BEA WebLogic Server™

WebLogic Server クラスタ ユーザーズ ガイド

BEA WebLogic Server バージョン 7.0
改訂 : 2003 年 10 月 20 日

著作権

限定的権利条項

商標または登録商標

WebLogic Server クラスタ ユーザーズ ガイド

パート番号	マニュアルの改訂	ソフトウェアのバージョン
なし	2003 年 3 月 17 日	BEA WebLogic Server バージョン 7.0

目次

4. クラスタでのロード バランシング

5. クラスタのフェイルオーバとレプリケーション

6. クラスタ アーキテクチャ

7. WebLogic クラスタの設定

8. 一般的な問題のトラブルシューティング

A. WebLogic クラスタの API

B. クラスタに関する BIG-IP™ ハードウェアのコンフィグレーション

このマニュアルの内容

このマニュアルでは、BEA WebLogic Server™ クラスタについて説明し、WebLogic Server 7.0 におけるクラスタの開発の概要について述べます。

このマニュアルの構成は次のとおりです。

- 第 1 章「WebLogic Server クラスタ化の概要」
- 第 2 章「クラスタでの通信」
- 第 3 章「クラスタのコンフィグレーションとアプリケーションのデプロイメント」
- 第 4 章「クラスタでのロード バランシング」
- 第 5 章「クラスタのフェイルオーバとレプリケーション」
- 第 6 章「クラスタ アーキテクチャ」
- 第 7 章「WebLogic クラスタの設定」
- 第 8 章「一般的な問題のトラブルシューティング」
- 付録 A「WebLogic クラスタの API」
- 付録 B「クラスタに関する BIG-IP™ ハードウェアのコンフィグレーション」

対象読者

このマニュアルは、1 つまたは複数のクラスタ上での Web ベース アプリケーションのデプロイメントに関心があるアプリケーション開発者および管理者を対象としています。HTTP、HTML コード、および Java プログラミング (サーブレット、JSP、または EJB のデプロイメント) に読者が精通していることを前提として書かれています。

e-docs Web サイト

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

このマニュアルの印刷方法

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

このマニュアルの PDF 版は、WebLogic Server の 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。

関連情報

- 『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』の「WebLogic Server EJB コンテナとサポートされるサービス」
- 『WebLogic HTTP サーブレット プログラマーズ ガイド』
- 『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』の「Web アプリケーション コンポーネントのコンフィグレーション」

サポート情報

WebLogic Server 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 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
太字のテキスト	用語集で定義されている用語を示す。
〔Ctrl〕+〔Tab〕	複数のキーを同時に押すことを示す。
斜体	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:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

太字の等幅テキスト	コード内の変数を示す。 例: `void **commit** ( )`
斜体の等幅テキスト	コード内の変数を示す。 例: `String *expr*`

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 SIGNON OR
{ }	構文の中で複数の選択肢を示す。実際には、この括弧は入力しない。
[ ]	構文の中で任意指定の項目を示す。実際には、この括弧は入力しない。 例： `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
|	構文の中で相互に排他的な選択肢を区切る。実際には、この記号は入力しない。
...	コマンドラインで以下のいずれかを示す。 ■ 引数を複数回繰り返すことができる。 ■ 任意指定の引数が省略されている。 ■ パラメータや値などの情報を追加入力できる。 実際には、この省略符号は入力しない。 例： `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
. . .	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。 実際には、この省略符号は入力しない。

1 WebLogic Server クラスタ化の概要

この章では、WebLogic Server クラスタの概要について簡単に説明します。説明する内容は以下のとおりです。

WebLogic Server クラスタとは

WebLogic Server クラスタは、同時に動作し、共同で高度なスケーラビリティと信頼性を実現する WebLogic Server プログラムの複数のコピーで構成されます。クラスタはクライアントからは単一の WebLogic Server インスタンスのように見えます。クラスタを構成する複数のサーバ インスタンスは同じマシン上で実行することも、複数のマシンに分散配置することもできます。クラスタの能力は、既存のマシン上のクラスタにサーバ インスタンスを追加することによって強化できます。また、新たにサーバ インスタンスを配置するためのマシンをクラスタに追加することもできます。クラスタ内の各サーバ インスタンスでは、同じバージョンの WebLogic Server が動作している必要があります。

クラスタとドメインの関係

クラスタは特定の WebLogic Server ドメインの一部です。

ドメインとは、関連性があり 1 つの単位として管理される WebLogic Server リソースの集合のことです。ドメインには 1 つ以上の WebLogic Server インスタンスが含まれます。これらのインスタンスはクラスタ構成にも非クラスタ構成にもでき、クラスタ化されたインスタンス群とそうでないインスタンス群を組み合わせて構成することもできます。ドメインには、複数のクラスタを構成できます。またドメインには、ドメインにデプロイされるアプリケーション コンポーネントと、それらのアプリケーション コンポーネントおよびドメイン内のサーバ インスタンスが必要とするリソースおよびサービスも含まれます。アプリケーションおよびサーバ インスタンスで使用されるリソースとサービスの例には、マシン定義、オプションのネットワーク チャネル、コネクタ、スタートアップ クラスなどがあります。

WebLogic Server インスタンスは、さまざまな基準によってドメインに分類できます。たとえば、稼働するアプリケーションの論理区分、地理的な考慮事項、あるいは管理対象リソースの数または複雑さに基づいて、複数のドメインにリソースを選択的に割り当てることができます。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。

各ドメイン内で、1 つの WebLogic Server インスタンスが管理サーバとして機能します。このサーバ インスタンスでは、ドメイン内のその他のサーバ インスタンスおよびリソースのすべてをコンフィグレーション、管理、およびモニタします。各管理サーバでは 1 つのドメインだけを管理します。ドメインに複数のクラスタが含まれる場合、ドメイン内の各クラスタは同じ管理サーバによって管理されます。

あるクラスタ内のすべてのサーバ インスタンスは同じドメイン内になければなりません。クラスタを複数のドメインに「分割する」ことはできません。同様に、コンフィグレーション対象のリソースまたはサブシステムを複数のドメイン間で共有することはできません。たとえば、あるドメイン内に JDBC 接続プールを作成する場合、別のドメイン内のサーバ インスタンスまたはクラスタからその接続プールを使用することはできません (代わりに、そのサーバ インスタンスまたはクラスタが属するドメイン内に同様の接続プールを作成する必要があります)。

クラスタ化される WebLogic Server インスタンスの動作は、フェイルオーバとロード バランシングの機能を備えること以外は、クラスタ化されないインスタンスと同様です。クラスタ化される WebLogic Server インスタンスのコンフィグレーションに使用するプロセスおよびツールは、クラスタ化されないインスタンスの場合と同じです。ただし、クラスタ化によって可能になるロード バランシングとフェイルオーバの効果を実現するためには、クラスタのコンフィグレーションに関する特定のガイドラインに従う必要があります。

WebLogic Server で使用されるフェイルオーバおよびロード バランシングのメカニズムと、個別のコンフィグレーション オプションとの関係については、4-1 ページの「クラスタでのロード バランシング」および 5-1 ページの「クラスタのフェイルオーバとレプリケーション」を参照してください。

コンフィグレーション上の推奨事項については、7-1 ページの「WebLogic クラスタの設定」の手順説明で詳しく示しています。

クラスタ化の利点

WebLogic Server クラスタを利用することによってもたらされる利点には、以下のものがあります。

- スケーラビリティ

  WebLogic Server クラスタにデプロイされるアプリケーションの能力を、必要に応じて動的に増強することができます。サービスを中断させることなく、つまり、クライアントおよびエンド ユーザへの影響なしにアプリケーションを動作させ続けながら、クラスタにサーバ インスタンスを追加できます。

- 高可用性

  WebLogic Server クラスタでは、サーバ インスタンスで障害が発生してもアプリケーションの処理を継続できます。アプリケーション コンポーネントの「クラスタ化」は、クラスタ内の複数のサーバ インスタンス上にコンポーネントをデプロイすることによって行います。そのため、コンポーネントが動作しているサーバ インスタンスに障害が発生した場合、同じコンポーネントがデプロイされている別のサーバ インスタンスがアプリケーションの処理を継続できます。

WebLogic Server インスタンスのクラスタ化は、アプリケーション開発者およびクライアントからは意識されません。ただし、クラスタ化を可能にする技術インフラストラクチャを理解しておくことは、プログラマおよび管理者が、各自が扱うアプリケーションのスケーラビリティと可用性を最大限に高める上で役立ちます。

クラスタの重要な機能

この節では、スケーラビリティと高可用性を実現する重要なクラスタ化の機能を、技術的でない観点から定義します。

- フェイルオーバ

  フェイルオーバは単純に言うと、特定の「ジョブ」、つまり何らかの処理タスクの集合を実行しているアプリケーション コンポーネント (以下の節では一般に「オブジェクト」と呼称します) が何らかの理由によって使用不可になったときに、障害を起こしたオブジェクトのコピーがそのジョブを引き継いで完了することを意味します。

  障害を起こしたオブジェクトの処理を新しいオブジェクトが引き継ぐためには、以下の条件が満たされている必要があります。

  - ジョブを引き継ぐことのできる、障害を起こしたオブジェクトのコピーが存在していること。
  - すべてのオブジェクトの場所および動作状態を定義する情報が、他のオブジェクトと、フェイルオーバを管理するプログラムから参照できること。この情報は、最初のオブジェクトがそのジョブを完了する前に障害を起こしたことを特定するために必要です。
  - 実行中のジョブの進捗状況についての情報が、他のオブジェクトと、フェイルオーバを管理するプログラムから参照できること。中断されたジョブを引き継ぐオブジェクトは、この情報を基に、最初のオブジェクトで障害が発生した時点でジョブがどの程度完了しているか (たとえば、どのデータが変更され、プロセス内のどの手順が完了しているか) を認識します。

  WebLogic Server では、標準に基づいた通信技術および通信機能であるマルチキャスト、IP ソケット、および JNDI (Java Naming and Directory Interface) を利用して、クラスタ内のオブジェクトの可用性に関する情報を共有および維持します。WebLogic Server ではこれらの技術によって、ジョブを完了す

る前にオブジェクトが停止したことと、中断されたジョブを完了するためのオブジェクトのコピーが存在する場所を特定します。

あるジョブに関してどのような処理が完了しているかについての情報をステートと呼びます。WebLogic Server では、セッション レプリケーションおよびレプリカ対応スタブと呼ばれる技術を利用して、ステートについての情報を維持します。レプリケーション技術により、特定のオブジェクトが不意にそのジョブの実行を停止したとき、障害を起こしたオブジェクトがどこで停止したかをオブジェクトの別のコピーが認識し、代わりにそのジョブを完了することが可能になります。

- ロード バランシング

  ロード バランシングとは、環境内のコンピューティング リソースおよびネットワーキング リソース間で、ジョブとそれに関連する通信を均等に分配することです。ロード バランシングを行うためには、以下の条件が満たされている必要があります。

  - 特定のジョブを実行できるオブジェクトの複数のコピーが存在していること。
  - すべてのオブジェクトの場所および動作状態についての情報が入手できること。

  WebLogic Server では、オブジェクトをクラスタ化、つまり複数のサーバ インスタンス上にデプロイすることにより、同じジョブを実行する代替オブジェクトを用意することができます。WebLogic Server は、デプロイされるオブジェクトの可用性および場所の情報を、マルチキャスト、IP ソケット、および JNDI を利用して共有し、維持します。

WebLogic Server での通信およびレプリケーション技術の利用形態について、詳しくは第 2 章「クラスタでの通信」を参照してください。

クラスタ化可能なオブジェクトの種類

クラスタ化されるアプリケーションまたはアプリケーション コンポーネントは、クラスタ内の複数の WebLogic Server インスタンス上で利用可能なものです。オブジェクトをクラスタ化すると、そのオブジェクトに対してフェイルオーバとロード バランシングが有効になります。クラスタの管理、保守、およびトラブルシューティングの手順を簡素化するには、オブジェクトを均一に、つまりクラスタ内のすべてのサーバ インスタンスにデプロイします。

Web アプリケーションは、エンタープライズ JavaBeans (EJB)、サーブレット、Java Server Pages (JSP) などを含むさまざまな種類のオブジェクトで構成できます。それぞれのオブジェクトの種類ごとに、制御、呼び出し、およびアプリケーション内部での機能に関連する振る舞いのユニークな集合が定義されています。この理由から、クラスタ化をサポートし、またその結果としてロード バランシングとフェイルオーバを実現するために WebLogic Server で利用される手法は、オブジェクトの種類ごとに異なる可能性があります。WebLogic Server のデプロイメントでは、次の種類のオブジェクトのクラスタ化が可能です。

- サーブレット
- JSP
- EJB
- Remote Method Invocation (RMI) オブジェクト
- Java Message Service (JMS) の送り先
- Java Database Connectivity (JDBC) 接続

種類の異なるオブジェクト間で、一部の振る舞いが共通している可能性があります。その場合、類似したオブジェクト タイプについては、クラスタ化のサポートおよび実装上の考慮事項が一致することがあります。以降の節では、次のオブジェクト タイプの組み合わせ別に説明および手順を示しています。

- サーブレットと JSP
- EJB と RMI オブジェクト

以降の節では、WebLogic Server でのクラスタ化、フェイルオーバ、およびロード バランシングのサポートについて、オブジェクトのタイプ別に簡単に説明します。

サーブレットと JSP

WebLogic Server は、クラスタ化されたサーブレットと JSP にアクセスするクライアントの HTTP セッション ステートをレプリケートすることで、サーブレットと JSP 向けのクラスタ化サポートを提供します。WebLogic Server では、HTTP セッション ステートをメモリ、ファイルシステム、またはデータベースに保持できます。

サーブレットまたは JSP の自動フェイルオーバを有効にするには、セッション ステートをメモリに保持する必要があります。サーブレットまたは JSP でのフェイルオーバの仕組み、および関連する要件とプログラミングにおける考慮事項については、5-3 ページの「HTTP セッション ステートのレプリケーション」を参照してください。

WebLogic Server プロキシプラグインまたは外部ロード バランシング ハードウェアを使用して、クラスタ間でのサーブレットおよび JSP のロード バランシングが可能になります。WebLogic Server プロキシ プラグインは、ラウンド ロビンのロード バランシングを実行します。外部のロード バランサは通常、さまざまなセッション ロード バランシング メカニズムをサポートしています。詳細については、4-1 ページの「サーブレットと JSP のロード バランシング」を参照してください。

EJB と RMI オブジェクト

EJB と RMI オブジェクトのロード バランシングとフェイルオーバは、レプリカ対応スタブを使用して処理されます。このスタブは、クラスタ全体の中からオブジェクトのインスタンスを見つけ出すためのメカニズムです。レプリカ対応スタブは、オブジェクトのコンパイル処理の結果として EJB および RMI オブジェクトに対して作成されます。EJB と RMI オブジェクトは均一に、つまりクラスタ内のすべてのサーバ インスタンスにデプロイされます。

EJB と RMI オブジェクトのフェイルオーバは、オブジェクトのレプリカ対応スタブを使用して実現されます。クライアントがレプリカ対応スタブを通じて障害が発生したサービスに対して呼び出しを行うと、スタブはその障害を検出し、別のレプリカに対してその呼び出しを再試行します。さまざまな種類のオブジェクトに対するフェイルオーバのサポートについては、5-16 ページの「EJB と RMI のレプリケーションとフェイルオーバ」を参照してください。

WebLogic Server クラスタは、クラスタ化される EJB および RMI オブジェクト間でロード バランシングを行うための複数のアルゴリズム (ラウンド ロビン、重みベース、ランダム) をサポートしています。デフォルトでは、WebLogic Server クラスタはラウンド ロビン方式を使用します。Administration Console で、他の方式を使用するようにクラスタをコンフィグレーションできます。選択した方法は、クラスタ化されたオブジェクト用に取得したレプリカ対応スタブ内で保持されます。詳細については、4-5 ページの「EJB と RMI オブジェクトのロード バランシング」を参照してください。

JDBC 接続

WebLogic Server では、クラスタがホストとなるアプリケーションの可用性を向上させるために、データ ソース、接続プール、マルチプールなどの JDBC オブジェクトをクラスタ化できます。クラスタ用にコンフィグレーションした各 JDBC オブジェクトは、クラスタ内の各管理対象サーバに存在する必要があります。JDBC オブジェクトをコンフィグレーションする場合は、各オブジェクトの対象をクラスタにします。

- データ ソース — クラスタ内で、外部クライアントは、JNDI ツリー内の JDBC データ ソースを介して接続を取得する必要があります。データ ソースは、WebLogic Server RMI ドライバを使用して接続を取得します。外部クライアント アプリケーションの WebLogic データ ソースはクラスタ対応なので、接続をホストするサーバ インスタンスに障害が発生した場合、クライアントは別の接続を要求できます。必須ではありませんが、サーバサイド クライアントも JNDI ツリー内のデータ ソースを介して接続を取得することをお勧めします。
- 接続プール — 接続プールは、既成のデータベース接続の集まりです。接続プールが起動すると、同一の物理的なデータベース接続が、指定した数だけ作成されます。接続プールでは起動時に接続を確立するので、アプリケーションごとにデータベース接続を作成する手間を省くことができます。クライアントサイドとサーバサイドの両方のアプリケーションで、JNDI ツリー内のデータ ソースを介して接続プールから接続を取得することをお勧めします。接続が完了したら、アプリケーションは接続を接続プールに返します。
- マルチプール — マルチプールは、基本的な接続プールを多重化したものです。アプリケーションにとって、マルチプールは基本的なプールとして認識されますが、マルチプールでは接続プールのプールを確立できます。接続属性は、接続プールごとに異なります。接続プール内の接続はすべて同一ですが、あるプールで予期された障害が起こっても、マルチプール内の他のプールが無効にならないという点で、マルチプール内の各接続プールの接続は異なっています。通常、これらのプールは、同じデータベースの異なるインスタンスを対象にします。

  マルチプールは、アプリケーション接続を同じように処理できる、複数の異なるデータベース インスタンスがある場合にのみ役立ちます。アプリケーション システムでは、アプリケーションの作業がデータベース間に分散されている場合に、データベースの同期化を行います。同じデータベース インスタンスに対する複数のプールを異なるユーザとして持つと便利な場合もあり

ます。これは、DBA があるユーザを無効にし、別のユーザを有効にしている場合に便利です。

デフォルトでは、クラスタ化されたプールは高可用性 (DBMS フェイルオーバ) を提供しますロード バランシングを提供するためにマルチプールをコンフィグレーションすることもできます。

JDBC の詳細については、『管理者ガイド』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)」を参照してください。

クラスタ化された JDBC との接続の取得

JDBC リクエストをどのクラスタ メンバーでも処理できるようにするには、クラスタ内の各管理対象サーバが、同じように命名または定義されたプール、および可能であればマルチプールを持つ必要があります。外部クライアントで使用することを想定したデータ ソースの場合は、クラスタを対象とする必要があります。これによりデータ ソースはクラスタ対応となり、どのクラスタ メンバーへの接続も可能になります。

- 外部クライアント接続 — データベース接続を必要とする外部クライアントは、JNDI ルックアップを実行し、データ ソースのレプリカ対応スタブを取得します。データ ソースのスタブには、データ ソースのホストとなるサーバ インスタンス (クラスタ内の全管理対象サーバ) のリストが格納されています。レプリカ対応スタブには、ホスト サーバ インスタンス間に負荷を分散するためのロード バランシング ロジックが含まれています。
- サーバサイド クライアント接続 — サーバサイドで使用する場合、アプリケーション コードは、JNDI ルックアップとデータ ソースの代わりに直接サーバサイド プール ドライバを使用できますが、データ ソースが使用されている場合、そのデータ ソースはローカル オブジェクトになります。サーバサイド データ ソースは、その JDBC 接続については別のクラスタ メンバーに向かうことはありません。データ ソースは、参照先のプールから接続を取得します。データベース トランザクションの間、およびアプリケーション コードがそれを保持している間 (接続が閉じられるまで)、接続はローカル サーバ インスタンスに固定されます。

JDBC 接続のフェイルオーバとロード バランシング

JDBC オブジェクトをクラスタ化しても接続のフェイルオーバは有効になりませんが、接続に障害が発生した場合の再接続のプロセスを簡略化することができます。レプリケートされたデータベース環境では、データベースのフェイルオーバ、および必要に応じて接続のロード バランシングをサポートするために、マルチプールをクラスタ化できます。詳細については、以下のトピックを参照してください。

- クラスタ化された JDBC オブジェクトの障害発生時の動作については、5-27 ページの「フェイルオーバと JDBC 接続」を参照してください。
- マルチプールのクラスタ化によって接続のロード バランシングが実現される仕組みについては、4-14 ページの「JDBC 接続のロード バランシング」を参照してください。
- クラスタ化された JDBC オブジェクトのコンフィグレーション手順については、7-26 ページの「クラスタ化された JDBC をコンフィグレーションする」を参照してください。

JMS

WebLogic JMS (Java Messaging Service) のアーキテクチャでは、クラスタ内のあらゆる WebLogic Server サーバ インスタンスから送り先への透過的なアクセスをクラスタ全体でサポートすることによって、複数の JSP サーバのクラスタ化を実装しています。WebLogic Server では、JMS の送り先および接続ファクトリをクラスタ全体に分散させることができますが、同じ JMS トピックまたは JMS キューは引き続き、クラスタ内の個々の WebLogic Server インスタンスによって個別に管理されます。

ロード バランシングは JMS に対してサポートされています。ロード バランシングを有効にするには、JMS サーバの対象をコンフィグレーションする必要があります。ロード バランシングと JSM コンポーネントの詳細については、4-13 ページの「JMS のロード バランシング」を参照してください。クラスタ化された JMS の設定手順については、7-25 ページの「固定サービスの移行可能対象をコンフィグレーションする」および 7-33 ページの「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。

自動フェイルオーバは、このリリースの WebLogic JMS ではサポートされていません。

クラスタ化できないオブジェクトの種類

以下の API および内部サービスは、WebLogic Server でクラスタ化できません。

- File サービス
- Time サービス
- WebLogic Event (WebLogic Server 6.0 より非推奨)
- ワークスペース (WebLogic Server 6.0 より非推奨)

これらのサービスは、クラスタ内の個々の WebLogic Server インスタンスでは使用できます。ただし、これらのサービスに関してロード バランシングやフェイルオーバ機能は利用できません。

WebLogic Server 7.0 の新しいクラスタ化機能

以降の節では、クラスタ化に関連する WebLogic Server 7.0 の新機能について説明します。

1 台のマシン上で 1 つの IP アドレスを使用してクラスタを作成可能

WebLogic Server 7.0 では、サーバ インスタンスがリスン アドレスを共有するものの異なるリスン ポートを使用するクラスタを設定し、クラスタ内の複数のサーバ インスタンスが単一の非マルチホーム マシンに存在するようにすること

ができます。1 台のマシン上で 1 つの IP アドレスだけでクラスタを設定できることにより、設定とサポートが容易になるため、この機能はデモンストレーション、開発、およびテスト用の環境で役立ちます。

旧リリースのクラスタでは、各サーバ インスタンスが異なるリスン アドレス、同じリスン ポート番号を持っていなければなりませんでした。1 台のコンピュータ上にクラスタを作成するには、1 台のコンピュータ上で複数の IP アドレスを使用するマルチホーム環境を設定する必要がありました。

また必要であれば、以前のリリースと同様に、1 台のマシン上で複数の IP アドレスと 1 つのリスン ポート番号を使用してクラスタを設定することもできます。

クラスタ内の複数のサーバ インスタンスで 1 つの IP アドレスを共有する場合、クラスタ内の各サーバ インスタンスにユニークなリスン ポート番号を割り当てます。クラスタ内の各サーバ インスタンスの IP アドレスが異なる場合、各インスタンスが使用するリスン ポート番号は同じであってもそれぞれ異なっていてもかまいません。必要なのは、サーバ インスタンスごとにユニークな IP アドレスとリスン ポートの組み合わせです。

コンフィグレーション ウィザードによる新しいクラスタの作成

WebLogic Server 7.0 には、ドメインおよびクラスタを容易に作成するための新しいユーティリティであるドメイン コンフィグレーション ウィザードが付属します。ウィザードの機能の詳細とその使用方法については、『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。

JMS の可用性の向上

バージョン 7.0 より前の WebLogic Server のクラスタには、JMS などのクラスタ化されないサービスで、クラスタが提供する冗長性を利用するためのメカニズムがありませんでした。WebLogic Server 7.0 では、管理者が、クラスタ化されないサービスを、あるサーバ インスタンスからクラスタ内の別のインスタンスに移行することができます。これは、サーバ インスタンスの障害への対応として、あ

るいは定期的な保守の一環として行います。この機能により、ホストのサーバインスタンスで障害が発生した場合に冗長サーバ インスタンス上でサービスを速やかに再開できるため、クラスタ内のクラスタ化されないサービスの可用性が向上します。

WebLogic JMS 7.0 では、管理者は複数の送り先をクラスタ内部で単一の分散送り先セットの一部としてコンフィグレーションできます。プロデューサおよびコンシューマは、分散送り先との送受信ができます。クラスタの内部で単一のサーバ インスタンスに障害が発生した場合、WebLogic JMS は、分散送り先セットに登録され、その時点で使用可能なすべての物理送り先に負荷を分散します。

バージョン 7.0 では、クラスタ化環境向けに WebLogic Server コアで実装された移行フレームワークも利用されます。これにより、WebLogic JMS は、移行要求に適切に対応することができ、JMS サーバを通常の方法でオンラインやオフラインにすることができます。これには、あらかじめ予定された移行と、WebLogic Server インスタンスの障害に対応して行われる移行の両方が含まれます。

2　クラスタでの通信

WebLogic Server クラスタは、ロード バランシングとフェイルオーバの 2 つの重要な機能を実装しています。以下の節では、アーキテクトと管理者が、特定の Web アプリケーションに適したクラスタをコンフィグレーションするために役立つ情報を示します。

クラスタでの WebLogic Server の通信

クラスタ内の WebLogic Server インスタンスは、2 つの基本的なネットワーク技術を利用して互いに通信します。

- IP マルチキャスト。サーバ インスタンスが、サービスの可用性と、継続的な可用性を示すハートビートをブロードキャストするために使用します。
- IP ソケット。クラスタ化されたサーバ インスタンス間でピア ツー ピア通信を行うための導管として機能します。

WebLogic Server での IP マルチキャストおよびソケット通信の使用形態は、クラスタをどのようにコンフィグレーションするかに影響します。

IP マルチキャストを使用した 1 対多通信

IP マルチキャストは、複数のアプリケーションが指定された IP アドレスとポート番号に「サブスクライブ」して、メッセージをリスンできるようにする単純なブロードキャスト技術です。マルチキャスト アドレスは、範囲が 224.0.0.0 ～ 239.255.255.255 の IP アドレスです。

IP マルチキャストはアプリケーションに対してメッセージをブロードキャストしますが、メッセージが実際に届くことは保証されていません。アプリケーションのローカル マルチキャスト バッファがいっぱいの場合、新規のマルチキャスト メッセージをそのバッファに書き込めないので、アプリケーションにはメッセージが「届いた」ことがわかりません。この制約のため、WebLogic Server インスタンスでは、IP マルチキャストに対してブロードキャストされたメッセージが届かない可能性を考慮しています。

WebLogic Server は、クラスタ内のサーバ インスタンス間の 1 対多通信で IP マルチキャストを使用します。次の通信が対象です。

- クラスタワイドの JNDI の更新 - クラスタ内の個々の WebLogic Server インスタンスはマルチキャストを使用して、ローカルでデプロイされたり、削除されたりしたクラスタ化されたオブジェクトが使用可能かどうかを通知します。クラスタ内の各サーバ インスタンスはこれらの通知をモニタし、クラスタ化されるオブジェクトの最新のデプロイメント状況に合わせてインスタンスのローカル JNDI ツリーを更新します。詳細については、2-10 ページの「クラスタワイドの JNDI ネーミング サービス」を参照してください。
- クラスタの「ハートビート」- WebLogic Server は、マルチキャストを使用して、クラスタ内の個々のサーバ インスタンスが使用可能であることを通知するために、定期的に「ハートビート」メッセージをブロードキャストします。ハートビート メッセージをモニタすることにより、クラスタ内のサーバ インスタンスは、どの時点でサーバ インスタンスに障害が発生したかを特定します (クラスタ化されたサーバ インスタンスは、サーバ インスタンスで障害が発生した時点を特定するためのより直接的な手段として、IP ソケットのモニタも行います)。

マルチキャストとクラスタのコンフィグレーション

マルチキャスト通信は、(2-10 ページの「クラスタワイドの JNDI ネーミング サービス」で説明するように) 障害の検出とクラスタワイドの JNDI ツリーの保守に関わる重要な機能を制御するので、クラスタのコンフィグレーションもマルチキャスト通信とのネットワーク トポロジ インタフェースも重要ではありません。以降の節では、クラスタ内でのマルチキャスト通信に伴う問題を回避するためのガイドラインを示します。

クラスタが WAN 内の複数のサブネットにまたがる場合

多くのデプロイメント構成で、クラスタ化されるサーバ インスタンス群は単一のサブネットの内部に置かれます。これは、マルチキャスト メッセージの確実な転送を保証するための構成です。ただし必要に応じて、ワイド エリア ネットワーク (WAN) 内の複数のサブネット間に WebLogic Server クラスタを分散させて冗長性を確保したり、あるいはクラスタ化されるサーバ インスタンスをより広範な地理的領域に分散させることができます。

クラスタを WAN (または複数のサブネット) 上に分散する場合、マルチキャスト メッセージがクラスタ内のすべてのサーバ インスタンスに必ず転送されるようにネットワーク トポロジを計画およびコンフィグレーションします。特に、ネットワークが以下の要件を満たす必要があります。

- IP マルチキャスト パケットの伝播の完全サポート。つまり、すべてのルータおよびその他のトンネリング技術がマルチキャスト メッセージをクラスタ化されているサーバ インスタンスに伝播するようにコンフィグレーションされている必要があります。
- ネットワーク レイテンシが十分に低く、大半のマルチキャスト メッセージがその最終的な宛先に 200 ～ 300 ミリ秒以内に確実に到達すること。
- クラスタのマルチキャスト TTL (Time-To-Live: 生存時間) の値は、マルチキャスト パケットが最終的な宛先に届くまでにルータがパケットを破棄しないような大きさの値でなければなりません。マルチキャスト TTL パラメータの設定手順については、7-39 ページの「マルチキャスト存続時間 (TTL) をコンフィグレーションする」を参照してください。

注意： WAN 上に WebLogic Server クラスタを分散するには、上記のマルチキャスト要件を満たす以外にも、ネットワーク機能を追加しなければならない場合があります。たとえば、不要なネットワーク ホップを経由せずにクライアント リクエストを最短経路でサーバ インスタンスに送るには、ロード バランシング ハードウェアをコンフィグレーションする必要があります。

ファイアウォールがマルチキャスト通信を遮断することがある

マルチキャスト トラフィックをファイアウォール内にトンネリングすることは可能ですが、WebLogic Server クラスタではお勧めしません。個々の WebLogic Server クラスタは、Web アプリケーションのクライアントに対して 1 つまたは複

数の別個のサービスを提供する論理単位として扱うようにします。複数の異なったセキュリティ ゾーン間にこの論理単位を分割しないでください。さらに、IP トラフィックの速度低下や中断につながる可能性のある技術は、ハートビートの不達が誤って障害として判断されることによって、WebLogic Server クラスタを分断させることがあります。

クラスタのマルチキャスト アドレスを他のアプリケーションと共有しない

複数の WebLogic Server クラスタは単一の IP マルチキャスト アドレスおよびポートを共有できますが、クラスタによって使用されているマルチキャスト アドレスおよびポートは、他のアプリケーションでブロードキャストまたはサブスクライブしないでください。つまり、クラスタのホストとなっている 1 台または複数台のマシンが、マルチキャスト通信を使用する他のアプリケーションのホストも兼ねている場合、それらのアプリケーションでは必ず、クラスタが使用しているものと異なるマルチキャスト アドレスおよびポートを使用してください。

クラスタのマルチキャスト アドレスを他のアプリケーションと共有すると、クラスタ化されたサーバ インスタンスは不必要なメッセージを処理しなければならなくなり、オーバーヘッドが増します。また、マルチキャスト アドレスの共有は、IP マルチキャスト バッファの過負荷と、WebLogic Server ハートビート メッセージの送信遅延につながる可能性があります。ハートビートが適切なタイミングで受信されないため、こうした遅延によって、WebLogic Server インスタンスがエラーとしてマークされる可能性があります。

こうした理由から、WebLogic Server クラスタ用には専用のマルチキャスト アドレスを割り当てて、そのアドレスを使用するすべてのクラスタのトラフィックをそのアドレスでブロードキャストできるようにします。

マルチキャスト ストームが起こったら

クラスタ内のサーバ インスタンスで受信メッセージが適時に処理されない場合は、NAK メッセージやハートビートの再送信を含むネットワーク トラフィックが増大します。ネットワーク上でマルチキャスト パケットが繰り返し送信されることをマルチキャス ストームと呼び、それが発生するとネットワークとそのネットワークに接続されたステーションが圧迫を受け、エンド ステーションでハングまたは障害が発生する可能性があります。マルチキャスト バッファのサイズを増やすと、通知が送信および受信されるレートが改善されて、マルチキャスト ストームを防止できます。7-40 ページの「マルチキャスト バッファのサイズをコンフィグレーションする」を参照してください。

IP ソケットを使用したピア ツー ピア通信

IP ソケットは、2 つのアプリケーション間でメッセージおよびデータを転送するための、シンプルでパフォーマンスの高いメカニズムです。クラスタ化される WebLogic Server インスタンスは、次の目的で IP ソケットを使用します。

- 異なったマシン上でクラスタ化されている別のサーバ インスタンスにデプロイされた、クラスタ化されていないオブジェクトへのアクセス
- HTTP セッション ステートとステートフル セッション EJB のステートの、プライマリ サーバ インスタンスとセカンダリ サーバ インスタンス間でのレプリケーション
- リモート サーバ インスタンス上にあるクラスタ化されたオブジェクトへのアクセス (これは一般に、6-7 ページの「推奨多層アーキテクチャ」で説明するような多層クラスタ アーキテクチャでのみ発生します)。

注意： WebLogic Server での IP ソケットの使い方はクラスタ関連にとどまらず、たとえば、Java クライアント アプリケーションがリモート オブジェクトにアクセスする場合など、すべての RMI 通信がソケットを使って行われます。

WebLogic Server クラスタのパフォーマンスは、ソケットのコンフィグレーションによって大きく左右されます。WebLogic Server でのソケット通信の効率は、次の 2 つの要因によって決まります。

- サーバ インスタンスのホスト システムがネイティブ ソケット リーダー実装と pure-Java ソケット リーダー実装のどちらを使用しているか
- pure-Java ソケット リーダーを使用しているシステムの場合は、サーバ インスタンスが十分な数のソケット リーダー スレッドに対応するようコンフィグレーションされているかどうか

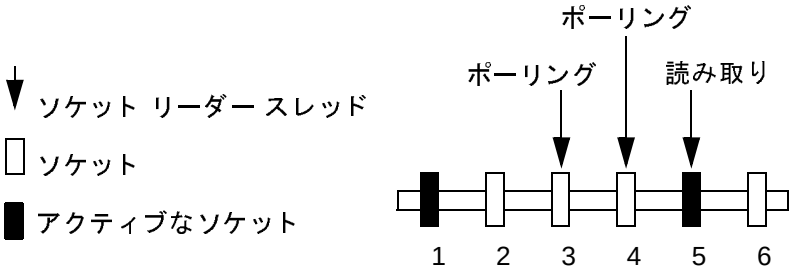
pure-Java とネイティブ ソケット リーダーの実装の比較

ソケット リーダー スレッドの pure-Java 実装は、ピア ツー ピア通信を行うための、信頼性が高く移植可能な方法ですが、ソケットに大きな負荷がかかる WebLogic Server クラスタでの使い方に最適なパフォーマンスは提供できません。pure-Java ソケット リーダーを使用すると、スレッドは、ソケットにデータが

入っているかどうかを調べるために、オープンしているソケットにポーリングする必要があります。つまり、ソケット リーダー スレッドはソケットのポーリングで常に「ビジー」となります。これはソケットがデータを持っていない場合でも変わりません。この不必要なオーバーヘッドにより、パフォーマンスが低下する可能性があります。

パフォーマンスの問題は、サーバ インスタンスがソケット リーダー スレッドよりも多くの開かれたソケットを持つとき (各リーダー スレッドが複数の開いたソケットをポーリングしなければならないとき) に顕著になります。ソケット リーダーはアクティブでないソケットに遭遇すると、別のソケットにサービスを提供する前にタイムアウトを待ちます。このタイムアウト待ちの間、次の図に示すように、アクティブでないソケットをソケット リーダーがポーリングする一方で、アクティブなソケットが未読のままになる場合があります。

図 2-1　pure-Java ソケット リーダー スレッドがアクティブでないソケットをポーリングする



ソケットの最適なパフォーマンスを実現するには、pure-Java 実装ではなく、オペレーティング システムに対応したネイティブ ソケット リーダー実装を使用するよう WebLogic Server ホスト マシンをコンフィグレーションします。ネイティブ ソケット リーダーは、ソケット上のデータの有無を非常に効率的な手法で調べます。ネイティブ ソケット リーダー実装では、リーダー スレッドはネイティブ ソケットをポーリングする必要がなく、アクティブなソケットに対してだけサービスとして、特定のソケットがアクティブになった場合には、(割り込みによって) 直ちに通知されます。

注意 : アプレットはネイティブ ソケット リーダー実装を使用できないので、ソケット通信では十分な効果が出ません。

オペレーティング システムに対応したネイティブ ソケット リーダーの実装を使用するように WebLogic Server ホスト マシンをコンフィグレーションする方法については、7-38 ページの「サーバ インスタンスのホスト マシン上でネイティブの IP ソケット リーダーをコンフィグレーションする」を参照してください。

Java ソケット実装用にリーダー スレッドをコンフィグレーションする

pure-Java ソケット リーダー実装を使用する場合は、サーバ インスタンスごとにソケット リーダー スレッドの数を適切にコンフィグレーションすることで、ソケット通信のパフォーマンスを向上できます。最適なパフォーマンスを実現するには、WebLogic Server でのソケット リーダー スレッドの数を、同時にオープンするソケットの最大数に合わせる必要があります。このコンフィグレーションにより、リーダー スレッドが複数のソケットにサービスしなければならない状況が回避され、ソケット データの即時読み取りが保証されます。

クラスタ内のサーバ インスタンスについて、リーダー スレッドの適切な数を決定する方法については、次の節「ソケットの使用数を確定する」を参照してください。

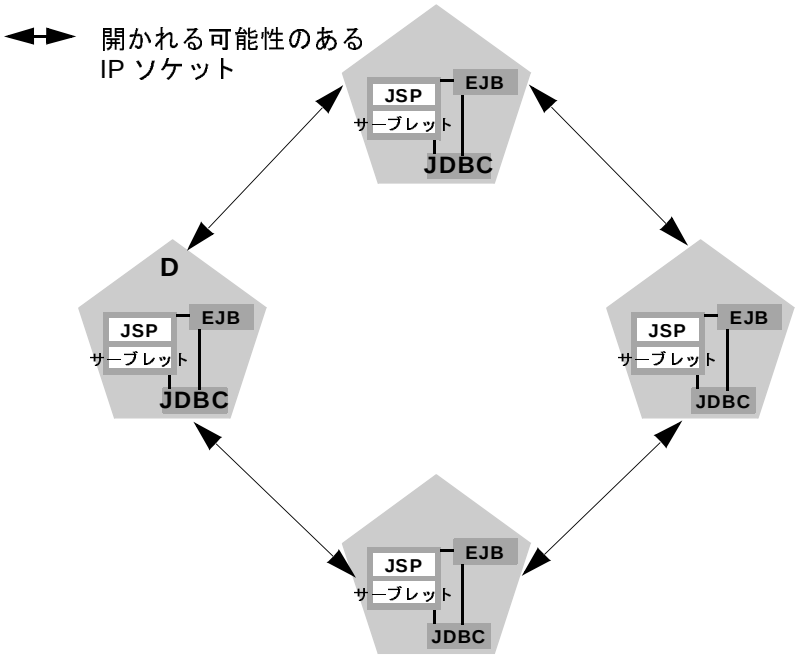
ソケット リーダー スレッドをコンフィグレーションする手順については、7-38 ページの「サーバ インスタンスのホスト マシン上のリーダー スレッドの数を設定する」を参照してください。

ソケットの使用数を確定する

個々の WebLogic Server インスタンスは、クラスタ内で自身を除くすべてのサーバ インスタンスに対してソケットをオープンする可能性があります。ただし、実際に使用されるソケットの最大数は、クラスタのコンフィグレーションによって異なります。実際には、クラスタ化されるシステムではオブジェクトが均一に (クラスタ内の各サーバ インスタンスに) デプロイされるため、他のどのサーバ インスタンスに対してもソケットがオープンされないのが一般的です。

クラスタで HTTP セッション ステートのインメモリ レプリケーションを使用しており、オブジェクトを均一にデプロイする場合、次の図に示すように、各サーバ インスタンスが最大で 2 個のソケットしかオープンしない可能性があります。

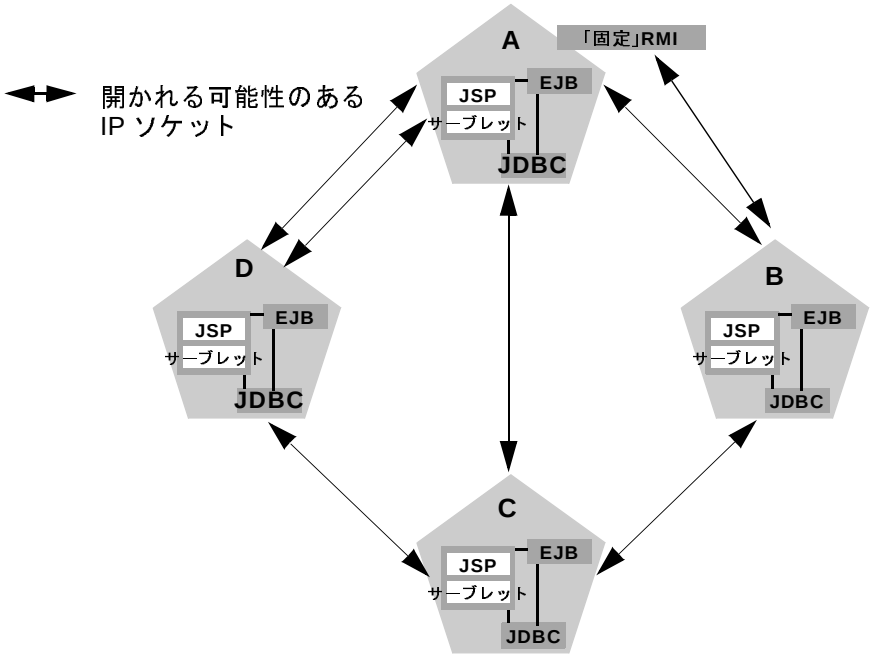
図 2-2　均一なデプロイメントによってソケットの必要数が最小限に抑えられる



この例の 2 つのソケットは、プライマリ サーバ インスタンスとセカンダリ サーバ インスタンスとの間で HTTP セッション ステートをレプリケートするために使用されます。クラスタ化されたオブジェクトにアクセスする場合、WebLogic Server が連結の最適化を行うために、ソケットは不要となります (これらの最適化については、4-9 ページの「連結されたオブジェクトの最適化」で説明しています)。このコンフィグレーションでは、デフォルトのソケット リーダー スレッド コンフィグレーションで十分です。

「固定」サービスのデプロイメント - サーバ インスタンスは、固定オブジェクトにアクセスするために追加のソケットを開く必要があるので、一度に 1 つのみのサーバ上でアクティブになっている複数のサービスで、ソケット使用率が増加することがあります (これは、固定オブジェクトにリモート サーバ インスタンスが実際にアクセスする場合にのみ考慮する必要があります)。次の図は、クラスタ化されていない RMI オブジェクトをサーバ A にデプロイすることによる影響を示しています。

図 2-3 クラスタ化されていないオブジェクトがソケットの潜在的な必要数を増加させる



この例では、各サーバ インスタンスは、HTTP セッション ステートをレプリケートするためと、サーバ A 上の固定 RMI オブジェクトにアクセスするために、最大で同時に 3 つのソケットをオープンする可能性があります。

注意： 6-12 ページの「多層アーキテクチャのコンフィグレーションに関する注意」で説明するように、多層クラスタ アーキテクチャのサーブレット クラスタでは、さらにソケットが必要な場合があります。

ソケット経由のクライアント通信

クラスタのクライアントは、ソケット リーダー スレッドの Java 実装を使用します。WebLogic Server 7.0 の Java クライアント アプリケーションは、クライアントがファイアウォールを通して接続するときでも、以前のバージョンの WebLogic Server のクライアントよりも多くの IP ソケットをオープンできます。

バージョン 7.0 では、クラスタ内の複数のサーバ インスタンスにリクエストを発行する場合、クライアントはそれぞれのサーバ インスタンスに対して個別のソケットをオープンします。

最高のパフォーマンスを引き出すには、クライアントを実行する Java 仮想マシン (JVM) において十分な数のソケット リーダー スレッドをコンフィグレーションします。この手順については、7-39 ページの「クライアント マシン上のリーダー スレッド数を設定する」を参照してください。

クラスタワイドの JNDI ネーミング サービス

クラスタ化されない WebLogic Server サーバ インスタンスのクライアントは、JNDI 準拠のネーミング サービスを使用してオブジェクトとサービスにアクセスします。JNDI ネーミング サービスには、サーバ インスタンスが提供する公開サービスの、ツリー構造のリストが含まれています。WebLogic Server インスタンスは、そのサービスを表す名前を JNDI ツリーにバインドすることによって新しいサービスを提供します。クライアントはサーバ インスタンスに接続し、バインドされたサービス名をルックアップすることによってサービスを取得します。

クラスタ内のサーバ インスタンスは、クラスタワイドの JNDI ツリーを利用します。クラスタワイドの JNDI ツリーは、ツリーが使用可能なサービスのリストを格納しているという点では、1 つのサーバ インスタンスの JNDI ツリーと似ています。ただし、クラスタワイドの JNDI ツリーは、ローカル サービスの名前だけでなく、クラスタ内の他のサーバ インスタンス上にあるクラスタ化されたオブジェクト (EJB や RMI クラス) が提供するサービスも格納します。

クラスタ内の各 WebLogic Server インスタンスは、クラスタワイドの論理 JNDI ツリーをローカルにコピーして保守します。以降の節では、クラスタワイドの JNDI ツリーが維持される方式と、クラスタ環境で発生しうる名前の衝突を防ぐ方法について説明します。

警告 :　クラスタワイドの JNDI ツリーは、アプリケーション データの永続性メカニズムまたはキャッシング メカニズムとして使用しないでください。

WebLogic Server は、クラスタ化されたサーバ インスタンスの JNDI エントリをクラスタ内の他のサーバ インスタンスにレプリケートしますが、元のインスタンスで障害が発生すると、これらのエントリはクラスタから削除されます。また、JNDI ツリー内に大きなオブジェクトを格納すると、マルチキャスト トラフィックが過負荷状態になり、クラスタの通常の処理の妨げとなる場合があります。

WebLogic Server によるクラスタワイドの JNDI ツリー作成のしくみ

クラスタ内の各 WebLogic Server は、クラスタのすべてのメンバーが提供するサービスが入ったクラスタワイドの JNDI ツリーをローカルにコピーして保守します。クラスタワイドの JNDI ツリーの作成では、まず各サーバ インスタンスをローカルの JNDI ツリーにバインドします。サーバ インスタンスが起動する (または新規サービスが動作中のサーバ インスタンスに動的にデプロイされる) と、サーバ インスタンスはまず、それらのサービスの実装をローカルの JNDI ツリーにバインドします。実装が JNDI ツリーにバインドされるのは、名前が他のサービスと重複していない場合だけです。

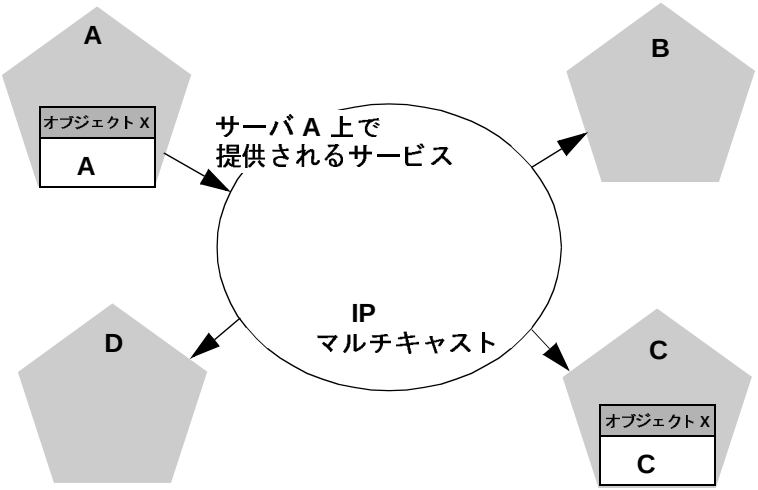
注意 : クラスタで管理対象サーバを起動すると、サーバ インスタンスは、`ClusterMBean` の `MemberWarmupTimeoutSeconds` パラメータで指定したウォームアップ時間の後で、ハートビートをリスンしてクラスタ内で実行されている他のサーバ インスタンスを識別します。デフォルトのウォームアップ時間は 30 秒です。

サーバ インスタンスがサービスをローカルの JNDI ツリーにバインドすると、レプリカ対応スタブを使用するクラスタ化されたオブジェクト向けに、次の手順が実行されます。クラスタ化されたオブジェクトの実装をローカルの JNDI ツリーにバインドしたら、サーバ インスタンスはそのオブジェクトのスタブを他のクラスタ メンバーに送信します。クラスタの他のメンバーはマルチキャスト アドレスをモニタして、リモート サーバ インスタンスが提供するサービスを追加したことを検出します。

次の図は、JNDI をバインドするプロセスの一部を示しています。

図 2-4 サーバ A がオブジェクトをその JNDI ツリーにバインドし、オブジェク

トが使用可能であることをマルチキャストする



前の図で、サーバ A はクラスタ化されたオブジェクト X の実装をローカルの JNDI ツリーにバインドしました。オブジェクト X はクラスタ化されているので、このサービスはクラスタ内の他のすべてのメンバーに提供されます。サーバ C はオブジェクト X の実装をバインド中です。

クラスタ内でマルチキャスト アドレスをリスンしている他のサーバ インスタンスは、サーバ A がクラスタ化されたオブジェクト X の新規サービスを提供し始めたことを検出します。これらのサーバ インスタンスは、自身のローカル JNDI ツリーを更新して、新規サービスをツリーに取り込みます。

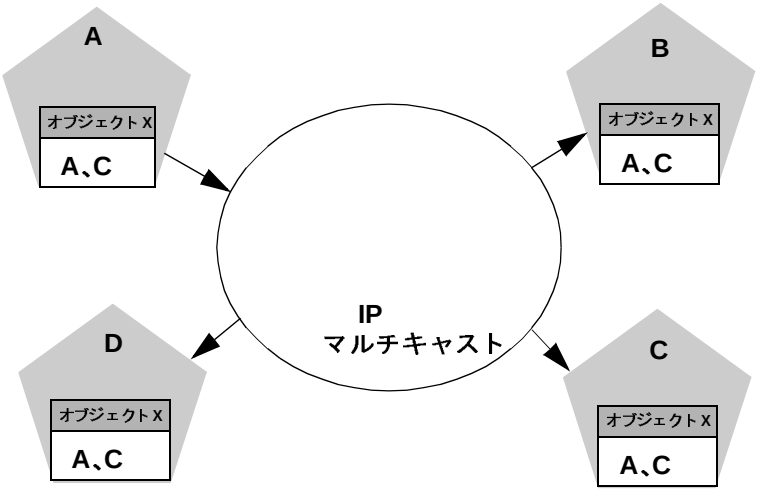
ローカルの JNDI バインドは次のいずれかの方法で更新されます。

- クラスタ化されたサービスがローカルの JNDI ツリーにバインドされていない場合、サーバ インスタンスは、サーバ A 上でオブジェクト X が使用可能であることを示す新規のレプリカ対応スタブをローカル ツリーにバインドします。これらのサーバ インスタンスにはクラスタ化されたオブジェクト X がデプロイされていないので、サーバ B とサーバ D は自身のローカル JNDI ツリーを更新して、これを反映させます。
- サーバ インスタンスがクラスタ対応サービスを既にバインドしている場合、サーバはローカルの JNDI ツリーを更新して、サービスのレプリカもサーバ A 上で使用可能であることを示します。サーバ C はクラスタ化されたオブ

ジェクト X をバインド済みなので、自身の JNDI ツリーを更新してそれを反映させます。

この方法によって、クラスタ内の各サーバ インスタンスはクラスタワイドの JNDI ツリーのコピーを独自に作成します。サーバ C が、オブジェクト X がローカルの JNDI ツリーにバインドされたことを通知する場合にも、処理順序は同じです。すべてのブロードキャスト メッセージが受信されたら、クラスタ内の各サーバ インスタンスのローカル JNDI ツリーは次の図のように、サーバ A とサーバ C 上でオブジェクトが使用可能であることを同じように示します。

図 2-5 マルチキャスト メッセージの受信後は各サーバの JNDI ツリーの内容は同じ



注意： 実際のクラスタでは、オブジェクト X は均一にデプロイされ、オブジェクトを呼び出すことのできる実装は 4 つのサーバ インスタンスすべてで使用可能になります。

JNDI 名の衝突が発生するしくみ

単純な JNDI 名の衝突は、サーバ インスタンスが JNDI ツリー内にバインド済みでクラスタ化されていないサービスと同じ名前を使って、別のクラスタ化されていないサービスをバインドしようとすると発生します。クラスタレベルの JNDI

名の衝突は、サーバ インスタンスが JNDI ツリー内にバインド済みでクラスタ化されていないサービスと同じ名前を使って、クラスタ化されているオブジェクトをバインドしようとしたときに発生します。

WebLogic Server は、ローカルの JNDI ツリーにサービスをバインドするときに、(クラスタ化されていないサービスの) 単純な名前の衝突を検出します。クラスタレベルの JNDI の衝突は、マルチキャストを使って新規サービスの通知を行うときに発生します。たとえば、クラスタ内のあるサーバ インスタンス上の固定 RMI オブジェクトをデプロイする場合、同じオブジェクトでレプリカ対応のものを別のサーバ インスタンス上にデプロイすることはできません。

クラスタ内の 2 つのサーバ インスタンスが、クラスタ化された別々のオブジェクトを同じ名前を使ってバインドしようとした場合、どちらのバインドもローカルには成功します。ただし、JNDI 名の衝突が発生するので、各サーバ インスタンスでは、他のサーバ インスタンスのレプリカ対応スタブを JNDI ツリーにバインドすることはできません。このタイプの衝突は、2 つのサーバ インスタンスのどちらかが終了するか、どちらかのサーバ インスタンスが衝突の原因となったクラスタ化オブジェクトをアンデプロイするまで解決しません。これと同じ衝突は、固定オブジェクトを両方のサーバ インスタンスが同じ名前でデプロイしようとした場合にも発生します。

均一なデプロイメントによってクラスタレベルの JNDI の衝突を回避する

クラスタレベルの JNDI の衝突を回避するには、すべてのレプリカ対応オブジェクトを、クラスタ内のすべての WebLogic Server インスタンスに均一にデプロイする必要があります。デプロイメントが WebLogic Server インスタンス間でバランスを欠いていると、起動時または再デプロイメント時に JNDI 名の衝突が発生する可能性が高まります。また、クラスタ内の処理負荷のバランスも取れなくなることがあります。

特定の RMI オブジェクトまたは EJB を個々のサーバ インスタンスに固定する必要がある場合は、そのオブジェクトのバインドをクラスタ間でレプリケートしないようにします。

WebLogic Server による JNDI ツリー更新のしくみ

クラスタ化されたオブジェクトが削除 (サーバ インスタンスからアンデプロイ) されるとき、JNDI ツリーの更新は、新規サービスの追加時に行われる更新と似た方法で処理されます。アンデプロイされたサービスがあったサーバ インスタンスは、そのサービスが提供されなくなったことを示すメッセージをブロードキャストします。既に説明したように、マルチキャスト メッセージを監視しているクラスタ内の他のサーバ インスタンスは、JNDI ツリーのローカル コピーを更新して、オブジェクトをアンデプロイしたサーバ インスタンス上でそのサービスが使用できなくなったことを反映させます。

クライアントがレプリカ対応スタブを取得すると、クラスタ内のサーバ インスタンスは、クラスタ化されたオブジェクトのホスト サーバの追加と削除を続行できます。JNDI ツリー内の情報が変更されると、クライアントのスタブも更新されます。その後の RMI リクエストには、クライアントのスタブが常に最新の情報を持つように、必要に応じて更新情報が含まれます。

クライアントとクラスタワイドの JNDI ツリーとの対話

WebLogic Server クラスタに接続して、クラスタ化されたオブジェクトをルックアップするクライアントは、そのオブジェクトのレプリカ対応スタブを取得します。このスタブには、そのオブジェクトの実装のホストとして使用可能なサーバ インスタンスのリストが入っています。スタブには、ホスト サーバ間で負荷を分散するためのロード バランシング ロジックも含まれています。

EJB および RMI クラス用のレプリカ対応スタブの詳細については、5-16 ページの「EJB と RMI のレプリケーションとフェイルオーバ」を参照してください。

WebLogic JNDI がクラスタ環境に実装されるしくみ、およびユーザ固有のオブジェクトを JNDI クライアントで使用可能にする方法については、『WebLogic JNDI プログラマーズ ガイド』の「クラスタ環境での WebLogic JNDI の使い方」を参照してください。

3 クラスタのコンフィグレーションとアプリケーションのデプロイメント

以下の節では、クラスタのコンフィグレーション情報が格納および維持される仕組みについて説明します。

- 3-1 ページの「クラスタのコンフィグレーションと config.xml」
- 3-2 ページの「管理サーバの役割」
- 3-5 ページの「動的コンフィグレーションの仕組み」
- 3-6 ページの「アプリケーションのデプロイメントについて」
- 3-11 ページの「クラスタをコンフィグレーションする方法」

注意： この章の情報の多くは、サーバ インスタンスがクラスタ化されていない WebLogic ドメインのコンフィグレーション プロセスに関連するものです。

クラスタのコンフィグレーションと config.xml

`config.xml` ファイルは、WebLogic Server ドメインのコンフィグレーションを記述した XML ドキュメントです。`config.xml` ファイルの内容および構造は、関連付けられた文書型定義 (DTD) である `config.dtd` ファイルに定義されます。

`config.xml` は XML 要素群で構成されています。Domain 要素はトップレベルの要素であり、Domain 内のすべての要素は Domain 要素の子です。Domain 要素には、Server、Cluster、Application などの子要素があります。子要素の中にさらに

子要素がある場合もあります。たとえば、Server 要素には、子要素として WebServer、SSL、および Log があります。Application 要素には EJBComponent と WebAppComponent という子要素があります。

各要素には、1 つ以上のコンフィグレーション可能な属性があります。`config.dtd` に定義されている属性には、コンフィグレーション API に対応する属性があります。たとえば、Server 要素には `ListenPort` 属性があり、同様に、`weblogic.management.configuration.ServerMBean` にも `ListenPort` 属性があります。コンフィグレーションできる属性は読み書きが可能です。たとえば、`ServerMBean` には `getListenPort` メソッドと `setListenPort` メソッドがあります。

`config.xml` の詳細については、『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。

管理サーバの役割

管理サーバは、そのドメイン内の WebLogic Server インスタンス群をコンフィグレーションおよび管理する WebLogic Server インスタンスです。

ドメインには複数の WebLogic Server クラスタと、クラスタ化されない WebLogic Server インスタンスが存在できます。厳密に言うと、ドメインは 1 つの WebLogic Server インスタンスだけでも構成できます。ただしその場合、各ドメインには必ず 1 つの管理サーバが存在しなければならないため、その唯一のサーバ インスタンスが管理サーバになります。

管理サーバのサービスを呼び出してコンフィグレーション作業を実施する方法は、3-11 ページの「クラスタをコンフィグレーションする方法」で説明するようにいくつか用意されています。どの方法を用いる場合でも、コンフィグレーションを修正するときはクラスタの管理サーバが稼働している必要があります。

管理サーバが起動すると、ドメインの `config.xml` がロードされます。管理サーバは、カレントディレクトリ内で `config.xml` を探します。ドメイン作成時に別のディレクトリを指定しない限り、`config.xml` は以下の場所に格納されます。

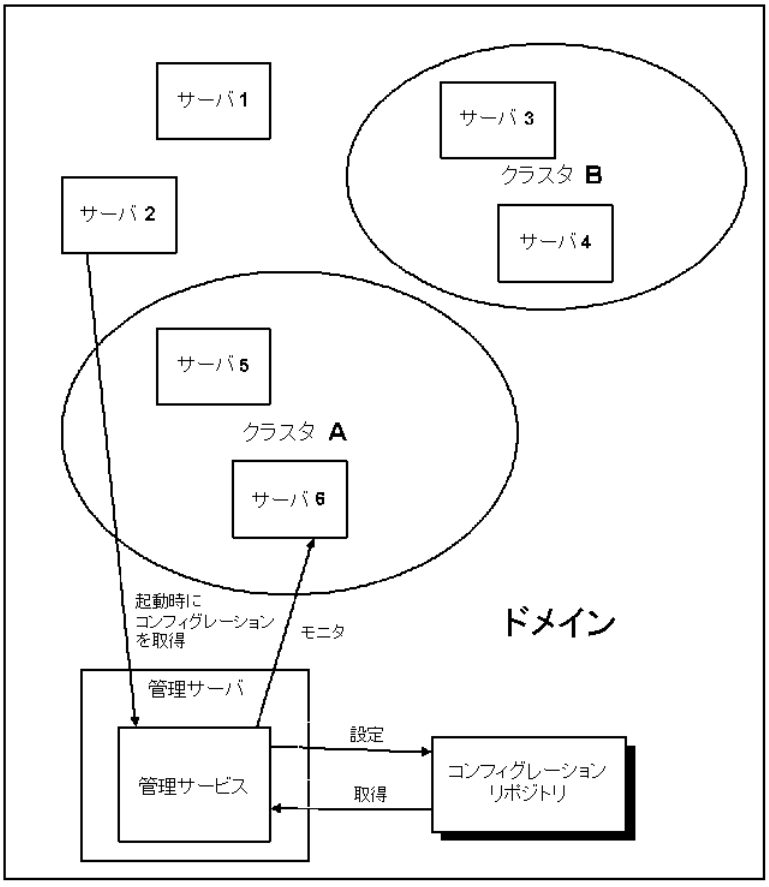
`BEA_HOME``/user_projects/`*`mydomain`*

mydomain はドメインに固有のディレクトリで、その名前はドメインと同じです。

管理サーバが正常に起動するたびに、`config.xml.booted` という名前のバックアップ コンフィグレーション ファイルがドメイン ディレクトリに作成されます。サーバ インスタンスの有効期間中に万一 `config.xml` ファイルが破損した場合、このファイルを使用して以前のコンフィグレーションに戻すことができます。

次の図は、1 つの管理サーバと複数の WebLogic Server インスタンスで構成される典型的なプロダクション環境を示しています。このようなドメインでサーバ インスタンスを起動すると、まず管理サーバが起動します。その他の各サーバ インスタンスは、起動時に管理サーバにアクセスしてそのコンフィグレーション情報を取得します。このように、管理サーバはドメイン全体のコンフィグレーションの一元的な制御エンティティとして動作します。

図 3-1　WebLogic Server のコンフィグレーション



管理サーバに障害が発生した場合

ドメインの管理サーバで障害が発生しても、ドメイン内の管理対象サーバの動作には影響しません。管理対象のサーバインスタンスが (クラスタ化されているかいないかには関係なく) 動作しているときにドメインの管理サーバが使用できなくなっても、その管理対象サーバは処理を続けます。ドメインにクラスタ化され

たサーバ インスタンスがある場合は、管理サーバに障害が発生しても、ドメイン コンフィグレーションでサポートされているロード バランシングおよびフェイルオーバ機能を使用できます。

注意： ホスト マシン上のハードウェアまたはソフトウェアに障害が発生したために管理サーバに障害が発生した場合、同じマシン上の他のサーバ インスタンスも同様に影響を受けることがあります。ただし、管理サーバ自体で障害が発生しても、ドメイン内の管理対象サーバの動作には影響しません。

管理サーバの再起動の手順については、『WebLogic Server ドメイン管理』の「管理対象サーバの動作中における管理サーバの再起動」を参照してください。

動的コンフィグレーションの仕組み

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

コンフィグレーションへのすべての変更が動的に適用されるわけではありません。たとえば、WebLogic Server インスタンスの `ListenPort` 値を変更した場合、新しいポートは対象のサーバ インスタンスを次に起動するまで使用されません。更新された値は XML ファイルに保存されますが、実行時の現在の値は影響を受けません。コンフィグレーション属性の実行時の値と、`config.xml` に格納されている値が異なるとき、Administration Console でその属性の隣に警告アイコンが表示され、コンフィグレーションの変更を有効にするためにはサーバ インスタンスの再起動が必要であることを知らせます。

Administration Console では、ユーザが変更した各属性に対して検証チェックが行われます。実行される検証には、範囲外エラーおよびデータ型不一致エラーのチェックがあります。どちらのエラーが見つかった場合にも、そのことを知らせるダイアログ ボックスが表示されます。

Administration Console の起動後に、管理サーバに割り当てられたリスン ポートを別のプロセスが獲得した場合、ポートを獲得したプロセスを停止することをお勧めします。リスン ポートを獲得したプロセスを削除できない場合、`config.xml` ファイルを編集して、`ListenPort` 値を変更します。

アプリケーションのデプロイメントについて

この節では、アプリケーションのデプロイメント プロセスについて簡単に説明します。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。

一般的なデプロイメント作業の実行手順については、7-29 ページの「アプリケーションをデプロイする」を参照してください。

デプロイメントの方法

以下の方法で、クラスタにアプリケーションをデプロイすることができます。

- WebLogic Server Administration Console

  Administration Console は、BEA Administration Service にアクセスするためのグラフィカル ユーザ インタフェース (GUI) です。

- WebLogic Builder

  WebLogic Builder は、J2EE コンポーネントをアセンブルし、そのデプロイメント記述子を作成および編集し、コンポーネントを WebLogic Server にデプロイするためのグラフィカル ツールです。

- weblogic.Deployer

  weblogic.Deployer ユーティリティは、WebLogic Server デプロイメント API にコマンドライン インタフェースを提供する Java ベースのデプロイメント ツールです。

これらのデプロイメント ツールについては、『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」で説明しています。

どのデプロイメント ツールを使用するかに関係なく、デプロイメント プロセスを開始するときは、デプロイするコンポーネントと、デプロイメント先の対象 (クラスタ、あるいはクラスタまたはドメイン内部の個別のサーバ インスタンス) を指定します。

ドメインの管理サーバはプロセス全体を通じて、クラスタ内の管理対象サーバと通信することによってデプロイメント プロセスを管理します。各管理対象サーバはデプロイメント対象のコンポーネントをダウンロードし、ローカルでのデプロイメント タスクを開始します。デプロイメントの状態は、デプロイメント対象のコンポーネントと関連付けられた MBean に維持されます。詳細については、『WebLogic Server アプリケーションの開発』の「デプロイメント管理 API」を参照してください。

注意： コンポーネントは、WebLogic Server にデプロイする前にパッケージ化します。アプリケーションのパッケージ化の詳細については、『WebLogic Server アプリケーションの開発』の「WebLogic Server アプリケーションのパッケージ化」を参照してください。

2 フェーズ デプロイメントの概要

WebLogic Server 7.0 のアプリケーションのデプロイメントには、準備と活性化の 2 つのフェーズがあります。

ユーザがクラスタへのアプリケーションのデプロイメントを開始すると、準備フェーズが開始される前に、まず WebLogic Server がクラスタ内の管理対象サーバの可用性を確認します。

デプロイメントの準備

デプロイメントの準備フェーズでは、ユーザの選択に従ってアプリケーション コンポーネントが対象のサーバ インスタンスに配信され、計画されたデプロイメントが検証されます。クラスタでは、アプリケーションは、個々の管理対象サーバではなく、クラスタに割り当てられる必要があります。準備フェーズの目的は、アプリケーション コンポーネントを正常にデプロイできるかどうかを検証することです。このフェーズの間、デプロイ中のアプリケーションに対するユーザからのリクエストは受け付けられません。

準備フェーズの開始後に障害が発生すると、(準備が正常に完了したサーバ インスタンスを含む) すべてのサーバ インスタンスでデプロイメントが中止されます。ステージングされたファイルは、削除されません。ただし、準備中に行われたコンテナ側での変更は元に戻されます。

デプロイメントの活性化

活性化の間、アプリケーションとそのコンポーネントが対象のサーバ インスタンスに完全にデプロイされ、デプロイされたアプリケーションがクライアントから利用可能になります。

活性化フェーズの間にサーバ インスタンスで障害が発生すると、そのインスタンスへのデプロイメントはキャンセルされます。1 つの管理対象サーバでそのような障害があっても、クラスタ化されている他のサーバ インスタンスでデプロイメントの活性化が妨げられることはありません。

クラスタ メンバーがアプリケーションのデプロイメントに失敗した場合、クラスタの一貫性を確保するためにそのメンバーは起動できなくなります (管理対象サーバ上でクラスタにデプロイされたアプリケーションに障害があると、その管理対象サーバは起動が中止されます)。

クラスタへのデプロイメントのガイドライン

デプロイメント プロセスの間は、クラスタ内のすべての管理対象サーバが実行されて利用可能になっているのが理想です。クラスタの一部のメンバーが利用できない状態でアプリケーションをデプロイすることはお勧めしません。クラスタにアプリケーションをデプロイする前に、できる限り、クラスタ内のすべての管理対象サーバが管理サーバからアクセスできる状態にあるようにしてください。

デプロイメント プロセスの間は、クラスタのメンバシップが変わらないようにしてください。デプロイメントが始まったら、以下のことはしないでください。

- 対象クラスタでの管理対象サーバの追加または削除
- 対象クラスタの管理対象サーバの停止

WebLogic Server 7.0 のデプロイメントの制限

WebLogic Server 7.0 では、以下の節で説明されているようにクラスタへのデプロイメントで制限が課されていました。

- 3-9 ページの「WebLogic Server 7.0 ではクラスタの一部分へのデプロイメントはできない」
- 3-9 ページの「WebLogic Server 7.0 では固定サービスの複数の管理対象サーバへのデプロイメントはできない」

WebLogic Server 7.0 ではクラスタの一部分へのデプロイメントはできない

WebLogic Server 7.0 では、クラスタの一部分へのデプロイメントはできません。クラスタ内の 1 つまたは複数の管理対象サーバが利用できない場合、デプロイメント プロセスは終了し、エラー メッセージが生成されます。そのメッセージは、デプロイメントを試みる前に、アクセス不能な管理対象サーバを再起動するか、クラスタから削除する必要があることを示します。

WebLogic Server 7.0 では固定サービスの複数の管理対象サーバへのデプロイメントはできない

WebLogic Server 7.0 では、クラスタ内の複数の管理対象サーバに固定サービスをデプロイすることはできません。WebLogic Server 7.0 では、アプリケーションがクラスタにデプロイされない場合に、そのアプリケーションをクラスタ内の 1 つの管理対象サーバ (かつその 1 つだけ) にデプロイできます。

WebLogic Server 7.0 SP1 以降でのデプロイメント ルールの「緩和」

WebLogic Server 7.0 SP01 では、3-9 ページの「WebLogic Server 7.0 のデプロイメントの制限」で説明されているデプロイメント ルールが緩和されました。デプロイメントにおいてユーザの自由裁量が認められています。詳細については、以下の節を参照してください。

- 3-10 ページの「WebLogic Server 7.0 SP01 以降ではクラスタの一部分にデプロイできる」

- 3-10 ページの「WebLogic Server 7.0 SP1 以降でのクラスタ全体へのデプロイメント」
- 3-10 ページの「WebLogic Server 7.0 SP01 以降での固定デプロイメント」

WebLogic Server 7.0 SP01 以降ではクラスタの一部分にデプロイできる

デフォルトの WebLogic Server 7.0 SP01 以降では、クラスタの一部分にデプロイできます。クラスタの 1 つまたは複数の管理対象サーバが利用できない場合は、次のメッセージが表示されます。

```
Cannot deploy to the following server(s) because they are
unreachable: "managed_server_n". Make sure that these servers are
currently shut down. Deployment will continue on the remaining
servers in the cluster "clustering". Once deployment has commenced,
do not attempt to start or shutdown any servers until the
application deployment completes.
```

アクセス不能な管理対象サーバが利用可能になると、そのサーバ インスタンスへのデプロイメントが開始されます。デプロイメント プロセスが完了するまで、その管理対象サーバでは欠けているクラスまたは古いクラスに関連する障害が発生する可能性があります。

WebLogic Server 7.0 SP1 以降でのクラスタ全体へのデプロイメント

weblogic.Deployer で `enforceClusterConstraints` フラグを設定すると、クラスタ内のすべての管理対象サーバがアクセス可能な場合にのみデプロイメントが実行されるようにすることができます。`enforceClusterConstraints` が「true」の場合は、3-9 ページの「WebLogic Server 7.0 のデプロイメントの制限」で説明されているルールに従ってデプロイメントが実行されます。

WebLogic Server 7.0 SP01 以降での固定デプロイメント

WebLogic Server 7.0 SP01 以降では、クラスタ内の複数の管理対象サーバを固定サービスの対象とすることができます。この操作はお勧めしません。固定サービスをクラスタ内の複数の管理対象サーバにデプロイすると、クラスタのロードバランシング機能とスケーラビリティが悪影響を受けるおそれがあります。複数の管理対象サーバを固定サービスの対象とすると、次のメッセージがサーバ ログに出力されます。

```
Adding server servername of cluster clustername as a target for
module modulename. This module also includes server servername that
belongs to this cluster as one of its other targets. Having multiple
individual servers a cluster as targets instead of having the entire
```

```
cluster as the target can result in non-optimal load balancing and
scalability. Hence this is not usually recommended.
```

クラスタをコンフィグレーションする方法

config.xml 内のコンフィグレーション情報を修正する複数の方法が用意されています。

- ドメイン コンフィグレーション ウィザード

  新しいドメインまたはクラスタの作成は、ドメイン コンフィグレーション ウィザードを使用して行うことをお勧めします。ウィザードを使用して実行できるタスクの一覧については、この章で後から説明する「ドメイン コンフィグレーション ウィザードの機能」を参照してください。

- WebLogic Server Administration Console

  Administration Console は、BEA Administration Service にアクセスするためのグラフィカル ユーザ インタフェース (GUI) です。コンソールからは、各種のドメイン コンフィグレーションおよびモニタ機能を実行できます。コンソールを使用して実行できるタスクの一覧については、3-12 ページの「Administration Console の機能」を参照してください。

- WebLogic Server アプリケーション プログラミング インタフェース (API)

  WebLogic Server に付属するコンフィグレーション API に基づいて、コンフィグレーション属性を変更するプログラムを記述することができます。この方法は、初期のクラスタ実装に対してはお勧めできません。

- WebLogic Server コマンドライン ユーティリティ

  WebLogic Server コマンドライン ユーティリティを使ってドメインの属性にアクセスできます。このユーティリティでは、ドメイン管理を自動化するスクリプトを作成できます。この方法は、初期のクラスタ実装に対してはお勧めできません。

ドメイン コンフィグレーション ウィザードの機能

ドメイン コンフィグレーション ウィザードでは、あらかじめ定義済みのドメイン テンプレートを使用して、ドメインとそのサーバ インスタンスの作成プロセスを簡素化しています。ウィザードではドメイン テンプレートを選択し、作成するサーバ インスタンスのマシン アドレス、名前、ポート番号などの重要な情報を指定します。

注意 : ドメイン コンフィグレーション ウィザードでは、リモート マシン上で動作する管理対象サーバ上のドメインに適したディレクトリ構造およびスクリプトを、管理サーバからインストールすることができます。この機能は、管理対象サーバをドメインのバックアップ管理サーバとして使用する必要がある場合に役立ちます。

ウィザードでは、4 つの典型的なドメイン コンフィグレーションから 1 つを選択します。

- 単一のサーバ - 単一の WebLogic Server インスタンスで構成されるドメイン
- 管理サーバと管理対象サーバ - 1 台の管理サーバと、クラスタ化されない 1 台以上の管理対象サーバで構成されるドメイン
- 管理サーバとクラスタ化された管理対象サーバ - 1 台の管理サーバと、クラスタ化された 1 台以上の管理対象サーバで構成されるドメイン
- 管理対象サーバ (管理コンフィグレーションを所有する)

目的のコンフィグレーション タイプを選択したら、ドメインとそのサーバ インスタンスについての関連情報を指定します。

ドメイン コンフィグレーション ウィザードの使用方法について、詳しくは『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。

Administration Console の機能

この節では、WebLogic Server Administration Console を使用して実行できるタスクの概要を示します。コンソールを使用してこれらのタスクを実行する具体的な手順については、Administration Console オンライン ヘルプを参照してください。

WebLogic Server のコンフィグレーション タスク

- サーバをコンフィグレーションする
- サーバを複製する
- サーバを削除する
- サーバ上でのガベージ コレクションを強制する
- XML レジストリをサーバに割り当てる
- メール セッションをサーバに割り当てる
- File T3 をサーバに割り当てる
- JDBC 接続プールをサーバに割り当てる
- WLEC 接続プールをサーバに割り当てる

WebLogic クラスタのコンフィグレーション タスク

- サーバ インスタンスのクラスタをコンフィグレーションする
- サーバ インスタンスのクラスタを複製する
- サーバ インスタンスをクラスタに割り当てる
- クラスタを削除する

デプロイメント タスク

- EJB をサーバ インスタンス上にデプロイする
- Web アプリケーション コンポーネントをサーバ インスタンス上にデプロイする
- 起動クラスと停止クラスをサーバ インスタンス上にデプロイする

表示とモニタのタスク

- サーバ インスタンスのログを表示する

- サーバ インスタンスの JNDI ツリーを表示する
- サーバ インスタンスの実行キューを表示する
- サーバ インスタンスの実行スレッドを表示する
- サーバ インスタンスのソケットを表示する
- サーバ インスタンスの接続を表示する
- サーバ インスタンスのバージョンを表示する
- サーバ インスタンスのセキュリティをモニタする
- クラスタをモニタする
- サーバ インスタンス上のすべての EJB のデプロイメント状況をモニタする
- サーバ インスタンス上のすべての Web アプリケーション コンポーネントをモニタする
- サーバ インスタンスに割り当てられているすべての WLEC 接続プールをモニタする
- クラスタに割り当てられているサーバ インスタンスをモニタする

4　クラスタでのロード バランシング

この章では、各種のオブジェクトを対象とした WebLogic Server クラスタでのロード バランシングのサポートについて説明し、設計者および管理者向けに、ロード バランシングに関連する計画およびコンフィグレーション上の考慮事項を示します。説明する内容は以下のとおりです。

- 4-1 ページの「サーブレットと JSP のロード バランシング」
- 4-5 ページの「EJB と RMI オブジェクトのロード バランシング」
- 4-13 ページの「JMS のロード バランシング」
- 4-14 ページの「JDBC 接続のロード バランシング」

サーブレットと JSP のロード バランシング

サーブレットと JSP のロード バランシングは、WebLogic プロキシ プラグインに組み込まれたロード バランシング機能を使用するか、またはロード バランシング ハードウェアを別途用意することによって実現できます。

注意： 外部ロード バランサは、HTTP トラフィック以外にも、t3 およびデフォルト チャネルを使用した Java クライアントからの初期コンテキスト リクエストを分散することもできます。WebLogic Server でのオブジェクト レベルのロード バランシングについては、4-5 ページの「EJB と RMI オブジェクトのロード バランシング」を参照してください。

プロキシ プラグインによるロード バランシング

WebLogic プロキシ プラグインは、クラスタ化されたサーブレットまたは JSP のホストである WebLogic Server インスタンスのリストを維持し、単純なラウンドロビン方式を使用して HTTP リクエストをそれらのインスタンスに転送します。このロード バランシング方法については、4-6 ページの「ラウンドロビンのロード バランシング」で説明します。

このプラグインは、WebLogic Server インスタンスで障害が発生した場合に、クライアントの HTTP セッション ステートのレプリカを見つけるために必要なロジックも備えています。

WebLogic Server は、以下の Web サーバおよび関連プラグインをサポートしています。

- WebLogic Server と `HttpClusterServlet`
- Netscape Enterprise Server と Netscape (プロキシ) プラグイン
- Apache と Apache Server (プロキシ) プラグイン
- Microsoft Internet Information Server と Microsoft-IIS (プロキシ) プラグイン

プロキシ プラグインの設定方法については、7-20 ページの「プロキシ プラグインをコンフィグレーションする」を参照してください。

プロキシ プラグインによるセッションの接続とフェイルオーバの仕組み

プロキシ プラグインによるクラスタ内の HTTP セッションの接続およびフェイルオーバの詳細については、5-9 ページの「クラスタ化されたサーブレットと JSP へのプロキシ経由のアクセス」を参照してください。

外部ロード バランサによる HTTP セッションのロード バランシング

ハードウェアによるロード バランシング ソリューションを利用するクラスタでは、ハードウェアが対応しているすべてのロード バランシング アルゴリズムを利用できます。この中には、個々のマシンの利用状況をモニタする負荷ベースの高度な調整方式を採用しているものもあります。

ロード バランサのコンフィグレーション要件

プロキシ プラグインではなくロード バランシング ハードウェアを使用する場合、互換性のあるパッシブまたはアクティブなクッキーの永続性メカニズムと、SSL の永続性にハードウェアが対応している必要があります。

- パッシブなクッキーの永続性

  パッシブなクッキーの永続性を使用すると、WebLogic Server は、ロード バランサを介したクライアントへのセッション パラメータ情報を含むクッキーを記述できます。セッション クッキーの詳細、およびロード バランサがセッション パラメータ データを使用して、HTTP セッション ステートをホストするプライマリ WebLogic Server とクライアントの関係を保持する仕組みについては、4-4 ページの「ロード バランサと WebLogic セッション クッキー」を参照してください。

- アクティブなクッキーの永続性

  ロード バランサが WebLogic Server クッキーを変更しないのであれば、特定のアクティブなクッキーの永続性メカニズムを WebLogic Server クラスタで使用することもできます。WebLogic Server クラスタでは、WebLogic HTTP セッション クッキーを上書きするか、または変更するアクティブなクッキーの永続性メカニズムはサポートされていません。ロード バランサのアクティブなクッキーの永続性メカニズムが、クライアント セッションに独自のクッキーを追加するものである場合は、WebLogic Server クラスタでロード バランサを使用するにあたって、新たなコンフィグレーションは不要です。

- SSL 永続性

  SSL 永続性を使用する場合、クライアントと WebLogic Server クラスタの間でのデータの暗号化および復号化は、すべてロード バランサが行います。そ

して、ロード バランサは、WebLogic Server がクライアントに挿入したプレーン テキストのクッキーを使用して、クライアントとクラスタ内の特定サーバ間の関係を維持します。

ロード バランサと WebLogic セッション クッキー

パッシブなクッキーの永続性を使用するロード バランサは、WebLogic セッション クッキーに文字列を使用することで、クライアントをそのプライマリ HTTP セッション ステートのホストになるサーバに関連付けることができます。文字列は、クラスタ内のサーバ インスタンスをユニークに識別します。ロード バランサをコンフィグレーションする際に、文字列のオフセットと長さを指定する必要があります。オフセットと長さの正しい値は、セッション クッキーの形式によって異なります。

セッション クッキーの形式は次のとおりです。

```
sessionid!primary_server_id!secondary_server_id
```

各値の説明は次のとおりです。

- `sessionid` は、HTTP セッションの識別子で、ランダムに生成されます。値の長さは、アプリケーションの `weblogic.xml` ファイルの <session-descriptor> 要素にある `IDLength` パラメータでコンフィグレーションされます。デフォルトでは、`sessionid` の長さは 52 バイトです。
- `primary_server_id` と `secondary_server_id` は、セッションのプライマリ ホストおよびセカンダリ ホストを表す 10 文字の識別子です。

注意： 非レプリケート メモリ、クッキー、JDBC、またはファイルベースの永続性を使用するセッションの場合、`secondary_server_id` はありません。インメモリ レプリケーションを使用するセッションで、セカンダリ セッションが存在していない場合、`secondary_server_id` は「NONE」です。

一般的なロード バランサのコンフィグレーション手順については、7-19 ページの「パッシブなクッキーの永続性をサポートするロード バランサをコンフィグレーションする」を参照してください。ベンダ固有の手順については、「クラスタに関する BIG-IP™ ハードウェアのコンフィグレーション」を参照してください。

関連するプログラミングの考慮事項

クラスタ化されるサーブレットおよびJSPを対象とした、プログラミング上の重要な制約および考慮事項については、5-5 ページの「クラスタ化されるサーブレットおよびJSPのプログラミング上の考慮事項」を参照してください。

ロード バランサでのセッション接続およびフェイルオーバの仕組み

ロード バランシング ハードウェアによるクラスタ内のHTTPセッションの接続およびフェイルオーバの詳細については、5-12 ページの「クラスタ化されたサーブレットとJSPへのロード バランシング ハードウェアを利用したアクセス」を参照してください。

EJB と RMI オブジェクトのロード バランシング

WebLogic Server クラスタは、クラスタ化されるEJBおよびRMIオブジェクト間でロード バランシングを行うための複数のアルゴリズム（ラウンド ロビン、重みベース、ランダム）をサポートしています。デフォルトでは、WebLogic Server クラスタはラウンド ロビン方式を使用します。Administration Consoleで、他の方式を使用するようにクラスタをコンフィグレーションできます。選択した方式は、クラスタ化されたオブジェクト用に取得したレプリカ対応スタブの内部で維持されます。

注意： WebLogic Serverでは、オブジェクトのメソッド呼び出しに対して常にロード バランシングが実行されるわけではありません。理由については、4-9 ページの「連結されたオブジェクトの最適化」を参照してください。

ロード バランシング アルゴリズムをクラスタで使用するように指定する手順については、7-18 ページの「EJB と RMI のロード バランシング方式をコンフィグレーションする」を参照してください。

サポートされているロード バランシング アルゴリズムについては、以下を参照してください。

- 4-6 ページの「ラウンドロビンのロード バランシング」
- 4-7 ページの「重みベースのロード バランシング」
- 4-8 ページの「ランダム ロード バランシング」

WebLogic Server では、パラメータベースのカスタム ルーティングもサポートしています。詳細については、4-8 ページの「クラスタ化されたオブジェクトのパラメータベースのルーティング」を参照してください。

EJB と RMI オブジェクトのロード バランシング

この章の以降の節では、WebLogic Server クラスタで使用できる標準的なロード バランシングの方法について説明します。

ラウンドロビンのロード バランシング

WebLogic Server は、特定のアルゴリズムが指定されていない場合には、クラスタ化されたオブジェクトのロード バランシング方式としてラウンドロビン アルゴリズムを使用します。このアルゴリズムは、RMI オブジェクトおよび EJB 用にサポートされています。また、WebLogic プロキシ プラグインで使用される手法でもあります。

このアルゴリズムは、WebLogic Server インスタンスのリストを順番に循環します。クラスタ化されたオブジェクトの場合、サーバ リストはそのオブジェクトのホストとなる WebLogic Server インスタンスからなります。プロキシ プラグインの場合、リストは、クラスタ化されたサーブレットまたは JSP のホストとなるすべての WebLogic Server インスタンスからなります。

ラウンドロビン アルゴリズムの長所は、シンプルで、動作コストの面で有利で、非常に予測しやすいということです。主な短所は、コンボイの可能性が若干あることです。コンボイは、あるサーバがその他のサーバよりも著しく速度が低下したときに発生します。レプリカ対応スタブまたはプロキシは同じ順序でサーバに

アクセスするので、速度の遅いサーバがあると、リクエストがそのサーバ上で「同期」するため、その他のサーバが将来のリクエストに備えて停滞する可能性があります。

重みベースのロード バランシング

このアルゴリズムは、EJB および RMI オブジェクトのクラスタ化だけに適用されます。

重みベースのロード バランシングは、各サーバに事前に割り当てられる重みを考慮することによって、ラウンドロビン アルゴリズムを改良したものです。Administration Console の [サーバ | コンフィグレーション | クラスタ] タブの [クラスタの重み] フィールドで、1 ～ 100 の範囲の数値を設定し、クラスタ内の各サーバに重みを割り当てることができます。この値は、あるサーバにかかる負荷の割合を他のサーバとの相対比較で決定します。すべてのサーバの重みが同じ場合、各サーバには等しい割合の負荷がかかります。あるサーバの重みが 50 で、他のサーバがすべて重み 100 の場合、重み 50 のサーバへの割り当ては、他のサーバの半分になります。このアルゴリズムを使うと、均一でないクラスタに対してラウンドロビン アルゴリズムの利点を活かすことができます。

重みベースのアルゴリズムを使用する場合、各サーバ インスタンスに割り当てる相対的な重みを慎重に決定してください。考慮すべき要因には、以下のものがあります。

- 他のサーバと比較したサーバのハードウェアの処理能力 (WebLogic Server 専用の CPU の数と性能など)
- 各サーバをホストとする、クラスタ化されていない (「固定」) オブジェクトの数

サーバに指定した重みを変更してサーバを再起動すると、レプリカ対応スタブ経由で新しい重みの情報がクラスタ全体に伝播されます。詳細については、2-10 ページの「クラスタワイドの JNDI ネーミング サービス」を参照してください。

注意： このバージョンの WebLogic Server では、RMI/IIOP プロトコルを使用して通信するオブジェクトに関する重みベースのロード バランシングはサポートされていません。

ランダム ロード バランシング

ランダム方式のロード バランシングは、EJB および RMI オブジェクトのクラスタ化にのみ適用されます。

ランダム ロード バランシングでは、リクエストは無作為にサーバに振り分けられます。ランダム ロード バランシングは、同様なコンフィグレーションがなされたマシン上で各サーバ インスタンスが動作する、均一なクラスタ デプロイメント環境でのみ推奨されます。リクエストを無作為に割り当てる方式では、サーバ インスタンスが動作するマシン間で処理能力に差があることは問題となります。クラスタ内でサーバをホストしているマシンの処理能力が、クラスタ内の他のマシンよりも著しく劣っているような場合、ランダム ロード バランシングによって、より強力なマシンに割り当てられるのと同じ分量のリクエストが能力の低いマシンに割り当てられてしまいます。

ランダム ロード バランシングでは、クラスタ内のサーバ インスタンス間で均等にリクエストが分配され、累計のリクエスト数の増加とともに個別の分配量も増します。リクエストの数が少ない場合、負荷は厳密に均等には配分されない場合があります。

ランダム ロード バランシングの欠点としては、リクエストごとにランダムな番号を生成することで、プロセスに多少のオーバーヘッドが発生することです。また、リクエストの数が少ない場合は、負荷が均等に分散されない可能性もあります。

クラスタ化されたオブジェクトのパラメータベースのルーティング

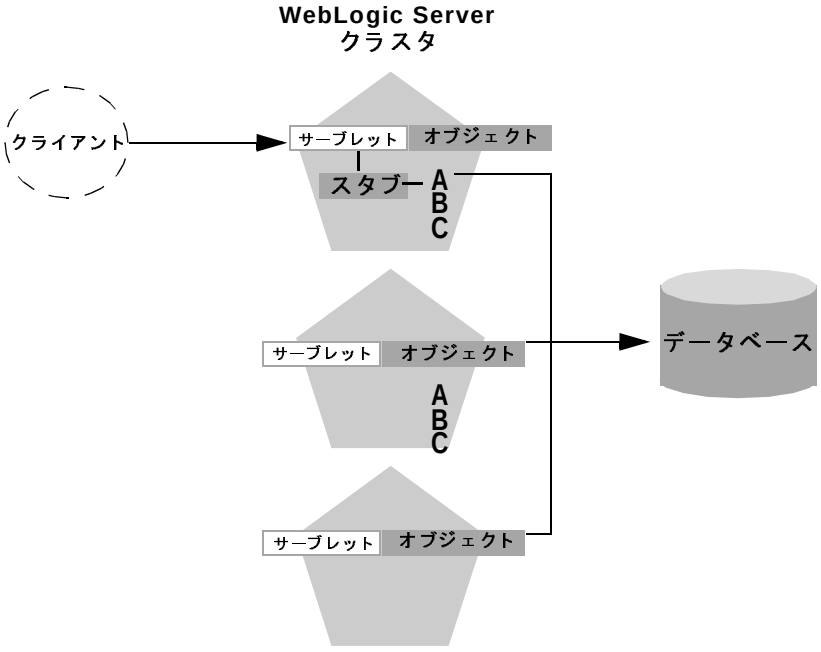
パラメータベースのルーティングでは、ロード バランシング動作をより詳細なレベルで制御できます。クラスタ化されたオブジェクトを `CallRouter` に割り当てることができます。これはパラメータにより各呼び出しの前に呼び出されるク

ラスです。CallRouter はそのパラメータを自由に調べ、呼び出しの送り先となるネーム サーバを返します。カスタムの CallRouter クラスを作成する手順については、「付録 A - WebLogic クラスタの API」を参照してください。

連結されたオブジェクトの最適化

レプリカ対応スタブはクラスタ化された EJB または RMI オブジェクトのロード バランシング ロジックを備えていますが、オブジェクトのメソッド呼び出しに対して、WebLogic Server が常にロード バランシングを実行するわけではありません。ほとんどの場合は、リモート サーバにあるレプリカを使用するよりも、スタブ自体と連結しているレプリカを使用する方が効率的です。次の図では、このことを示します。

図 4-1　連結の最適化はメソッド呼び出しに対してロード バランサが備えるロジックよりも優先する



上の例では、クライアントは、クラスタ内の最初の WebLogic Server インスタンスにあるサーブレットに接続します。クライアントの動作に対する応答として、サーブレットはオブジェクト A のレプリカ対応スタブを取得します。オブジェクト A のレプリカは同じサーバ インスタンス上にもあるので、そのオブジェクトはクライアントのスタブと連結していると判断されます。

WebLogic Server では、クラスタ内のオブジェクト A の他のレプリカにクライアントの呼び出しを分散しないで、常に、ローカルにある連結されたオブジェクト A のコピーを使用します。クラスタ内の他のサーバとのピア接続を確立するネットワーク オーバーヘッドが避けられるので、ローカル コピーを使用した方が効率的です。

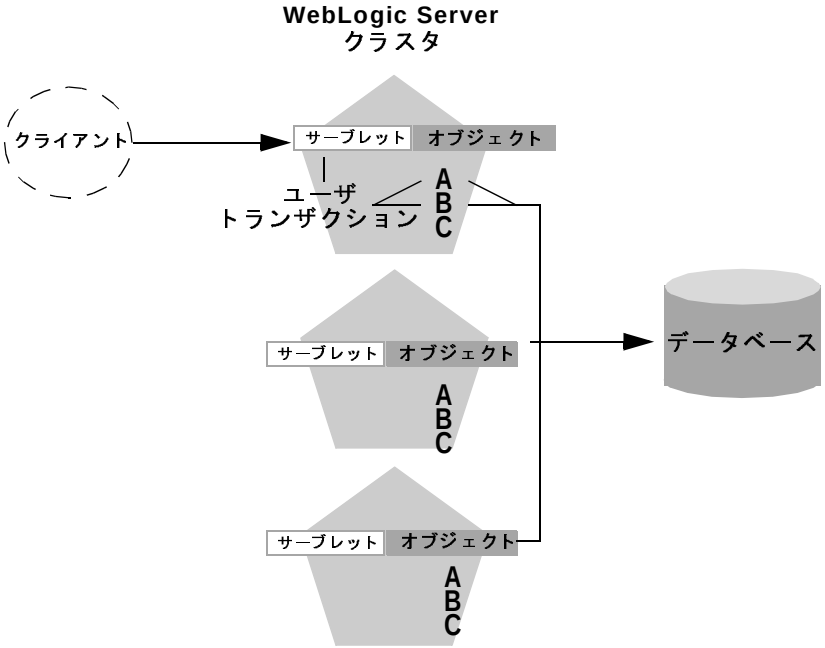
この最適化は、WebLogic Server クラスタの設計段階でよく見過ごされます。連結の最適化は、各メソッド呼び出しでのロード バランシングを必要としている管理者や開発者にとって混乱の元になることもよくあります。Web アプリケーションが単一のクラスタにデプロイされる場合、連結の最適化は、レプリカ対応スタブに固有のどのロード バランシング ロジックよりも優先されます。

クラスタ化されたオブジェクトに対する各メソッド呼び出しでロード バランシングが必要な場合、そのように WebLogic Server クラスタを設計する方法については、6-7 ページの「推奨多層アーキテクチャ」を参照してください。

トランザクションの連結

基本的な連結方式の拡張として、WebLogic Server では、同じトランザクションの一部として登録されているクラスタ化オブジェクトの連結も試みられます。クライアントによって UserTransaction オブジェクトが作成されると、WebLogic Server ではそのトランザクションと連結されているオブジェクトのレプリカが使用されます。次の図は、この最適化の仕組みを表しています。

図 4-2　連結の最適化がトランザクション内のその他のオブジェクトへと拡張する



この例では、クライアントは、クラスタ内の最初の WebLogic Server インスタンスに接続し、UserTransaction オブジェクトを取得します。新しいトランザクションが開始された後、クライアントはトランザクションの処理を実行するためにオブジェクト A とオブジェクト B をルックアップします。この状況では、A と B のスタブによるロード バランシングとは関係なく、WebLogic Server は常に UserTransaction オブジェクトと同じサーバにある A と B のレプリカを使用します。

このようなトランザクションの連結方式は、4-9 ページの「連結されたオブジェクトの最適化」で説明されている基本的な最適化よりも重要です。A と B のリモート レプリカが使用される場合は、トランザクションが終了するまでの間、余計なネットワーク オーバーヘッドが生じることになります。なぜなら、トランザクションがコミットされるまで A と B のピア接続がロックされるからです。その上、WebLogic Server ではトランザクションをコミットするために多層 JDBC 接続を利用しなければならないため、さらにネットワーク オーバーヘッドが生じることになります。

トランザクション コンテキストでクラスタ化されたオブジェクトを連結すると、WebLogic Server では個々のオブジェクトにアクセスするためのネットワーク負荷が削減されます。また、サーバでは多層接続ではなく単一層の JDBC 接続を利用してトランザクションの処理を実行できます。

JMS のロード バランシング

システム管理者は、複数の JMS サーバをコンフィグレーションし、対象を使用してそれらを定義済みの WebLogic Server に割り当てることで、クラスタ内の複数の JMS サーバにわたる送り先のロード バランシングを確立できます。各 JMS サーバは、厳密に 1 つの WebLogic Server にデプロイされ、送り先の集合に対するリクエストを処理します。コンフィグレーションの段階で、システム管理者が JMS サーバの対象を指定してロード バランシングを有効にします。対象の設定手順については、7-25 ページの「固定サービスの移行可能対象をコンフィグレーションする」を参照してください。JMS サーバを移行可能対象にデプロイする手順については、7-33 ページの「移行可能対象のサーバ インスタンスに JMS をデプロイする」を参照してください。

システム管理者は、複数の接続ファクトリをコンフィグレーションし、対象を使用してそれらを WebLogic Server に割り当てることで、クラスタ内のあらゆるサーバから送り先への透過的なアクセスをクラスタ全体にわたって確立できます。各接続ファクトリは、複数の WebLogic Server にデプロイできます。接続ファクトリの詳細については、『WebLogic JMS プログラマーズ ガイド』の「ConnectionFactory」を参照してください。

アプリケーションでは、Java Naming and Directory Interface (JNDI) を使用して接続ファクトリをルックアップし、JMS サーバとの通信を確立するための接続を作成します。各 JMS サーバでは、複数の送り先に対するリクエストが処理されます。JMS サーバによって処理されない送り先に対してのリクエストは、適切なサーバに転送されます。

分散 JMS 送り先のサーバ アフィニティ

サーバ アフィニティは、分散送り先機能を使用する JMS アプリケーションでサポートされています。スタンドアロンの送り先ではこの機能はサポートされていません。JMS 接続ファクトリのサーバ アフィニティをコンフィグレーションする場合、分散送り先の複数のメンバーにまたがる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サーバ インスタンスは、最初に同じサーバのインスタンスで実行されているすべての送り先メンバー間でロード バランシングを試みます。

JMS 接続ファクトリの [サーバ アフィニティを有効化] オプションが分散送り先メンバのロード バランシングの優先順位に与える影響の詳細については、『Programming WebLogic JMS』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。

分散送り先のサーバ アフィニティのコンフィグレーション手順については、『管理者ガイド』の「分散送り先のチューニング」を参照してください。

JDBC 接続のロード バランシング

JDBC 接続のロード バランシングでは、ロード バランシング用にコンフィグレーションされたマルチプールを使用する必要があります。ロード バランシング サポートは、マルチプールのコンフィグレーション時にユーザが選択できるオプションです。

ロード バランシング マルチプールは、5-27 ページの「フェイルオーバと JDBC 接続」で説明する機能を提供することに加えて、マルチプール内の接続プール間で負荷を分散します。マルチプールには、含まれている接続プールのリストがあります。ロード バランシング用にマルチプールをコンフィグレーションしていない場合は、必ずリスト内の最初の接続プールから接続が試行されます。ロード バランシング マルチプールでは、マルチプール内の各接続プールにラウンドロビン方式でアクセスします。マルチプール接続に対するクライアント リクエストがあるたびに、リストの先頭のプールが交代し、プールがリスト内でサイクルを作ります。

JDBC オブジェクトのクラスタ化の手順については、7-26 ページの「クラスタ化された JDBC をコンフィグレーションする」を参照してください。

5　クラスタのフェイルオーバとレプリケーション

クラスタが高可用性を提供するためには、サービスの障害からの回復が可能でなければなりません。この章の以下の節では、WebLogic Server でクラスタ内の障害が検出される仕組みについて説明し、フェイルオーバ方式の概要をオブジェクトのタイプ別に示します。

WebLogic Server で障害を検出する仕組み

クラスタ内の WebLogic Server インスタンスは、以下のものをモニタすることで、自身のピア サーバ インスタンスの障害を検出します。

- ピア サーバへのソケット接続
- サーバの定期的なハートビート メッセージ

IP ソケットを使用した障害検出

WebLogic Server インスタンスは、障害を直ちに検出するために、ピア サーバ インスタンス間で IP ソケットが使用されているかどうかをモニタします。サーバがクラスタ内のピアのいずれかに接続し、ソケットを使ってデータ転送を始めた場合、そのソケットが突然クローズされると、ピア サーバが「エラー」としてマークされ、その関連サービスが JNDI ネーミング ツリーから削除されます。

WebLogic Server の「ハートビート」

クラスタ化されたサーバ インスタンスがピア ツー ピア通信用にソケットをオープンしていない場合、障害が発生したサーバは WebLogic Server のハートビートによって検出できます。クラスタ内のすべてのサーバ インスタンスはマルチキャストを使用して、定期的なサーバ ハートビート メッセージを他のクラスタ メンバーにブロードキャストします。個々のハートビート メッセージには、メッセージの送信元のサーバを一意に識別するデータが入っています。サーバは、自身のハートビート メッセージを 10 秒間隔で定期的にブロードキャストします。同時に、クラスタ内の各サーバはマルチキャスト アドレスをモニタして、すべてのピア サーバのハートビート メッセージが送信されているかどうかを確認します。

マルチキャスト アドレスをモニタ中のサーバにピア サーバからのハートビート メッセージが 3 回届かなかった場合 (つまり、モニタする側のサーバが他のサーバから 30 秒以上ハートビートを受信していない場合)、モニタする側のサーバはピア サーバを「エラー」としてマークします。次に、必要であればローカルの JNDI ツリーを更新して、障害が発生したサーバでホストされていたサービスを削除します。

このようにして、サーバは、ピア ツー ピア通信でソケットがオープンされていない場合でも、障害を検出できます。

注意 : WebLogic Server での IP ソケットとマルチキャスト通信の使用について、詳しくは 2-1 ページの「クラスタでの WebLogic Server の通信」を参照してください。

サーブレットと JSP のレプリケーションとフェイルオーバ

WebLogic プロキシ プラグインと組み合わせて Web サーバを利用しているクラスタでは、クライアントから意識されない形でプロキシ プラグインがフェイルオーバを処理します。サーバに障害が発生した場合、プラグインはセカンダリ サーバ上にレプリケートされている HTTP セッション ステートを探し、その結果に従ってクライアントからのリクエストをリダイレクトします。

サポート対象のハードウェア ロード バランシング ソリューションを使用しているクラスタの場合、ロード バランシング ハードウェアは、WebLogic Server クラスタ内で使用可能な任意のサーバにクライアントのリクエストを単純にリダイレクトします。クラスタ自身は、クライアントの HTTP セッション ステートのレプリカをクラスタ内のセカンダリ サーバから取得します。

HTTP セッション ステートのレプリケーション

WebLogic Server では、サーブレットと JSP の HTTP セッション ステートの自動フェイルオーバを、セッション ステートをメモリ内にレプリケートすることによって実現しています。WebLogic Server はクライアントが最初に接続するサーバ上にプライマリ セッション ステートを作成し、クラスタ内の別の WebLogic Server インスタンス上に予備のレプリカを作成します。サーブレットのホストとなっているサーバで障害が起きた場合に使用できるように、レプリカは最新の状態に保たれます。サーバ インスタンス間でセッション ステートをコピーするプロセスは、インメモリ レプリケーションと呼ばれます。

注意： WebLogic Server では、ファイルベースまたは JDBC ベースの永続性メカニズムを利用して、サーブレットまたは JSP の HTTP セッション ステートを維持することもできます。それぞれの永続性メカニズムの詳細については、『Web アプリケーションのアセンブルとコンフィグレーション』の「セッション永続性のコンフィグレーション」を参照してください。

HTTP セッション ステートのレプリケーションに関する必要条件

HTTP セッション ステートのインメモリ レプリケーションを利用するためには、WebLogic プロキシ プラグインのコンフィグレーションが一致した Web サーバの集合、またはロード バランシング ハードウェアのどちらかを使用して WebLogic Server クラスタにアクセスする必要があります。

サポート対象のサーバ ソフトウェアとプロキシ ソフトウェア

WebLogic プロキシ プラグインは、クラスタ化されたサーブレットまたは JSP のホストである WebLogic Server インスタンスのリストを維持し、単純なラウンドロビン方式を使用して HTTP リクエストをそれらのインスタンスに転送します。このプラグインは、WebLogic Server インスタンスで障害が発生した場合に、クライアントの HTTP セッション ステートのレプリカを見つけるために必要なロジックも備えています。

HTTP セッション ステートのインメモリ レプリケーションは、以下の Web サーバおよびプロキシ ソフトウェアでサポートされています。

- WebLogic Server と `HttpClusterServlet`
- Netscape Enterprise Server と Netscape (プロキシ) プラグイン
- Apache と Apache Server (プロキシ) プラグイン
- Microsoft Internet Information Server と Microsoft-IIS (プロキシ) プラグイン

プロキシ プラグインの設定方法については、7-20 ページの「プロキシ プラグインをコンフィグレーションする」を参照してください。

ロード バランサの必要条件

プロキシ プラグインではなくロード バランシング ハードウェアを使用する場合、互換性のあるパッシブまたはアクティブなクッキーの永続性メカニズムと、SSL の永続性にハードウェアが対応している必要があります。これらの必要条件の詳細については、4-3 ページの「ロード バランサのコンフィグレーション要件」を参照してください。ロード バランサの設定手順については、7-19 ページの「パッシブなクッキーの永続性をサポートするロード バランサをコンフィグレーションする」を参照してください。

クラスタ化されるサーブレットおよび JSP のプログラミング上の考慮事項

この節では、クラスタ環境にデプロイするサーブレットおよび JSP を対象とした、プログラミング上の重要な制限および考慮事項について説明します。

- セッション データはシリアライズ可能でなければならない

  HTTP セッション ステートのインメモリ レプリケーションを行うには、サーブレットと JSP のセッション データがすべてシリアライズ可能でなければなりません。

注意： シリアライゼーションとは、複雑なデータ構造を変換するプロセスのことです。この例には、データの並列な配置 (多数のビットが並列チャネルを通じて同時に転送される) からシリアル形式 (1 ビットずつ順番に転送される) への変換などがあります。シリアル インタフェースでは、この変換によってデータ転送を可能にしています。

オブジェクトをシリアル化可能であると定義するための条件は、オブジェクトのすべてのフィールドがシリアル化可能または一時的であることです。サーブレットまたは JSP でシリアライズ可能なオブジェクトと不可能なオブジェクトが組み合わせて使用される場合、WebLogic Server ではシリアライズ不可能なオブジェクトのセッション ステートがレプリケートされません。

- setAttribute を使用してセッション ステートを変更する

  `javax.servlet.http.HttpSession` を実装する HTTP サーブレットでは、セッション オブジェクトの属性変更には (非推奨となった `putValue` の代わりとなる) `HttpSession.setAttribute` を使用します。`setAttribute` を使用してセッション オブジェクトの属性を設定する場合、オブジェクトとその属性はインメモリ レプリケーションを使用してクラスタにレプリケートされます。その他の set メソッドを使用してセッションの内部でオブジェクトを変更する場合、WebLogic Server ではその変更はレプリケートされません。セッション内にあるオブジェクトに対して変更が行われるたびに、`setAttribute()` を呼び出してクラスタ全体でそのオブジェクトを更新する方式が推奨されます。

  同様に、セッション オブジェクトから属性を削除するには、非推奨となった `removeValue` に代わって `removeAttribute` を使用します。

注意： 非推奨の `putValue` メソッドおよび `removeValue` メソッドを使用しても、セッション属性はレプリケートされます。

- シリアライゼーションのオーバーヘッドを考慮する

  セッション データをシリアライズすると、セッション ステートのレプリケートでオーバーヘッドが生じます。オーバーヘッドは、シリアライズされるオブジェクトのサイズに比例して大きくなります。セッション内で非常にサイズの大きいオブジェクトを作成するような設計では、サーブレットのパフォーマンスをテストして適切なレベルを確保してください。

- フレームからのセッション データへのアクセスを制御する

  複数のフレームを利用する Web アプリケーションを設計する場合は、指定したフレームセットのフレームが同時にリクエストを送らないようにすることを心がけてください。たとえば、論理的にはクライアントが 1 つのセッションを作成する場合でも、1 つのフレームセットの複数のフレームがクライアント アプリケーションに代わって複数のセッションを作成する可能性があります。

  クラスタ環境では、フレーム リクエストを適切に調整しないとアプリケーションの予期しない動作が発生することがあります。プロキシ プラグインは各リクエストを他のリクエストに関係なく処理するので、複数のフレーム リクエストによって、アプリケーションとクラスタ化されたインスタンスとの関連付けが「リセット」される場合が考えられます。また、フレームセット内の複数のフレームを介して同じセッションの属性を変更することで、アプリケーションがセッション データを壊してしまう可能性もあります。

  アプリケーションの予期しない動作を防ぐには、フレームからのセッション データへのアクセスを注意深く設計します。以下のいずれかの規則に従うと、よくある問題を防ぐことができます。

  - 指定のフレームセットでは、1 つのフレームだけがセッション データを作成および変更するようにする。
  - 必ず、アプリケーションで使用する最初のフレームセットのフレームでセッションを作成する (たとえば、最初に訪れる HTML ページでセッションを作成する)。セッションを作成した後は、最初のフレームセット以外のフレームセットでセッション データにアクセスします。

レプリケーション グループを使用する

デフォルトでは、WebLogic Server はプライマリ セッション ステートが置かれるものとは別のマシン上にセッション ステートのレプリカを作成しようと試みます。それ以外にも、レプリケーション グループを使用することにより、セカン

ダリ ステートが置かれる場所を独自に制御することができます。レプリケーション グループは、セッション ステートのレプリカを格納するために使用されるクラスタ内のサーバの優先順リストです。

WebLogic Server Console を使用して、個別のサーバ インスタンスのホストとなるユニークなマシン名を定義できます。それらのマシン名を新しい WebLogic Server インスタンスに関連付けると、そのサーバがシステムのどこにあるのかを識別できます。

マシン名は一般に、同じマシン上で動作するサーバを表すために使用します。たとえば、同じマシンまたはサーバ ハードウェア上で動作するすべてのサーバ インスタンスに同じマシン名を割り当てます。

1 台のマシン上で複数の WebLogic Server インスタンスを実行しない場合、WebLogic Server マシン名を指定する必要はありません。マシン名のないサーバは、それぞれ別個のマシン上にあるものとして扱われます。マシン名を設定する手順について、詳しくは 7-40 ページの「マシン名をコンフィグレーションする」. を参照してください。

クラスタ化されるサーバ インスタンスをコンフィグレーションするとき、サーバをレプリケーション グループと優先セカンダリ レプリケーション グループに割り当てることができます。後者のグループは、そのサーバに対して作成されるプライマリ HTTP セッション ステートのレプリカの保存先となります。

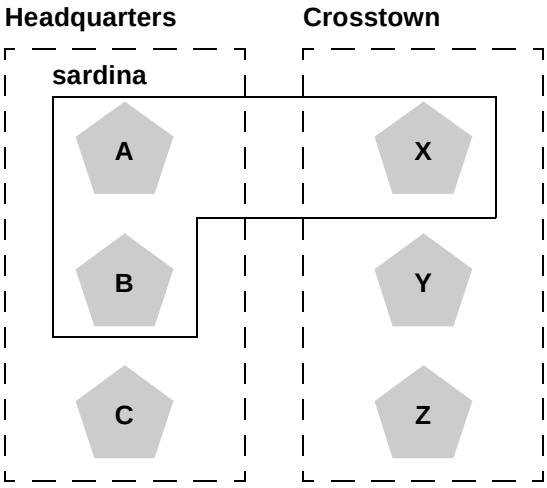
クライアントがクラスタ内のサーバに接続されて、プライマリ セッション ステートが作成されると、そのプライマリ ステートのホスト サーバではセカンダリのホスト サーバを決めるためにクラスタ内の他のサーバがランク付けされます。サーバのランクは、そのサーバがプライマリ サーバと同じマシン上にあるかどうか、およびプライマリ サーバの優先レプリケーション グループに属しているかどうか、という 2 つの情報を組み合わせて判断されます。次の表は、クラスタ内のサーバの相対的なランキングを示しています。

サーバのランク	別のマシンにあるのか	優先レプリケーション グループのメンバーか
1	はい	はい
2	いいえ	はい

サーバのランク	別のマシンにあるのか	優先レプリケーション グループのメンバーか
3	はい	いいえ
4	いいえ	いいえ

プライマリ WebLogic Server は、このルールに従ってクラスタ内のその他のサーバをランク付けし、最もランクの高いサーバをセカンダリ セッション ステートのホストとして選択します。たとえば、次の図は地理的な分類に基づいてコンフィグレーションされたレプリケーション グループを示しています。

図 5-1　地理的に分類されたレプリケーション グループ



この例では、サーバ A、B、および C は「Headquarters」というレプリケーション グループのメンバーであり、「Crosstown」という優先的なセカンダリ レプリケーション グループを使用します。逆に、サーバ X、Y、および Z は「Crosstown」というグループのメンバーであり、「Headquarters」という優先的なセカンダリ レプリケーション グループを使用します。サーバ A、B、および X は、「sardina」という同じマシン上にあります。

クライアントがサーバ A に接続し、HTTP セッション ステートを作成する場合の動作は次のようになります。

- サーバ Y および Z は別のマシン上にあり、サーバ A の優先セカンダリ グループのメンバーであるため、これらのサーバが最も高い確率でこのステートのレプリカのホストとなります。
- サーバ X は、次のランクに位置しています。プライマリと同じマシン上ではありますが、サーバ X も優先レプリケーション グループのメンバーだからです。
- サーバ C は、マシンは別ですが、優先的なセカンダリ グループのメンバーではないためランクは 3 番目になります。
- サーバ B は最低のランクです。なぜなら、サーバ A と同じマシン上にあり (したがってハードウェアに障害があると A と一緒にダウンする可能性がある)、かつ優先的なセカンダリ グループのメンバーではないからです。

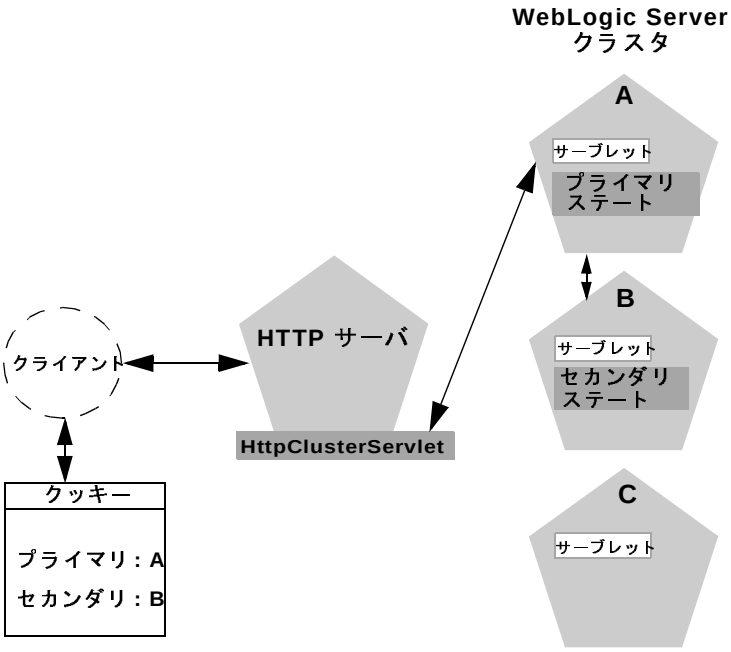
レプリケーション グループ内のサーバのメンバーシップをコンフィグレーションする手順、あるいはサーバの優先セカンダリ レプリケーション グループを割り当てる手順については、7-24 ページの「レプリケーション グループをコンフィグレーションする」を参照してください。

クラスタ化されたサーブレットと JSP へのプロキシ経由のアクセス

この節では、クラスタ化されたサーブレットおよび JSP にプロキシ経由で送られてくるリクエストを対象とした、接続およびフェイルオーバのプロセスについて説明します。プロキシ プラグインの設定方法については、7-20 ページの「プロキシ プラグインをコンフィグレーションする」を参照してください。

次の図は、クラスタでホストされるサーブレットにクライアントがアクセスする状況を表しています。この例では、静的な HTTP リクエストのみを処理する 1 つの WebLogic Server を使用します。すべてのサーブレット リクエストは、`HttpClusterServlet` を通じて WebLogic Server クラスタに転送されます。

図 5-2　サーブレットと JSP へのプロキシ経由のアクセス



注意： 以下の解説は、WebLogic Server と `HttpClusterServlet` ではなくサードパーティの Web サーバと WebLogic プロキシ プラグインを使用する場合にも有効です。

プロキシ接続の手順

HTTP クライアントがサーブレットを要求すると、`HttpClusterServlet` がプロキシとして機能し、WebLogic Server クラスタにそのリクエストを転送します。`HttpClusterServlet` は、クラスタ内の全サーバのリストと、クラスタへのアクセス時に使用するロード バランシング ロジックを管理します。上の例では、`HttpClusterServlet` はクライアントのリクエストを WebLogic Server A にあるサーブレットに転送します。WebLogic Server A は、クライアントのサーブレット セッションをホストするプライマリ サーバになります。

サーブレットのフェイルオーバ サービスを提供するために、プライマリ サーバではクライアントのサーブレット セッション ステートがクラスタ内のセカンダリ WebLogic Server にレプリケートされます。このような処理により、たとえばネットワークの障害によってプライマリ サーバがダウンしても、セッション ステートのレプリカを利用することができます。上の例では、サーバ B がセカンダリとして選択されています。

サーブレット ページは `HttpClusterServlet` を通じてクライアントに返され、クライアント ブラウザはサーブレット セッション ステートのプライマリとセカンダリの位置を示すクッキーを記述するように指示されます。クライアント ブラウザでクッキーがサポートされていない場合、WebLogic Server では代わりに URL 書き換えを利用できます。

URL 書き換えを利用してセッション レプリカを追跡する

デフォルト コンフィグレーションの WebLogic Server では、クライアントサイドのクッキーを使用して、クライアントのサーブレット セッション ステートのホストであるプライマリ サーバとセカンダリ サーバが追跡されます。クライアント ブラウザでクッキーが無効になっている場合、WebLogic Server では URL 書き換えを利用してもプライマリ サーバとセカンダリ サーバを追跡できます。URL 書き換えを利用する場合は、クライアント セッション ステートの両方の位置が、クライアントとプロキシ サーバの間で渡される URL に挿入されます。この機能をサポートするには、WebLogic Server クラスタで URL 書き換えを有効にする必要があります。URL 書き換えを有効にする方法については、『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』の「URL 書き換えの使い方」を参照してください。

プロキシ フェイルオーバのプロセス

プライマリ サーバで障害が発生すると、`HttpClusterServlet` はクライアントのクッキー情報を利用して、セッション ステートのレプリカのホストであるセカンダリ WebLogic Server の位置を確認します。`HttpClusterServlet` は、クライアントの次の HTTP リクエストを自動的にセカンダリ サーバにリダイレクトします。フェイルオーバは、クライアントには意識されません。

障害の発生後は、WebLogic Server B がサーブレット セッション ステートのプライマリ サーバになり、新しいセカンダリ サーバが作成されます (前の例ではサーバ C)。HTTP 応答では、今後のフェイルオーバに備えて、新しいプライマリ サーバとセカンダリ サーバを反映するためにプロキシによってクライアントのクッキーが更新されます。

2 つのサーバで構成されるクラスタでは、クライアントはセカンダリ セッション ステートのホスト サーバに透過的にフェイルオーバされます。ただし、もう 1 つの WebLogic Server がクラスタで利用可能にならない限り、クライアントのセッション ステートのレプリケーションは継続されません。たとえば、元のプライマリ サーバが再起動されるか、ネットワークに再び接続されると、そのサーバはセカンダリ セッション ステートのホストとして使用されます。

クラスタ化されたサーブレットと JSP へのロード バランシング ハードウェアを利用したアクセス

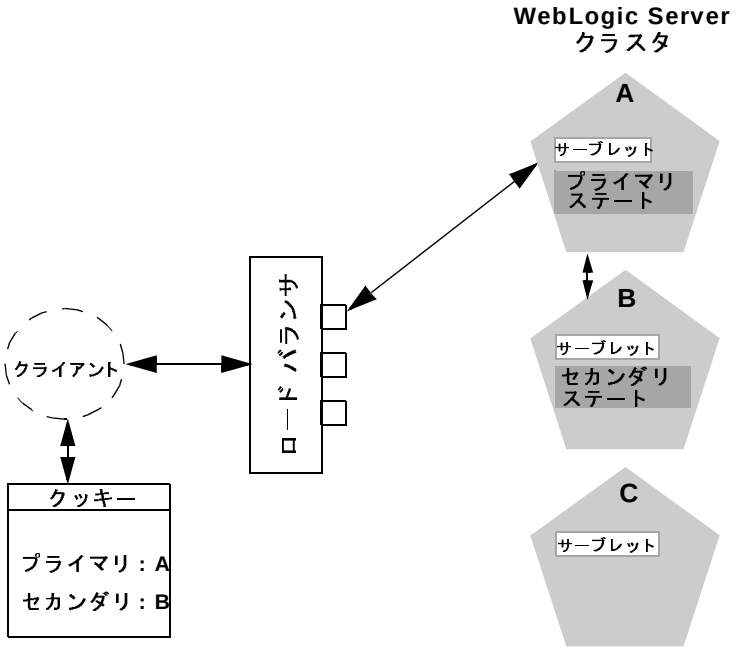
ロード バランシング ハードウェアを経由したクライアントの直接アクセスを可能にするために、WebLogic Server のレプリケーション システムでは、クライアントがフェイルオーバ先のサーバとは無関係にセカンダリ セッション ステートを使用できます。WebLogic Server は、プライマリ サーバとセカンダリ サーバの位置を記録する手段としてクライアント側のクッキーまたは URL 書き換えを使用します。ただし、この情報はサーブレット セッション ステートの位置の履歴としてのみ使用されます。ロード バランシング ハードウェアを通じてクラスタにアクセスする場合、クライアントは障害発生後にサーバを能動的に見つける手段としてはクッキー情報を使用しません。

以下の節では、HTTP セッション ステートのレプリケーションをロード バランシング ハードウェアと組み合わせて使用する場合の、接続およびフェイルオーバのプロセスについて説明します。

ロード バランシング ハードウェアを利用した接続

次の図は、ロード バランサを通じてクラスタにアクセスしているクライアントの接続手順を示しています。

図 5-3　ロード バランシング ハードウェアを利用した接続



Web アプリケーションのクライアントがパブリックな IP アドレスを使用してサーブレットを要求する場合は、次のようなプロセスが行われます。

1. ロード バランサはクライアントの接続要求を、コンフィグレーション済みのポリシーに従って WebLogic Server クラスタに転送します。クラスタはリクエストを WebLogic Server A に転送します。
2. WebLogic Server A は、クライアントのサーブレット セッション ステートのプライマリ ホストとして機能します。プライマリ ホストは、5-6 ページの「レプリケーション グループを使用する」で説明されているランキング システムを使用して、セッション ステートのレプリカのホストとなるサーバを選択します。上の例では、WebLogic Server B がレプリカのホストとして選択されています。

3. クライアントは、WebLogic Server インスタンス A と B の位置をローカルのクッキーに記録するように指示されます。クライアントでクッキーを利用できない場合、プライマリ サーバとセカンダリ サーバの位置は URL 書き換えを利用してクライアントに返される URL に記録できます。

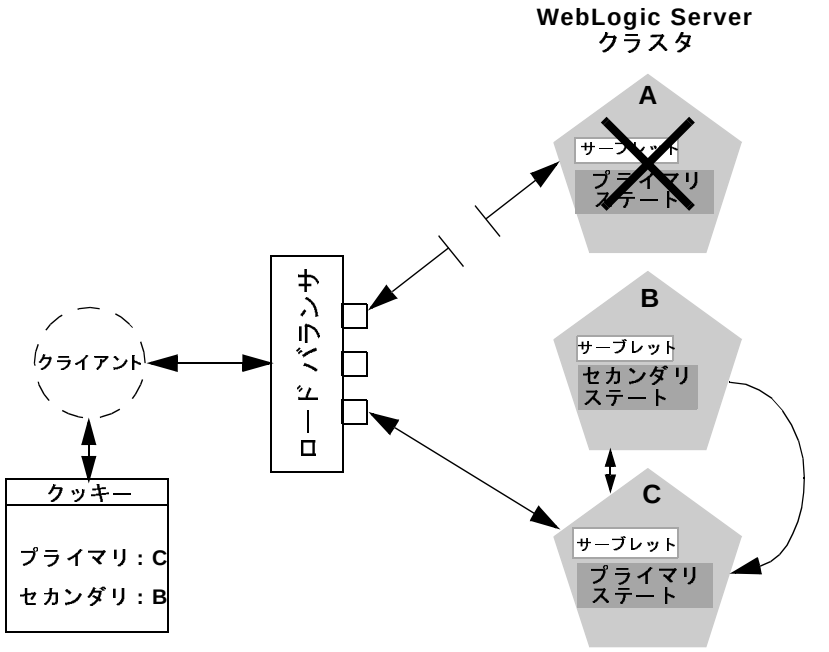
注意： クッキーを無効にしているクライアントに対応するには、5-11 ページの「URL 書き換えを利用してセッション レプリカを追跡する」で説明しているように、WebLogic Server の URL 書き換え機能を有効にする必要があります。

4. クライアントがクラスタに対してさらに要求を行う場合、ロード バランサはクライアント側のクッキーに記録された識別子を利用して、リクエストが引き続き、クラスタ内の別のサーバではなく WebLogic Server A に確実に転送されるようにします。このような処理によって、クライアントはセッションが終了するまでプライマリ セッション オブジェクトのホスト サーバと関係を維持することできます。

ロード バランシング ハードウェアを利用したフェイルオーバ

クライアントのセッションの途中でサーバ A に障害が発生すると、次の図に示すように、そのクライアントによるサーバ A への次の接続要求は失敗します。

図 5-4　ロード バランシング ハードウェアを利用したフェイルオーバ



接続が失敗した場合は、次のようなプロセスが行われます。

1. ロード バランシング ハードウェアは、コンフィグレーションされているポリシーを使用して、クラスタ内の利用可能な WebLogic Server にリクエストを転送します。上の例では、WebLogic Server A で障害が起こった後、クライアントのリクエストは WebLogic Server C に転送されます。
2. クライアントが WebLogic Server C に接続すると、そのサーバはクライアントのクッキーにある情報 (URL 書き換えが使用される場合は HTTP リクエストの情報) を使用して WebLogic Server B にあるセッション ステートのレプリカを取得します。このフェイルオーバ プロセスは、クライアントではまったく意識されません。

WebLogic Server C はクライアントのプライマリ セッション ステートの新しいホストになり、WebLogic Server B は引き続きセッション ステートのレプリカを保持します。プライマリ ホストとセカンダリ ホストに関するこの新しい情報は、クライアントのクッキーまたは URL 書き換えで再び更新されます。

EJB と RMI のレプリケーションとフェイルオーバ

クラスタ化される EJB と RMI のフェイルオーバは、オブジェクトのレプリカ対応スタブを使用して実現されます。クライアントがレプリカ対応スタブを通じて障害が発生したサービスに対して呼び出しを行うと、スタブはその障害を検出し、別のレプリカに対してその呼び出しを再試行します。

クラスタ化されたオブジェクトについては、オブジェクトが多重呼び出し不変の場合にだけ自動的なフェイルオーバが行われます。メソッドを何回呼び出しても 1 回呼び出したときと効果に違いがなければ、オブジェクトは多重呼び出し不変となります。このことは、恒久的な副作用を持たないメソッドに関して常に当てはまります。副作用のあるメソッドは、多重呼び出し不変性に留意して作成する必要があります。

ここでショッピング カートに商品を追加するショッピング カード サービス呼び出し `addItem()` を考えてみます。クライアント C がこの呼び出しをサーバ S1 上のレプリカで行うものとします。S1 が呼び出しを受け取った後、呼び出しを C に返す前に、S1 がクラッシュしたとします。この時点では、商品がショッピング カートに追加されていますが、レプリカ対応スタブは例外を受け取っています。スタブがサーバ S2 でこのメソッドを再試行すれば、商品がショッピング カートに 2 回追加されることになります。この理由から、レプリカ対応スタブは、デフォルトでは、リクエストが送られた後、そのリクエストが返ってくるまでは、メソッドの再試行は行いません。この動作は、サービスを多重呼び出し不変としてマークすることで、オーバーライドできます。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「クラスタ内のセッション EJB」を参照してください。

レプリカ対応スタブによるオブジェクトのクラスタ化

EJB または RMI オブジェクトをクラスタ化すると、オブジェクトのインスタンスがクラスタ内のすべての WebLogic Server にデプロイされます。クライアントでは、オブジェクトのどのインスタンスを呼び出すのかを選択できます。オブジェクトの各インスタンスはレプリカと呼ばれます。

WebLogic Server では、レプリカ対応スタブという技術を土台としてオブジェクトのクラスタ化を実現しています。クラスタ化をサポート (デプロイメント記述子で定義) する EJB をコンパイルすると、`ejbc` によって EJB のインタフェースが `rmic` コンパイラに渡されてその Bean のレプリカ対応スタブが生成されます。RMI オブジェクトの場合は、`rmic` に渡されるコマンドライン オプションを使用して明示的にレプリカ対応スタブを生成します。詳細については、『WebLogic RMI プログラマーズ ガイド』の「WebLogic RMI コンパイラ」を参照してください。

レプリカ対応スタブは、呼び出し側では通常の RMI スタブとして認識されます。しかしながら、スタブは 1 つのオブジェクトではなく複数のレプリカを表します。レプリカ対応スタブは、オブジェクトがデプロイされるすべての WebLogic Server インスタンスで EJB クラスまたは RMI クラスを見つけるためのロジックを備えています。クラスタ対応の EJB オブジェクトまたは RMI オブジェクトをデプロイすると、その実装は JNDI ツリーにバインドされます。2-10 ページの「クラスタワイドの JNDI ネーミング サービス」で説明されているように、クラスタ化された WebLogic Server インスタンスではそのオブジェクトを利用できるすべてのサーバをリストするために JNDI ツリーを更新することができます。クライアントがクラスタ化されたオブジェクトにアクセスすると、その実装はクライアントに送信されるレプリカ対応スタブで置き換えられます。

スタブは、オブジェクトに対するメソッド呼び出しを負荷分散するためのロード バランシング アルゴリズム (呼び出しルーティング クラス) を備えています。呼び出しが行われるたびに、スタブではそのロード アルゴリズムを利用して呼び出すレプリカを選択できます。この機能により、呼び出し側からは意識されない方法でクラスタ全体でのロード バランシングが実現されます。呼び出しの途中でエラーが起きると、スタブによってその例外が横取りされ、別のレプリカで呼び出しが再試行されます。この機能により、同じように呼び出し側からは意識されないフェイルオーバが実現されます。

各種の EJB でのクラスタ化サポート

EJB は、2 つの異なったレプリカ対応スタブを生成できる点が通常の RMI オブジェクトと異なります。1 つは `EJBHome` インタフェースに対して、もう 1 つは `EJBObject` インタフェースに対して生成されます。つまり、EJB では以下の 2 段階でロード バランシングとフェイルオーバのメリットを実現できるのです。

- クライアントが `EJBHome` スタブを使用して EJB オブジェクトをルックアップするとき
- クライアントが `EJBObject` スタブを使用して EJB に対するメソッド呼び出しを行うとき

以降の節では、EJB のタイプ別のクラスタ化サポートについて説明します。各タイプの EJB に対するクラスタ化の動作について、詳しくは『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』の「WebLogic Server クラスタにおける EJB」を参照してください。

EJB ホームのスタブ

すべての Bean ホームはクラスタ化可能です。Bean がサーバにデプロイされると、そのホームはクラスタワイドのネーミング サービスにバインドされます。ホームはクラスタ化可能なので、各サーバではそのホームのインスタンスを同じ名前でバインドできます。クライアントがこのホームをルックアップすると、そのクライアントでは Bean がデプロイされた各サーバ上のホームへの参照を持つレプリカ対応スタブが取得されます。`create()` または `find()` が呼び出されると、その呼び出しはレプリカ対応スタブによってレプリカの 1 つに転送されます。ホームのレプリカは、`find()` の結果を受信するか、またはそのサーバで Bean のインスタンスを作成します。

ステートレス EJB

ホームでステートレス Bean が作成されると、Bean がデプロイされたどのサーバにでも呼び出しを転送できるレプリカ対応 `EJBObject` スタブが返されます。ステートレス Bean ではステートが保持されないので、スタブでは Bean のホストであるどのサーバにでも呼び出しを転送できます。また、Bean はクラスタ化されるので、スタブでは障害が起きたときに自動的にフェイルオーバを実行できま

す。スタブでは、Bean は自動的には多重呼び出し不変として扱われないので、あらゆる障害から自動的に回復することはありません。Bean が多重呼び出し不変メソッドを利用して記述されている場合は、そのことをデプロイメント記述子で示すことができ、そうすればあらゆる場合で自動フェイルオーバが有効になります。

ステートフル EJB

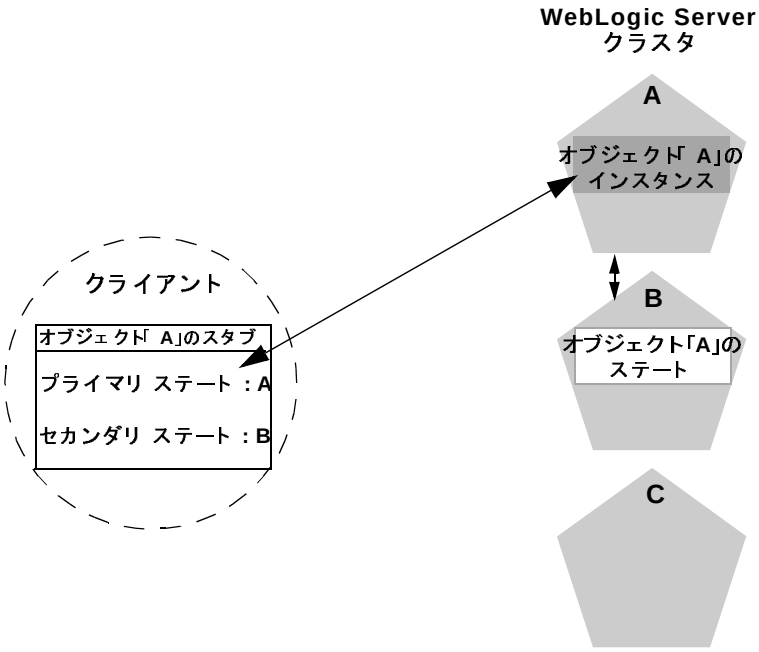
あらゆる EJB の場合と同じように、クラスタ化されたステートフル セッション EJB でもレプリカ対応 `EJBHome` スタブが利用されます。ステートフル セッション EJB のレプリケーションを使用する場合、EJB ではそのプライマリ ステートとセカンダリ ステートの位置を保持するレプリカ対応 `EJBObject` スタブも利用されます。EJB のステートは、HTTP セッション ステートの場合と同様のレプリケーション方式で維持されます。ステートフル セッション EJB でのレプリケーションについては、次の節で説明します。

ステートフル セッション Bean のレプリケーション

ステートフル セッション EJB に対して WebLogic Server で使用されるステートレプリケーションの方式は、HTTP セッション ステートのレプリケーションに使用されるものと似ています。クライアントで `EJBObject` スタブが作成されると、接続先の WebLogic Server インスタンスでは EJB のレプリケートされたステートのホストとなるセカンダリ サーバ インスタンスが自動的に選択されます。セカンダリ サーバ インスタンスは、5-6 ページの「レプリケーション グループを使用する」で定義されているものと同じルールに従って選択されます。たとえば、レプリケートするステートフル セッション EJB データのホストとなるレプリケーション グループとして動作するように、WebLogic Server インスタンスの集合を定義できます。

クライアントでは、EJB のステートをホストするクラスタ内のプライマリ サーバとセカンダリ サーバの位置が記録されたレプリカ対応スタブが受信されます。次の図は、クラスタ化されたステートフル セッション EJB にクライアントがアクセスする状況を示しています。

図 5-5　ステートフル セッション EJB にアクセスするクライアント



プライマリ サーバは、クライアントが対話する EJB の実際のインスタンスのホストとして機能します。セカンダリ サーバは、EJB のレプリケートされたステートのみを保持します。ステートのみなので、少しのメモリしか消費されません。セカンダリ サーバでは、フェイルオーバのとき以外は EJB の実際のインスタンスは作成されません。このため、セカンダリ サーバではリソースの使用が最小限に抑えられます。EJB ステートのレプリケートに備えて追加の EJB リソースをコンフィグレーションする必要はありません。

EJB ステートの変更をレプリケートする

クライアントによって EJB のステートが変更されると、ステートの変更部分がセカンダリ サーバのインスタンスにレプリケートされます。トランザクションに関係している EJB の場合、レプリケーションはトランザクションのコミットの直後に行われます。トランザクションに関係していない EJB の場合、レプリケーションは各メソッド呼び出しの後に行われます。

両方の場合で、EJB のステートの実際の変更部分だけがセカンダリ サーバにレプリケートされます。このため、レプリケーション プロセスに伴うオーバーヘッドが最小限に抑えられます。

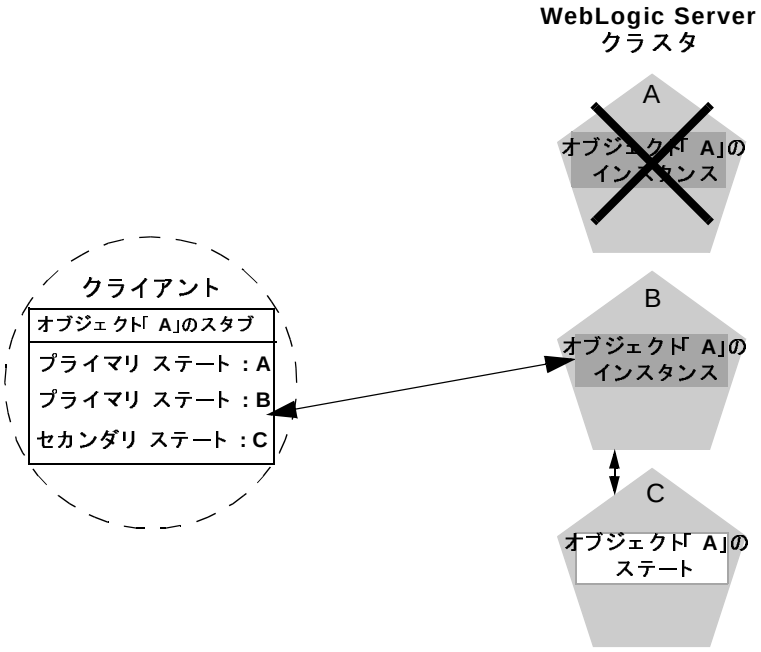
注意： EJB 仕様で説明されているように、ステートフル EJB の実際のステートはトランザクション非対応です。可能性は低いですが、EJB の現在のステートが失われることもあり得ます。たとえば、EJB の関与しているトランザクションがクライアントによってコミットされ、ステートの変更がレプリケートされる前にプライマリ サーバで障害が起きると、クライアントは以前に格納されていた EJB のステートにフェイルオーバされます。どのような障害が起きても EJB のステートを維持する必要がある場合は、ステートフル セッション EJB ではなくエンティティ EJB を使用してください。

ステートフル セッション EJB のフェイルオーバ

プライマリ サーバで障害が起きると、以降のリクエストはクライアントの EJB スタブによって自動的にセカンダリ WebLogic Server インスタンスにリダイレクトされます。この時点で、レプリケートされたステート データを使用してセカンダリ サーバで新しい EJB インスタンスが作成され、セカンダリ サーバで処理が継続されます。

フェイルオーバの後、WebLogic Server では EJB セッション ステートをレプリケートする新しいセカンダリ サーバが選択されます (クラスタ内に利用可能な別のサーバがある場合)。新しいプライマリとセカンダリのサーバ インスタンスの位置は、次に示すように、次のメソッド呼び出しのときにクライアントのレプリカ対応スタブで自動的に更新されます。

図 5-6　フェイルオーバ後にレプリカ対応スタブが更新される



エンティティ EJB

エンティティ Bean には、読み書き対応エンティティと読み込み専用エンティティの 2 種類があります。

- 読み書き対応エンティティ

  ホームで読み書き対応エンティティ Bean が検索または作成される場合は、ローカル サーバでインスタンスが取得され、そのサーバに固定されたスタブが返されます。ロード バランシングとフェイルオーバはホームのレベルでのみ行われます。クラスタにはエンティティ Bean の複数のインスタンスが存在する可能性があるため、各インスタンスは各トランザクションの前にデータベースから読み込まれ、各コミットで書き込まれなければなりません。

- 読み込み専用エンティティ

  ホームで読み込み専用エンティティ Bean が検索または作成される場合は、レプリカ対応スタブが返されます。このスタブでは、すべての呼び出しでロード バランシングが行われますが、回復可能な呼び出しエラーが発生したときに自動的にフェイルオーバは行われません。読み込み専用 Bean は、データベース読み込みを防止するためにすべてのサーバでキャッシュされます。

クラスタでの EJB の使用方法について、詳しくは『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』の「WebLogic Server EJB コンテナとサポートされるサービス」を参照してください。

エンティティ Bean および EJB ハンドルのフェイルオーバ

エンティティ Bean および EJB ハンドルのフェイルオーバでは、クラスタ内のすべてのサーバ インスタンスだけにマップされる DNS 名としてクラスタ アドレスを指定する必要があります。クラスタの DNS 名は、クラスタのメンバー以外のサーバ インスタンスにマップされません。

RMI オブジェクトのクラスタ化のサポート

WebLogic RMI には、クラスタ化されたリモート オブジェクトをビルドするための特殊な拡張機能があります。それらの拡張機能は、EJB のセクションで説明されているレプリカ対応スタブをビルドするために使用します。クラスタでの RMI の使用方法について、詳しくは『WebLogic RMI プログラマーズ ガイド』の「WebLogic RMI の機能とガイドライン」を参照してください。

オブジェクト デプロイメントの必要条件

WebLogic Server クラスタで使用する EJB をプログラムする場合は、この節の解説と『WebLogic Server エンタープライズ JavaBeans プログラマーズ ガイド』の「WebLogic Server EJB コンテナとサポートされるサービス」を参照して、クラスタ内での各種の EJB の機能について理解します。EJB のデプロイメント記述

子でクラスタ化を有効にします。クラスタ化用の XML デプロイメント要素については、『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』の「weblogic-ejb-jar.xml 文書型定義」で説明しています。

EJB またはカスタム RMI オブジェクトを開発している場合は、『WebLogic JNDI プログラマーズ ガイド』の「クラスタ環境での WebLogic JNDI の使い方」を参照して、クラスタ化されたオブジェクトの実装を JNDI ツリーにバインドすることの意味について理解してください。

他のフェイルオーバの例外

クラスタ化されたオブジェクトが多重呼び出し不変でない場合でも、WebLogic Server は、`ConnectException` または `MarshalException` が発生したときに自動フェイルオーバを実行します。どちらの例外もオブジェクトを変更できなかったことを示しているため、別のインスタンスにフェイルオーバすることでデータの矛盾が生じる危険性はありません。

固定サービスの移行

HTTP セッション ステートと EJB については、WebLogic Server クラスタはオブジェクトまたはサービスをクラスタ内の冗長サーバ上に複製することによって、高可用性とフェイルオーバを実現します。一方、JMS サーバや JTA トランザクション回復サービスなどの一部のサービスは、クラスタ内で動作しているサービスのアクティブなインスタンスが常に 1 つだけ存在するという前提のもとで設計されています。これらのタイプのサービスは、同時に 1 つのサーバ インスタンス上でのみアクティブな状態を保つため、「固定」サービスと呼ばれます。

WebLogic Server 7.0 では、あるサーバ インスタンスからクラスタ内の別のインスタンスに管理者の手で固定サービスを移行することができます。これは、サーバ インスタンスの障害への対応として、あるいは定期的な保守の一環として行います。この機能により、ホストのサーバで障害が発生した場合に冗長サーバ上でサービスを速やかに再開できるため、クラスタ内の固定サービスの可用性が向上します。

このリリースでは、JMS サーバと JTA トランザクション回復サービスの移行だけが可能です。これらのサービスはクラスタの内部であるサーバから別にサーバに移すことができるため、このマニュアルではこれらのサービスを移行可能サービスと呼びます。JMS については、単一の分散送り先セットの一部として複数の物理送り先 (キューとトピック) をコンフィグレーションできるため、単独の WebLogic Server で障害が発生した場合のサービスの継続可能性も改善されています。

注意： このリリースの WebLogic Server では、固定サービスの自動移行 (フェイルオーバ) を行うことはできません。固定サービスを手動で移行する方法については、『WebLogic JMS プログラマーズ ガイド』の「WebLogic Server の障害からの回復」を参照してください。

固定サービスの移行の仕組み

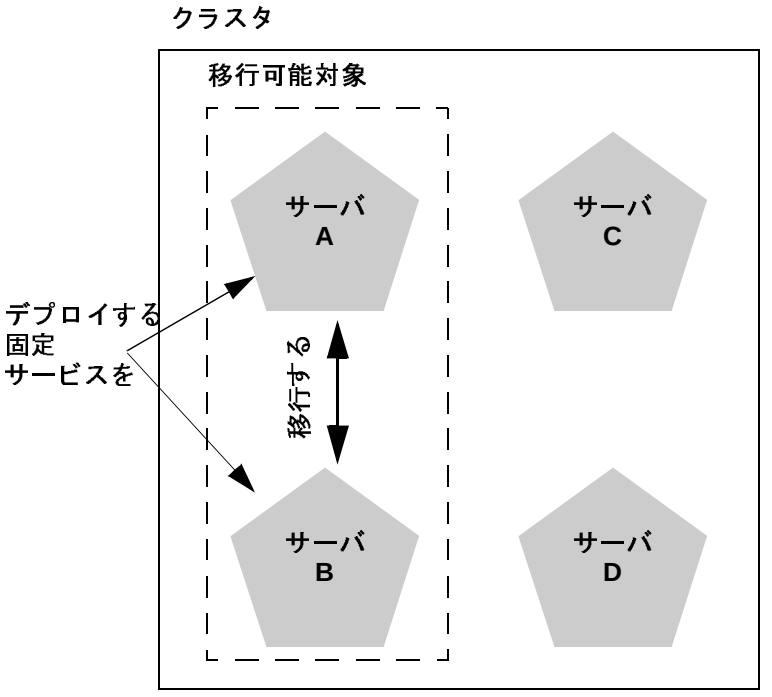
クライアントは、移行対応の RMI スタブを使用してクラスタ内の移行可能サービスにアクセスします。RMI スタブは、各時点でどのサーバが固定サービスのホストとなっているかを追跡し、それに従ってクライアントからのリクエストを転送します。たとえば、クライアントが最初に固定サービスにアクセスしたとき、スタブはクライアントのリクエストを、その時点でサービスのホストとなっているクラスタ内のサーバ インスタンスに転送します。クライアントからの次のリクエストまでにサービスが別の WebLogic Server に移動している場合、スタブはそのリクエストを最新のホスト サーバに透過的にリダイレクトします。

JMS サーバや JTA トランザクション回復サービスがクラスタ内に位置し、コンフィグレーションで移行が有効にされているとき、WebLogic Server 7.0 はそれらのサービスに対する移行対応の RMI スタブを実装します。

クラスタ内の移行可能対象サーバの定義

デフォルトでは、WebLogic Server は JTA トランザクション回復サービスまたは JMS サーバを、クラスタ内の他のどのサーバにでも移行できます。また必要に応じて、固定サービスのホストとなることのできるクラスタ内のサーバのリストを手動で編集できます。このサーバ リストは移行可能対象と呼ばれ、サービスを移行することのできるサーバを管理します。JMS の場合、移行可能対象は JMS サーバをデプロイ可能なサーバのリストも定義します。

たとえば、次の図では 4 つのサーバで構成されるクラスタを示しています。クラスタ内で、サーバ A および B は JMS サーバの移行可能対象としてコンフィグレーションされます。



上の図では、移行可能対象の設定に従って、管理者は固定サービスである JMS サーバをサーバ A からサーバ B に、またはサーバ B からサーバ A にのみ移行できます。同様に、JMS サーバをクラスタにデプロイするとき、管理者はデプロイメント先としてサーバ A または B のどちらかを選択して、サービスの移行を有効にします (管理者が移行可能対象を使用しない場合、JMS サーバはクラスタ内で使用可能などのサーバにもデプロイまたは移行できます)。

WebLogic Server では、JTA トランザクション回復サービスと JMS サーバに対して個別の移行可能対象を作成できます。これにより、必要に応じて、クラスタ内の別々のサーバ上で常に各サービスを動作させ続けることが可能になっていま

す。また逆に、JTA と JMS の両方に共通する移行可能対象として同じサーバ群を選択し、両方のサービスが確実にクラスタ内の同じサーバ上に配置されるようにすることもできます。

フェイルオーバと JDBC 接続

JDBC は、クライアント と DBMS 間の高度にステートフルなプロトコルです。JDBC では、DBMS 接続とトランザクションのステートが、DBMS プロセスとクライアント (ドライバ) の間のソケットに直接関連付けられます。このため、接続のフェイルオーバはサポートされません。WebLogic Server インスタンスが消滅すると、そのインスタンスで管理していた JDBC 接続も消滅し、DBMS は処理中のトランザクションをロールバックします。関連するアプリケーションは、現在のトランザクションを最初から始める必要があります。切断された接続に関連付けられているすべての JDBC 接続も切断されます。クラスタ化された JDBC では、簡単に再接続できます。外部クライアント アプリケーションの WebLogic データ ソースはクラスタ対応なので、接続をホストしていたサーバ インスタンスに障害が発生した場合でも、クライアントは他の接続を要求できます。

データベース インスタンスを既にレプリケートし、同期させている場合には、JDBC マルチプールを使用してデータベースのフェイルオーバをサポートできます。その場合、接続プールが存在しないか、またはプールからのデータベース接続がダウンしたために、クライアントがマルチプール内のある接続プールから接続を取得できない場合でも、WebLogic Server は、接続プールのリストの次の接続プールから接続を取得しようとします。

JDBC オブジェクトのクラスタ化の手順については、7-26 ページの「クラスタ化された JDBC をコンフィグレーションする」を参照してください。

注意： すべての接続が使用中であるプールに対してクライアントが接続を要求した場合は、例外が生成され、WebLogic Server は他のプールから接続を取得しようとしなくなります。接続プール内の接続数を増やすと、この問題に対処できます。

予約時に接続をテストするように、マルチプールに割り当てられた接続プールをコンフィグレーションする必要があります。これは、プールが接続が良好かどうかを確認し、マルチプールがリスト内の次のプールにフェイルオーバするタイミングを認識する唯一の方法です

6　クラスタ アーキテクチャ

以降の節では、WebLogic Server クラスタの各種のアーキテクチャについて説明します。

アーキテクチャとクラスタ関連の用語

この節では、クラスタおよび Web アプリケーションの構成要素を指してこのマニュアルで使用する用語を定義します。

アーキテクチャ

この節では、「アーキテクチャ」という用語は、アプリケーションの層が 1 つまたは複数のクラスタにデプロイされる仕組みを指します。

Web アプリケーションの「層」

Web アプリケーションは、複数の「層」に分けられます。「層」は、アプリケーションで提供される論理的なサービスに対応します。すべての Web アプリケーションが似通っているわけではないので、アプリケーションによっては、以下で説明する層のすべてを利用しない場合があります。また、層はアプリケーションのサービスの論理的な区分を示すものであり、必ずしもハードウェアまたはソフトウェアのコンポーネント間の物理的な区分を示すものではありません。1 つの WebLogic Server インスタンスを実行している 1 台のマシンが、以下で説明するすべての層を提供する場合もあります。

- Web 層

  Web 層では、Web アプリケーションのクライアントに対して静的なコンテンツ (単純な HTML ページなど) が提供されます。Web 層は、通常、外部クライアントが Web アプリケーションにアクセスする場合の最初のポイントになります。単純な Web アプリケーションでは、WebLogic Express、Apache、Netscape Enterprise Server、または Microsoft Internet Information Server を実行している 1 つまたは複数のマシンで構成される Web 層が存在する場合もあります。

- プレゼンテーション層

  プレゼンテーション層では、Web アプリケーションのクライアントに対して動的なコンテンツ (サーブレットや JavaServer Pages (JSP) など) が提供されます。Web アプリケーションのプレゼンテーション層は、サーブレットや JSP のホストになる WebLogic Server インスタンスのクラスタで構成されます。クラスタでアプリケーションの静的な HTML ページも提供される場合は、クラスタには Web 層とプレゼンテーション層の両方が含まれます。

- オブジェクト層

  オブジェクト層では、Web アプリケーションのクライアントに対して Java オブジェクト (エンタープライズ JavaBean や RMI クラスなど) や、それらに関連付けられたビジネス ロジックが提供されます。EJB のホストになる WebLogic Server クラスタは、オブジェクト層を提供します。

組み合わせ層アーキテクチャ

Web アプリケーションのすべての層を単一の WebLogic Server クラスタにデプロイするようなクラスタ アーキテクチャを組み合わせ層アーキテクチャと呼びます。

非武装地帯 (DMZ)

非武装地帯 (DMZ) とは、外部の、信頼性のないソースから使用できるハードウェアおよびサービスの論理的な集合のことです。ほとんどの Web アプリケーションでは、Web サーバのバンクは DMZ にあります。DMZ では、ブラウザベースのクライアントからの静的な HTML コンテンツへのアクセスが許可されています。

DMZ には、ハードウェアおよびソフトウェアに対する外部からの攻撃に備えてセキュリティが用意されている場合もあります。ただし、DMZ は信頼性のないソースから使用できるので、その安全性は内部システムよりは劣ります。たとえば、内部システムは、外部からのアクセスをすべて拒否するファイアウォールによって保護できます。DMZ は、アクセス先の個々のマシン、アプリケーション、またはポート番号を隠すファイアウォールによって保護できますが、信頼性のないクライアントからそれらのサービスにアクセスすることは可能です。

ロード バランサ

ロード バランサという用語は、このマニュアルでは、クライアントの接続リクエストを 1 つまたは複数の別々の IP アドレスに分散させる技術を指します。たとえば、単純な Web アプリケーションでは、DNS ラウンドロビン アルゴリズムがロード バランサとして使用される場合があります。大規模なアプリケーションでは、通常、Alteon WebSystems などから発売されているハードウェアベースのロード バランシング ソリューションが使用されます。このようなソリューションには、ファイアウォールのようなセキュリティ機能も用意されています。

ロード バランサには、クライアント接続をクラスタ内の特定のサーバに関連付ける機能があります。この機能は、クライアント セッション情報のインメモリ レプリケーションを使用する場合に必要です。一部のロード バランシング製品

では、インメモリ レプリケーションで使用されるプライマリ サーバおよびセカンダリ サーバを追跡する WebLogic Server クッキーを上書きしないように、クッキーの永続性メカニズム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、7-19 ページの「パッシブなクッキーの永続性をサポートするロード バランサをコンフィグレーションする」を参照してください。

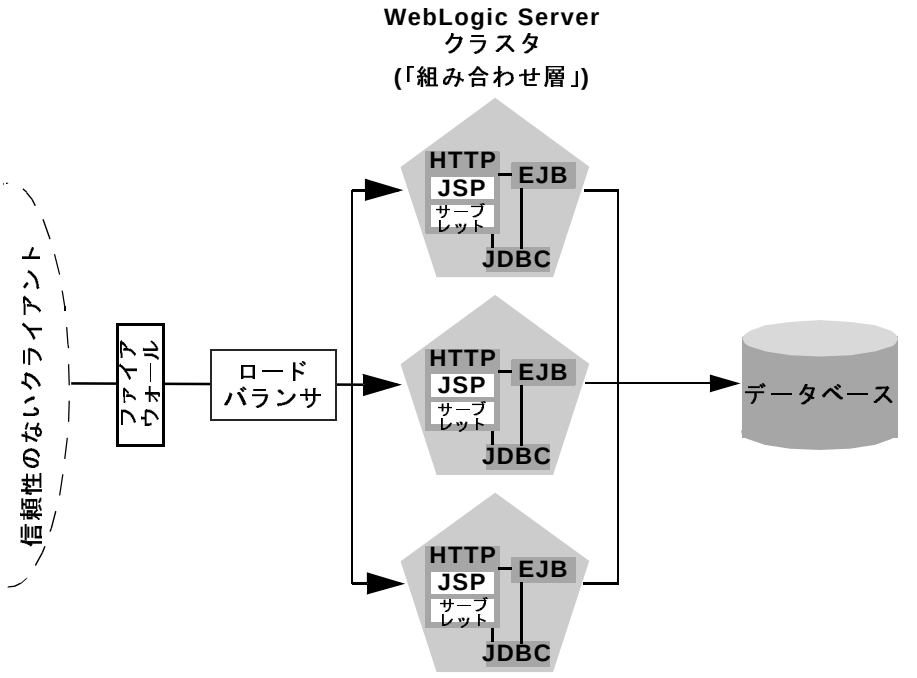
プロキシ プラグイン

プロキシ プラグインとは、WebLogic Server クラスタによって提供される、クラスタ化されたサーブレットにアクセスする Apache、Netscape Enterprise Server、または Microsoft Internet Information Server に対する WebLogic Server の拡張機能のことです。プロキシ プラグインには、WebLogic Server クラスタ内のサーブレットおよび JSP へのアクセスに対するロード バランシング ロジックが含まれます。また、クライアントのセッション状態のホストになっている主要な WebLogic Server に障害が発生した場合の、セッション状態のレプリカへのアクセスに対するロジックも含まれます。

推奨基本アーキテクチャ

推奨基本アーキテクチャは、Web アプリケーションのすべての層が同じ WebLogic Server クラスタにデプロイされる組み合わせ層アーキテクチャです。このアーキテクチャを次の図に示します。

図 6-1　推奨基本アーキテクチャ



推奨基本アーキテクチャの利点には、以下のものがあります。

- 管理の容易さ

  1 つのクラスタが静的な HTTP ページ、サーブレット、および EJB のホストになるので、WebLogic Server コンソールを使用して、Web アプリケーション全体をコンフィグレーションしたり、オブジェクトをデプロイまたはアンデプロイしたりできます。クラスタ化されたサーブレットからの恩恵を受けるために、Web サーバのバンクを別々に保持したり、WebLogic Server プロキシ プラグインをコンフィグレーションしたりする必要はありません。

- 柔軟性の高いロード バランシング

  WebLogic Server クラスタの前で直接ロード バランシング ハードウェアを使用することで、HTML コンテンツとサーブレット コンテンツ両方へのアクセスに対して高度なロード バランシング ポリシーを使用できます。たとえば、現在のサーバの負荷を検出し、クライアントのリクエストを適切に送信するよう、ロード バランサをコンフィグレーションできます。

- 堅牢なセキュリティ

  ロード バランシング ハードウェアの前にファイアウォールを配置することで、Web アプリケーションに対して、最小限のファイアウォール ポリシーを使用する非武装地帯 (DMZ) を設定できます。

- 最適なパフォーマンス

  組み合わせ層アーキテクチャでは、プレゼンテーション層のほとんどまたはすべてのサーブレットまたは JSP が通常オブジェクト層のオブジェクト (EJB または JDBC オブジェクトなど) にアクセスするアプリケーションに対して、最適なパフォーマンスを提供します。

注意 : インメモリ セッション レプリケーションでサードパーティ製のロード バランサを使用する場合は、プライマリ セッション ステートのホストとなる WebLogic Server インスタンス (接続先サーバ) へのクライアント接続を、ロード バランサが維持するようにしなければなりません。詳細については、7-19 ページの「パッシブなクッキーの永続性をサポートするロード バランサをコンフィグレーションする」を参照してください。

組み合わせ層アーキテクチャを使用しない状況

推奨基本アーキテクチャなどの組み合わせ層アーキテクチャは多くの Web アプリケーションのニーズに適していますが、クラスタのロード バランシングおよびフェイルオーバ機能を活用できる度合いは多少限定されます。ロード バランシングおよびフェイルオーバは、Web アプリケーションの各層間のインタフェースでのみ導入できます。そのため、すべての層を単一のクラスタにデプロイするときは、クライアントとクラスタの間でのロード バランシングしか行うことができません。

ロード バランシングとフェイルオーバの大部分は、クライアントとクラスタ自身の間で発生するので、組み合わせ層アーキテクチャでも、ほとんどの Web アプリケーションのニーズを満たすことができます。

しかし、組み合わせ層クラスタには、クラスタ化された EJB へのメソッド呼び出しに対してロード バランシングを実行する機会がありません。クラスタ化されたオブジェクトは、クラスタ内のすべての WebLogic Server インスタンスにデプロイされるので、各オブジェクト インスタンスは各サーバでローカルに使用できます。WebLogic Server では、常にローカルのオブジェクト インスタンスを

選択することにより、クラスタ化された EJB に対するメソッド呼び出しが最適化されます。リクエストをリモート オブジェクトに分散させないので、ネットワークのオーバーヘッドが増加しません。

この連結方式は、ほとんどの場合で、異なるサーバへの各メソッド リクエストに対してロード バランシングを行うよりも効率的です。ただし、各サーバの負荷が不均衡な場合は、ローカルでメソッドを処理するよりも、リモート オブジェクトにメソッド呼び出しを送信する方が結果的に効率的になる場合もあります。

クラスタ化された EJB へのメソッド呼び出しに対してロード バランシングを行うには、次の節で説明するように、Web アプリケーションのプレゼンテーション層とオブジェクト層を物理的に異なるクラスタ上に分割する必要があります。

組み合わせ層アーキテクチャと多層アーキテクチャのどちらかに決定する際には、プレゼンテーション層によるオブジェクト層の呼び出しの頻度を考慮してください。プレゼンテーション オブジェクトが通常オブジェクト層を呼び出す場合、組み合わせ層アーキテクチャは、多層アーキテクチャよりも高いパフォーマンスを示す可能性があります。

推奨多層アーキテクチャ

この節では、推奨多層アーキテクチャについて説明します。このアーキテクチャでは、アプリケーションの各層が複数の異なったクラスタにデプロイされます。

推奨多層アーキテクチャでは、2 つの異なる WebLogic Server クラスタを使用します。1 つは静的な HTTP コンテンツを提供するクラスタであり、もう 1 つはクラスタ化された EJB を提供するクラスタです。多層クラスタは、以下のような Web アプリケーションにお勧めします。

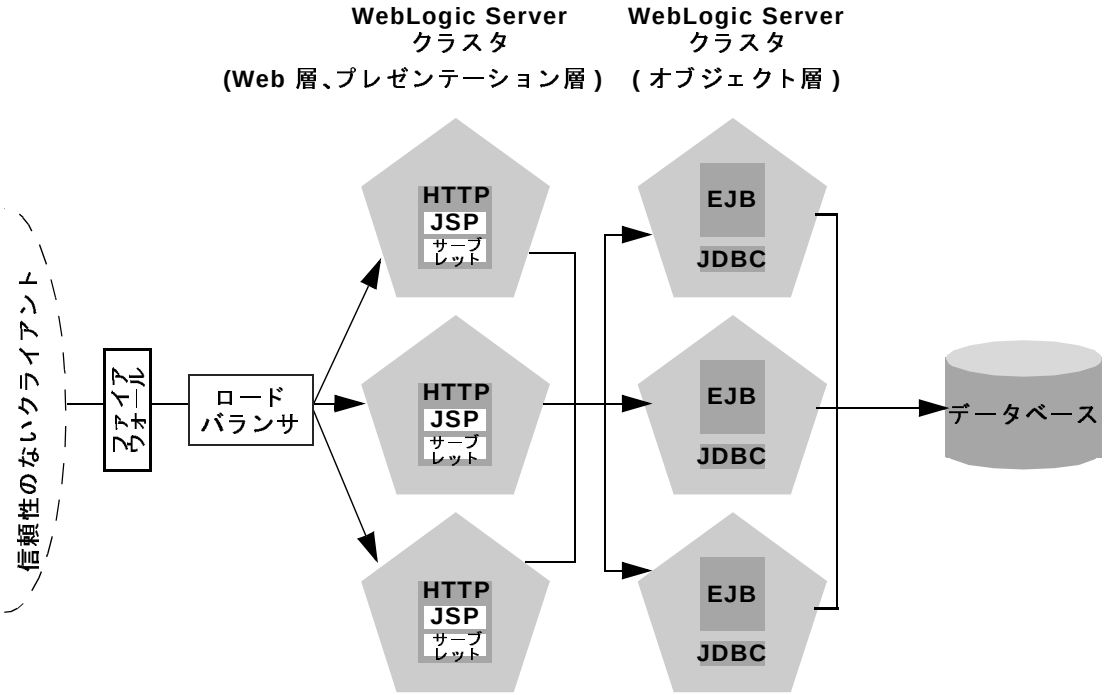
- クラスタ化された EJB へのメソッド呼び出しに対するロード バランシングが必要なアプリケーション
- HTTP コンテンツを提供するサーバとクラスタ化されたオブジェクトを提供するサーバの間により柔軟性の高いロード バランシングが必要なアプリケーション

- より高い可用性 (シングル ポイント障害がより少ないこと) が必要なアプリケーション

注意 : 多層アーキテクチャを考慮する場合は、プレゼンテーション層からオブジェクト層への呼び出しの頻度を考慮してください。プレゼンテーション オブジェクトが通常オブジェクト層を呼び出す場合、組み合わせ層アーキテクチャは、多層アーキテクチャよりも高いパフォーマンスを示す可能性があります。

次の図に、推奨多層アーキテクチャを示します。

図 6-2　推奨多層アーキテクチャ



ハードウェアとソフトウェアの物理レイヤ

推奨多層アーキテクチャでは、アプリケーションの各層が 2 つの異なった物理レイヤ (ハードウェアとソフトウェア) 上に配置されます。

Web/ プレゼンテーション レイヤ

Web/ プレゼンテーション レイヤは、静的な HTTP ページ、サーブレット、および JSP の専用ホストになっている WebLogic Server インスタンスのクラスタで構成されます。このサーブレット クラスタは、クラスタ化されたオブジェクトのホストにはなりません。その代わりに、プレゼンテーション層クラスタ内のサーブレットは、オブジェクト レイヤにある異なる WebLogic Server クラスタ内のクラスタ化されたオブジェクトのクライアントとして機能します。

オブジェクト レイヤ

オブジェクト レイヤは、Web アプリケーションで必要なクラスタ化されたオブジェクト (EJB オブジェクトおよび RMI オブジェクト) 専用のホストになる WebLogic Server インスタンスのクラスタで構成されます。専用クラスタにオブジェクト層のホストを配置することで、クラスタ化されたオブジェクトへのアクセスに対するデフォルトの連結の最適化 (4-9 ページの「連結されたオブジェクトの最適化」参照) が失われます。ただし、次の節で説明するように、特定のクラスタ化されたオブジェクトへの各メソッド呼び出しに対するロード バランシングは可能になります。

多層アーキテクチャの利点

多層アーキテクチャには、以下の利点があります。

- EJB メソッドのロード バランシング

  サーブレットと EJB のホストを異なるクラスタに配置することで、サーブレットの EJB へのメソッド呼び出しで、複数のサーバにわたるロード バランシングが可能になります。このプロセスについては、6-10 ページの「多層アーキテクチャでのクラスタ化オブジェクトのロード バランシング」で詳しく説明しています。

- 改良されたサーバ ロード バランシング

  プレゼンテーション層とオブジェクト層を別々のクラスタに分割することで、Web アプリケーションの負荷分散に関するオプションが増加します。たとえば、アプリケーションが EJB コンテンツよりも頻繁に HTTP およびサーブレット コンテンツにアクセスする場合は、プレゼンテーション層のクラスタで数多くの WebLogic Server インスタンスを使用し、EJB のホストになるサーバを少数にしてアクセスを集中させることができます。

- より高い可用性

  追加の WebLogic Server インスタンスを利用することで、多層アーキテクチャの障害ポイントは基本クラスタ アーキテクチャよりも少なくなります。たとえば、EJB のホストになっている WebLogic Server に障害が発生しても、Web アプリケーションの HTTP やサーブレットのホストとなる機能には影響しません。

- 改良されたセキュリティ オプション

  プレゼンテーション層とオブジェクト層を別々のクラスタに分割することで、DMZ にサーブレット /JSP クラスタだけを配置するファイアウォール ポリシーを使用できます。信頼性のないクライアントからの直接アクセスを拒否することで、クラスタ化されたオブジェクトのホストになっているサーバの保護を強化できます。詳細については、6-21 ページの「クラスタ アーキテクチャのセキュリティ オプション」を参照してください。

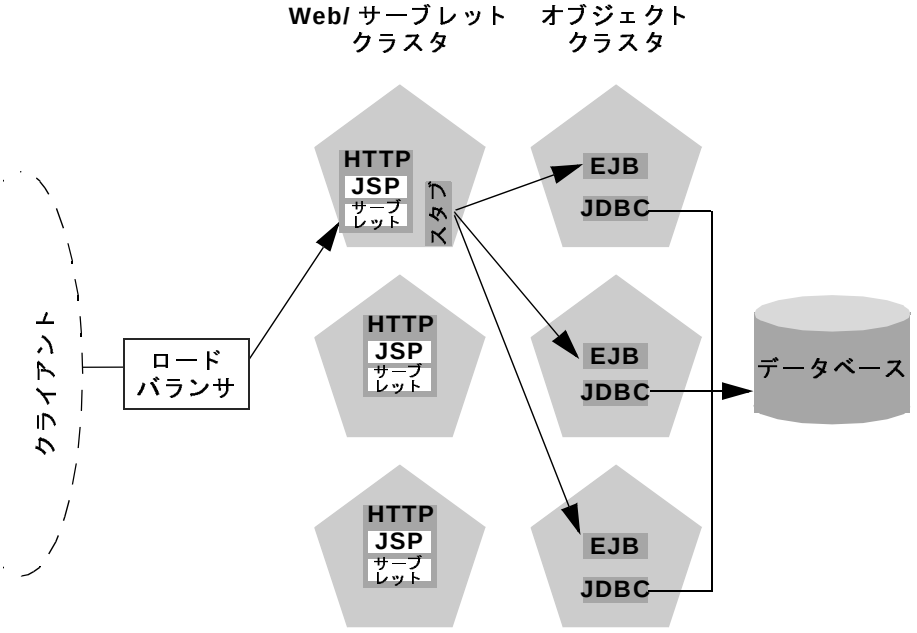
多層アーキテクチャでのクラスタ化オブジェクトのロード バランシング

4-9 ページの「連結されたオブジェクトの最適化」で説明しているように、クラスタ化されたオブジェクトに対する WebLogic Server の連結の最適化は、クラスタ化されたオブジェクト (EJB または RMI クラス) のホストが、オブジェクトを呼び出すレプリカ対応スタブと同じサーバ インスタンスに配置されることに基づいています。

オブジェクト層を分離させることの最終的な効果は、クライアント (HTTP クライアント、Java クライアント、またはサーブレット) が、クラスタ化されたオブジェクトのホストになっている同じサーバ上のレプリカ対応スタブを取得しないことにあります。このため、WebLogic Server では、連結の最適化を使用できず

(4-9 ページの「連結されたオブジェクトの最適化」参照)、クラスタ化されたオブジェクトに対するサーブレット呼び出しでは、レプリカ対応スタブに含まれるロジックに従って自動的にロード バランシングが実行されます。次の図に、多層アーキテクチャ内のクラスタ化された EJB インスタンスにアクセスするクライアントを示します。

図 6-3 多層アーキテクチャでのオブジェクトのロード バランシング



クライアント接続の経路をトレースすると、異なるハードウェアとソフトウェアにオブジェクト層を分離させる意味が理解できます。

1. HTTP クライアントは、Web/ サーブレット クラスタ内の複数ある WebLogic Server インスタンスのいずれか 1 つに接続し、ロード バランサを経由して最初のサーバに到達します。
2. クライアントは WebLogic Server クラスタにホストが配置されたサーブレットにアクセスします。

3. サーブレットは Web アプリケーションで必要になるクラスタ化されたオブジェクトのクライアントとして機能します。上記の例では、サーブレットはステートレス セッション EJB にアクセスします。

   サーブレットは、クラスタ化されたオブジェクトのホストになる、WebLogic Server クラスタにある EJB をルックアップします。サーブレットは、Bean のレプリカ対応スタブを取得します。スタブには、Bean のホストになるすべてのサーバのアドレスと、Bean のレプリカへのアクセスに対するロード バランシング ロジックが示されています。

   注意： EJB のレプリカ対応スタブおよび EJB ホームのロード アルゴリズムは、EJB デプロイメント記述子の各要素を使用して指定します。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「weblogic-ejb-jar.xml 文書型定義」を参照してください。

4. 次に EJB にアクセスしたとき (たとえば、別のクライアントに対する応答として)、サーブレットは、Bean のスタブに示されているロード バランシング ロジックを使用してレプリカを見つけます。上記の例では、複数のメソッド呼び出しがロード バランシングのラウンドロビン アルゴリズムを使用して送信されます。

この例では、同じ WebLogic Server クラスタがサーブレットと EJB の両方のホストになっている場合 (推奨基本アーキテクチャ参照)、WebLogic Server では EJB のリクエストに対するロード バランシングは実行されません。その代わり、サーブレットは常に、ローカル サーバがホストになっている EJB レプリカでメソッドを呼び出します。ローカルの EJB インスタンスを使用した方が、別のサーバにある EJB に対してリモート メソッド呼び出しを行うよりも効率的です。しかし、多層アーキテクチャでは、EJB メソッド呼び出しのロード バランシングを必要とするアプリケーションに対してリモート EJB アクセスが可能です。

多層アーキテクチャのコンフィグレーションに関する注意

多層アーキテクチャではクラスタ化されたオブジェクトへの呼び出しに対するロード バランシングが提供されるので、システムでは通常、組み合わせ層アーキテクチャよりも多い IP ソケットが利用されます。特に、ソケット使用のピーク時は、サーブレットおよび JPS のホストになるクラスタ内の各 WebLogic Server では以下のソケットが最大限に使用される可能性があります。

- プライマリ サーバとセカンダリ サーバの間で HTTP セッション状態をレプリケートするためのソケット
- EJB クラスタ内の各 WebLogic Server に 1 つずつある、リモート オブジェクトにアクセスするためのソケット

たとえば、図 6-2 では、サーブレット /JSP クラスタ内の各サーバは最大で 5 つのソケットをオープンする可能性があります。この最大数は、プライマリおよびセカンダリの各セッション ステートが均等にサーブレット クラスタ全体に分散していて、サーブレット クラスタ内の各サーバがオブジェクト クラスタ内の各サーバのリモート オブジェクトに同時にアクセスしたという、最悪のケースを表しています。ほとんどの場合、実際に使用されるソケット数は最大数より小さくなります。

多層アーキテクチャで pure-Java ソケット実装を使用する場合は、ソケットの最大使用に対応できるだけのソケット リーダー スレッドをコンフィグレーションしていることを確認してください。詳細については、2-7 ページの「Java ソケット実装用にリーダー スレッドをコンフィグレーションする」を参照してください。

多層アーキテクチャではハードウェア ロード バランサを使用するため、インメモリ セッション ステート レプリケーションを使用する場合は、クライアントの接続先サーバに対するセッション維持型の接続を保持するように、ロード バランサをコンフィグレーションしなければなりません。詳細については、7-18 ページの「EJB と RMI のロード バランシング方式をコンフィグレーションする」を参照してください。

多層アーキテクチャに関する制限

この節では、多層クラスタ アーキテクチャでの制限の概要を示します。

連結の最適化が行われない

推奨多層アーキテクチャでは、連結方式を使用してオブジェクト呼び出しを最適化できないので、Web アプリケーションでは、クラスタ化されたオブジェクトに対するすべてのメソッド呼び出しでネットワークのオーバーヘッドが発生しま

す。ただし、このオーバーヘッドは、6-9 ページの「多層アーキテクチャの利点」で説明した利点のいずれかが Web アプリケーションで必要な場合には許容できる場合もあります。

たとえば、Web クライアントがサーブレットおよび JSP を頻繁に使用するものの、クラスタ化されたオブジェクトへのアクセスは比較的少ない場合、多層アーキテクチャではサーブレットおよびオブジェクトの負荷を適切に集中させることができます。各サーバの処理能力を最大限に利用しながら、10 個の WebLogic Server インスタンスを含むサーブレット クラスタと 3 個の WebLogic Server インスタンスを含むオブジェクト クラスタをコンフィグレーションできます。

ファイアウォールに関する制限

多層アーキテクチャのサーブレット クラスタとオブジェクト クラスタの間にファイアウォールを配置する場合は、オブジェクト クラスタ内のすべてのサーバを IP アドレスではなく、外部に公開されている DNS 名にバインドさせる必要があります。サーバを IP アドレスにバインドさせると、アドレス変換に関する問題が発生し、サーブレット クラスタが各サーバ インスタンスにアクセスできなくなる場合があります。

WebLogic Server インスタンスの内部 DNS 名と外部 DNS 名が同じでない場合、サーバ インスタンスの `ExternalDNSName` 属性を使用して、サーバの外部 DNS 名を定義します。ファイアウォールの外で、`externalDNSName` はサーバの外部 IP アドレスに変換されます。Administration Console の [サーバ | コンフィグレーション | 一般] タブで、この属性を設定します。Administration Console オンライン ヘルプの「[サーバ] --> [コンフィグレーション] --> [一般]」を参照してください。

ファイアウォールがネットワーク アドレス変換を行うコンフィグレーションでは、クライアントが t3 およびデフォルト チャネルを使用して WebLogic Server にアクセスする場合を除いて、`ExternalDNSName` を使用する必要があります。たとえば、ファイアウォールがネットワーク アドレス変換を行い、クライアントがプロキシプラグイン経由で HTTP を使用して WebLogic Server にアクセスするコンフィグレーションでは、`ExternalDNSName` を使用する必要があります。

注意： `ExternalDNSName` には IP アドレスを使用しないでください。ExternalDNSName は実際のドメイン名でなければなりません。

推奨プロキシ アーキテクチャ

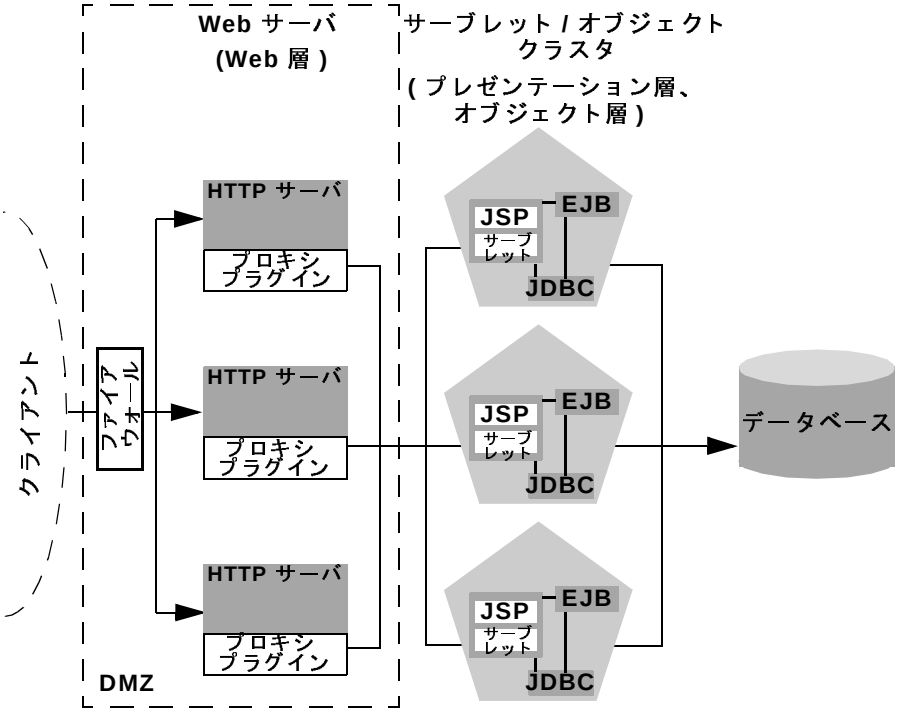
既存の Web サーバと連携して動作するよう、WebLogic Server クラスタをコンフィグレーションすることができます。そのようなアーキテクチャでは Web サーバのバンクは Web アプリケーションの静的な HTTP コンテンツを提供し、WebLogic プロキシ プラグインまたは `HttpClusterServlet` を使用してクラスタにサーブレット リクエストおよび JSP リクエストを送信します。

以下の節では、2 種類の代替プロキシ アーキテクチャについて説明します。

2 層プロキシ アーキテクチャ

次の図に示すような 2 層プロキシ アーキテクチャは、静的な HTTP サーバのホストが Web サーバのバンクに配置されるという点以外は、6-4 ページの「推奨基本アーキテクチャ」とほぼ同じです。

図 6-4　2 層プロキシ アーキテクチャ



ハードウェアとソフトウェアの物理レイヤ

2 層プロキシ アーキテクチャには、ハードウェアとソフトウェアの物理レイヤが 2 つあります。

Web レイヤ

プロキシ アーキテクチャでは、アプリケーションの Web 層を提供するタスクに特化したハードウェアとソフトウェアのレイヤが利用されます。この物理的な Web レイヤは、以下のアプリケーションの組み合わせのいずれか 1 つのホストになる、1 つまたは複数の同じようにコンフィグレーションされたマシンで構成されます。

- WebLogic Server と `HttpClusterServlet`
- Apache と WebLogic Server Apache (プロキシ) プラグイン
- Netscape Enterprise Server と WebLogic Server NSAPI (プロキシ) プラグイン
- Microsoft Internet Information Server と WebLogic Server Microsoft-IIS (プロキシ) プラグイン

選択する Web サーバ ソフトウェアに関係なく、Web サーバの物理層では静的な Web ページのみが提供されるようにする必要があります。動的なコンテンツ (サーブレットや JSP) は、プロキシ プラグインまたは `HttpClusterServlet` を経由して、プレゼンテーション層のサーブレットや JSP のホストになる WebLogic Server クラスタにプロキシされます。

サーブレット / オブジェクト レイヤ

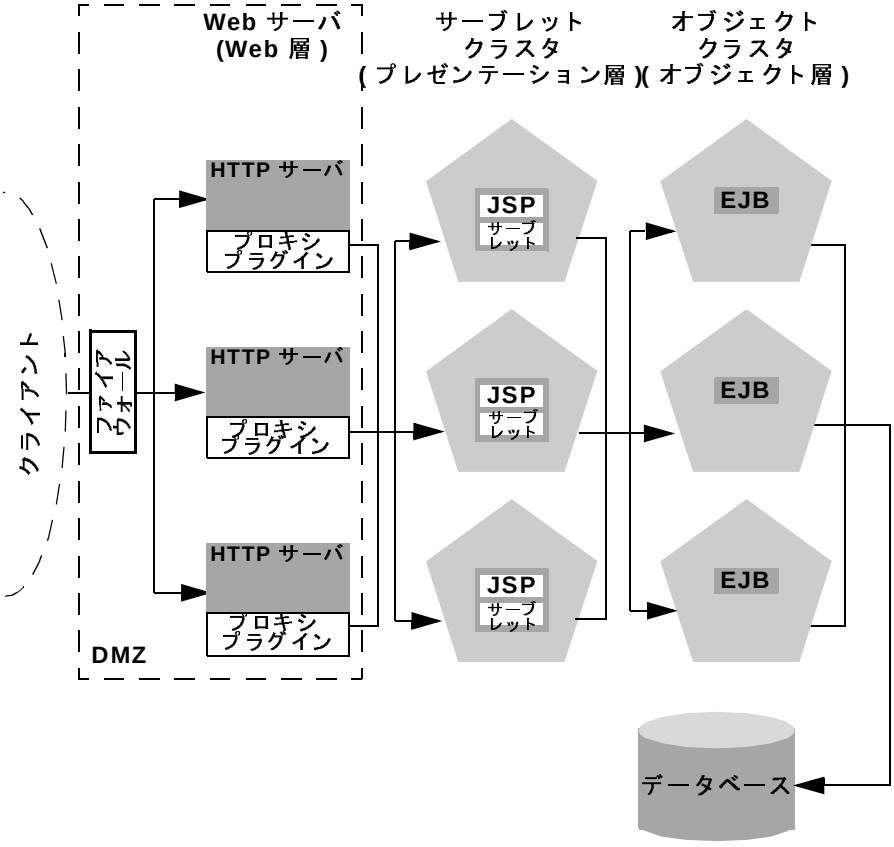
推奨 2 層プロキシ アーキテクチャでは、プレゼンテーション層およびオブジェクト層のホストは WebLogic Server インスタンスのクラスタに配置されます。このクラスタは、単一のマシンまたは複数の異なるマシンにデプロイできます。

サーブレット / オブジェクト レイヤは、組み合わせ層クラスタ (「推奨基本アーキテクチャ」参照) とは、アプリケーションのクライアントに静的な HTTP コンテンツを提供しないという点で異なります。

多層プロキシ アーキテクチャ

また、Web サーバのバンクを、プレゼンテーション層およびオブジェクト層のホストになる 1 対の WebLogic Server クラスタに対するフロント エンドとして使用することもできます。このアーキテクチャを次の図に示します。

図 6-5　多層プロキシ アーキテクチャ



このアーキテクチャには、「推奨多層アーキテクチャ」と同じ利点 (および同じ制限) があります。異なる点は、WebLogic プロキシ プラグインを利用する Web サーバの異なるバンクに Web 層が配置されることだけです。

プロキシ アーキテクチャの利点

スタンドアロン Web サーバとプロキシ プラグインの使用には、以下のような利点があります。

- 既存のハードウェアの利用

  静的な HTTP コンテンツをクライアントに提供する Web アプリケーション アーキテクチャが既に存在する場合は、1 つまたは複数の WebLogic Server クラスタを持つ既存の Web サーバを簡単に統合して、動的な HTTP およびクラスタ化されたオブジェクトを提供できます。

- 一般的なファイアウォール ポリシー

  Web アプリケーションのフロントエンドで Web サーバ プロキシを使用することで、一般的なファイアウォール ポリシーを使用して DMZ を定義できます。通常は、アーキテクチャの残りの WebLogic Server クラスタへの直接接続を禁止している間は、DMZ 内に引き続き Web サーバを配置することができます。上記の図には、この DMZ ポリシーが示されています。

プロキシ アーキテクチャの制限

スタンドアロン Web サーバとプロキシ プラグインの使用には、以下のような、Web アプリケーションに関する制限があります。

- 管理の追加

  プロキシ アーキテクチャ内の Web サーバは、サードパーティ ユーティリティを使用してコンフィグレーションする必要があり、WebLogic Server 管理ドメインには表示されません。また、クラスタ化されたサーブレットへのアクセスおよびフェイルオーバの恩恵を受けるためには、Web サーバに WebLogic プロキシ プラグインをインストールおよびコンフィグレーションする必要があります。

- ロード バランシング オプションの制限

  プロキシ プラグインまたは `HttpClusterServlet` を使用して、クラスタ化されたサーブレットにアクセスする場合、ロード バランシング アルゴリズムは単純なラウンドロビン方式に制限されます。

プロキシ プラグインとロード バランサ

WebLogic Server クラスタでロード バランサを直接使用すると、サーブレット リクエストのプロキシに関する利点がもたらされます。まず、ロード バランサを持つ WebLogic Server を使用することにより、クライアントの設定における追加管理が不要になります。HTTP サーバのレイヤを別に設定し保持する必要もなく、1 つまたは複数のプロキシ プラグインをインストールおよびコンフィグレーションする必要もありません。また、Web プロキシ レイヤの削除により、クラスタへのアクセスに必要なネットワーク接続数も削減されます。

ロード バランシング ハードウェアを使用することで、システムの能力に適合したロード バランシング アルゴリズムをより柔軟に定義できるようになります。使用するロード バランシング ハードウェアでサポートされている、任意のロード バランシング方式 (ロードベースのポリシーなど) を使用できます。プロキシ プラグインまたは `HttpClusterServlet` を使用する場合、クラスタ化されたサーブレットへのリクエストについては、単純なラウンドロビン アルゴリズムに制限されます。

ただし、インメモリ セッション ステート レプリケーションを使用している場合、サードパーティ製のロード バランサを使用するには、さらにコンフィグレーションを行う必要があります。この場合は、クライアントがプライマリ セッション ステート情報にアクセスできるようにするため、クライアントと接続先のサーバ間でセッション維持型の接続を保持するように、ロード バランサをコンフィグレーションしなければなりません。プロキシは自動的にセッション維持型の接続を保持するため、プロキシ プラグインを使用する場合は特別なコンフィグレーションは不要です。

クラスタ アーキテクチャのセキュリティ オプション

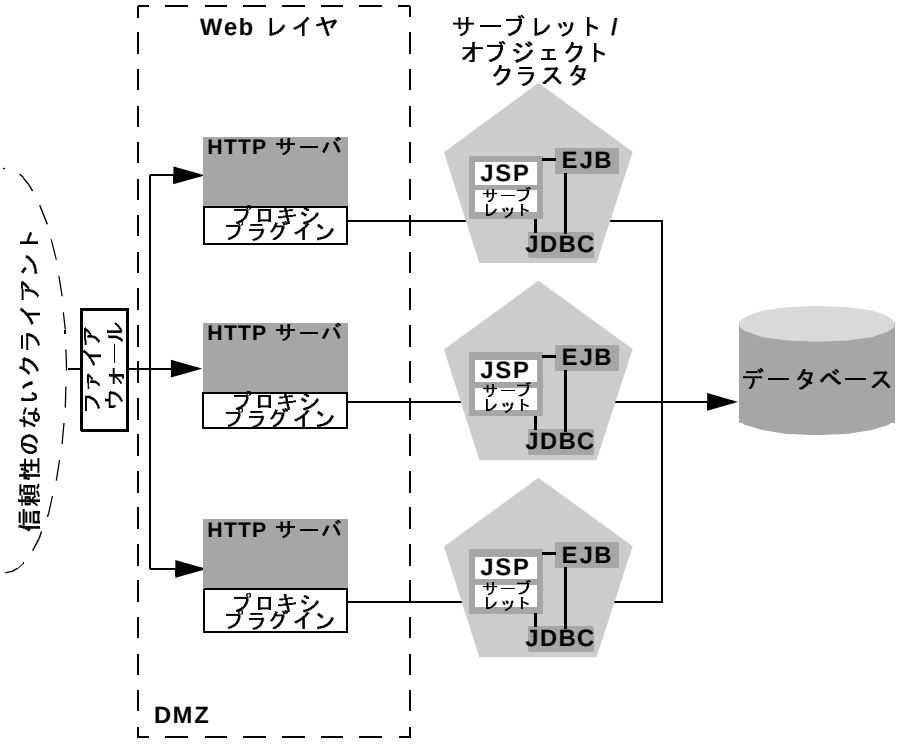
推奨コンフィグレーションにあるハードウェアとソフトウェアの物理レイヤの間の境界には、Web アプリケーションの非武装地帯 (DMZ) を定義できるポイントがあります。ただし、すべての境界で物理的なファイアウォールがサポートされているわけではありません。特定の境界で典型的なファイアウォール ポリシーのサブセットがサポートされているだけです。

以降の節では、DMZ を定義して、さまざまなレベルのアプリケーション セキュリティを作成する方法について説明します。

プロキシ アーキテクチャの基本ファイアウォール

基本ファイアウォール コンフィグレーションでは、信頼性のないクライアントと Web サーバ レイヤの間に 1 つのファイアウォールを使用します。このファイアウォール コンフィグレーションは、推奨基本アーキテクチャまたは推奨多層アーキテクチャで使用できます。

図 6-6　ファイアウォールを使用する基本プロキシ アーキテクチャ



上記のコンフィグレーションでは、1 つのファイアウォールで任意のポリシー (アプリケーションレベルの制限、NAT、IP マスカレード) の組み合わせを使用して、3 つの HTTP サーバへのアクセスをフィルタ処理しています。ファイアウォールの最も重要な役割は、システム内のその他のサーバへのアクセスを拒否することです。つまり、信頼性のないクライアントからは、サーブレット レイヤ、オブジェクト レイヤ、およびデータベースにはアクセスできないようにする必要があります。

物理的なファイアウォールは、DMZ 内の Web サーバの前にも後ろにも配置できます。Web サーバの前にファイアウォールを配置すると、Web サーバへのアクセスを許可し、その他のシステムへのアクセスを拒否するだけで済むので、ファイアウォール ポリシーを簡素化できます。

注意： 3 つの Web サーバと WebLogic Server クラスタの間にファイアウォールを配置する場合は、すべてのサーバ インスタンスを IP アドレスではなく、外部に公開されている DNS 名にバインドさせる必要があります。この作業を行うことで、プロキシ プラグインは自由にクラスタ内の各サーバに接続できるようになり、6-30 ページの「ファイアウォールについての考慮事項」で説明したようなアドレス変換エラーが発生しなくなります。

WebLogic Server インスタンスの内部 DNS 名と外部 DNS 名が同じでない場合、サーバ インスタンスの `ExternalDNSName` 属性を使用して、サーバの外部 DNS 名を定義します。ファイアウォールの外で、`externalDNSName` はサーバの外部 IP アドレスに変換されます。Administration Console の [サーバ | コンフィグレーション | 一般] タブで、この属性を設定します。Administration Console オンライン ヘルプの「[サーバ] --> [コンフィグレーション] --> [一般]」を参照してください。

たとえば、クライアントがデフォルト チャネルと T3 を使用して WebLogic Server にアクセスする場合、WebLogic Server インスタンスの内部 DNS 名と外部 DNS 名が異なっていても、`ExternalDNSName` 属性を設定しないでください。

基本ファイアウォール コンフィグレーションの DMZ

Web サーバ レイヤへのアクセス以外のすべてのアクセスを拒否することによって、基本ファイアウォール コンフィグレーションでは、3 つの Web サーバのみが含まれる小規模な DMZ が作成されます。ただし、DMZ をどんなに慎重に定義しても、悪意のあるクライアントがプレゼンテーション層とオブジェクト層のホストになっているサーバにアクセスする可能性があることは考慮に入れておいてください。

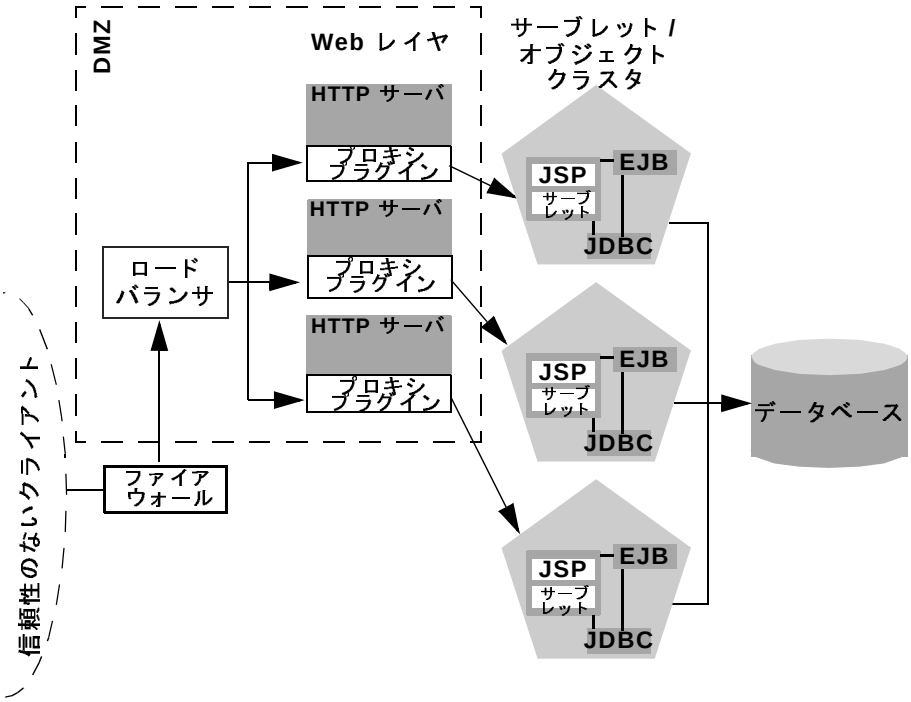
たとえば、ハッカーが Web サーバのホストになっているマシンの 1 つにアクセスしたと仮定します。アクセスのレベルによっては、動的なコンテンツを求めて Web サーバがアクセスする、プロキシされたサーバに関する情報を、ハッカーが入手できる場合もあります。

DMZ をより慎重に定義する場合には、追加のファイアウォールを配置できます (6-27 ページの「共有データベースに対するセキュリティの追加」参照)。

ファイアウォールとロード バランサを組み合わせる

推奨クラスタ アーキテクチャでロード バランシング ハードウェアを使用する場合は、基本ファイアウォールとの関連を考慮してハードウェアのデプロイ方法を決定する必要があります。数多くのハードウェア ソリューションでは、ロード バランシング サービスに加えてセキュリティ機能も提供されていますが、ほとんどのサイトでは Web アプリケーションの防御の最前線としてファイアウォールが使用されています。通常、ファイアウォールでは、Web トラフィックを制限するための、最もよくテストされた一般的なセキュリティ ソリューションが提供されます。ファイアウォールは、次の図に示すようにロード バランシング ハードウェアの前で使用する必要があります。

図 6-7　ファイアウォールとロード バランサを使用する基本プロキシ アーキテクチャ



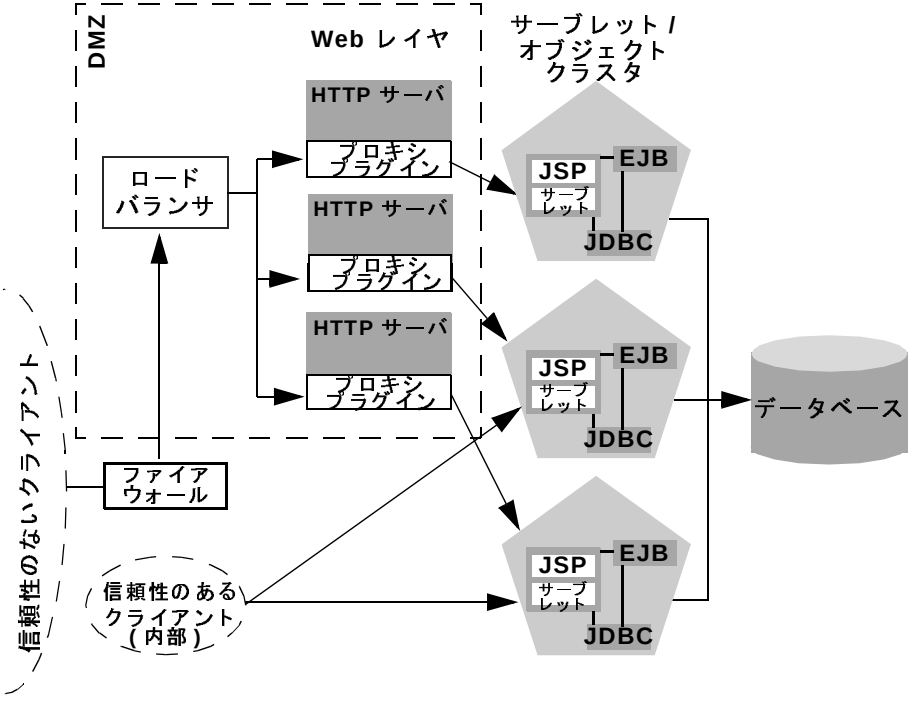
上記の設定では、Web 層を含む DMZ 内にロード バランサが配置されています。このコンフィグレーションでファイアウォールを使用すると、ファイアウォールはロード バランサへのアクセスを制限するだけで済むので、セキュリティ ポリシーの管理を簡素化できます。また、この設定では、以下で説明するような、Web アプリケーションにアクセスする内部クライアントをサポートするサイトの管理を簡素化することもできます。

内部クライアントに対してファイアウォールを拡張する

Web アプリケーションへの直接アクセスを必要とする内部クライアント (独自の Java アプリケーションを実行するリモート マシンなど) をサポートする場合は、プレゼンテーション層への制限されたアクセスを許可できるよう、基本ファイアウォール コンフィグレーションを拡張できます。アプリケーションへのアクセスを拡張する方法は、リモート クライアントを信頼性のある接続として扱うか、または信頼性のない接続として扱うかによって変わります。

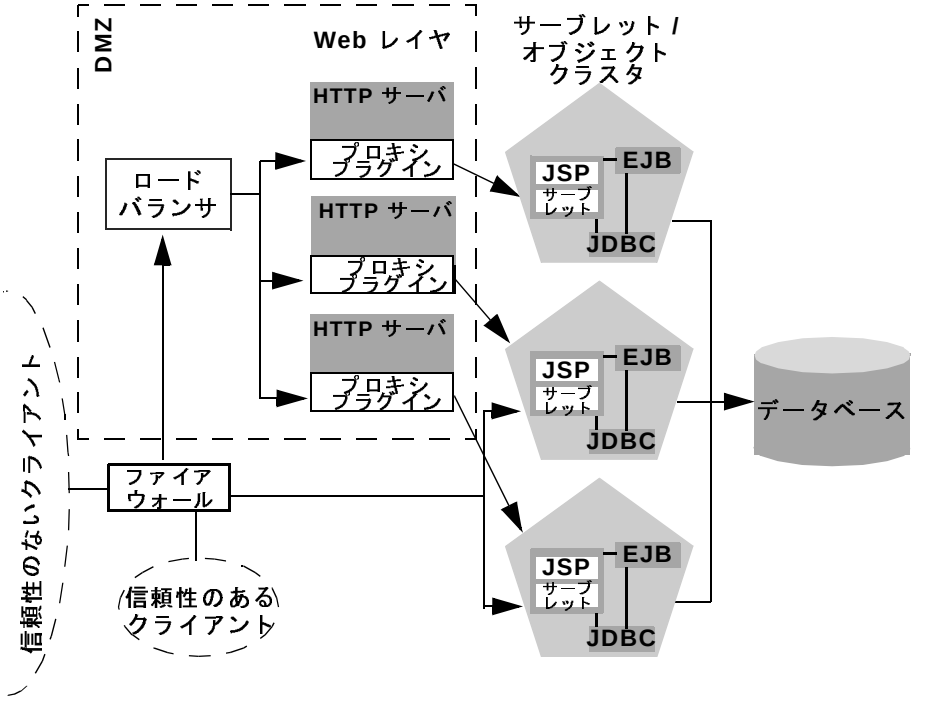
仮想プライベート ネットワーク (VPN) を使用して、リモート クライアントをサポートする場合は、クライアントを信頼性のある接続として扱うことができ、クライアントはファイアウォールの向こう側のプレゼンテーション層に直接接続できます。このコンフィグレーションは、次のようになります。

図 6-8　VPN ユーザのファイアウォール経由のアクセスは制限される



VPN を使用しない場合は、Web アプリケーションへのすべての接続を (独自のクライアント アプリケーションを使用するリモート サイトからの接続であっても) 信頼性のない接続として扱う必要があります。この場合は、次の図に示すように、プレゼンテーション層のホストになっている WebLogic Server インスタンスへのアプリケーションレベルの接続を許可するよう、ファイアウォール ポリシーを変更できます。

図 6-9 アプリケーション コンポーネントのファイアウォール経由のアクセスは制限される



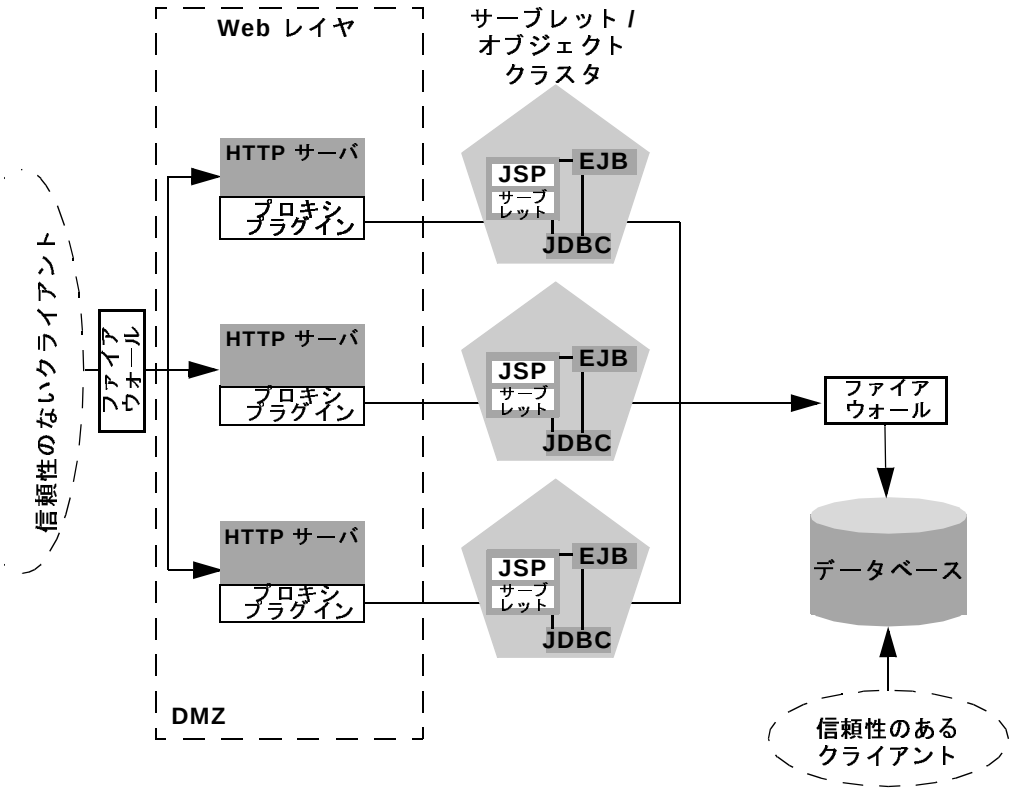
共有データベースに対するセキュリティの追加

Web アプリケーションの内部データと外部から入手できるデータの両方をサポートする 1 つのデータベースを使用する場合は、データベースにアクセスするオブジェクト レイヤの間に強固な境界を配置することを検討する必要があります。この場合、ファイアウォールを追加することによって、6-21 ページの「プロキシ アーキテクチャの基本ファイアウォール」で説明されている DMZ 境界を簡単に強化できます。

ファイアウォールが 2 つあるコンフィグレーションの DMZ

次のコンフィグレーションには、Web アプリケーションと内部 (信頼性のある) クライアントによって共有されているデータベース サーバの前に追加のファイアウォールが配置されています。このコンフィグレーションでは、最初のファイアウォールが万一破られた場合や、ハッカーが最終的にオブジェクト層のホストになっているサーバにアクセスした場合のための追加のセキュリティが提供されています。プロダクション環境では、この環境はあり得ません。サイトでは、ハッカーがオブジェクト レイヤにあるマシンにアクセスするよりもずっと前に、悪意のある侵入を検出し、くい止める機能が必要になります。

図 6-10　ファイアウォールが 2 つあるアーキテクチャの DMZ



上記のコンフィグレーションでは、オブジェクト層とデータベースの間の境界は追加のファイアウォールによって強固になっています。ファイアウォールは、オブジェクト層のホストになっている WebLogic Server からの JDBC 接続以外のすべての接続を拒否する厳密なアプリケーションレベルのポリシーを保持します。

問題の回避

以降の節では、クラスタ アーキテクチャを決定する際に留意すべき考慮事項を示します。

管理サーバについての考慮事項

クラスタに参加している WebLogic Server インスタンスを起動する場合、各管理対象サーバは、そのクラスタを含むドメインのコンフィグレーション情報を管理している管理サーバに接続できなくてはなりません。セキュリティ上の理由から、管理サーバは WebLogic Server クラスタと同じ DMZ 内に配置する必要があります。

管理サーバは、クラスタに参加しているすべてのサーバ インスタンスのコンフィグレーション情報を保持します。管理サーバ上にある `config.xml` ファイルには、クラスタ化されているかどうかに関係なく、管理サーバのドメイン内の全サーバのコンフィグレーション データが格納されます。クラスタ内のサーバごとに個別のコンフィグレーション ファイルは作成しません。

クラスタ化された WebLogic Server インスタンスが起動するためには、管理サーバが使用可能になっている必要があリます。ただし、いったんクラスタが起動したら、管理サーバに障害が発生しても実行中のクラスタの動作には影響しません。

管理サーバをクラスタに参加させないでください。管理サーバがサーバの管理 (コンフィグレーション データの保持、サーバの起動とシャットダウン、およびアプリケーションのデプロイとアンデプロイ) プロセスだけを受け持つような構成を採ることをお勧めします。管理サーバにクライアントからのリクエストも処理させると、管理タスクの実行に遅れが生じるリスクが発生します。

管理サーバをクラスタ化する利点はありません。管理オブジェクトをクラスタ化することはできません。また、管理サーバで障害が発生した場合に、他のクラスタ メンバーにフェイルオーバもされません。管理サーバにアプリケーションをデプロイすると、サーバおよびサーバが提供している管理機能の安定性が損なわれる可能性があります。管理サーバにデプロイしたアプリケーションが予期しない動作を見せた場合、管理サーバの動作に影響する恐れもあります。

以上の理由から、管理サーバの IP アドレスがクラスタワイドの DNS 名に含まれていないことを確認してください。

ファイアウォールについての考慮事項

1 つまたは複数のファイアウォールを利用するクラスタ アーキテクチャでは、すべての WebLogic Server インスタンスを IP アドレスではなく、外部に公開されている DNS 名を使用して識別することが重要です。DNS 名を使用することで、信頼性のないクライアントに対して内部 IP アドレスをマスクする場合に使用されるアドレス変換ポリシーに関連する問題を回避できます。

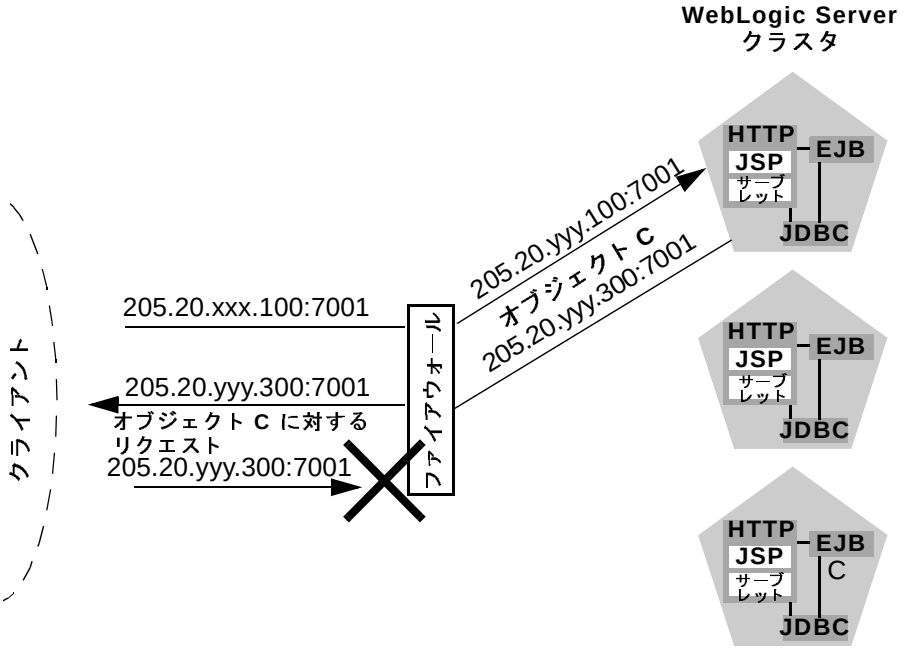
WebLogic Server インスタンスの内部 DNS 名と外部 DNS 名が同じでない場合、サーバ インスタンスの `ExternalDNSName` 属性を使用して、サーバの外部 DNS 名を定義します。ファイアウォールの外で、`externalDNSName` はサーバの外部 IP アドレスに変換されます。Administration Console の [サーバ | コンフィグレーション | 一般] タブで、この属性を設定します。Administration Console オンライン ヘルプの「[サーバ] --> [コンフィグレーション] --> [一般]」を参照してください。

注意： ファイアウォールがネットワーク アドレス変換を行うコンフィグレーションでは、クライアントが t3 およびデフォルト チャネルを使用して WebLogic Server にアクセスする場合を除いて、`ExternalDNSName` を使用する必要があります。たとえば、ファイアウォールがネットワーク アドレス変換を行い、クライアントがプロキシ プラグイン経由で HTTP を使用して WebLogic Server にアクセスするコンフィグレーションでは、`ExternalDNSName` を使用する必要があります。

`ExternalDNSName` には IP アドレスを使用しないでください。ExternalDNSName は実際のドメイン名でなければなりません。

次の図に、WebLogic Server インスタンスの識別に IP アドレスを使用する場合に発生する可能性のある問題を示します。この図では、ファイアウォールは、サブネット「xxx」の外部 IP リクエストを、サブネット「yyy」の内部 IP アドレスに変換しています。

図 6-11　サーバが IP アドレスによって識別されるときに変換エラーが発生する可能性がある



以下の手順では、接続プロセスと、考えられる障害ポイントについて説明します。

1. クライアントは、205.20.xxx.100:7001 にある最初のサーバへの接続を要求して WebLogic Server クラスタへのアクセスを開始します。ファイアウォールは、このアドレスを 205.20.yyy.100:7001 という IP アドレスに変換し、クライアントをそのアドレスに接続します。
2. クライアントは、クラスタ内の 3 番目の WebLogic Server インスタンスにある、ピン固定オブジェクト C の JNDI ルックアップを実行します。オブジェクト C のスタブには、そのオブジェクトのホストになっているサーバの内部 IP アドレス 205.20.yyy.300:7001 が含まれています。

3. オブジェクト C をインスタンス化しようとする場合、クライアントは IP アドレス 205.20.yyy.300:7001 を使用してオブジェクト C のホストになっているサーバへの接続を要求します。ファイアウォールはこの接続を拒否します。クライアントが、外部に公開されているサーバのアドレスではなく、制限されている内部 IP アドレスを使用して要求したことが原因です。

外部 IP アドレスと内部 IP アドレスの間の変換が行われなかった場合は、上記のようなクライアント リクエストがファイアウォールで問題なく処理されます。ただし、ほとんどのセキュリティ ポリシーでは内部 IP アドレスへのアクセスは拒否されます。

プロダクション環境で使用する前にクラスタのキャパシティを評価する

クラスタのアーキテクチャは、システムのキャパシティに影響します。プロダクション環境で使用するためにアプリケーションをデプロイする前に、パフォーマンスを評価して、実際の運用におけるクライアント負荷を処理するためにサーバまたはサーバ ハードウェアを追加する必要があるかどうか、また必要であればどの場所に追加するべきかを判断してください。Mercury Interactive の LoadRunner のようなテスト ソフトウェアを使用すると、多数のクライアントによる利用時の負荷をシミュレートできます。

7　WebLogic クラスタの設定

以降の節では、WebLogic Server クラスタをコンフィグレーションするためのガイドラインと手順を示します。

始める前に

この節では、WebLogic Server クラスタを設定するための前提となる作業および情報についてまとめています。

クラスタ ライセンスを取得する

WebLogic Server インスタンスをクラスタ構成でインストールするには、有効なクラスタ ライセンスが必要です。クラスタ ライセンスの入手方法については、BEA 販売代理店にお問い合わせください。

コンフィグレーション プロセスについて

クラスタのコンフィグレーション プロセスと、コンフィグレーション タスクの実施方法の基本を理解しておくと、この節で示す情報がより有益なものとなります。

WebLogic Server が備えるコンフィグレーション機能と、それらの機能を通じて実施できるタスクの詳細については、3-1 ページの「クラスタのコンフィグレーションとアプリケーションのデプロイメント」を参照してください。

クラスタ アーキテクチャを決定する

どのクラスタ アーキテクチャが、対象アプリケーションのニーズに最も適しているかを決定します。アーキテクチャ上の重要な決定には、以下のものがあります。

- すべてのアプリケーション層を 1 つのクラスタにまとめるか、それともアプリケーションの各層を複数のクラスタ間に分離するか
- クラスタ内のサーバ インスタンス間での負荷調整の方法。次のうちどの方法を使用するか
  - WebLogic Server の基本ロード バランシング機能を使用する
  - サード パーティ製のロード バランサを実装する
  - アプリケーションの Web 層を 1 つ以上のセカンダリ HTTP サーバにデプロイし、Web 層へのリクエストをプロキシで中継する
- Web アプリケーションに対して、1 つ以上のファイアウォールを備えた非武装地帯 (DMZ) を定義するか

これらを決定するためには、6-1 ページの「クラスタ アーキテクチャ」および 4-1 ページの「クラスタでのロード バランシング」の情報が参考になります。

選択するアーキテクチャによって、クラスタの設定方法も変わります。クラスタ アーキテクチャによっては、ロード バランサ、HTTP サーバ、プロキシ プラグインなどの、他のリソースのインストールまたはコンフィグレーションが必要になる場合もあります。

ネットワーク トポロジとセキュリティ トポロジを考慮する

アプリケーションのセキュリティ要件は、適切なセキュリティ トポロジを設計するための基礎となります。多様なレベルのアプリケーション セキュリティを提供する各種の代替アーキテクチャの詳細については、6-21 ページの「クラスタ アーキテクチャのセキュリティ オプション」を参照してください。

注意： 一部のネットワーク トポロジは、マルチキャスト通信に干渉することがあります。WAN にクラスタをデプロイしている場合の注意事項については、2-3 ページの「クラスタが WAN 内の複数のサブネットにまたがる場合」を参照してください。

ファイアウォールを越えてクラスタにサーバ インスタンスをデプロイしないでください。ファイアウォールを通じてマルチキャスト トラフィックをトンネリングすることの影響については、2-3 ページの「ファイアウォールがマルチキャスト通信を遮断することがある」を参照してください。

クラスタをインストールするマシンを選択する

WebLogic Server をインストールする予定の 1 台以上のマシン (この節では「ホスト」と呼ぶ) を確定し、各マシンが必要なリソースを備えていることを確認します。システムおよびソフトウェアの前提条件の一覧については、『インストール ガイド』の「WebLogic Server のインストール準備」を参照してください。

注意： WebLogic Server バージョン 7.0 では、クラスタを非マルチホーム環境の 1 台のマシン上に設定できます。この新しい機能は、デモンストレーションまたは開発用の環境で役立ちます。

IP アドレスが動的に割り当てられるマシンに WebLogic Server をインストールしないでください。

マルチ CPU マシン上の WebLogic Server インスタンス

BEA WebLogic Server には、クラスタ内のサーバ インスタンス数に関する制限はありません。したがって、Sun Microsystems, Inc. の Sun Enterprise 10000 などの大規模マルチプロセッサ サーバを、大規模なクラスタまたは複数のクラスタのホストとすることができます。

ほとんどの場合、WebLogic Server クラスタは、2 つの CPU につき 1 つの WebLogic Server インスタンスの割合でデプロイするのが最適です。ただし、どのようなキャパシティ プランニングにも共通することですが、サーバ インスタンスの最適数および分散方法を決定する場合は、対象となる Web アプリケー

ションで実際のデプロイメントを事前にテストする必要があります。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「マルチ CPU マシンのパフォーマンスに関する考慮事項」を参照してください。

ホスト マシンのソケット リーダー実装をチェックする

ソケットの最適なパフォーマンスを実現するには、pure-Java 実装ではなく、オペレーティング システムに対応したネイティブ ソケット リーダー実装を使用するよう WebLogic Server ホスト マシンをコンフィグレーションします。ネイティブ ソケットをコンフィグレーションする、または pure-Java ソケット通信を最適化する理由とその手順については、2-5 ページの「IP ソケットを使用したピア ツー ピア通信」を参照してください。

切断された Windows マシン上でのクラスタの設定

切断された単一の Windows マシン上で 1 つの WebLogic Server クラスタを示す場合は、TCP/IP スタックをロードするよう Windows に強制する必要があります。デフォルトでは、Windows は物理的なネットワーク接続を検出しないと TCP/IP スタックをロードしません。

TCP/IP スタックをロードするよう Windows に強制するには、http://support.microsoft.com/default.aspx?scid=kb;ja;239924 の「Windows で TCP/IP のメディア検出機能を無効にする方法」にある手順に従って、Windows のメディア検出機能を無効にします。

名前とアドレスを識別する

クラスタのコンフィグレーション プロセスの間、クラスタとそのメンバーについてアドレス情報 (IP アドレスまたは DNS 名、およびポート番号) を指定します。

クラスタ内通信の概要と、クラスタ内通信を利用したロード バランシングとフェイルオーバの仕組みについては、2-1 ページの「クラスタでの WebLogic Server の通信」を参照してください。

クラスタを設定するときは、以下の要素の位置情報を指定する必要があります。

- 管理サーバ
- 管理対象サーバ
- マルチキャスト ロケーション

以降の節では、指定しなければならない情報と、リソースの識別に用いる手法に影響する要因について説明しています。

リスン アドレスの問題を回避する

クラスタをコンフィグレーションするときには、クラスタおよびそのクラスタを構成するサーバ インスタンスのアドレス情報を IP アドレスまたは DNS 名を使用して指定できます。

DNS 名と IP アドレス

DNS 名と IP アドレスのどちらを使用するかを決定するときは、クラスタの目的を考慮してください。プロダクション環境では一般に、DNS 名を使用することをお勧めします。IP アドレスを使用すると、以下の状況で変換エラーが発生するおそれがあります。

- クライアントがファイアウォールを経由してクラスタに接続する場合
- プレゼンテーション層とオブジェクト層の間にファイアウォールを設ける場合。たとえば、推奨多層クラスタの説明箇所で示しているように、サーブレット クラスタと EJB クラスタの間にファイアウォールを置く場合など

変換エラーは、個別のサーバ インスタンスのアドレスを DNS 名にバインドすることによって回避できます。サーバ インスタンスの DNS 名が、ファイアウォールの両側で必ず一致するようにしてください。また、ネットワーク上の NT システムの名前でもある DNS 名は使用しないでください。

IP アドレスの代わりに DNS 名を使用する場合の注意事項については、6-30 ページの「ファイアウォールについての考慮事項」を参照してください。

内部 DNS 名と外部 DNS 名が異なる場合

WebLogic Server インスタンスの内部 DNS 名と外部 DNS 名が同じでない場合、サーバ インスタンスの `ExternalDNSName` 属性を使用して、サーバの外部 DNS 名を定義します。ファイアウォールの外で、`externalDNSName` はサーバの外部

IP アドレスに変換されます。Administration Console の [サーバ | コンフィグレーション | 一般] タブで、この属性を設定します。Administration Console オンライン ヘルプの「[サーバ] --> [コンフィグレーション] --> [一般]」を参照してください。

注意 : クライアントがデフォルト チャネルと T3 を使用して WebLogic Server にアクセスする場合、WebLogic Server インスタンスの内部 DNS 名と外部 DNS 名が異なっていても、`ExternalDNSName` 属性を設定しないでください。

外部 DNS 名の使用

`ExternalDNSName` には IP アドレスを使用しないでください。ExternalDNSName は実際のドメイン名でなければなりません。

localhost の考慮事項

サーバ インスタンスのリスン アドレスを localhost として識別すると、非ローカルのプロセスがそのサーバ インスタンスに接続できなくなります。サーバ インスタンスのホスト マシン上のプロセスのみ、そのサーバ インスタンスに接続できます。(たとえば localhost に接続する管理スクリプトがある場合などに) サーバ インスタンスが localhost としてアクセス可能でなければならない場合で、かつリモート プロセスからもアクセス可能でなければならない場合は、リスン アドレスを空白にします。リスン アドレスを空白にすると、サーバ インスタンスはマシンのアドレスを判別してそのアドレスでリスンします。

名前を WebLogic Server リソースに割り当てる

WebLogic Server 環境でコンフィグレーション可能な各リソースの名前がユニークであることを確認します。ドメイン、サーバ、マシン、クラスタ、JDBC 接続プール、仮想ホスト、またはその他のリソースのそれぞれの名前は、ユニークでなければなりません。

管理サーバのアドレスとポート

クラスタに対して使用する管理サーバの DNS 名または IP アドレスおよびリスン ポートを識別します。

管理サーバは、そのドメイン内のすべての管理対象サーバをコンフィグレーションおよび管理するために使われる WebLogic Server インスタンスです。管理対象サーバを起動するときには、その管理サーバのホストとポートを識別します。

管理対象サーバのアドレスとリスン ポート

クラスタ用の各管理対象サーバの DNS 名と IP アドレスを識別します。

クラスタ内の各管理対象サーバについて、アドレスとリスン ポート番号の組み合わせはユニークでなければなりません。非マルチホーム環境の 1 台のマシン上でクラスタ化されるサーバ インスタンスについては、インスタンス間でアドレスが重複してもかまいませんが、使用するリスン ポートはインスタンスごとに異なっている必要があります。

クラスタのマルチキャスト アドレスとマルチキャスト ポート

クラスタのマルチキャスト通信専用のアドレスとポートを識別します。

クラスタ内のサーバ インスタンスは、マルチキャストを使用して互いに通信します。具体的には、マルチキャストを使用して各自のサービスを全体に通知し、インスタンスが継続的に使用可能であることを知らせるハートビートを一定間隔で出します。

クラスタのマルチキャスト アドレスは、クラスタ通信以外の目的には使用しないことをお勧めします。クラスタのマルチキャスト アドレスが存在するマシンが、マルチキャスト通信を使用するクラスタ外部のプログラムのホストであるか、またはそのようなプログラムによってアクセスされる場合、そのマルチキャスト通信では必ず、クラスタのマルチキャスト ポートとは異なるポートを使用してください。

マルチキャストとマルチキャスト クラスタ

ネットワーク上の複数のクラスタは、必要に応じてマルチキャスト アドレスとマルチキャスト ポートの組み合わせを共有できます。

マルチキャストと多層クラスタ

第 6 章「クラスタ アーキテクチャ」で説明しているような、クラスタ間にファイアウォールを設ける推奨多層アーキテクチャを設定している場合、専用のマルチキャスト アドレスが 2 つ必要になります。1 つはプレゼンテーション (サーブレット) クラスタ用であり、もう 1 つはオブジェクト クラスタ用です。2 つのマルチキャスト アドレスを使用することにより、ファイアウォールはクラスタの通信に干渉しなくなります。

クラスタ アドレス

クラスタをコンフィグレーションする際に、クラスタ内の管理対象サーバを識別するクラスタ アドレスを定義します。クラスタ アドレスは、URL のホスト名部分を構築するためにエンティティ Bean およびステートレス Bean で使用されます。クラスタ アドレスが設定されていない場合、EJB ハンドルは正しく動作しません。

プロダクション環境でのクラスタ アドレス

プロダクション環境では、クライアント アプリケーションは、クラスタ内の各 WebLogic Server インスタンスの IP アドレスまたは DNS 名にマップされる DNS 名としてクラスタ アドレスを指定する必要があります。クラスタ内の管理対象サーバにマップされる DNS 名としてクラスタ アドレスを定義しなかった場合、5-23 ページの「エンティティ Bean および EJB ハンドルのフェイルオーバ」で説明したように、フェイルオーバはエンティティ Bean および EJB ハンドルに関して機能しません。

クラスタ アドレスを DNS 名として定義した場合、クラスタ メンバーのリスン ポートは、クラスタ アドレスで指定されません。クラスタ内の各管理対象サーバが同じリスン ポート番号を持っていると見なされます。クラスタ内の各サーバ インスタンスは、ユニークなアドレスとリスン ポートの組み合わせを持つ必要があるので、クラスタ アドレスが DNS 名の場合、クラスタ内の各サーバ インスタンスには以下が必要です。

- ユニークなアドレス
- 共通のリスン ポート番号

クライアントがクラスタ DNS 名を提供して初期 JNDI コンテキストを取得すると、`weblogic.jndi.WLInitialContextFactory` はその DNS 名にマップされているすべてのアドレスのリストを取得します。このリストは WebLogic Server インスタンスによってキャッシュされ、新しい初期コンテキスト リクエストは、キャッシュされているリスト内のアドレスをラウンドロビン アルゴリズムで使うことによって実行されます。キャッシュされているリスト内のサーバ インスタンスが使用できない場合、そのサーバはリストから削除されます。アドレス リストは、サーバ インスタンスがキャッシュ内のどのアドレスにもアクセスできない場合にのみ、DNS サービスによって更新されます。

キャッシュされているアドレス リストを使用すると、DNS ラウンドロビンだけを利用する場合の問題を避けることができます。たとえば、DNS ラウンドロビンでは、アドレスがアクセス可能かどうかに関わりなく、ドメイン名にマップされているすべてのアドレスが使用され続けます。アドレス リストをキャッシュすると、WebLogic Server はアクセス不能なアドレスを削除できるので、初期コンテキスト リクエストで接続の失敗が繰り返されることはありません。

注意： 管理サーバをクラスタに参加させないでください。管理サーバの IP アドレスがクラスタワイドの DNS 名に含まれていないことを確認してください。詳細については、6-29 ページの「管理サーバについての考慮事項」を参照してください。

開発およびテスト環境でのクラスタ アドレス

クラスタ アドレスに対してクラスタ DNS 名を使用することは、前節で推奨したプロダクション環境だけではなく、開発環境とテスト環境でも役立ちます。

代わりに、次の例で示すように、DNS 名 (または IP アドレス)、およびクラスタ内の各管理対象サーバのリスン ポートを格納するリストとしてクラスタ アドレスを定義することもできます。

```
DNSName1:port1,DNSName1:port2,DNSName1:port3
```

```
IPaddress1:port1,IPaddress2:port2;IPaddress3:port3
```

各クラスタ メンバーは、ユニークなアドレスとポートの組み合わせを持っています。

単一のマルチホーム マシンでのクラスタ アドレス

マルチホーム環境の 1 台のマシン上でクラスタが動作しており、クラスタ内の各サーバ インスタンスで異なった IP アドレスを使用する場合、クラスタ内のサーバ インスタンスの IP アドレスにマップする DNS 名を使用してクラスタ アドレスを定義します。クラスタ アドレスを DNS 名として定義する場合、クラスタ内の各管理対象サーバで共通のリスン ポート番号を指定します。

クラスタ実装の手順

この節では、アプリケーションをクラスタ構成にして実行する手順を、WebLogic Server のインストールからアプリケーション コンポーネントの初期デプロイメントまで順を追って説明します。

コンフィグレーションのロードマップ

ここでは、クラスタ実装における典型的な作業の流れと、コンフィグレーションの選択に影響する重要な考慮事項を示します。実際のプロセスは、環境ごとにユニークな特性およびアプリケーションの性質によって決まります。この節では、以下の作業について説明します。

1. 7-11 ページの「WebLogic Server をインストールする」
2. 7-12 ページの「クラスタ化されたドメインを作成する」
3. 7-17 ページの「ノード マネージャをコンフィグレーションする」
4. 7-18 ページの「EJB と RMI のロード バランシング方式をコンフィグレーションする」
5. 7-19 ページの「パッシブなクッキーの永続性をサポートするロード バランサをコンフィグレーションする」
6. 7-20 ページの「プロキシ プラグインをコンフィグレーションする」
7. 7-24 ページの「レプリケーション グループをコンフィグレーションする」
8. 7-25 ページの「固定サービスの移行可能対象をコンフィグレーションする」

9. 7-26 ページの「クラスタ化された JDBC をコンフィグレーションする」
10. 7-29 ページの「デプロイメント用にアプリケーションをパッケージ化する」
11. 7-29 ページの「アプリケーションをデプロイする」
12. 7-33 ページの「移行可能サービスをデプロイ、活性化、および移行する」
13. 7-36 ページの「インメモリ HTTP レプリケーションをコンフィグレーションする」
14. 7-37 ページの「コンフィグレーションに関するその他のトピック」

クラスタの実装によっては、一部の手順が不要である場合があります。また、ここで示す以外の手順が必要になる場合もあります。

WebLogic Server をインストールする

まだインストールしていない場合、WebLogic Server をインストールします。手順については、『インストール ガイド』を参照してください。

- クラスタを 1 台のマシン上で実行する場合、/bea ディレクトリの下に、クラスタ内のすべてのインスタンスに対して使用する単体の WebLogic Server をインストールします。
- ネットワーク上のリモート マシンについては、各マシンに同じバージョンの WebLogic Server をインストールします。各マシンは次の要件を満たしている必要があります。
  - 永続的に割り当てられる静的な IP アドレスを持つこと。クラスタ化環境では、動的に割り当てられる IP アドレスは使用できません。
  - クライアントからアクセス可能であること。サーバ インスタンスとクライアントの間にファイアウォールがある場合、各サーバ インスタンスには、クライアントから到達できるパブリックな静的 IP アドレスを割り当てる必要があります。
  - すべてのマシンが同じローカル エリア ネットワーク (LAN) 上にあり、IP マルチキャストによって通信できること。

注意： 共有ファイルシステムと 1 つのインストールを使用して、異なるマシン上で複数の WebLogic Server インスタンスを実行しないでください。共有ファイルシステムを使用すると、クラスタにシングル ポイントの競合が発生します。すべてのサーバ インスタンスが、ファイルシステムにアクセスする (また場合によっては、個別のログ ファイルへの書き込みを行う) ために競合しなければならなくなります。さらに、共有ファイルシステムに障害が発生した場合には、クラスタ化されたサーバ インスタンスを起動できなくなることもあります。

クラスタ化されたドメインを作成する

ここでは、BEA ドメイン コンフィグレーション ウィザードを使用してクラスタを作成する手順を示します。

注意： クラスタのコンフィグレーションを作成および管理する方法は他にもあります。3-11 ページの「クラスタをコンフィグレーションする方法」を参照してください。

クラスタでカスタムのネットワーク チャネルを使用する場合の手順については、『WebLogic Server ドメイン管理』の「クラスタにおけるネットワーク チャネルのコンフィグレーション」を参照してください。その節で示されている手順に従って管理対象サーバを作成し、クラスタを作成し、チャネルを作成して割り当て、マルチキャスト アドレスを設定します。

1. [スタート | プログラム | BEA | WebLogic Platform | Configuration Wizard] を選択して、コンフィグレーション ウィザードを起動します。
2. [ドメインのタイプと名前を選択します] ウィンドウで、[WLS Domain] テンプレートをクリックします。
3. [名前] フィールドに英数字でドメイン名を入力します。スペースは使用できません。[次へ] をクリックして次に進みます。
4. [サーバ タイプを選択します] ウィンドウで、[Admin Server with Clustered Managed Server(s)] をクリックします。[次へ] をクリックして次に進みます。

5. [ドメインの場所を選択します] ウィンドウで [次へ] をクリックして、表示されている中から、または別のディレクトリからデフォルト ドメインを選択します。

   [ドメインの場所を選択します] ウィンドウに次の要素が表示されます。

   - ドメイン ディレクトリが作成されるディレクトリ
   - 作成されるドメイン ディレクトリの絶対パス

     ドメイン ディレクトリの名前は、[ドメインのタイプと名前を選択します] ウィンドウで指定したドメイン名と同じになります。

   ドメイン ディレクトリが *BEA_HOME*\user_projects\ ディレクトリの下にあることを確認してください。*BEA_HOME* は、WebLogic Platform がインストールされているディレクトリです。

6. [クラスタ化サーバをコンフィグレーションします] ウィンドウが表示されるので、[追加] をクリックして、クラスタの最初の管理対象サーバをコンフィグレーションします。

7. [サーバを追加] ウィンドウで、次の項目を設定します。

   - [サーバ名] - 英数字を使用してサーバ名を入力します。スペースは使用できません。

     作成した各サーバ インスタンスにユニークな名前を割り当てます。WebLogic Server 環境の他の管理対象サーバまたは管理サーバと同じ名前は割り当てないでください。

   - [サーバ リスン アドレス] - マシンのアドレスまたは名前を入力します。

     リスン アドレスを指定する方法については、7-5 ページの「リスン アドレスの問題を回避する」を参照してください。

   - [サーバ リスン ポート] - 数値を入力します。指定できる値の範囲は 1 ～ 65535 です。

   [追加] をクリックして次に進みます。

8. [サーバを追加] ウィンドウで、[追加] をクリックしてクラスタに管理対象サーバを追加し、クラスタ内の個々の管理対象サーバについて前の手順を繰り返します。管理対象サーバの追加が終了したら、[次へ] をクリックして次に進みます。

9. [クラスタをコンフィグレーションします] ウィンドウで、次の項目を設定します。

- [クラスタ名] - デフォルト値の「mycluster」が入力されています。英数字を使用してクラスタの名前を入力します。スペースは使用できません。
- [クラスタ マルチキャスト アドレス] - デフォルト値の「237.0.0.1」が入力されています。指定できる値の範囲は 224.0.0.0 ～ 239.255.255.255 です。
- [クラスタ マルチキャスト ポート] - デフォルト値の「777」が入力されています。
- [クラスタ アドレス] - クラスタ内の各サーバ インスタンスのアドレスとポートの組み合わせで構成されるクラスタ アドレスが入力されています。

  クラスタ アドレスを変更する場合は、適切なアドレス形式を使用します。この形式は、クラスタをプロダクション環境で使用するかそうでないかによって異なります。プロダクション環境とそれ以外の環境でのクラスタ アドレスの形式についての詳細は、7-8 ページの「クラスタ アドレス」を参照してください。

[次へ] をクリックして次に進みます。

10. [Configure Admin Server (with Cluster)] ウィンドウで、次の項目を設定します。
    - [サーバ名] - デフォルト値の「myserver」が入力されています。英数字を使用してクラスタの管理サーバの名前を入力します。スペースは使用できません。

      作成した各サーバ インスタンスにユニークな名前を割り当てます。WebLogic Server 環境の他の管理対象サーバまたは管理サーバと同じ名前は割り当てないでください。
    - [サーバ リスン アドレス] - マシンのアドレスまたは名前を入力します。

      リスン アドレスを指定する方法については、7-5 ページの「リスン アドレスの問題を回避する」を参照してください。
    - [サーバ リスン ポート] - デフォルトのポート番号は 7001 です。指定できる値の範囲は 1 ～ 65535 です。
    - [サーバ SSL リスン ポート] - デフォルトのポート番号は 7002 です 指定できる値の範囲は 1 ～ 65535 です。

    [次へ] をクリックして次に進みます。

11. [システム ユーザ名およびパスワードを作成します] ウィンドウで、[ユーザ名] および [パスワード] の各フィールドを設定します。パスワードは 8 文字以上で入力します。

    ユーザ名とパスワードは、クラスタ内のサーバ インスタンスを起動するために必要です。ユーザ名は、サーバ インスタンスを起動する権限のあるロールに属していなければなりません。ロールの詳細については、『管理者ガイド』の「システム管理操作の保護」を参照してください。

    [次へ] をクリックして次に進みます。

12. [サーバを Windows サービスとしてインストールします] ウィンドウで、次の操作を行います。

    [はい] をクリックして、ドメインを Windows のサービスとしてインストールします。これにより、Windows システムが起動するたびに WebLogic Server サービスが自動的に起動するようになります。

    または

    WebLogic を Windows のサービスとして実行しない場合は、[いいえ] をクリックします。この場合、Windows システムの起動時に WebLogic Server サービスは自動的には起動しません。

    注意 : `beaSvc` は、`domainname_ServerName` 変数内のサービス名です。

13. [ドメインを Windows の [スタート] メニューに追加します] ウィンドウで、次の操作を行います。

    [はい] をクリックして、新しいドメインを起動するための項目を Windows の [スタート] メニューに追加します。

    または

    Windows のスタート メニューにドメインの項目を追加しない場合は、[いいえ] をクリックします。

    [次へ] をクリックして次に進みます。

14. [コンフィグレーションの概要] で、コンフィグレーションの要約情報を確認します。以下のどちらかをクリックします。

    前のウィンドウに戻ってコンフィグレーション情報を修正する場合は [Previous]。

    または

    ドメインを作成する場合は [作成]。

15. [コンフィグレーション ウィザード完了] ウィンドウで、[コンフィグレーションウィザードを終了します] をクリックしてウィザードを終了します。

コンフィグレーションの以前のステップでの設定を変更し、コンフィグレーション プロセスを完了するには、Administration Console を使用します。Administration Console 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。

注意 : クラスタ コンフィグレーションに変更を加える場合は、クラスタを含むドメインの管理サーバを実行する必要があります。管理サーバを起動するには、次の節の手順 1 ～ 6 に従います。

WebLogic Server クラスタを起動する

この節では、クラスタを起動するための手順を示します。まずクラスタの管理サーバを起動し、次に、クラスタ内の個々の管理対象サーバを起動します。個々のサーバ インスタンスは、個別のコマンド シェルで実行するコマンドによって起動されます。

サーバ インスタンスの起動と停止については、『管理者ガイド』の「WebLogic Server の起動と停止」を参照してください。

1. コマンド シェルを開きます。
2. コンフィグレーション ウィザードで作成したドメイン ディレクトリに移動します。
3. 次のコマンドを入力して管理サーバを起動します。

   ```
   StartWebLogic
   ```

4. 「Enter username to boot WebLogic Server」というプロンプトが表示されたら、ドメインのユーザ名を入力します。
5. 「Enter password to boot WebLogic Server」というプロンプトが表示されたら、ドメインのパスワードを入力します。

   起動プロセスの状態を知らせるメッセージがコマンド シェルに表示されます。
6. 管理対象サーバを起動するために、別のコマンド シェルを開きます。

7. コンフィグレーション ウィザードで作成したドメイン ディレクトリに移動します。
8. 次のコマンドを入力します。

   `StartManagedWebLogic` *`server_name address`*`:`*`port`*

   各値の説明は次のとおりです。

   server_name : 起動する管理対象サーバの名前

   address : ドメインの管理サーバの IP アドレスまたは DNS 名

   port : ドメインの管理サーバのリスン ポート
9. 「Enter username to boot WebLogic Server」というプロンプトが表示されたら、ドメインのユーザ名を入力します。
10. 「Enter password to boot WebLogic Server」というプロンプトが表示されたら、ドメインのパスワードを入力します。

    起動プロセスの状態を知らせるメッセージがコマンド シェルに表示されます。

    注意 : 管理対象サーバを起動すると、クラスタ内の他の実行中サーバ インスタンスからハートビートがリスンされます。管理対象サーバは、2-15 ページの「WebLogic Server による JNDI ツリー更新のしくみ」で説明しているように、クラスタワイドの JNDI ツリーのローカル コピーを構築し、クラスタ内の実行中の管理対象サーバとの同期が取れたら、ステータス メッセージを表示します。同期化プロセスには、1 分ほどかかります。
11. クラスタ内の別のサーバ インスタンスを起動するには、手順 6. から手順 10. までを繰り返します。
12. クラスタ内のすべての管理対象サーバを起動したら、クラスタの起動プロセスは完了です。

ノード マネージャをコンフィグレーションする

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、管理サーバとは異なるマシン上にある管理対象サーバの起動に便利です。またノード マネージャには、クラスタ内にある管理対象サーバの可用性の

向上に役立つ機能もあります。ノード マネージャの詳細と、コンフィグレーション方法および使用方法については、『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。

EJB と RMI のロード バランシング方式をコンフィグレーションする

EJB と RMI オブジェクトに対して、重みベースまたはランダム方式のロード バランシングを使用する場合は、この節の手順に従います。

その他の方式を明示的に指定しない場合、WebLogic Server はクラスタ化されるオブジェクトのスタブに対して、デフォルトのロード バランシング方式であるラウンドロビン アルゴリズムを使用します。それ以外のロード バランシング方式については、4-5 ページの「EJB と RMI オブジェクトのロード バランシング」を参照してください。デフォルトのロード バランシング アルゴリズムを変更するには、次の手順に従います。

1. WebLogic Server Administration Console を起動します。
2. [クラスタ] ノードを選択します。
3. クラスタを選択します。
4. [デフォルトのロード バランス アルゴリズム] の横のドロップダウン矢印をクリックして、ロード バランシング アルゴリズムの選択肢を表示します。
5. 選択肢のリストから、使用するロード バランシング アルゴリズムを選択します。
6. [サービス期間しきい値] フィールドに設定値を入力します。この属性の詳細については、Administration Console オンライン ヘルプを参照してください。
7. [適用] をクリックして変更を保存します。

パッシブなクッキーの永続性をサポートするロード バランサをコンフィグレーションする

パッシブなクッキーの永続性をサポートするロード バランサは、セッションをホストする WebLogic Server インスタンスをクライアントと関連付けるために WebLogic Server セッション クッキーの情報を使用できます。セッション クッキーには、ロード バランサがセッションのプライマリ サーバ インスタンスを識別する文字列が含まれています。

外部ロード バランサ、セッション クッキーの永続性、および WebLogic Server のセッション クッキーの詳細については、4-3 ページの「外部ロード バランサによる HTTP セッションのロード バランシング」を参照してください。

クラスタで動作するようにロード バランサをコンフィグレーションするには、ロード バランサの機能を使用して、文字列定数のオフセットと長さを定義します。

セッション クッキーのセッション ID 部分が、デフォルト長の 52 バイトに設定されている場合、次のように設定します。

- 文字列のオフセットを、デフォルトのランダム セッション ID 長に、区切り文字用の 1 バイトを加えた 53 バイトに設定する。
- 文字列長を 10 バイトに設定する。

アプリケーションまたは環境によって、ランダム セッション ID の長さをデフォルト値の 52 バイト以外に変更しなければならない場合は、それに応じて文字列のオフセットもロード バランサで設定します。文字列のオフセットは、セッション ID の長さに、区切り文字用の 1 バイトを加えたものと同じ値にする必要があります。

注意： WebLogic Server で Big-IP ロード バランサをコンフィグレーションする際のベンダ固有の手順については、B-1 ページの「クラスタに関する BIG-IP™ ハードウェアのコンフィグレーション」を参照してください。

プロキシ プラグインをコンフィグレーションする

プロキシ プラグインを使用してサーブレットおよび JSP のロード バランシングを行う場合は、この節で示す手順を参考にしてください。プロキシ プラグインは、リクエストを Web サーバからクラスタ内の WebLogic サーバ インスタンスに中継し、プロキシを経由する HTTP リクエストのロード バランシングとフェイルオーバを可能にします。

プロキシ プラグインによるロード バランシングの詳細については、4-2 ページの「プロキシ プラグインによるロード バランシング」を参照してください。プロキシ プラグインによる接続とフェイルオーバの詳細については、5-3 ページの「サーブレットと JSP のレプリケーションとフェイルオーバ」および 5-9 ページの「クラスタ化されたサーブレットと JSP へのプロキシ経由のアクセス」を参照してください。

- BEA が提供する Web サーバを使用する場合、7-21 ページの「HttpClusterServlet を設定する」の指示に従って、関連プラグインの `HttpClusterServlet` を設定する。
- サポートされているサード パーティ製 Web サーバを使用する場合、その製品に固有のプラグインを設定する。サポートされているサーバ製品の一覧については、4-2 ページの「プロキシ プラグインによるロード バランシング」を参照してください。

  サード パーティ製の Web サーバ用のプラグインを設定するには、『WebLogic Server における Web サーバ プラグインの使い方』の手順に従ってください。

注意： リクエストをクラスタにプロキシ経由で中継する各 Web サーバでは、プラグインが同じようにコンフィグレーションされている必要があります。

HttpClusterServlet を設定する

この節では、HttpClusterServlet のデプロイ手順を示します。

1. HttpClusterServlet を含む Web アプリケーション、およびこの節で説明する要素を含むデプロイメント記述子ファイルを作成します。Web アプリケーションおよびデプロイメント記述子を手動で作成することも、WebLogic Builder ツールで作成することもできます。その背景と手順については、以下を参照してください。
   - 『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』の「Web アプリケーションの概要」
   - 『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』の「Web アプリケーション作成の主な手順」
   - 『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』の「Web アプリケーションのデプロイメント記述子の記述」
   - WebLogic Builder Online Help
2. Web アプリケーションのデプロイメント記述子ファイル web.xml を作成します。完全なデプロイメント記述子の例については、7-23 ページの「HttpClusterServlet 用デプロイメント記述子のサンプル」を参照してください。
   a. <servlet> 要素に、Web アプリケーション デプロイメント記述子の HttpClusterServlet を登録します。HttpClusterServlet のクラス名は weblogic.servlet.proxy.HttpClusterServlet です。

      ```
      <servlet-name>HttpClusterServlet</servlet-name>
      <servlet-class>
         weblogic.servlet.proxy.HttpClusterServlet
      </servlet-class>
      ```

   b. 『WebLogic Server における Web サーバ プラグインの使い方』の「Web サーバ プラグインのパラメータ」で説明したように、必要に応じて追加パラメータを定義します。7-23 ページの「クラスタ コンフィグレーションとプロキシ プラグイン」を参照してください。
   c. プロキシ サーブレットを <url-pattern> にマップします。特に、プロキシを通すファイルの拡張子 (*.jsp、*.html など) をマップします。

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

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

次に例を示します。

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

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

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

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

3. Administration Console で、この Web アプリケーションをコンフィグレーションします。Administration Console による Web アプリケーションのコンフィグレーションとデプロイメントについては、Administration Console オンライン ヘルプの「新しい Web アプリケーションまたは Web サービスのコンフィグレーション」を参照してください。
   a. ドメインに新しいサーバ インスタンスを作成します。
   b. デフォルト Web アプリケーションとして作成した Web アプリケーションを、作成したばかりのサーバ インスタンスに割り当てます。
   c. Web アプリケーションをサーバにデプロイします。

   注意： weblogic.Deployer ツールを使用して Web アプリケーションをデプロイすることもできます。詳細については、『管理者ガイド』の「Deployer」を参照してください。

クラスタ コンフィグレーションとプロキシ プラグイン

HttpClusterServlet の動作を制御するために、WebLogic Server の 2 つのコンフィグレーション属性をクラスタ レベルで設定できます。

- WeblogicPluginEnabled—HttpClusterServlet からリクエストを受け取るクラスタでこの属性を true に設定すると、サーブレットは Web サーバのアドレスを返す代わりに、専用の WL-Proxy-Client-IP ヘッダにあるブラウザ クライアントのアドレスを使用して、getRemoteAddr 呼び出しへ応答します。
- ClientCertProxy Enabled—HttpClusterServlet からリクエストを受け取るクラスタでこの属性を true に設定すると、プラグインはクライアント証明書を特別な WL-Proxy-Client-Cert ヘッダに格納して送信します。そのため、ユーザ認証はプロキシ サーバ上で行われます。

  注意： 詳細については、Administration Console オンライン ヘルプの「[クラスタ]-->[コンフィグレーション]-->[一般]」を参照してください。

HttpClusterServlet 用デプロイメント記述子のサンプル

次に示すのは、HttpClusterServlet を使用するための、Web アプリケーション デプロイメント記述子 web.xml の設定例です。

コード リスト 7-1 HttpClusterServlet を使用するときの web.xml の設定例

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.
 //DTD Web Application 2.2//EN"
 "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>

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

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

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

</servlet>

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

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

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

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

</web-app>
```

レプリケーション グループをコンフィグレーションする

WebLogic Server では、HTTP セッション ステートをメモリ内にレプリケートすることによって、サーブレットおよび JSP の自動フェイルオーバを行います。それ以外にも、レプリケーション グループを使用することにより、セカンダリ ステートが置かれる場所を独自に制御することができます。レプリケーション グループは、セッション ステートのレプリカの格納先として使用する、クラスタ内のサーバ インスタンスの優先順を定義するリストです。

クラスタがサーブレットまたはステートフル セッション EJB のホストになる場合は、WebLogic Server インスタンスのレプリケーション グループを作成して、セッション ステートのレプリカのホストにすることができます。

各レプリケーション グループに参加させるサーバ インスタンスと、各サーバ インスタンスの優先レプリケーション グループを決定する手順については、5-6 ページの「レプリケーション グループを使用する」を参照してください。

次に、個々の WebLogic Server インスタンスについて、次の手順に従ってレプリケーション グループをコンフィグレーションします。

WebLogic Server インスタンスのレプリケーション グループをコンフィグレーションするには、次の手順を実行します。

1. WebLogic Server Console を起動します。
2. [サーバ] ノードを選択します。
3. コンフィグレーションするサーバを選択します。
4. [クラスタ] タブを選択します。
5. 以下の属性フィールドに値を入力します。
   - [レプリケーション グループ]: このサーバ インスタンスが属するレプリケーション グループ名を入力します。
   - [セカンダリ プリファレンス グループ]: このサーバ インスタンスをレプリケートされた HTTP セッション ステートのホストにする場合に使用するレプリケーション グループ名を入力します。
6. 変更を適用します。

固定サービスの移行可能対象をコンフィグレーションする

WebLogic Server では、オプションの移行可能対象をコンフィグレーションできます。移行可能対象には、JMS サーバや Java Transaction API (JTA) トランザクション回復サービスなどの移行可能サービスのホストとなることのできる、クラスタ内のサーバ インスタンスのリストを定義します。移行可能対象を使用する場合、クラスタ内でサービスをデプロイまたは活性化する前に、対象サーバ リストをコンフィグレーションします。

クラスタ内の移行可能対象をコンフィグレーションしない場合、移行可能サービスは、クラスタ内のどの WebLogic Server インスタンスにも移行できます。詳細については、5-24 ページの「固定サービスの移行」を参照してください。

JTA または JMS に対して移行可能対象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. 対象のクラスタの管理サーバを起動し、Administration Console にログインします。
2. 左ペインで [サーバ] ノードを選択し、コンフィグレーションするクラスタのメンバーであるサーバ インスタンスを選択します。
3. 右ペインで [制御 | 移行コンフィグレーション] タブを選択して、JMS サービスの移行可能対象をコンフィグレーションします。または、右ペインで [制御 | JTA 移行コンフィグレーション] タブを選択して、JTA トランザクション回復サービスの移行可能対象をコンフィグレーションします。
4. [選択可] カラムで 1 つまたは複数のサービス名を選択し、矢印ボタンを使用して [選択済み] カラムにサービスを移動します。[選択済み] カラムのサービス名は、移行可能サービスのホストとなることができるサーバ インスタンスの一覧を表します。
5. [適用] をクリックして、移行可能対象に対する変更を適用します。

クラスタ化された JDBC をコンフィグレーションする

この節では、Administration Console による JDBC コンポーネントのコンフィグレーション手順を示します。JDBC コンポーネントのコンフィグレーション時に選択した内容は、クラスタを含む WebLogic Server ドメインの `config.xml` ファイルに反映されます。

接続プール、および必要に応じて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しているときに、データ ソース属性の 1 つとして接続プールまたはマルチプールを指定します。これによって、そのデータ ソースを特定の接続プールまたはマルチプールに関連付けます。

- WebLogic Server クラスタ内での JDBC オブジェクトの動作については、1-8 ページの「JDBC 接続」を参照してください。

- クラスタ化された JDBC がアプリケーションの可用性をどのように向上させるかについては、5-27 ページの「フェイルオーバと JDBC 接続」を参照してください。
- クラスタ化された JDBC がロード バランシングをどのようにサポートするかについては、4-14 ページの「JDBC 接続のロード バランシング」を参照してください。

接続プールをクラスタ化する

次の手順を実行し、クラスタ内の基本接続プールを設定します。

1. 接続プールを作成します。

   手順については、Administration Console オンライン ヘルプの「1 つまたは複数のサーバまたはクラスタへの JDBC 接続プールの割り当て」を参照してください。

2. 接続プールをクラスタに割り当てます。

   手順については、「1 つまたは複数のサーバまたはクラスタへの JDBC 接続プールの割り当て」を参照してください。

3. データ ソースを作成します。[プール名] 属性に、前の手順で作成した接続プールを指定します。

   手順については、Administration Console オンライン ヘルプの「JDBC データ ソースの作成とコンフィグレーション」を参照してください。

4. データ ソースをクラスタに割り当てます。

   手順については、「サーバまたはクラスタへの JDBC データ ソースの割り当て」を参照してください。

マルチプールをクラスタ化する

可用性を向上させ、必要に応じてロード バランシングを提供するために、次の手順を実行してクラスタ化されたマルチプールを作成します。

注意： 通常、マルチプールは、レプリケートされて同期を取られているデータベース インスタンスに対する接続の可用性を向上させ、ロード バランシングを提供するために使用します。詳細については、1-8 ページの「JDBC 接続」を参照してください。

1. 複数の接続プールを作成します。

   手順については、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。

2. 各接続プールをクラスタに割り当てます。

   手順については、「1 つまたは複数のサーバまたはクラスタへの JDBC 接続プールの割り当て」を参照してください。

3. マルチプールを作成します。前の手順で作成した接続プールをマルチプールに割り当てます。

   手順については、「JDBC マルチ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。

4. マルチプールをクラスタに割り当てます。

   手順については、「1 つまたは複数のサーバへの JDBC マルチプールの割り当て」を参照してください。

5. データ ソースを作成します。[プール名] 属性に、前の手順で作成したマルチプールを指定します。

   手順については、「JDBC データ ソースの作成とコンフィグレーション」を参照してください。

6. データ ソースをクラスタに割り当てます。

   手順については、「サーバまたはクラスタへの JDBC データ ソースの割り当て」を参照してください。

デプロイメント用にアプリケーションをパッケージ化する

『WebLogic Server アプリケーションの開発』の「WebLogic Server アプリケーションのパッケージ化」の指示に従って、デプロイメント用にアプリケーションを準備します。アプリケーションのパッケージ化は、デプロイメントのための前提条件です。

アプリケーションをデプロイする

この節では、一般的なデプロイメント作業の手順を示します。クラスタ環境でのアプリケーションのデプロイメントについては、3-6 ページの「アプリケーションのデプロイメントについて」を参照してください。デプロイメントに関する全般的なトピックについては、『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。

アプリケーションをクラスタにデプロイする

WebLogic Server Administration Console を使用してアプリケーションをコンフィグレーションおよびデプロイするには、この節の手順に従います。

注意： Administration Console を使用してアプリケーションをクラスタにデプロイするときは、クラスタ内のすべてのサーバ インスタンスを動作させておくことをお勧めします。

1. WebLogic Server 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. Administration Console の左ペインで、[デプロイメント] をクリックします。
4. Administration Console の左ペインで [アプリケーション] をクリックします。Administration Console の右ペインに、すべてのデプロイメント済みアプリケーションを示すテーブルが表示されます。
5. [新しい Application のコンフィグレーション] オプションを選択します。

6. WebLogic Server で使用するためにコンフィグレーションする .ear、.war、.jar、または .rar ファイルを見つけます。「分解された」アプリケーションまたはコンポーネント ディレクトリをコンフィグレーションすることもできます。WebLogic Sever は、指定したディレクトリおよびその下位ディレクトリで見つかった全コンポーネントをデプロイします。
7. 名前の左側にある WebLogic アイコンをクリックしてディレクトリまたはファイルを選択し、次の手順に進みます。
8. 表示されたフィールドにアプリケーションまたはコンポーネントの名前を入力して、[作成] をクリックします。
9. 以下の情報を入力します。
   - [ステージング モード] - ステージング モードを指定します。server、nostage、および stage の各オプションから選択します。
   - [デプロイ] - チェック ボックスを使用して、.ear、.war、.jar、または .rar ファイルを作成と同時にデプロイするかどうかを指定します。
10. アプリケーションのコンポーネントをコンフィグレーションするために、[アプリケーションのコンフィグレーション] をクリックします。
11. [コンポーネント] テーブルが表示されます。コンフィグレーションするコンポーネントをクリックします。
12. 使用できるタブで、以下の情報を入力します。
    - [コンフィグレーション] ー ステージング モードを編集し、デプロイメント順を入力します。
    - [対象] - アプリケーションを [選択可] リストから [選択済み] リストに移動することによって、このコンフィグレーション対象のアプリケーションの対象を指定します。

注意： WebLogic Server でクラスタ化するオブジェクトは、均一にデプロイすることをお勧めします。オブジェクトにレプリカ対応スタブが含まれる場合は、Administration Console でクラスタ名を使用してオブジェクトをデプロイします。

均一なデプロイメントが確実に行われるようにするには、対象を選択するときに、クラスタ内の個別の WebLogic Server インスタンスではなくクラスタ名を使用します。

Administration Console では、クラスタへのレプリカ対応オブジェクトのデプロイメントが自動化されます。アプリケーションまたはオブジェクトをクラスタにデプロイする場合、Administration Console では、アプリケーションまたはオブジェクトがクラスタの全メンバーに自動的にデプロイされます。メンバーは、管理サーバ マシンのローカルにあっても、リモート マシン上にあってもかまいません。

- [デプロイ] - 選択したすべての対象にアプリケーションをデプロイするか、またはすべての対象からアプリケーションをアンデプロイします。
- [モニタ] - アプリケーションに関連するモニタ情報を表示します。
- [メモ] - アプリケーションについての注記事項を入力します。

[適用] をクリックします。

サーバ インスタンスにデプロイする (固定デプロイメント)

クラスタのすべてのメンバーでなく、1 つのサーバ インスタンスにアプリケーションをデプロイする形態を固定デプロイメントと呼びます。固定デプロイメントでは特定のサーバ インスタンスが対象となりますが、デプロイメント プロセスの間は、クラスタ内のすべてのサーバ インスタンスが動作していなければなりません。

固定デプロイメントは、Administration Console から実行することも、`weblogic.Deployer` を使用してコマンドラインから実行することもできます。

コマンドラインからの固定デプロイメント

コマンド シェルから、次の構文を使用して対象のサーバ インスタンスを指定します。

```
java weblogic.Deployer -activate -name ArchivedEarJar -source
C:/MyApps/JarEar.ear -target server1
```

Administration Console を使用しての固定デプロイメント

対象サーバ インスタンスに対して次の作業を行います。

1. Administration Console で [デプロイメント] ノードを開きます。
2. デプロイするアプリケーションまたはコンポーネントをクリックします。

3. 右ペインで [対象 | クラスタ] タブを選択し、クラスタが [選択済み] リストではなく [選択可] リストにあることを確認します。
4. [対象] タブを選択し、サーバが [選択済み] リストにあることを確認します。
5. [適用] をクリックします。

クラスタのデプロイメントをキャンセルする

Administration Console を使用するか、または `weblogic.Deployer` を使用してコマンドラインからデプロイメントをキャンセルすることができます。

コマンドラインからデプロイメントをキャンセルする

コマンド シェルから、次の構文を使用してデプロイメント タスク ID をキャンセルします。

```
java weblogic.Deployer -adminurl http://admin:7001 -cancel -id tag
```

Administration Console を使用してデプロイメントをキャンセルする

Administration Console で [タスク] ノードを開き、現在のデプロイメント タスクを表示してキャンセルします。

デプロイ済みアプリケーションを表示する

デプロイ済みのアプリケーションを Administration Console で表示するには、次の操作を行います。

1. Administration Console で、[デプロイメント] をクリックします。
2. [アプリケーション] オプションをクリックします。
3. Administration Console 内のテーブルに、デプロイ済みアプリケーションの一覧が表示されます。

デプロイ済みアプリケーションをアンデプロイする

デプロイ済みのアプリケーションを WebLogic Server Administration Console からアンデプロイするには、次の操作を行います。

1. Administration Console で、[デプロイメント] をクリックします。
2. [アプリケーション] オプションをクリックします。
3. 表示されたテーブルで、アンデプロイするアプリケーションの名前をクリックします。
4. [コンフィグレーション] タブをクリックし、[デプロイ] チェック ボックスをオフにします。
5. [適用] をクリックします。

移行可能サービスをデプロイ、活性化、および移行する

以降の節では、移行可能サービスをデプロイ、活性化、および移行するためのガイドラインと手順を示します。移行可能サービスについては、5-24 ページの「固定サービスの移行」を参照してください。

移行可能対象のサーバ インスタンスに JMS をデプロイする

ここで作成する移行可能対象は、移行可能サービスのホストとなることができるクラスタ内のサーバ インスタンスのスコープを定義します。後から対象サーバ リストの内部でサービスを移行するためには、移行可能対象として登録されているいずれかのサーバ インスタンス上で固定サービスをデプロイまたは活性化する必要があります。次の手順に従って移行可能対象上に JMS サーバをデプロイするか、後から移行できるように JTA トランザクション回復システムを活性化します。

注意： 移行可能対象をコンフィグレーションしていない場合は、単純に JMS サーバをクラスタ内の任意の WebLogic Server インスタンスにデプロイします。その後、クラスタ内の任意の別サーバ インスタンスに JMS サーバを移行できます (移行可能対象は使用しません)。

Administration Console を使用して JMS サーバを移行可能対象にデプロイするには、次の手順に従います。

1. まだ作成していない場合、7-25 ページの「固定サービスの移行可能対象をコンフィグレーションする」の指示に従って、クラスタに対して移行可能対象を作成します。
2. 対象のクラスタの管理サーバを起動し、Administration Console にログインします。
3. 左ペインで [JMS] ノードを選択し、このノードの下にある [サーバ] ノードを選択します。
4. クラスタにデプロイする、コンフィグレーション済み JMS サーバの名前を選択します。JMS サーバのコンフィグレーションが右ペインに表示されます。
5. 右ペインで [対象 | 移行できる対象] タブを選択します。
6. ドロップダウン リストから、サーバ インスタンスの名前を選択します。このドロップダウン リストには、移行可能対象の一部として定義されているサーバの名前が表示されます。
7. [適用] をクリックして、選択した WebLogic Server インスタンスに JMS サーバを適用します。

移行可能サービスとして JTA を活性化する

JTA 回復サービスは、クラスタの移行可能対象として登録されているいずれかのサーバ インスタンス上で自動的に起動されます。

選択したサーバ インスタンスにサービスをデプロイする必要はありません。JTA 移行可能対象をコンフィグレーションしない場合、WebLogic Server はクラスタ内で使用可能な任意の WebLogic Server インスタンス上でサービスを活性化します。JTA サービスのホストである現在のサーバ インスタンスを変更するには、7-35 ページの「対象サーバ インスタンスに固定サービスを移行する」で示されている手順に従います。

対象サーバ インスタンスに固定サービスを移行する

移行可能サービスをデプロイした後で、Administration Console を使用して、サービスをクラスタ内の別のサーバ インスタンスに移行できます。サービスに対して移行可能対象がコンフィグレーションされている場合、移行可能対象として登録されているサーバ インスタンスであれば、動作していない場合でもそのサーバ インスタンスにサービスを移行することができます。移行可能対象をコンフィグレーションしない場合、クラスタ内のその他の任意のサーバ インスタンスにサービスを移行できます。

停止しているサーバ インスタンスにサービスを移行する場合、そのサーバ インスタンスは次回の起動時に直ちにサービスを活性化します。動作中の WebLogic Server インスタンスにサービスを移行する場合、移行は直ちに有効になります。

Administration Console を使用して固定サービスを移行するには、次の操作を行います。

1. まだデプロイしていない場合、7-33 ページの「移行可能対象のサーバ インスタンスに JMS をデプロイする」の指示に従って、固定サービスをクラスタにデプロイします。
2. 対象のクラスタの管理サーバを起動し、Administration Console にログインします。
3. 左ペインで [サーバ] ノードを選択し、コンフィグレーションするクラスタのメンバーであるサーバ インスタンスを選択します。
4. JMS サービスを移行する場合は [制御 | 移行] を、JTA トランザクション回復サービスを移行する場合は [制御 | JTA 移行] をそれぞれ選択します。

   [現在のサーバ] フィールドには、その時点で固定サービスのホストとなっている WebLogic Server インスタンスが表示されています。[送り先サーバ] ドロップダウン リストには、サービスを移行できるサーバ インスタンスが表示されます。
5. [送り先サーバ] ドロップダウン リストを使用して、新しく固定サービスのホストとなるサーバ インスタンスを選択します。
6. [移行] をクリックして、固定サービスを現在のサーバから移行先サーバへ移行します。

7. 停止している WebLogic Server インスタンスへのサービス移行を選択した場合、サーバ インスタンスが停止していることが通知され、移行を続けるかどうかが確認されます。停止しているサーバ インスタンスへの移行を続けるには [続行] を、移行を中止して別のサーバ インスタンスを選択するには [取り消し] をそれぞれクリックします。
8. サーバ インスタンスのコンフィグレーションによっては、移行プロセスを完了するまでに数分以上かかる場合があります。ただし、移行タスクを処理しながら、Administration Console で他の作業を続けることができます。後から移行の状況を確認するには、左ペインで [タスク] ノードをクリックし、対象のドメインに対して実行中であるタスクを表示します。次に、移行タスクの説明を選択して、現在の状況を表示します。

インメモリ HTTP レプリケーションをコンフィグレーションする

WebLogic Server では、HTTP セッション ステートをメモリ内にレプリケートすることによって、サーブレットおよび JSP の自動フェイルオーバを行います。

注意： WebLogic Server では、ファイルベースまたは JDBC ベースの永続性メカニズムを利用して、サーブレットまたは JSP の HTTP セッション ステートを維持することもできます。それぞれの永続性メカニズムの詳細については、『WebLogic HTTP サーブレット プログラマーズ ガイド』の「セッションの永続化」を参照してください。

インメモリでの HTTP セッション ステート レプリケーションは、デプロイするアプリケーションごとに個別に制御されます。レプリケーションを制御するパラメータ (`PersistentStoreType`) は、対象アプリケーションの WebLogic デプロイメント記述子ファイルである `weblogic.xml` の `session-descriptor` 要素の内部にあります。

domain_directory/applications/*application_directory*/Web-Inf/`weblogic.xml`

クラスタ内のサーバ インスタンス間でインメモリでの HTTP セッション ステート レプリケーションを使用するには、`PersistentStoreType` を `replicated` に設定します。次に示すのは、`weblogic.xml` での適切な設定例です。

```
<session-descriptor>
        <session-param>
                <param-name> PersistentStoreType </param-name>
                <param-value> replicated </param-value>
        </session-param>
</session-descriptor>
```

コンフィグレーションに関するその他のトピック

以降の節では、クラスタの特殊なコンフィグレーションを行うときに役に立つヒントを示します。

IP ソケットをコンフィグレーションする

最適なパフォーマンスを得るために、WebLogic Server インスタンスのホストであるマシン上では、pure-Java 実装ではなくネイティブのソケット リーダー実装を使用することをお勧めします。

ホスト マシンで pure-Java ソケット リーダー実装を使用しなければならない場合でも、サーバ インスタンスおよびクライアント マシンごとにソケット リーダー スレッドの数を適切にコンフィグレーションすることによって、ソケット通信のパフォーマンスを向上させることができます。

- クラスタ内での IP ソケットの使用方法と、ネイティブ ソケット リーダー スレッドが最適なパフォーマンスをもたらす理由については、2-5 ページの「IP ソケットを使用したピア ツー ピア通信」および 2-9 ページの「ソケット経由のクライアント通信」を参照してください。
- クラスタで必要なソケット リーダー スレッドの数を決定する方法については、2-7 ページの「ソケットの使用数を確定する」を参照してください。多層クラスタ アーキテクチャにサーブレット クラスタをデプロイしている場合、6-12 ページの「多層アーキテクチャのコンフィグレーションに関する注意」で説明しているように、これは必要なソケットの数に影響します。

以降の節では、ホスト マシンに対してネイティブ ソケット リーダー スレッドをコンフィグレーションする方法と、ホストおよびクライアント マシンに対してリーダー スレッドの数を設定する方法について説明します。

サーバ インスタンスのホスト マシン上でネイティブの IP ソケット リーダーをコンフィグレーションする

ネイティブのソケット リーダー スレッド実装を使用するように WebLogic Server インスタンス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. WebLogic Server Console を起動します。
2. [サーバ] ノードを選択します。
3. コンフィグレーション対象のサーバ インスタンスを選択します。
4. [チューニング] タブを選択します。
5. [ネイティブ IO を有効化] ボックスをチェックします。
6. 変更を適用します。

サーバ インスタンスのホスト マシン上のリーダー スレッドの数を設定する

デフォルトでは、WebLogic Server インスタンスは起動時に 3 つのソケット リーダー スレッドを作成します。クラスタ システムがピーク時に 4 つ以上のソケットを使用する可能性がある場合は、ソケット リーダー スレッド数を増やします。

1. WebLogic Server Console を起動します。
2. [サーバ] ノードを選択します。
3. コンフィグレーション対象のサーバ インスタンスを選択します。
4. [チューニング] タブを選択します。
5. [ソケット リーダー] 属性フィールドで Java リーダー スレッドの割合を編集します。Java ソケット リーダーの数が、合計実行スレッド数 ([実行スレッド] 属性フィールドに表示されます) の割合として計算されます。
6. 変更を適用します。

クライアント マシン上のリーダー スレッド数を設定する

クライアント マシン上では、クライアントを実行する Java 仮想マシン (JVM) 内でソケット リーダーの数をコンフィグレーションできます。クライアントの Java コマンドラインで `-Dweblogic.ThreadPoolSize=value` オプションおよび `-Dweblogic.ThreadPoolPercentSocketReaders=value` オプションを定義してソケット リーダーを指定します。

マルチキャスト存続時間 (TTL) をコンフィグレーションする

クラスタが WAN 内の複数のサブネットにまたがっている場合、マルチキャスト パケットが最終の送り先に到達する前にルータがパケットを破棄しないように、クラスタのマルチキャスト存続時間 (Time-To-Live: TTL) パラメータの値を十分に大きく設定する必要があります。マルチキャスト存続時間パラメータには、パケットが破棄されるまでにマルチキャスト メッセージが経由できるネットワーク ホップ数を設定します。マルチキャスト存続時間パラメータを適切に設定することにより、クラスタ内のサーバ インスタンス間で送受信されるマルチキャスト メッセージが消失するリスクが少なくなります。

マルチキャスト メッセージが確実に転送されるようにネットワーク トポロジを設計する方法については、2-3 ページの「クラスタが WAN 内の複数のサブネットにまたがる場合」を参照してください。

クラスタのマルチキャスト存続時間をコンフィグレーションするには、Administration Console で、対象となるクラスタの [マルチキャスト] タブにある [マルチキャスト TTL] の値を変更します。`config.xml` の以下の抜粋部分は、マルチキャスト存続時間値に 3 を指定したクラスタを示しています。この値によって、破棄される前にクラスタのマルチキャスト メッセージを 3 つのルータに渡すことができます。

```
<Cluster
        Name="testcluster"
        ClusterAddress="wanclust"
        MulticastAddress="wanclust-multi"
        MulticastTTL="3"
/>
```

マルチキャスト バッファのサイズをコンフィグレーションする

クラスタ内のサーバ インスタンスが受信メッセージを適時に処理しないことが原因でマルチキャスト ストームが発生する場合は、マルチキャスト バッファのサイズを増やすことができます。マルチキャスト ストームの詳細については、2-4 ページの「マルチキャスト ストームが起こったら」を参照してください。

TCP/IP カーネル パラメータは、UNIX の ndd ユーティリティを使用してコンフィグレーションできます。`udp_max_buf` パラメータでは、UDP ソケットの送信および受信バッファのサイズをバイト単位で管理します。`udp_max_buf` の適切な値は、デプロイメントによって異なります。マルチキャスト ストームが発生する場合は、`udp_max_buf` の値を 32KB 単位で増やして、その変更の効果を評価します。

必要な場合以外は、`udp_max_buf` の値を変更しないでください。`udp_max_buf` を変更する前に、『Solaris Tunable Parameters Reference Manual』(http://docs.sun.com/?p=/doc/806-6779/6jfmsfr7o&) の「TCP/IP Tunable Parameters」という章の「UDP Parameters with Additional Cautions」に記載されいてる警告を読んでください。

マシン名をコンフィグレーションする

クラスタ内の各サーバ インスタンスにマシン名を定義することができます。マシン名は必須ではありませんが、複数のマシンでクラスタを構成する場合と、複数のサーバ インスタンスがクラスタ内の 1 台のマシン上で実行される場合には設定することをお勧めします。

WebLogic Server では、コンフィグレーションされたマシン名を使用して、2 つのサーバ インスタンスが物理的に同じハードウェアに存在しているかどうかを調べることができます。マシン名は一般に、マルチホーム WebLogic Server インスタンスのホストとなるマシンで使用されます。そのようなインストール用のマシン名を定義していない場合、各インスタンスは物理的に異なるハードウェア上に存在するものとして扱われます。このことは、5-6 ページの「レプリケーション グループを使用する」で説明するように、セカンダリ HTTP セッション ステートのレプリカのホストになるサーバ インスタンスの選択に悪影響を与えることがあります。

1 台のマシン上で複数のサーバ インスタンスを実行する予定の場合、それらのサーバ インスタンスを作成する前に、サーバ インスタンスのホストとなるマシンの名前を次のようにして定義します。

1. 管理サーバを起動します。起動方法については、『管理者ガイド』の「管理サーバの起動」を参照してください。
2. [マシン] ノードを選択します。
3. [新しい Machine のコンフィグレーション] を選択して Windows NT マシンを定義するか、または [新しい Unix Machine のコンフィグレーション] を選択します。
4. [名前] 属性フィールドに新しいマシンのユニークな名前を入力します。
5. [作成] をクリックして、新しいマシンの定義を作成します。
6. 新しい UNIX サーバのその他の属性を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 ヘルプを参照してください。

- クラスタ内の 1 つまたは複数の WebLogic Server インスタンスのホストになるマシンごとに、上記の手順を繰り返します。

多層アーキテクチャのコンフィグレーションに関する注意

多層アーキテクチャのクラスタのコンフィグレーションについては、6-12 ページの「多層アーキテクチャのコンフィグレーションに関する注意」のガイドラインを参照してください。

- JMS サーバについては、単独サーバの対象または移行可能対象を識別できます。移行可能対象は、移行可能サービスのホストとなることができるクラスタ内の WebLogic Server インスタンスの集合です。移行可能対象の詳細については、「JMS 移行できる対象のコンフィグレーション」を参照してください。
- 接続ファクトリでは、単一サーバの対象とクラスタの対象を識別できます。対象とは、クラスタ化をサポートするために接続ファクトリに関連付けられた WebLogic Server のインスタンスです。

これらのコンフィグレーション属性の詳細については、『管理者ガイド』の「JMS サーバのコンフィグレーション」または「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。

注意： 同じ送り先を複数の JMS サーバにデプロイすることはできません。また、1 つの JMS サーバを複数の WebLogic Server にデプロイすることもできません。

必要に応じて、クラスタの内部で単独の分散送り先セットの一部として物理送り先をコンフィグレーションすることができます。詳細については、『管理者ガイド』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。

URL 書き換えを有効にする

デフォルト コンフィグレーションの WebLogic Server では、クライアント側のクッキーを使用して、クライアントのサーブレット セッション ステートのホストであるプライマリ サーバ インスタンスとセカンダリ サーバ インスタンスが追跡されます。クライアントのブラウザでクッキーが無効になっている場合、WebLogic Server では URL 書き換えによってもプライマリ サーバ インスタンスとセカンダリ サーバ インスタンスを追跡できます。URL 書き換えを利用する場合は、クライアント セッション ステートの両方の位置が、クライアントとプロキシ サーバの間で渡される URL に挿入されます。この機能をサポートするには、WebLogic Server クラスタで URL 書き換えを有効にする必要があります。URL 書き換えを有効にする方法については、『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』の「URL 書き換えの使い方」を参照してください。

8 一般的な問題のトラブルシューティング

この節では、クラスタに関連する問題の発生を防ぎ、問題に対処するためのガイドラインを示します。

クラスタを起動する前に

クラスタを起動する前に、問題を避けるためのいくつかの確認を行うことができます。

クラスタ ライセンスのチェック

WebLogic Server のライセンスにクラスタ化機能が含まれている必要があります。クラスタ化ライセンスがない状態でクラスタを起動しようとすると、「**Unable to find a license for clustering**」というエラー メッセージが表示されます。

サーバのバージョン番号のチェック

クラスタ内のすべての管理対象サーバとクラスタの管理サーバは、WebLogic Server の同じバージョンで実行する必要があります。メジャーおよびマイナーバージョン番号 (6.1 など)、サービス パック、および適用されたパッチのレベルがクラスタ全体で同じでなければなりません。

マルチキャスト アドレスのチェック

マルチキャスト アドレスに関する問題は、クラスタが起動しないか、またはサーバがクラスタに参加できないことの最も一般的な理由の 1 つです。

マルチキャスト アドレスはクラスタごとに必要です。マルチキャスト アドレスには 224.0.0.0 ～ 239.255.255.255 の範囲の IP アドレスか、またはその範囲内の IP アドレスを持つホスト名を使用できます。

クラスタのマルチキャスト アドレスとポートは、WebLogic Server コンソールを使用してチェックできます。

ネットワーク上の各クラスタで、マルチキャスト アドレスとポートの組み合わせはユニークでなければなりません。ネットワーク上の 2 つのクラスタで同じマルチキャスト アドレスを使用する場合、それぞれのクラスタのポートは異なっている必要があります。複数のクラスタでマルチキャスト アドレスが異なっていれば、それらのクラスタで同じポートまたはデフォルトの 7001 番ポートを使用できます。

クラスタを起動する前に、クラスタのマルチキャスト アドレスとポートが正しいことと、ネットワーク上の他のクラスタとの間でマルチキャスト アドレスとポートの組み合わせが重複していないことを確認してください。

マルチキャスト アドレスが不正な場合に最も起こりやすいエラーには、以下のものがあります。

- **クラスタ化のためのマルチキャスト ソケットを作成できない**
- **マルチキャスト ソケットの送信エラー**
- **マルチキャスト ソケットの受信エラー**

CLASSPATH の値のチェック

CLASSPATH の値が、クラスタ内のすべての管理対象サーバ間で一致していることを確認します。CLASSPATH は setEnv スクリプトによって設定されます。このスクリプトは、startManagedWebLogic を実行して管理対象サーバを起動する前に実行します。

デフォルトでは、setEnv は CLASSPATH の値を次のように設定します (Windows システム上での表現形式)。

```
set WL_HOME=C:\bea\weblogic700
set JAVA_HOME=C:\bea\jdk131
.
.
set CLASSPATH=%JAVA_HOME%\lib\tools.jar;
       %WL_HOME%\server\lib\weblogic_sp.jar;
       %WL_HOME%\server\lib\weblogic.jar;
       %CLASSPATH%
```

ある管理対象サーバで CLASSPATH の値を変更するか、または setEnv による CLASSPATH の設定内容を変更する場合、クラスタ内のすべての管理対象サーバで同じ変更を行う必要があります。

スレッド カウントのチェック

クラスタ内の各サーバには実行スレッド カウントが割り当てられます。このカウントは Administration Console からチェックできます ([**サーバ | (対象のサーバ名) | モニタ | すべてのアクティブなキューのモニタ | Execute Queue のコンフィグレーション**] をクリックし、次にサーバ リスト内の **default** をクリックします)。

管理対象サーバを起動する前に、そのスレッド カウント属性をチェックします。デフォルト値は 15 であり、最小値は 5 です。スレッド カウントの値が 5 未満の場合、管理対象サーバが起動時にハングしないように値を 5 以上に変更します。

クラスタ起動後の作業

この節では、クラスタを起動する際に問題が発生した場合に、最初に実行するトラブルシューティングの手順について説明します。

コマンドのチェック

クラスタが起動できないかまたはサーバがクラスタに参加できない場合は、まず最初に、`startManagedWebLogic` コマンドや `java` インタプリタ コマンドなどの入力間違いをチェックします。

このリリースでは、システム名とパスワードに `weblogic` を使用してサーバが起動する点に注意してください。

ログ ファイルの生成

クラスタ関連の問題について BEA テクニカル サポートにお問い合わせになる前に、いくつかの診断情報を収集する必要があります。最も必要な情報は、管理対象サーバからの複数のスレッド ダンプが出力されたログ ファイルです。ログ ファイルは特に、クラスタのフリーズやデッドロックの問題に対処する場合に重要です。

複数のスレッド ダンプが出力されたログ ファイルは、問題を診断するための前提条件となります。

ログ ファイルを作成するには、管理サーバまたは管理対象サーバ上で次の手順に従います。

1. サーバを停止します。
2. この時点でログ ファイルがあればすべて削除するか、バックアップします。既存のログ ファイルに追加書き込みを行うよりも、サーバを起動するたびに新しいログ ファイルを作成することをお勧めします。

3. 次のコマンドでサーバを起動します。このコマンドでは冗長ガベージ コレクションを有効にし、標準エラーと標準出力の両方をログ ファイルにリダイレクトします。

```
% java -ms64m -mx64m -verbose:gc -classpath $CLASSPATH
        -Dweblogic.domain=mydomain
-Dweblogic.Name=clusterServer1
        -Djava.security.policy==$WL_HOME/lib/weblogic.policy
        -Dweblogic.admin.host=192.168.0.101:7001
         weblogic.Server > logfile.txt 2>&1
```

注意： 構文は、UNIX と NT の両方で似ています。このコマンドでは、logfile.txt が上書きされます。上書きではなく追加にするには、最初の「>」を「>>」に変えます。

標準エラーと標準出力の両方をリダイレクトすることにより、サーバの通知メッセージとエラー メッセージを含む、適切なコンテキストのスレッド ダンプ情報が得られ、問題を解析しやすくなります。

4. 問題が再発生するまで、クラスタの実行を継続します。
5. サーバがハングした場合は、`kill -3` コマンドまたは `<Ctrl>-<Break>` を使用して、問題を診断するために必要なスレッド ダンプを作成します。デッドロックの診断を行うには、この作業を各サーバに対して、約 5 ～ 10 秒間隔で何度か行うようにしてください。

注意： Linux 環境で JRockit JVM を実行している場合は、8-5 ページの「Linux 環境での JRockit スレッド ダンプの取得」を参照してください。

6. UNIX ユーティリティを使用してログ ファイルを圧縮します。

```
% tar czf logfile.tar logfile.txt
```

または、Windows の zip ユーティリティを使用して圧縮します。

7. BEA テクニカル サポート担当者宛の電子メールに、圧縮したログ ファイルを添付します。メールの本文にログ ファイルの内容を切り取って貼り付けないでください。
8. 圧縮してもログ ファイルのサイズがまだ大きすぎる場合、BEA カスタマ サポート FTP サイトを使用できます。

Linux 環境での JRockit スレッド ダンプの取得

Linux 環境で JRockit JVM を使用する場合、以下のいずれかの方法でスレッド ダンプを生成できます。

- `weblogic.admin THREAD_DUMP` コマンドを使用する。手順と制限については、『管理者ガイド』の「THREAD_DUMP」を参照してください。
- `-Xmanagement` オプションで JVM を起動して JVM の管理サーバが有効になっている場合、JRockit Management Console を使用してスレッド ダンプを生成できる。
- `Kill -3 PID` を使用する。*PID* はプロセス ツリーのルートです。

  ルート PID を取得するには、次のコマンドを実行します。

  ```
  ps -efHl | grep 'java' **. **
  ```

  `grep` の文字列引数には、プロセス スタック内でサーバの起動コマンドと一致する文字列を使用します。`ps` コマンドを別のルーチンにパイプしていなければ、出力された最初の PID がルート プロセスとなります。

  Linux 環境では、各実行スレッドが Linux プロセス スタック下の独立したプロセスとして表示されます。Linux 上で Kill -3 を使用する場合、指定する PID が、WebLogic のメイン実行スレッドの PID と一致しなければなりません。一致していないと、スレッド ダンプは生成されません。

ガベージ コレクションのチェック

クラスタで問題が発生している場合、管理対象サーバ上でガベージ コレクションをチェックすることもお勧めします。ガベージ コレクションに時間がかかりすぎる場合、サーバが使用可能であることを他のクラスタ メンバーに通知する定期的なハートビート シグナルをサーバから出すことができなくなります。

ガベージ コレクション (最初または 2 番目の世代) が 10 秒以上かかる場合、システム上のヒープ割り当て (`msmx` パラメータ) を調整する必要があります。

utils.MulticastTest の実行

いずれかの管理対象サーバから `utils.MulticastTest` を実行することにより、マルチキャストが機能していることを検証できます。

A　WebLogic クラスタの API

以下の節では、WebLogic クラスタ API について説明します。

- API の使い方
- カスタム呼び出しルーティングと連結の最適化

API の使い方

WebLogic クラスタの公開 API は、単一インタフェース `weblogic.rmi.cluster.CallRouter` に含まれています。

```
Class java.lang.Object
   Interface weblogic.rmi.cluster.CallRouter
      (extends java.io.Serializable)
```

パラメータベースのルーティングを可能にするには、このインタフェースを実装するクラスを RMI コンパイラ (`rmic`) に与えなければなりません。以下のオプションを使って (すべて 1 行に入力します)、サービス実装時に `rmic` を実行します。

```
$ java weblogic.rmic -clusterable -callRouter
    <callRouterClass> <remoteObjectClass>
```

リモート メッセージが呼び出されるたびに、クラスタ化可能なスタブからコール ルータを呼び出します。コール ルータは、その呼び出しの宛先のサーバの名前を返します。

クラスタ内の各サーバは、WebLogic Server Console で定義された名前でユニークに識別されます。これらの名前は、メソッド ルータがサーバを識別するための名前となります。

例 : `ExampleImpl` というクラスを例に説明します。このクラスは、メソッド `foo` でリモート インタフェース Example を実装します。

```
public class ExampleImpl implements Example {
  public void foo(String arg) { return arg; }
}
```

この CallRouter を実装した ExampleRouter では、'arg' < "n" の場合にすべての foo 呼び出しが server1 (server1 に届かない場合は server3) に送られ、'arg' > "n" の場合にすべての呼び出しが server2 (server2 に届かない場合は server3) に送られます。

```
public class ExampleRouter implements CallRouter {
  private static final String[] aToM = { "server1", "server3" };
  private static final String[] nToZ = { "server2", "server3" };

  public String[] getServerList(Method m, Object[] params) {
    if (m.GetName().equals("foo")) {
      if (((String)params[0]).charAt(0) < 'n') {
        return aToM;
    } else {
        return nToZ;
      }
    } else {
         return null;
    }
  }
}
```

次の rmic 呼び出しは、ExampleRouter と ExampleImpl を関連付けて、パラメータベースのルーティングを有効にします。

```
$ rmic -clusterable -callRouter ExampleRouter ExampleImpl
```

カスタム呼び出しルーティングと連結の最適化

オブジェクトがレプリカを呼び出している同じサーバ インスタンス上にレプリカがある場合、ローカル レプリカを使用する方が効率的なので、その呼び出しはロード バランシングの対象にはなりません。詳細については、4-9 ページの「連結されたオブジェクトの最適化」を参照してください。

B　クラスタに関する BIG-IP™ ハードウェアのコンフィグレーション

この章の内容は以下のとおりです。

- 概要
- BIG-IP および WebLogic Server の使用時に URL 書き換えを利用する
- BIG-IP および WebLogic Server の使用時にセッションの永続性を利用する

概要

この節では、WebLogic Server クラスタで動作するように F5 BIG-IP コントローラをコンフィグレーションする方法について説明します。ここでは、読者が BIG-IP のコンフィグレーション作業を理解していることを前提にしています。

BIG-IP のコンフィグレーション手順の一部については、順を追って説明します。設定および管理の詳細な手順については、F5 製品のマニュアルを参照してください。

WebLogic Server が外部ロード バランサとどのように連携して動作するかについては、4-3 ページの「外部ロード バランサによる HTTP セッションのロード バランシング」を参照してください。

BIG-IP および WebLogic Server の使用時に URL 書き換えを利用する

BIG-IP および WebLogic Server インスタンスを使用している場合に URL 書き換えを利用するには、BIG-IP がバージョン 4.5 以降、すなわち `Rewrite cookie persistence` にコンフィグレーションされたものである必要があります。BIG-IP の永続性の設定がそれ以外の場合、フェイルオーバが成功しないこともあります。

URL 書き換えを利用するように WebLogic Server をコンフィグレーションする手順については、『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』の「URL 書き換えの使い方」を参照してください。

BIG-IP および WebLogic Server の使用時にセッションの永続性を利用する

クラスタがクライアント セッションのステート用にイン メモリ レプリケーションを使用する場合、クッキーの挿入モードを使用するよう BIG-IP をコンフィグレーションしなければなりません。挿入モードを使用すると、元の WebLogic Server クッキーが上書きされることがなくなるので、クライアントがプライマリ WebLogic Server に接続できなかった場合にそのクッキーを使用できます。

BIG-IP クッキーの挿入モード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. BIG-IP コンフィグレーション ユーティリティを開きます。
2. ナビゲーション ペインで [Pools] オプションを選択します。
3. コンフィグレーションするプールを選択します。
4. [Persistence] タブを選択します。
5. [Active HTTP Cookie] を選択して、クッキーのコンフィグレーションを開始します。

6. 方法リストから [Insert mode] を選択します。
7. クッキーのタイムアウト値を入力します。タイムアウト値は、挿入されたクッキーが有効期限切れになるまでクライアントに保存される時間を指定します。タイムアウト値は WebLogic Server セッションのクッキーには影響せず、挿入される BIG-IP クッキーについてのみ有効です。

   ラウンドロビン方式に基づいたリクエストをロード バランシングするには、タイムアウト値を 0 に設定します。これにより、ラウンドロビン方式に従って、同じクライアントからの複数のリクエストは同じ管理対象サーバに転送され、別のクライアントからのリクエストはクラスタ内の別の管理対象サーバに転送されるようになります。

   タイムアウト値を 0 以上に設定すると、そのタイムアウト期間中はロード バランサによりすべてのクライアントからのすべてのリクエストが、WebLogic Server クラスタ内の同じ管理対象サーバに送信されます。つまりそのタイムアウト期間中は、別のクライアントからのリクエストがロード バランシングされません。
8. 変更を適用して、ユーティリティを終了します。