはコンフィグレーション済み WebLogic Server ドメインにすでに存在している場合がありますが、generateconfig=true 属性を使用して開発用の新しい単一サーバ ドメインを作成することもできます。

Ant スクリプトで使用する場合、wlsrver タスクは指定のサーバが利用可能で接続をリスンするようになるまで制御を返しません。wlsrver を使用してサーバインスタンスを起動した場合、そのサーバ プロセスは Ant VM が終了した後に自動的に終了します。wlsrver タスクを使用して動作中のサーバに接続するだけの場合、サーバ プロセスは Ant が完了した後も動作を続けます。

wlsrver を使用する基本的な手順

wlsrver Ant タスクを使用するには、次の手順を行います。

1. 環境を設定します。

Windows NT では、setWLSEnv.cmd コマンドを実行します。

WL_HOME\server\bin ディレクトリにあります。WL_HOME は WebLogic プラットフォームがインストールされている最上位ディレクトリです。

UNIX では、setWLSEnv.sh コマンドを実行します。WL_HOME/server/bin ディレクトリにあります。WL_HOME は WebLogic プラットフォームがインストールされている最上位ディレクトリです。

注意： wlsrver タスクは、WebLogic Server に付属している Ant のバージョンで事前定義されています。独自の Ant でこのタスクを使用する場合は、ビルド ファイルで次のタスク定義を追加してください。

```
<taskdef name="wlsrver"  
  classname="weblogic.ant.taskdefs.management.WLServer"/>
```

2. サーバの起動、停止、再起動、または接続を行うための、wlsrver タスクの呼び出しを構築スクリプトに追加します。wlsrver の属性とデフォルト動作については、C-5 ページ「wlsrver Ant タスクのリファレンス」を参照してください。
3. ステージング ディレクトリで ant と入力し、必要であればこのコマンドにターゲットの引数を渡して、build.xml ファイルで指定された Ant タスク (1 つまたは複数) を実行します。

```
prompt> ant
```

ant -verbose を使用すると、wlsrver タスクからより詳しいメッセージを取得できます。

wlserver のサンプル build.xml ファイル

次に、すべてでデフォルト値を使用してカレントディレクトリでサーバを起動する最小限の wlserver ターゲットを示します。

```
<target name="wlserver-default">
  <wlserver/>
</target>
```

次のターゲットは、指定された接続パラメータ、およびユーザ名とパスワードの組み合わせを使用して、既存の動作中サーバに接続します。

```
<target name="connect-server">
  <wlserver host="127.0.0.1" port="7001" username="weblogic"
password="weblogic" action="connect"/>
</target>
```

次のターゲットは、config サブディレクトリにコンフィグレーションされた WebLogic Server インスタンスを起動します。

```
<target name="start-server">
  <wlserver dir="./config" host="127.0.0.1" port="7001"
action="start"/>
</target>
```

次のターゲットは、空のディレクトリに新しい単一サーバドメインを作成し、ドメインのサーバインスタンスを起動します。

```
<target name="new-server">
  <delete dir="./tmp"/>
  <mkdir dir="./tmp"/>
  <wlserver dir="./tmp" host="127.0.0.1" port="7001"
generateConfig="true" username="weblogic" password="weblogic"
action="start"/>
</target>
```

wlsrserver Ant タスクのリファレンス

次の表では、wlsrserver Ant タスクの属性について説明します。

表 13-2 wlsrserver Ant タスクの属性

属性	説明	データ型	必須 / 省略可能
policy	WebLogic Server ドメインのセキュリティ ポリシー ファイルのパス。この属性は、サーバインスタンスを起動する場合にのみ使用する。	ファイル	省略可能
dir	ドメイン コンフィグレーションを保持するパス (c:\bea\user_projects\mydomain など)。デフォルトでは、wlsrserver はカレント ディレクトリを使用する。	ファイル	省略可能
beahome	BEA ホーム ディレクトリのパス (c:\bea など)。	ファイル	省略可能
weblogic_home	WebLogic Server のインストール ディレクトリのパス (c:\bea\weblogic700 など)。	ファイル	省略可能
servername	起動、再起動、または接続するサーバの名前。	文字列	省略可能
domainname	サーバがコンフィグレーションされている WebLogic Server ドメインの名前。	文字列	省略可能
adminserverurl	ドメイン内の管理サーバにアクセスするための URL。この属性は、ドメインで管理対象サーバを起動する場合は必須。	文字列	管理対象サーバを起動する場合は必須
username	管理者アカウントのユーザ名。username 属性と password 属性の両方を省略すると、wlsrserver は boot.properties ファイルから暗号化されたユーザ名とパスワードの値を取得しようとする。	文字列	省略可能
password	管理者アカウントのパスワード。username 属性と password 属性の両方を省略すると、wlsrserver は boot.properties ファイルから暗号化されたユーザ名とパスワードの値を取得しようとする。	文字列	省略可能

C Ant タスクを使用した WebLogic Server ドメインのコンフィグレーション

表 13-2 wlsver Ant タスクの属性

属性	説明	データ型	必須 / 省略可能
pkpassword	SSL プライベート キー ファイルを復号化するためのプライベート キー パスワード。	文字列	省略可能
timeout	wlsver がサーバの起動を待機する最大時間 (秒単位)。動作中のサーバに接続するときの最大待機時間も指定する。	long	省略可能
productionmodeenabled	サーバインスタンスを開発モードとプロダクションモードのどちらで起動するかを指定する。	boolean	省略可能
host	サーバインスタンスがリスンする DNS 名または IP アドレス。	文字列	省略可能
port	サーバインスタンスがリスンする TCP ポートの番号。	int	省略可能
generateconfig	wlsver が、指定されたサーバの新しいドメインを作成するかどうかを指定する。この属性はデフォルトでは false。	boolean	省略可能
action	wlsver が実行するアクションを指定する (startup、shutdown、reboot、または connect)。 shutdown アクションを任意指定の forceshutdown 属性と共に使用すると、強制停止を実行できる。	文字列	省略可能
failonerror	WebLogic Server Ant タスクで使用されるグローバル属性。ビルド中にエラーが発生した場合、タスクが失敗するかどうかを指定する。この属性は、デフォルトで true に設定される。	Boolean	省略可能

表 13-2 wlservlet Ant タスクの属性

属性	説明	データ型	必須 / 省略可能
forceshutdown	この省略可能な属性は、強制停止を実行するために action="shutdown" 属性と一緒に使用する。次に例を示す。 <pre><wlservlet host="\${wls.host}" port="\${port}" username="\${wls.username}" password="\${wls.password}" action="shutdown" forceshutdown="true"/></pre>	Boolean	省略可能

wlconfig Ant タスクを使用した WebLogic Server ドメインのコンフィグレーション

wlconfig Ant タスクの機能

wlconfig Ant タスクを使用すると、動作中の管理サーバインスタンス上でコンフィグレーション MBean の作成、クエリ、変更を行うことで、WebLogic Server ドメインをコンフィグレーションできます。特に、wlconfig では以下のことを実行できます。

- 新しい MBean を作成し、必要に応じて新しい MBean オブジェクト名を Ant プロパティに格納する
- 管理サーバ上で使用可能な指定した MBean に属性値を設定する
- MBean を作成するコマンドの内部に属性を設定するコマンドをネストすることで、MBean の作成と属性の設定を 1 回の手順で行う
- MBean のクエリを実行し、必要に応じてクエリの結果を Ant プロパティ参照内に格納する
- MBean のクエリを実行し、一致するすべての結果に対して属性を設定する

- 作成コマンドを他の作成コマンドの内部にネストすることで、MBean 間の親子関係を確立する

wlconfig を使用する基本的な手順

1. コマンド シェルで環境を設定します。詳細については、C-3 ページ「wlserver を使用する基本的な手順」を参照してください。

注意： wlconfig タスクは、WebLogic Server に付属している Ant のバージョンで事前定義されています。独自の Ant でこのタスクを使用する場合は、ビルド ファイルで次のタスク定義を追加してください。

```
<taskdef name="wlconfig"
  classname="weblogic.ant.taskdefs.management.WLConfig"/>
```

2. wlconfig は通常、Ant タスクのコンテキストで作成された新しい WebLogic Server ドメインをコンフィグレーションするために wlserver と一緒に使用します。wlconfig を使用してそのようなドメインをコンフィグレーションする場合は、まず wlserver の属性を使用して新しいドメインを作成し、WebLogic Server インスタンスを起動します。
3. ドメインの管理サーバに接続するための、wlconfig タスクの最初の呼び出しを追加します。次に例を示します。

```
<target name="doconfig">
  <wlconfig url="t3://localhost:7001" username="weblogic"
    password="weblogic">
</target>
```

4. ドメインをコンフィグレーションするための、ネストされた create、delete、get、set、および query 要素を追加します。
5. ステージング ディレクトリで ant と入力し、必要であればこのコマンドにターゲットの引数を渡して、build.xml ファイルで指定された Ant タスク (1 つまたは複数) を実行します。

```
prompt> ant doconfig
```

ant -verbose を使用すると、wlconfig タスクからより詳しいメッセージを取得できます。

wlconfig のサンプル build.xml ファイル

完全な例

この例は、wlserver を使用して新しいドメインを作成し、wlconfig でさまざまなドメイン コンフィグレーション タスクを実行する 1 つの build.xml ファイルを示しています。

スクリプトは、新しいドメインを作成することから始まります。

```
<target name="sample.config">
  mkdir dir="config"/>
  <wlserver username="a" password="a" servername="SampleServer"
    domainname="sample" dir="config" host="localhost"
port="7000"
generateconfig="true"/>
```

次に、新しいサーバにアクセスして wlconfig タスクを開始します。

```
<wlconfig url="t3://localhost:7000" username="a" password="a">
```

wlconfig タスクの中では、query 要素がクエリを実行してサーバ MBean のオブジェクト名を取得し、その MBean を `${sampleserver}` Ant プロパティに格納します。

```
<query domain="sample" type="Server" name="SampleServer"
property="sampleserver"/>
```

create 要素を使用してドメインに新しい JDBC 接続プールが作成され、そのオブジェクト名が `${samplepool}` Ant プロパティに格納されます。create にネストされている set 要素は、新しく作成された Mbean の属性を設定します。新しいプールは、前のクエリで設定された `${sampleserver}` Ant プロパティを使用してサーバに割り当てられます。

```
<create type="JDBCConnectionPool" name="SamplePool"
property="samplepool">
  <set attribute="CapacityIncrement" value="1"/>
  <set attribute="DriverName"
value="com.pointbase.jdbc.jdbcUniversalDriver"/>
  <set attribute="InitialCapacity" value="1"/>
  <set attribute="MaxCapacity" value="10"/>
  <set attribute="Password" value="samplename"/>
  <set attribute="Properties" value="user=samplepwd"/>
  <set attribute="RefreshMinutes" value="0"/>
  <set attribute="ShrinkPeriodMinutes" value="15"/>
  <set attribute="ShrinkingEnabled" value="true"/>
  <set attribute="TestConnectionsOnRelease" value="false"/>
```

C Ant タスクを使用した WebLogic Server ドメインのコンフィグレーション

```
<set attribute="TestConnectionsOnReserve" value="false"/>
<set attribute="URL"
  value="jdbc:pointbase:server://localhost/demo"/>
<set attribute="Targets" value="${sampleserver}"/>
</create>
```

次に、上で作成された JDBC 接続プールを使用して JDBC TX データソースが作成されます。

```
<create type="JDBCTxDataSource" name="Medical Records Tx
DataSource">
  <set attribute="JNDIName" value="SampleTxDataSource"/>
  <set attribute="PoolName" value="SamplePool"/>
  <set attribute="Targets" value="${sampleserver}"/>
</create>
```

ネストされた set 要素を使用して、新しい JMS 接続ファクトリが作成されます。

```
<create type="JMSConnectionFactory" name="Queue">
  <set attribute="JNDIName"
value="jms/QueueConnectionFactory"/>
  <set attribute="XAServerEnabled" value="true"/>
  <set attribute="Targets" value="${sampleserver}"/>
</create>
```

新しい JMS JDBC ストアが、SamplePool を使用して作成されます。

```
<create type="JMSJDBCStore" name="SampleJDBCStore"
  property="samplejdbcstore">
  <set attribute="ConnectionPool" value="${samplepool}"/>
  <set attribute="PrefixName" value="Sample"/>
</create>
```

新しい JMS サーバを作成するときには、ネストされた create 要素を使用して JMS キュー (JMS サーバの子) を作成します。

```
<create type="JMSServer" name="SampleJMSServer">
  <set attribute="Store" value="${samplejdbcstore}"/>
  <set attribute="Targets" value="${sampleserver}"/>
  <create type="JMSQueue" name="Registration Queue">
    <set attribute="JNDIName"
value="jms/REGISTRATION_MDB_QUEUE"/>
  </create>
</create>
```

このスクリプトは、新しいメールセッションと起動クラスを作成します。

```
<create type="MailSession" name="Medical Records Mail Session">
  <set attribute="JNDIName" value="mail/SampleMailSession"/>
  <set attribute="Properties"
  value="mail.user=joe;mail.host=mail.mycompany.com"/>
  <set attribute="Targets" value="${sampleserver}"/>
</create>
```

```
<create type="StartupClass" name="StartBrowser">
  <set attribute="Arguments" value="port=${listenport}"/>
  <set attribute="ClassName"
    value="com.bea.sample.startup.StartBrowser"/>
  <set attribute="FailureIsFatal" value="false"/>
  <set attribute="Notes" value="Automatically starts a browser
on server boot."/>

  <set attribute="Targets" value="${sampleserver}"/>
</create>
```

最後に、**WebServer MBean** が取得され、ネストされた **set** 要素を使用してログファイル名が設定されます。

```
<query domain="sample" type="WebServer" name="SampleServer">
  <set attribute="LogFileName" value="logs/access.log"/>
</query>
</wlconfig>
</target>
```

クエリと削除の例

query 要素の内部にネストされる場合、その **query** 要素では **MBean** 名を指定する必要はありません。

```
<target name="queryDelete">
  <wlconfig url="${adminurl}" username="${user}"
password="${pass}"
  failonerror="false">
    <query query="${wlsdomain}:Name=MyNewServer2,*"
      property="deleteQuery">
      <delete/>
    </query>
  </wlconfig>
</target>
```

複数の属性値の設定例

set 要素では、属性値を **Ant** プロパティに格納された複数のオブジェクト名に設定できます。たとえば、次のターゲットは 2 つのサーバのオブジェクト名を別々の **Ant** プロパティに格納し、それらのプロパティを使用して両方のサーバを新しい **JDBC** 接続プールのターゲット属性に割り当てます。

```
<target name="multipleJDBCTargets">
  <wlconfig url="${adminurl}" username="${user}"
password="${pass}">
    <query domain="mydomain" type="Server" name="MyServer"
      property="myserver"/>
    <query domain="mydomain" type="Server" name="OtherServer"
```

```
        property="otherserver" />
        <create type="JDBCConnectionPool" name="sqlpool"
property="sqlpool">
        <set attribute="CapacityIncrement" value="1" />
[.....]
        <set attribute="Targets"
value="\${myserver};\${otherserver}" />
        </create>
    </wlconfig>
</target>
```

wlconfig Ant タスクのリファレンス

主な属性

次の表では、wlconfig Ant タスクの主な属性について説明します。

表 13-3 wlconfig Ant タスクの主な属性

属性	説明	データ型	必須 / 省略可能
url	ドメインの管理サーバの URL。	文字列	必須
username	管理者アカウントのユーザ名。	文字列	省略可能

表 13-3 wlconfig Ant タスクの主な属性

属性	説明	データ型	必須 / 省略可能
password	<p>管理者アカウントのパスワード。</p> <p>プレーンテキストのパスワードがビルドファイルや ps などのプロセス ユーティリティに表示されないようにするには、まず <code>weblogic.Admin STOREUSERCONFIG</code> コマンドを使用して有効なユーザ名と暗号化されたパスワードをコンフィグレーションファイルに格納する。次に、Ant ビルドファイルで <code>username</code> 属性と <code>password</code> 属性を両方とも省略する。これらの属性を省略すると、wlconfig はデフォルトのコンフィグレーションファイルから取得した値を使用してログインを試行する。</p> <p>デフォルト以外のコンフィグレーションファイルおよびキーファイルからユーザ名とパスワードを取得するには、wlconfig で <code>userconfigfile</code> 属性と <code>userkeyfile</code> 属性を使用する。</p>	文字列	省略可能
failonerror	<p>WebLogic Server Ant タスクで使用されるグローバル属性。ビルド中にエラーが発生した場合、タスクが失敗するかどうかを指定する。この属性は、デフォルトで <code>true</code> に設定される。</p>	Boolean	省略可能
userconfigfile	<p>管理用のユーザ名とパスワードを取得するために使用するユーザ コンフィグレーションファイルの位置を指定する。このオプションは、プレーンテキストのパスワードをインラインまたは ps などのプロセスレベルのユーティリティで表示したくない場合に、ビルドファイルで <code>username</code> 属性および <code>password</code> 属性の代わりに使用する。</p> <p><code>userconfigfile</code> 属性を指定する前に、まず、<code>weblogic.Admin STOREUSERCONFIG</code> コマンドを使用してファイルを生成する必要がある。</p>	ファイル	省略可能

表 13-3 wlconfig Ant タスクの主な属性

属性	説明	データ型	必須 / 省略可能
userkeyfile	ユーザ コンフィグレーション ファイル (userconfigfile 属性) に格納されたユーザ名とパスワードの情報を暗号化および復号化するために使用するユーザ キー ファイルの位置を指定する。userkeyfile 属性を指定する前に、まず、weblogic.Admin STOREUSERCONFIG コマンドを使用してファイルを生成する必要がある。	ファイル	省略可能

ネストされた要素

wlconfig にも、コンフィグレーション オプションを指定するためにネストできる要素があります。

- create
- delete
- set
- get
- query

create

create 要素は、WebLogic Server ドメインで新しい MBean を作成します。wlconfig タスクでは、create 要素をいくつでも使用できます。

create 要素では、新しく作成された MBean の属性を設定する set 要素をいくつでもネストできます。create 要素では、子 MBean を作成する追加の create 要素をネストすることもできます。

wlconfig Ant タスクを使用した WebLogic Server ドメインのコンフィグレーション

`create` 要素には以下の属性があります。

表 13-4 `create` 要素の属性

属性	説明	データ型	必須 / 省略可能
<code>name</code>	作成する新しい MBean オブジェクトの名前。	文字列	省略可能 (省略すると、 <code>wlconfig</code> がデフォルト名を提供する)
<code>type</code>	MBean の型。	文字列	必須
<code>property</code>	新しく作成された MBean のオブジェクト名を保持する省略可能な Ant プロパティの名前。	文字列	省略可能

delete

`delete` 要素は、WebLogic Server ドメインから既存の MBean を削除します。`delete` の属性は 1 つです。

表 13-5 `delete` 要素の属性

属性	説明	データ型	必須 / 省略可能
<code>mbean</code>	削除する MBean のオブジェクト名。	文字列	<code>delete</code> 要素が <code>wlconfig</code> タスクの直接の子である場合は必須。 <code>query</code> 要素にネストされている場合は省略可能

C Ant タスクを使用した WebLogic Server ドメインのコンフィグレーション

set

set 要素は、指定された MBean、新しく作成された MBean、またはクエリで取得された MBean の MBean 属性を設定します。set 要素は、wlconfig タスクの直接の子として使用するか、create または query 要素にネストすることができます。

set 要素には以下の属性があります。

表 13-6 set 要素の属性

属性	説明	データ型	必須 / 省略可能
attribute	設定する MBean 属性の名前。	文字列	必須
value	指定した MBean 属性に設定する値。 引用符で値のリスト全体を区切って、セミコロンでオブジェクト名を分ければ、Ant プロパティに格納された複数のオブジェクト名を値として指定できる。C-11 ページ「複数の属性値の設定例」を参照。	文字列	必須
mbean	値が設定される MBean のオブジェクト名。set 要素が wlconfig タスクの直接の子である場合のみ必須。create または query 要素のコンテキストに set 要素がネストされている場合は省略可能。	文字列	set 要素が wlconfig タスクの直接の子である場合のみ必須
domain	この属性は、セキュリティ MBean およびサードパーティ SPI MBean の JMX ドメイン名を指定する。ドメインが WebLogic Server ドメインに対応しているため、管理 MBean では省略可能。	文字列	省略可能

get

get 要素は、WebLogic Server ドメインの MBean から属性値を取得します。wlconfig タスクでは、get 要素をいくつでも使用できます。

get 要素には以下の属性があります。

表 13-7 get 要素の属性

属性	説明	データ型	必須 / 省略可能
attribute	値を取得する MBean 属性の名前。	文字列	必須
property	取得した MBean 属性の値を保持する Ant プロパティの名前。	文字列	必須
mbean	属性値を取得する MBean のオブジェクト名。	文字列	必須

query

query 要素は、検索パターンに一致する MBean を見つけます。query をネストした set 要素または delete 要素と一緒に使用すると、結果セットのすべての MBean で設定または削除操作を行うことができます。

wlconfig では、query 要素をいくつでもネストできます。

query には以下の属性があります。

表 13-8 query 要素の属性

属性	説明	データ型	必須 / 省略可能
domain	MBean を検索する WebLogic Server ドメインの名前。	文字列	省略可能
type	クエリを実行する MBean の型。	文字列	省略可能
name	クエリを実行する MBean の名前。	文字列	省略可能
pattern	JMX クエリ パターン。	文字列	省略可能
property	クエリ結果を格納する省略可能な Ant プロパティの名前。	文字列	省略可能

表 13-8 query 要素の属性

属性	説明	データ型	必須 / 省略可能
domain	この属性は、セキュリティ MBean およびサードパーティ SPI MBean の JMX ドメイン名を指定する。ドメインが WebLogic Server ドメインに対応しているため、管理 MBean では省略可能。	文字列	省略可能

D WebLogic SNMP エージェント コマンドライン リファレンス

WebLogic Server は、Simple Network Management Protocol (SNMP) を使用してエンタープライズ全体の管理システムと通信できます。WebLogic 管理データの収集、管理データの SNMP 通信モジュールへの変換 (トラップ通知)、およびサードパーティ製 SNMP 管理システムへのトラップ通知の転送を行う WebLogic Server サブシステムを WebLogic SNMP エージェントと呼びます。WebLogic SNMP エージェントは、管理サーバで実行し、ドメイン内のすべての管理対象サーバからの情報収集に使用します。

WebLogic SNMP エージェントのコマンドラインインタフェースを使用すると以下を実行できます。

- WebLogic Server MIB 内に管理対象オブジェクトとしてエクスポートされている WebLogic Server 属性の値を取得する。
- WebLogic Server トラップを生成および受信する。

以下の節では、WebLogic SNMP エージェントをコマンドラインインタフェースで操作する方法について説明します。

- D-2 ページの「SNMP コマンドラインインタフェースに必要な環境および構文」
- D-4 ページの「WebLogic Server 属性の値を取得するためのコマンド」
- D-11 ページの「トラップをテストするためのコマンド」

WebLogic Server での SNMP の使用についての詳細は、『SNMP 管理ガイド』を参照してください。

SNMP コマンドライン インタフェースに必要な環境および構文

WebLogic SNMP エージェントのコマンドライン インタフェースを使用するには、あらかじめ環境を設定し、以下の節で説明するコマンド構文を理解しておく必要があります。

環境

WebLogic SNMP エージェントのコマンドライン インタフェースを使用するための環境を設定するには次の手順に従います。

1. 『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。
2. **Administration Console** オンライン ヘルプの「**WebLogic SNMP エージェントの設定**」の説明に従って、**WebLogic SNMP エージェント**を有効にします。

注意: `snmpv1trap` コマンドおよび `snmptrapd` コマンドを使用する際は、**SNMP エージェント**を有効にしておく必要はありません。

3. コマンド プロンプト (シェル) を開き、次のスクリプトを呼び出します。

```
WL_HOME\server\bin\setWLSEnv.sh (Windows の場合は setWLSEnv.cmd)
```

`WL_HOME` は **WebLogic Server** のインストール ディレクトリです。

スクリプトを実行すると、サポートされる **JDK** がシェルの `PATH` 環境変数に追加され、**WebLogic Server** クラスが `CLASSPATH` 変数に追加されます。

共通の引数

すべての **WebLogic SNMP エージェント コマンド**は次の形式をとります。

```
java command-name arguments
```

表 D-1 に、ほとんどの WebLogic SNMP エージェント コマンドで共通の引数を示します。

表 D-1 共通のコマンドライン引数

引数	定義
-d	コマンドの出力に、デバッグ情報およびパケット ダンプを含める。
-c <i>snmpCommunity</i> [@ <i>server_name</i> @ <i>domain_name</i>]	<p>WebLogic SNMP エージェントで SNMP データの保護、および対話するオブジェクトをホストするサーバインスタンスの指定に使用するコミュニティ名を指定する。</p> <p>管理サーバ上のオブジェクトの値を要求するには次のように指定する。</p> <p><i>snmpCommunity</i></p> <p><i>snmpCommunity</i> は SNMP エージェントをコンフィグレーションした際に [コミュニティ プレフィックス] フィールドに設定した SNMP コミュニティ名 (Administration Console オンラインヘルプの「WebLogic SNMP エージェントの設定」を参照)。</p> <p>管理対象サーバ上のオブジェクトの値を要求するには次のように指定する。</p> <p><i>snmpCommunity@server_name</i></p> <p><i>server_name</i> は管理対象サーバの名前。</p> <p>ドメイン内のすべてのサーバインスタンスのオブジェクトの値を要求するには、次の形式でコミュニティ文字列を送信する。</p> <p><i>snmpCommunity@domain_name</i></p> <p>この値を指定しない場合は、-c public を指定したものと見なされ、管理サーバ上のオブジェクトの値が取得される。</p>
-p <i>snmpPort</i>	<p>WebLogic SNMP エージェントがリクエストをリスンするポート番号を指定する。</p> <p>この値を指定しない場合は、-p 161 を指定したものと見なされる。</p>
-t <i>timeout</i>	<p>コマンドが SNMP エージェントへの接続が完了するのを待機するミリ秒数を指定する。</p> <p>この値を指定しない場合は、-t 5000 を指定したものと見なされる。</p>
-r <i>retries</i>	<p>コマンドが SNMP エージェントへの接続を再試行する回数を指定する。</p> <p>この値を指定しない場合、接続は再試行されない。</p>

表 D-1 共通のコマンドライン引数

引数	定義
<i>host</i>	WebLogic Server 管理サーバをホストするコンピュータの DNS 名または IP アドレスを指定する。WebLogic SNMP エージェントはこのコンピュータで実行される。

WebLogic Server 属性の値を取得するためのコマンド

表 D-2 に、WebLogic Server MIB にエクスポートされている WebLogic Server MBean 属性の値を取得するコマンドの概要を示します。

表 D-2 WebLogic Server 属性の値を取得するためのコマンドの概要

コマンド	説明
<i>snmpwalk</i>	MIB ツリーで指定したノードの下にあるすべての管理対象オブジェクトの再帰的なリストを返す。 D-5 ページの「snmpwalk」を参照。
<i>snmpgetnext</i>	指定した OID の直後にある管理対象オブジェクトの記述を返す。 D-7 ページの「snmpgetnext」を参照。
<i>snmpget</i>	1 つまたは複数のオブジェクト インスタンス OID に対応する管理対象オブジェクトの記述を返す。 D-10 ページの「snmpget」を参照。

snmpwalk

MIB ツリーで指定したノードの下にあるすべての管理対象オブジェクトの再帰的なリストを返します。

オブジェクト タイプの **OID** を指定した場合は、そのタイプのすべてのインスタンスと、その子オブジェクト タイプのすべてのインスタンスのリストが返されます。

たとえば、**MBean** に対応するオブジェクト タイプの **OID** を指定すると、**MBean** のすべてのインスタンスと、**MBean** 内の属性のすべてのインスタンスの記述が返されます。

WebLogic Server MIB ツリーを表示するには、『SNMP MIB リファレンス』を参照してください。MIB の構造およびオブジェクト識別子 (OID) の詳細については、『SNMP 管理ガイド』の「WebLogic 用の SNMP MIB」を参照してください。

構文

```
java snmpwalk [-d] [-c snmpCommunity] [-p snmpPort]
               [-t timeout] [-r retries] host OID
```

引数	定義
<i>OID</i>	オブジェクト値の再帰的なリストを取得するノードのオブジェクト ID を指定する。 指定する値は「.」で始める。「.」を指定しないと、WebLogic Server MIB ではなく、標準の MIB (.1.3.6.1.2.1) への相対参照と見なされる。

上の表に示したコマンド引数の詳細については、D-3 ページの表 D-1 「共通のコマンドライン引数」を参照してください。

例

次の例では、管理サーバでホストされている `ServerRuntimeMBean` インスタンスのすべての属性を返します。`OID .1.3.6.1.4.1.140.625.360` は、WebLogic MIB 内の `serverRuntimeTable` オブジェクト タイプを参照しています。

```
java snmpwalk localhost .1.3.6.1.4.1.140.625.360
```

Examples Server を実行しているコンピュータからこのコマンドを呼び出すと、次のような出力 (一部を抜粋) が返されます。出力には、`serverRuntimeTable` オブジェクトの下にある各属性インスタンスの完全 **OID** が含まれています。

```
Object ID:
.1.3.6.1.4.1.140.625.360.1.1.32.101.98.52.50.55.97.53.101.55.101.
56.97.51.98.97.52.99.97.57.53.100.51.51.98.102.51.98.57.48.98.51.
55
STRING: eb427a5e7e8a3ba4ca95d33bf3b90b37
```

```
Object ID:
.1.3.6.1.4.1.140.625.360.1.5.32.101.98.52.50.55.97.53.101.55.101.
56.97.51.98.97.52.99.97.57.53.100.51.51.98.102.51.98.57.48.98.51.
55
STRING: ServerRuntime:examplesServer
```

```
Object ID:
.1.3.6.1.4.1.140.625.360.1.10.32.101.98.52.50.55.97.53.101.55.101
.56.97.51.98.97.52.99.97.57.53.100.51.51.98.102.51.98.57.48.98.51
.55
STRING: ServerRuntime
```

```
Object ID:
.1.3.6.1.4.1.140.625.360.1.15.32.101.98.52.50.55.97.53.101.55.101
.56.97.51.98.97.52.99.97.57.53.100.51.51.98.102.51.98.57.48.98.51
.55
STRING: examplesServer
...
```

次の例では、`examples` ドメイン内のすべてのサーバの名前を取得します。このサンプル コマンドで指定した **OID** は、**WebLogic Server MIB** が `serverRuntimeName` オブジェクトタイプに割り当てた数値です。

```
java snmpwalk -c public@examples localhost
.1.3.6.1.4.1.140.625.360.1.15
```

次の例では、`MS1` という管理対象サーバでホストされている `ServerRuntimeMBean` インスタンスのすべての属性を返します。**OID** `.1.3.6.1.4.1.140.625.360` は、**WebLogic MIB** 内の `serverRuntimeTable` オブジェクトを参照しています。

```
java snmpwalk -c public@MS1 localhost .1.3.6.1.4.1.140.625.360
```


snmpgetnext

指定した 1 つまたは複数の OID の直後にある管理対象オブジェクトの記述を返します。

snmpwalk コマンドでは再帰的なリストが返されましたが、このコマンドでは、指定した OID の次の OID を持つ 1 つの管理対象オブジェクトの記述を返します。複数の snmpgetnext コマンドを連続した文字列にすることで、snmpwalk コマンドと同じ結果が得られます。

オブジェクトタイプを指定すると、そのオブジェクトタイプのインスタンスがいくつ存在するかに関係なく、そのタイプの最初のインスタンスが返されます。

WebLogic Server MIB ツリーを表示するには、『SNMP MIB リファレンス』を参照してください。MIB の構造およびオブジェクト識別子 (OID) の詳細については、『SNMP 管理ガイド』の「WebLogic 用の SNMP MIB」を参照してください。

構文

```
java snmpgetnext [-d] [-c snmpCommunity] [-p snmpPort]
                 [-t timeout] [-r retries] host OID [OID]...
```

引数	定義
OID [OID]...	1 つまたは複数のオブジェクト ID を指定する。オブジェクトタイプまたはオブジェクトインスタンスの OID を指定できる。 指定する値は「.」で始める。「.」を指定しないと、WebLogic Server MIB ではなく、標準の MIB (.1.3.6.1.2.1) への相対参照と見なされる。

上の表に示したコマンド引数の詳細については、D-3 ページの表 D-1 「共通のコマンドライン引数」を参照してください。

例

次の例では、管理サーバにデプロイされているアプリケーションの名前を取得します。サンプル コマンドで指定した OID は applicationRuntimeName オブジェクトタイプの OID で、applicationRuntime MBean の Name 属性を表します。

```
java snmpgetnext localhost .1.3.6.1.4.1.140.625.105.1.15
```

このコマンドを呼び出すと、次のような出力が返されます。

```
Response PDU received from /127.0.0.1, community: public
Object ID:
.1.3.6.1.4.1.140.625.105.1.15.32.49.102.98.97.100.97.102.99.57.48
.50.102.48.98.53.54.100.100.49.54.50.54.99.54.99.49.97.97.98.53.
100.97
STRING: MyServer_uddiexplorer
```

管理サーバにデプロイされているアプリケーションが他にもあるかどうかを識別するには、最初の `snmpgetnext` コマンドの出力を次の `snmpgetnext` コマンドの入力として使用します。

```
java snmpgetnext localhost
.1.3.6.1.4.1.140.625.105.1.15.32.49.102.98.97.100.97.102.99.57.48
.50.102.48.98.53.54.100.100.49.54.50.54.99.54.99.49.97.97.98.53.
100.97
```

このコマンドを呼び出すと、次のような出力が返されます。

```
Response PDU received from /127.0.0.1, community: public
Object ID:
.1.3.6.1.4.1.140.625.105.1.15.32.54.98.49.101.57.56.54.98.98.50.
57.100.54.55.48.100.56.98.101.101.97.55.48.53.57.99.49.51.56.98.
97.99
STRING: MyServer_StartupEAR
```

次の例では、2つの **OID** を指定して、管理サーバにデプロイされているアプリケーションの名前と **JDBC** 接続プールの名前を取得します。サンプルコマンドで指定した **OID** は、**ApplicationRuntime MBean** の **Name** 属性を表す `applicationRuntimeName` オブジェクトタイプの **OID** と、**JDBCConnectionPoolRuntimeMBean** の **Name** 属性を表す `jdbcConnectionPoolRuntimeName` の **OID** です。

```
java snmpgetnext localhost .1.3.6.1.4.1.140.625.105.1.15
.1.3.6.1.4.1.140.625.190.1.15
```

このコマンドを呼び出すと、次のような出力が返されます。

```
Response PDU received from /127.0.0.1, community: public
Object ID:
.1.3.6.1.4.1.140.625.105.1.15.32.49.102.98.97.100.97.102.99.57.48
.50.102.48.98.53.54.100.100.49.54.50.54.99.54.99.49.97.97.98.53.
100.97
STRING: MyServer_uddiexplorer
Object ID:
.1.3.6.1.4.1.140.625.190.1.15.32.53.53.49.48.50.55.52.57.57.49.99
.102.55.48.98.53.50.54.100.48.100.53.53.52.56.49.57.49.49.99.99.
```

99
STRING: MyPool

snmpget

1つまたは複数のオブジェクトインスタンスの値を取得します。このコマンドでは、オブジェクトタイプの **OID** は指定できません。

構文

```
java snmpget [-d] [-c snmpCommunity] [-p snmpPort]  
              [-t timeout] [-r retries] host object-instance-OID  
              [object-instance-OID]...
```

引数	定義
<i>object-instance-OID</i> [<i>object-instance-OID</i>]...	オブジェクト インスタンスのオブジェクト ID。このコマンドでは、オブジェクト タイプの OID は指定できない。 指定する値は「.」で始める。「.」を指定しないと、 WebLogic Server MIB ではなく、標準の MIB への相対参照と見なされる。

例

次の例では、管理サーバの `serverRuntimeState` および `serverRuntimeListenPort` 属性インスタンスの値を取得します。

```
java snmpget localhost  
.1.3.6.1.4.1.140.625.360.1.60.32.102.100.48.98.101.102.100.99.102  
.52.98.97.48.48.49.102.57.53.51.50.100.102.53.55.97.101.52.56.99  
.99.97.99  
.1.3.6.1.4.1.140.625.360.1.35.32.102.100.48.98.101.102.100.99.102  
.52.98.97.48.48.49.102.57.53.51.50.100.102.53.55.97.101.52.56.99.  
99.97.99
```

このコマンドを呼び出すと、次のような出力が返されます。

```
Response PDU received from /127.0.0.1, community: public  
Object ID:  
.1.3.6.1.4.1.140.625.360.1.60.32.102.100.48.98.101.102.100.99.102  
.52.98.97.48.48.49.102.57.53.51.50.100.102.53.55.97.101.52.56.99.  
99.97.99  
STRING: RUNNING  
Object ID:  
.1.3.6.1.4.1.140.625.360.1.35.32.102.100.48.98.101.102.100.99.102  
.52.98.97.48.48.49.102.57.53.51.50.100.102.53.55.97.101.52.56.99.  
99.97.99  
INTEGER: 7001
```

トラップをテストするためのコマンド

表 D-3 に、テスト用にトラップを生成および受信するコマンドの概要を示します。

表 D-3 WebLogic Server に関する情報を取得するコマンドの概要

コマンド	説明
<code>snmpv1trap</code>	SNMPv1 トラップを構築し、指定したホストで実行され、指定したポート番号をリスンしている SNMP マネージャまたはトラップ デーモンに配布する。 D-12 ページの「 <code>snmpv1trap</code> 」を参照。
<code>snmptrapd</code>	トラップを受信するデーモンを開始し、トラップに関する情報を出力する。 D-15 ページの「 <code>snmptrapd</code> 」を参照。

snmpv1trap

SNMPv1 トラップを構築し、指定したホストで実行され、指定したポート番号をリスンしている SNMP マネージャまたはトラップ デーモンに配布します。トラップ デーモンの詳細については、D-15 ページの「snmptrapd」を参照してください。

このコマンドを呼び出す際は、送信するトラップ パケット内のフィールドの値を指定します。WebLogic Server MIB 内で定義したトラップに解決される値を指定する必要があります。WebLogic Server トラップおよびトラップ パケットに必要なフィールドについては、『SNMP 管理ガイド』の「WebLogic トラップ通知のフォーマット」を参照してください。

構文

```
java snmpv1trap [-d] [-c snmpCommunity] [-p TrapDestinationPort]
                TrapDestinationHost .1.3.6.1.4.140.625
                agent-addr generic-trap specific-trap timestamp
                [OID {INTEGER | STRING | GAUGE | TIMETICKS | OPAQUE |
                IPADDRESS | COUNTER} value] ...
```

引数	定義
<code>-c snmpCommunity</code>	トラップ内のデータを保護するパスワード (コミュニティ名) を指定する。 この値を指定しない場合は、 <code>-c public</code> を指定したものと見なされる。
<code>-p TrapDestinationPort</code>	SNMP マネージャまたはトラップ デーモンがリスンするポート番号を指定する。 この値を指定しない場合は、 <code>-p 162</code> を指定したものと見なされる。
<code>TrapDestinationHost</code>	SNMP マネージャまたはトラップ デーモンをホストするコンピュータの DNS 名または IP アドレスを指定する。
<code>.1.3.6.1.4.140.625</code>	トラップの <code>enterprise</code> フィールドの値を指定する。このフィールドには、すべての WebLogic Server トラップの OID の先頭部分が格納される。

引数	定義
<i>agent-addr</i>	<p>トラップの <code>agent address</code> フィールドの値を指定する。</p> <p>このフィールドは、トラップが生成されたコンピュータを示すために使用する。</p> <p><code>snmpv1trap</code> コマンドを使用してトラップを生成する場合は、有効であればどの DNS 名または IP アドレスでも指定できる。</p>
<i>generic-trap</i>	<p>トラップの <code>generic trap type</code> フィールドの値を指定する。</p> <p>指定できる値については、『SNMP 管理ガイド』の「WebLogic トラップ通知のフォーマット」を参照。</p>
<i>specific-trap</i>	<p>トラップの <code>specific trap type</code> フィールドの値を指定する。</p> <p>指定できる値については、『SNMP 管理ガイド』の「WebLogic トラップ通知のフォーマット」を参照。</p>
<i>timestamp</i>	<p>トラップの <code>timestamp</code> フィールドの値を指定する。</p> <p>このフィールドは、SNMP エージェントを最後に再初期化した時からトラップが発行された時までの時間を示すために使用する。</p> <p><code>snmpv1trap</code> コマンドを使用してトラップを生成する場合は、秒単位であればどのような値でも指定できる。</p>
OID {INTEGER STRING GAUGE TIMETICKS OPAQUE IPADDRESS COUNTER} <i>value</i>	<p>(省略可能) トラップの <code>variable bindings</code> フィールドの値を指定する。このフィールドは、トラップ通知をより詳細に示す名前と値の組み合わせで構成される。</p> <p>それぞれの名前と値の組み合わせには、OID、値タイプ、および値を指定する。</p> <p>たとえば、ログメッセージトラップには、トラップが生成された時刻を示す <code>trapTime</code> バインディングが含まれる。生成するテストトラップにこの変数バインドを含めるには、次のように <code>trapTime</code> 変数バインドの OID、STRING キーワード、および時刻を表す文字列を指定する。</p> <pre>.1.3.6.1.4.1.140.625.100.5 STRING "2:00 pm"</pre>

例

次の例では、trapTime および trapServerName 変数バインドを含むログメッセージトラップを生成します。トラップは、ポート 165 からブロードキャストされます。この例で指定する値の意味は以下のとおりです。

- 6 は「他の WebLogic Server トラップ」を指定する generic trap の値
- 60 は WebLogic Server がログメッセージトラップの識別に使用する specific trap の値
- .1.3.6.1.4.1.140.625.100.5 は trapTime 変数バインドの OID、
.1.3.6.1.4.1.140.625.100.10 は trapServerName 変数バインドの OID

```
java snmpv1trap -p 165 localhost .1.3.6.1.4.1.140.625 localhost 6 60
1000 .1.3.6.1.4.1.140.625.100.5 STRING "2:00 pm"
.1.3.6.1.4.1.140.625.100.10
STRING localhost
```

ポート番号 165 をリスンする SNMP マネージャ (またはトラップデーモン) がトラップを受信します。トラップデーモンが 165 をリスンしている場合であれば以下が返されます。

```
Trap received from: /127.0.0.1, community: public
Enterprise: .1.3.6.1.4.1.140.625
Agent: /127.0.0.1
TRAP_TYPE: 6
SPECIFIC NUMBER: 60
Time: 1000
VARBINDS:
Object ID: .1.3.6.1.4.1.140.625.100.5
STRING: 2:00 pm
Object ID: .1.3.6.1.4.1.140.625.100.10
STRING: localhost
```


snmptrapd

トラップを受信するデーモンを開始し、トラップに関する情報を出力します。

構文

```
java snmpv1trap [-d] [-c snmpCommunity] [-p TrapDestinationPort]
```

引数	定義
<code>-c <i>snmpCommunity</i></code>	トラップの生成に使用した SNMP エージェント (または <code>snmpv1trap</code> コマンド) のコミュニティ名を指定する。 この値を指定しない場合は、 <code>-c public</code> を指定したものと見なされる。
<code>-p <i>TrapDestinationPort</i></code>	トラップデーモンがトラップを受信するポート番号を指定する。 この値を指定しない場合は、 <code>-p 162</code> を指定したものと見なされる。

例

次のコマンドでは、トラップデーモンを開始し、ポート 165 のリクエストをリスンさせます。デーモンは、プロセスを強制停止するかシェルを終了するまでシェル内で実行されます。

```
java snmptrapd -p 165
```

コマンドが正常に実行されると、カーソルとともに空白行が返されます。トラップデーモンは、トラップを受信するまでこのままの状態で待機し、受信するとそのトラップを出力します。

例：トラップデーモンへのトラップの送信

WebLogic Server トラップを生成してトラップデーモンで受信するには次の手順に従います。

1. コマンドプロンプト (シェル) を開き、次のスクリプトを呼び出します。

D WebLogic SNMP エージェント コマンドライン リファレンス

WL_HOME\server\bin\setWLSEnv.sh (Windows の場合は setWLSEnv.cmd)
WL_HOME は WebLogic Server のインストール ディレクトリです。

2. トラップ デーモンを開始するには、次のコマンドを入力します。

```
java snmptrapd
```

3. 別のシェルを開き、次のスクリプトを呼び出します。

WL_HOME\server\bin\setWLSEnv.sh (Windows の場合は setWLSEnv.cmd)

4. トラップを生成するには、次のコマンドを入力します。

```
java snmpv1trap localhost .1.3.6.1.4.140.625 localhost 6 60  
1000
```

snmpv1trap コマンドによって serverStart トラップが生成され、ポート 162 からブロードキャストされます。

トラップ デーモンを実行しているシェルに以下が出力されます。

```
Trap received from: /127.0.0.1, community: public  
Enterprise: .1.3.6.1.4.140.625  
Agent: /127.0.0.1  
TRAP_TYPE: 6  
SPECIFIC NUMBER: 60  
Time: 1000  
VARBINDS:
```