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 6.1
マニュアルの日付：2003 年 4 月 24 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	マニュアルの日付	ソフトウェアのバージョン
なし	2002 年 6 月 24 日	BEA WebLogic Server バージョン 6.1

目次

4. オブジェクトのクラスタ化について

5. WebLogic Server クラスタのプランニング

6. WebLogic クラスタの管理

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 6.1 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 のクラスタ化の概要」では、WebLogic Server クラスタの概念、および WebLogic Server 6.1 におけるクラスタ化の変更点について説明します。
- 第 2 章「クラスタの機能とインフラストラクチャ」では、HTTP セッションとクラスタ化されたオブジェクトに対してクラスタで提供される基本的な機能について説明します。
- 第 3 章「HTTP セッション ステートのレプリケーションについて」では、自動ロード バランシングおよびフェイルオーバを提供するために、WebLogic Server クラスタがどのように HTTP セッション ステートをメモリ内でレプリケートするかについて説明します。
- 第 4 章「オブジェクトのクラスタ化について」では、クラスタ化された EJB オブジェクトと RMI オブジェクトに対して WebLogic Server で提供されるロード バランシングとフェイルオーバについて説明します。
- 第 5 章「WebLogic Server クラスタのプランニング」では、1 つまたは複数の WebLogic Server クラスタをデプロイする前に検討しておく必要のある事項について説明します。また、一般的な Web アプリケーションの推奨クラスタ アーキテクチャについても説明します。
- 第 6 章「WebLogic クラスタの管理」では、WebLogic Server クラスタの設定および実行方法など、WebLogic Server クラスタの管理について説明します。
- 付録 A 「一般的な問題のトラブルシューティング」では、クラスタに関する問題の解決に役立つチェックリストを提供します。
- 付録 B 「WebLogic クラスタの API」では、RMI オブジェクト用クラスタ API と、API を使用した開発について説明します。

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を一度に 1 ファイルずつ印刷できます。

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

Adobe Acrobat Reader をインストールしていない場合は、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」（http://edocs.beasys.co.jp/e-docs/wls61/ejb/EJB_environment.html）
- 『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

BEA 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@bea.com までお送りください。寄せられた意見については、WebLogic Server のドキュメントを作成および改訂する BEA の専門の担当者が直に目を通します。

電子メールのメッセージには、ご使用のソフトウェア名とバージョン名、およびマニュアルのタイトルと作成日付をお書き添えください。

本バージョンの BEA WebLogic Server について不明な点がある場合、または BEA WebLogic Server のインストールおよび動作に問題がある場合は、BEA WebSUPPORT（www.bea.com）を通じて BEA カスタマ サポートまでお問い合わせください。カスタマ サポートへの連絡方法については、製品パッケージに同梱されているカスタマ サポート カードにも記載されています。

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

- お名前、電子メール アドレス、電話番号、ファクス番号
- 会社の名前と住所
- お使いの機種とコード番号
- 製品の名前とバージョン

- 問題の状況と表示されるエラー メッセージの内容

表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

表記法	適用
太字	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	コード内の重要な箇所を示す。 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 のクラスタ化の概要

以下の節では、WebLogic Server クラスタについて説明します。

- WebLogic Server クラスタとは
- クラスタ化されたサービス
- WebLogic Server バージョン 6.1 のクラスタの新機能

WebLogic Server クラスタとは

WebLogic Server クラスタはより強力で、より信頼性のあるアプリケーション プラットフォームを提供するためのサーバ群です。クラスタはそのクライアントにとって単一のサーバに見えますが、実際には、一体で機能するサーバ群です。クラスタは、単一のサーバを越える下記の 2 つの重要な機能を提供します。

- **スケーラビリティ：**クラスタの能力は 1 つのサーバまたは 1 つのマシンに制限されません。新しいサーバをクラスタに動的に追加して能力を増大させることができます。ハードウェアがさらに必要な場合は、新しいサーバを新しいマシン上に追加できます。1 つのサーバが既存のマシンを完全に使用しきっていない場合は、そのマシンに別のサーバを追加できます。
- **高可用性：**クラスタは複数サーバの冗長性を利用して、クライアントを障害から保護します。クラスタ内の複数のサーバが同じサービスを提供できます。1 つのサーバで障害が発生しても、別のサーバが引き継ぎます。障害が発生したサーバから機能しているサーバへのフェイルオーバ機能によって、クライアントに対するアプリケーションの可用性が増大します。

WebLogic Server クラスタは、J2EE アプリケーションにスケーラビリティと高可用性を提供するよう設計されています。これらの機能はアプリケーションの作成者とクライアントには透過的な方法で提供されます。ただし、アプリケーションのスケーラビリティと可用性を最大にするには、プログラマと管理者がクラスタ化固有の問題を理解しておく必要があります。

クラスタ化されたサービス

クラスタ化されたサービスは、クラスタ内の複数のサーバで利用できる API とインタフェースです。WebLogic Server が提供する主なクラスタ サービスは、HTTP セッション ステートのクラスタ化とオブジェクトのクラスタ化の 2 つです。以降の節で説明するように、WebLogic Server では JMS 送り先と JDBC 接続のクラスタ化にも対応しています。

HTTP セッション ステート

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。HTTP セッション ステートのクラスタ化のメリットを活かすには、インメモリ レプリケーション、ファイルシステムの永続性、または JDBC の永続性をコンフィグレーションすることで、セッション ステートを永続的にしておく必要があります。3-1 ページの「HTTP セッション ステートのレプリケーションについて」では、クラスタ化されたサーブレットと JSP 向けのインメモリ レプリケーションについて説明しています。ファイルの永続性と JDBC 永続性の詳細については、『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。

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、特殊な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、クラスタ全体の中でオブジェクトのインスタンスを見つけることができます。EJB と RMI オブジェ

クト用のレプリカ対応スタブを作成するには、適切なデプロイメント プロパティを指定するか、またはコマンドライン オプションを `rmic` に追加します。クラスタ化されたオブジェクトをデプロイする場合は、クラスタ内のすべてのサーバ インスタンスにオブジェクトをデプロイします（均一デプロイメント）。オブジェクトのクラスタ化の詳細については、4-1 ページの「オブジェクトのクラスタ化について」を参照してください。

JMS

WebLogic の JMS アーキテクチャでは、複数の JMS サーバの「クラスタ化」を実装するため、クラスタ内のすべてのサーバから送り先へのクラスタワイドで透過的なアクセスがサポートされています。WebLogic Server は、JMS の送り先と接続ファクトリのクラスタ全体への分散をサポートしますが、JMS のキューおよびトピックはクラスタ内の個々の WebLogic Server インスタンスによって管理されます。

WebLogic JMS のクラスタ化には、次のような利点があります。

- 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クラスタワイドで透過的なアクセスを確立できます。各接続ファクトリは、複数の WebLogic Server にデプロイできます。
- 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サーバ インスタンスに送り先のロード バランシングを行うことができます。個々の JMS サーバは、ただ 1 つの WebLogic Server インスタンスにデプロイされて、一群の送り先に対する要求を処理します。

WebLogic JMS の詳細と、JMS を対象としたクラスタのサポートについては、『WebLogic JMS プログラマーズ ガイド』の「WebLogic JMS の概要」を参照してください。

JDBC 接続

WebLogic Server では、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して、クラスタ環境で運用されるアプリケーションの可用性を高めることができます。JDBC オブジェクトをコンフィグレーションし、そのターゲットをクラスタに設定するとき、クラスタ用にコンフィグレーションする各 JDBC オブジェクトは、クラスタ内の各管理対象サーバ上に存在している必要があります。

- データ ソース — クラスタ内で、外部クライアントは JNDI ツリー上の JDBC データ ソースを通じ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獲得します。外部クライアント アプリケーション内の WebLogic データ ソースはクラスタに対応しているため、以前の接続のホストであったサーバ インスタンスが停止した場合、クライアントは別の接続を要求できます。必ずしも従う必要はありませんが、BEA では、サーバ側のクライアントも JNDI ツリー上のデータ ソースを介して接続を取得する構成を推奨しています。
- 接続プール — 接続プールは、使用の準備ができたデータベース接続のコレクションです。接続プールはその起動時に、指定された数の同じ物理データベース接続を作成します。接続プールが起動時に接続を確立することにより、各アプリケーション用にデータベース接続を作成するオーバーヘッドが解消されます。BEA では、クライアント側とサーバ側の両方のアプリケーションが、JNDI ツリー上のデータ ソースを通じて接続プールから接続を取得する構成を推奨しています。アプリケーションは接続が不要になった時点で、その接続を接続プールに返却します。
- マルチプール — マルチプールは基本の接続プールを多重化したものです。マルチプールはアプリケーションにとっては基本のプールと同じように機能しますが、マルチプールを使用すると、接続プールのプールを確立して、接続プールごとに特性の異なる多様な接続を利用できるようになります。1 つの接続プール内の接続はすべて等質ですが、あるプールに予測される障害がマルチプール内の別のプールを巻き添えにしたりしないように、マルチプール内の接続プールごとに接続の性質はある程度異なっているのが一般的です。これらのプールは通常、同じデータベースの異なったインスタンスを接続先とします。

  マルチプールが効果を発揮するのは、アプリケーションの接続を同じように処理する複数の異なったデータベース インスタンスが存在し、アプリケーションの作業が複数のデータベース間に分散されるときにアプリケーション システムがデータベースの同期を正しく処理する場合に限られます。まれ

に、同じデータベース インスタンスを指す複数のプールを異なったユーザとして用意すると役に立つ場合があります。これは、DBA が他のユーザを有効にしたままある特定のユーザを無効にする場合などに役立ちます。

デフォルトでは、クラスタ化されるマルチプールは高可用性（DBMS のフェイルオーバ）を実現します。必要に応じて、ロード バランシングにも対応するようにマルチプールをコンフィグレーションすることができます。

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 を使用しての接続の取得

どのクラスタ メンバーによってもすべての JDBC リクエストを同様に処理できることを保証するには、クラスタ内の各管理サーバのプールと（使用する場合の）マルチプールが、同じ名前を付けて同じように定義される必要があります。外部クライアントでの使用を想定する場合、データ ソースのターゲットをクラスタに設定することをお勧めします。これによりデータ ソースがクラスタ対応になり、データ ソースの接続をどのクラスタ メンバーからも利用できるようになります。

- 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 ソースのホストであるサーバ インスタンスのリスト（通常はクラスタ内のすべての管理対象サーバ）が含まれます。レプリカ対応スタブは、ホストのサーバ インスタンス間で負荷を分散させるためのロード バランシング ロジックを備えています。
- サーバサイド クライアント接続 — サーバサイドでの使用のために、アプリケーション コードでは JNDI ルックアップとデータ ソースの代わりにサーバサイド プール ドライバを直接使用できますが、データ ソースが使用される場合、そのデータ ソースはローカル オブジェクトになります。サーバサイドのデータ 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 ソースはその参照先のプールから接続を取得します。データベース トランザクションが継続し、アプリケーション コードが接続を保持する間（接続がクローズされるまで）、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実現されませんが、接続に障害が発生したときの再接続のプロセスを簡略化することができます。複製されたデータベース環境では、マルチプールをクラスタ化することによってデータベースのフェイルオーバを、また必要に応じて接続のロード バランシングを実現できます。詳細については、以下の各節を参照してください。

- クラスタ化される JDBC オブジェクトがフェイルオーバ発生時にどのように動作するかについては、2-20 ページの「クラスタ化されたサービスのフェイルオーバ サポート」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しくみの詳細については、2-15 ページの「クラスタ化されたサービスのロード バランシング」を参照してください。

クラスタ化されていないサービスと API

一部の API と内部サービスは、WebLogic Server ではクラスタ化できません。その内容は次のとおりです。

- 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）
- ZAC

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 バージョン 6.1 のクラスタの新機能

WebLogic Server バージョン 6.1 では、Weblogic Server バージョン 5.x と比較して、サーバ インスタンスのクラスタとしてコンフィグレーションする際の以下の機能と改良点を新たに取り入れました。

ロード バランシング ハードウェアの統合サポート

WebLogic Server は、クライアントがサポート対象のロード バランシング ハードウェアを使ってクラスタに直接アクセスする場合に、クラスタ化されたサーブレットと JSP 向けのロード バランシングとフェイルオーバをサポートするようになりました。`HttpClusterServlet` または WebLogic Server のプロキシ プラグインを使用して HTTP リクエストをクラスタにプロキシする場合に、クラスタ化されたシステムは不要となりました。詳細については、「HTTP セッション ステートのレプリケーションについて」を参照してください。

ステートフル セッション EJB のクラスタ化

WebLogic Server は、ステートフル セッション EJB インスタンスの `EJBObject`（と `EJBHome` オブジェクト）のクラスタ化をサポートするようになりました。WebLogic Server は、HTTP セッション ステートをレプリケートするのと似た方法で、ステートフル セッション EJB をメモリ内にレプリケートします。詳細については、オブジェクトのクラスタ化についてを参照してください。

クラスタ化された JMS

WebLogic Server は、クラスタ全体への JMS の送り先と接続ファクトリの配布をサポートするようになりました。ただし、JMS のトピックとキューがクラスタ内の個々の WebLogic Server インスタンスによって管理される点は変わりません。

HTTP セッション ステート レプリケーションに関する変更

WebLogic Server バージョン 6.1 では、クラスタ化されたサーバ インスタンス間でサーブレットのセッション ステートをレプリケートするためのより堅牢なメカニズムを導入しました。以前のサーバ バージョンと同じく、WebLogic Server は、サーブレットのセッション ステートの 2 つのコピー（プライマリとセカンダリ）をクラスタ内の別々のサーバ インスタンス上で維持します。バージョン 6.1 では、次の方法によって、レプリケーション システムを改良しました。

- WebLogic Server がセッション ステートのレプリカを作成する場所を、管理者が制御できるようになりました。
- WebLogic Server クラスタ内でのクライアントの接続先とは無関係に、クライアントがセカンダリ セッション ステートにフェイルオーバできるようになりました。

これらの変更によって、より幅広いシナリオでのフェイルオーバに対応できるようになると共に、WebLogic Server がロード バランシング ハードウェアと直接連携して処理を行えるようになります。詳細については、HTTP セッション ステートのレプリケーションについてを参照してください。

WebLogic Server バージョン 6.1 での管理に関する変更

以下の変更に合わせて、WebLogic Server 6.1 でクラスタを管理する方法も WebLogic Server バージョン 5.x の方法から変更されました。

マルチキャスト メッセージに関する変更

複数の WebLogic Server バージョン 6.1 クラスタは、ブロードキャスト メッセージの競合を起こさずに、単一のマルチキャスト アドレスを「共有」できるようになりました。クラスタごとに専用のマルチキャスト アドレスを割り当てる必要はありません。

ただし、ほかのアプリケーションが WebLogic Server のマルチキャスト アドレスを使用しないようにする必要はあります。他のアプリケーションがクラスタのマルチキャスト アドレスに対してブロードキャストする場合、クラスタ内のすべてのサーバは、それらのメッセージをデシリアライズして、クラスタ関連のメッセージでないことを調べなければなりません。これによって無用なオーバーヘッドが生まれるので、サーバが実際のクラスタ関連のブロードキャスト メッセージの送信に失敗する可能性があります。

均一デプロイメント

WebLogic Server バージョン 6.1 は、クラスタ化されたオブジェクトの均一デプロイメントだけをサポートします。オブジェクトをクラスタ可能オブジェクトとしてコンパイルした場合（オブジェクトがレプリカ対応スタブを持つ場合）は、そのオブジェクトをクラスタのすべてのメンバーに対してデプロイする必要があります。

クラスタ化されていないオブジェクト（EJB と RMI クラス）は、クラスタ内の個々のサーバにデプロイできます。

管理サーバのコンフィグレーション

WebLogic Server 6.1 クラスタのコンフィグレーションはすべて、Administration Console を使用して行います。Administration Console は、クラスタのコンフィグレーション情報を 1 つの XML コンフィグレーション ファイルに保存します。また、Administration Console は、クラスタ メンバーへのオブジェクトおよび Web アプリケーションのデプロイメントも管理します。

WebLogic Server 6.1 のコンフィグレーション全般については、『管理者ガイド』を参照してください。Administration Console を使用したクラスタのコンフィグレーションについては、「WebLogic クラスタの管理」を参照してください。

2 クラスタの機能とインフラストラクチャ

以下の節では、クラスタ化されたオブジェクトと HTTP セッション ステートをサポートするために WebLogic Server クラスタで使われるインフラストラクチャについて説明します。

- 概要
- クラスタ内のサーバの通信
- クラスタワイドの JNDI ネーミング サービス
- クラスタ化されたサービスのロード バランシング
- クラスタ化されたサービスのフェイルオーバ サポート

概要

また、WebLogic Server クラスタ内で実行される API とサービスで利用可能な一般的な機能、ロード バランシングとフェイルオーバについても説明します。これらの内容を理解しておくことは、Web アプリケーションのニーズを満たすように WebLogic Server クラスタをプランニングし、コンフィグレーションする際に重要となります。

クラスタ内のサーバの通信

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。クラスタ化された WebLogic Server インスタンス間で 1 対多通信をブロードキャストします。
- 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 が IP マルチキャストとソケット通信を使用する方法は、管理者がクラスタを計画し、コンフィグレーションする方法と直接関係しています。

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する簡単な方法ですが、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 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 ー すべてのサーバ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。サーバは、ローカルの JNDI ツリーを更新して、クラスタ化されたオブジェクトの最新のデプロイメントを反映できるように、これらの通知をモニタします。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」ー 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。クラスタ内のすべてのサーバは、サーバの障害を調べる方法としてハートビート メッセージをリスンします（クラスタ化されたサーバは、サーバの障害をより早く調べる方法として、IP ソケットもモニタします）。詳細については、クラスタ化されたサービスのフェイルオーバ サポートを参照してください。

クラスタのプランニングとコンフィグレーションの意味

マルチキャストは、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基本的なネットワーク トポロジインタフェースも重要ではありません。WebLogic Server クラスタのコンフィグレーションとプランニングに際しては、常に以下の規則を考慮してください。

WAN クラスタのマルチキャスト要件

ほとんどのデプロイメントでは、クラスタ化されたサーバを 1 つのサブネットに制限すると、マルチキャスト メッセージが確実に転送されます。ただし、特別なケースでは、WebLogic Server クラスタをワイド エリア ネットワーク（WAN）内の複数のサブネット間に分散することもできます。この方法は、クラスタ化されたデプロイメント内で冗長性を高めたり、地理的に広い範囲でクラスタを分散したりする場合に適しています。

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に必ず転送されるようにネットワーク トポロジをプランニングおよびコンフィグレーションしなければなりません。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。

- ネットワークは、IP マルチキャスト パケットの伝播を完全サポートする必要があります。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 レイテンシは、マルチキャスト メッセージが最終的な宛先に 200 ～ 300 ミリ秒で届くような小さな値でなければなりません。

- マルチキャスト生存時間（TTL）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 の設定方法については、6-27 ページの「マルチキャスト生存時間（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。各 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必要があります。こうした論理単位は、異なるセキュリティ ゾーン間で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、ハートビートが届かないために誤ってエラーを発生させるので、WebLogic Server クラスタを分裂させることがあります。

WebLogic Server クラスタ専用のマルチキャスト アドレスを使用する

複数の WebLogic Server クラスタが 1 つの IP マルチキャスト アドレスとポート番号を共有することは可能ですが、他のアプリケーションは同じアドレスに対してブロードキャストしたり、サブスクライブしたりしてはなりません。他のアプリケーションと 1 つのマルチキャスト アドレスを「共有」すると、クラスタ化されたサーバが不要なメッセージを生成することになり、システムのオーバーヘッドが発生します。

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 ストームが発生する場合

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切なタイミングで処理しない結果として、NAK メッセージやハートビートの再転送などを含めて、ネットワーク トラフィックが増大する可能性があります。ネットワーク上でマルチキャスト パケットが繰り返し転送される現象は*マルチキャスト ストーム*と呼ばれます。この現象により、ネットワークとそれに接続されたステーションの負荷が増加し、ひいては末端のステーションの停止または障害に至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大きくすると、通知の転送率と受信率が改善されることによってマルチキャスト ストームを防止できる可能性があります。

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切なタイミングで処理していないことが理由でマルチキャスト ストームが発生する場合、マルチキャスト バッファのサイズを大きくすることで対処できます。

UNIX の ndd ユーティリティを使用して、TCP/IP カーネル パラメータを調整することができます。`udp_max_buf` パラメータは、UDP ソケットの送信バッファおよび受信バッファのサイズをバイト単位で設定します。`udp_max_buf` の適切な値はデプロイメント環境によって異なります。マルチキャスト ストームが発生している場合、`udp_max_buf` の値を 32K ずつ大きくして調整の効果を確認してください。

`udp_max_buf` パラメータは必要な場合以外は変更しないでください。`udp_max_buf` パラメータを変更する場合は、事前に『*Solaris Tunable Parameters Reference Manual*』（http://docs.sun.com/?p=/doc/806-6779/6jfmsfr7o&）の「TCP/IP Tunable Parameters」の章の「UDP Parameters with Additional Cautions」に記されている Sun からの警告を確認してください。

IP ソケットを使用したピア ツー ピア通信

クラスタ化されたサーバ間の 1 対多通信ではマルチキャストを使用するのに対し、WebLogic Server インスタンスどうしのピア ツー ピア通信では IP ソケットを使用します。IP ソケットは、2 つのアプリケーション間でメッセージとデータを転送するための単純で高性能のメカニズムです。クラスタ内の WebLogic Server インスタンスは、以下の目的で IP ソケットを使用します。

- クラスタ内のリモート サーバ インスタンス上にあるクラスタ化されていないオブジェクトにアクセスします。

- 高可用性を実現するために、HTTP セッション ステートとステートフル セッション EJB のステートをプライマリ サーバとセカンダリ サーバとの間でレプリケートします。
- リモート サーバ インスタンスにあるクラスタ化されたオブジェクトにアクセスします（「推奨多層アーキテクチャ」で説明するように、一般には多層クラスタ アーキテクチャの場合のみ）。

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。

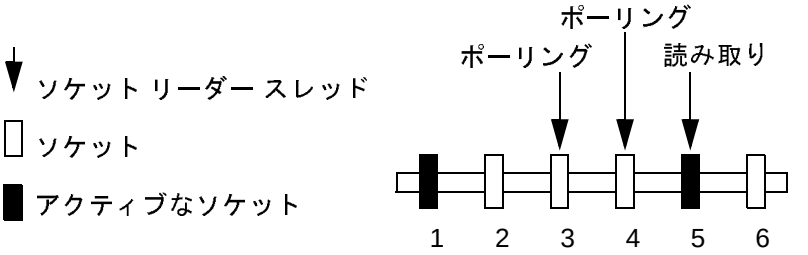
WebLogic Server クラスタのパフォーマンスは、ソケットのコンフィグレーションによって大きく左右されます。WebLogic Server でのソケット通信の効率は、次の 2 つの要因によって決まります。

- 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
- 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。

この問題は、サーバがソケット リーダー スレッドの数よりも多くのソケットをオープンしている場合により顕著になります。こうした場合、各リーダー スレッドは、ソケットが非アクティブであると判断するためにタイムアウト条件が満たされるまで待ちながら、複数のオープン ソケットをポーリングしなければなりません。タイムアウトが発生したら、以下に示すように、スレッドは待機中の別のソケットに移動します。



オープンしているソケットの数が使用可能なソケット リーダー スレッドの数を超えると、アクティブなソケットは、使用可能なソケット リーダー スレッドにポーリングされるまで、使用不可になる場合があります。

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。

ネイティブ ソケット リーダー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、6-25 ページの「サーバ インスタンスのホストであるマシンに対してネイティブ 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。

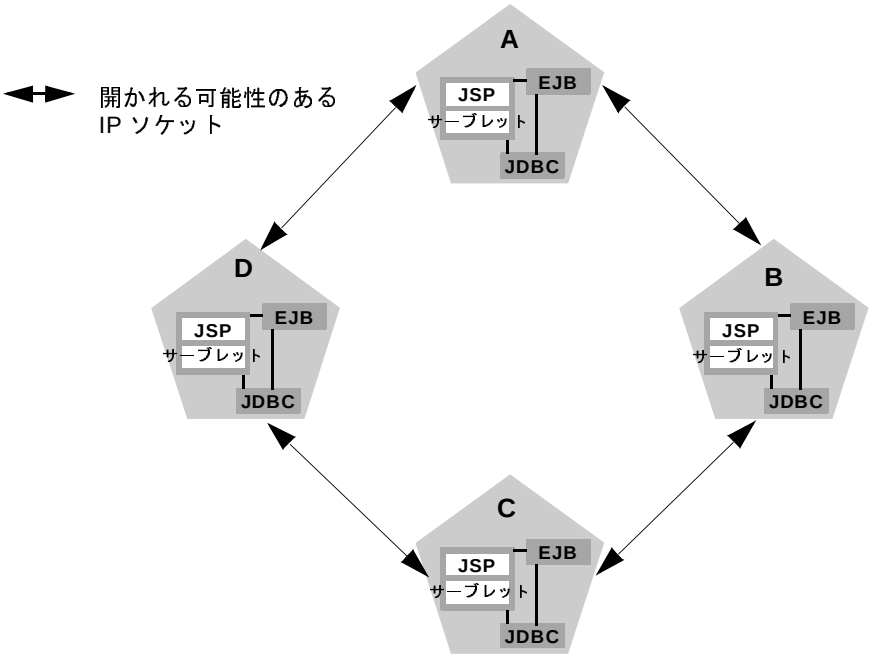
Java ソケット実装用のリーダー スレッドのコンフィグレーション

pure-Java ソケット リーダー実装を使用する場合は、適切な数のソケット 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。これにより、リーダー スレッドを複数のソケットで「共有」せずに済むので、ソケットのデータを直ちに読み取ることができます。

ソケットの使用数の確定

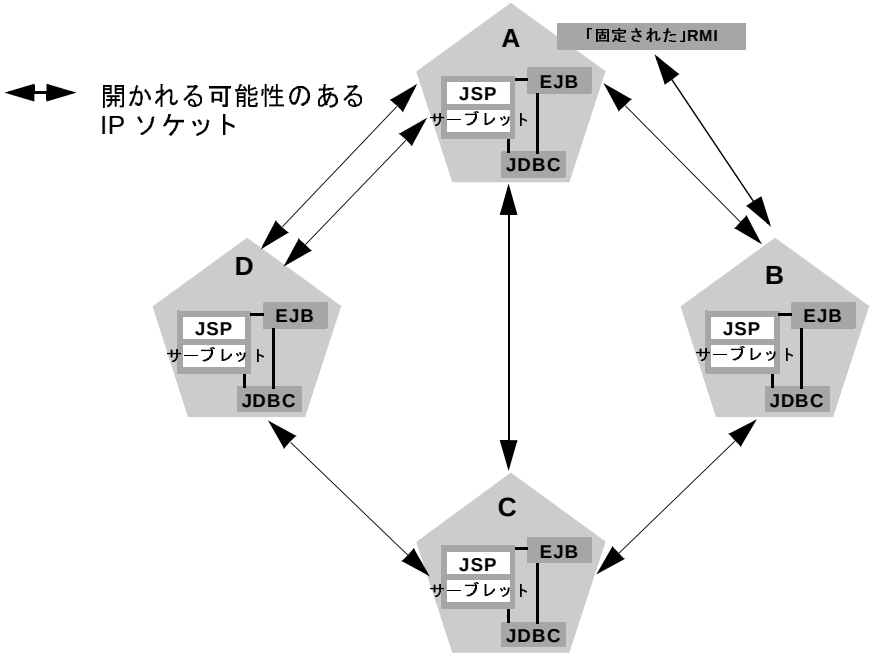
各 WebLogic Server インスタンスは、クラスタ内で自身を除くすべてのサーバ インスタンスに対してソケットをオープンする可能性があります。ただし、実際に使用されるソケットの最大数は、クラスタのコンフィグレーションによって決まります。実際には、クラスタ化されたサービスをデプロイする方法のため、クラスタ化されたシステムが他のサーバ インスタンスすべてに対してソケットをオープンすることはありません。

たとえば、クラスタがインメモリ HTTP セッション ステートのレプリケーションを使用し、クラスタ化されたオブジェクトだけをすべての WebLogic Server インスタンスにデプロイする場合、各サーバがオープンするソケットは、次に示すように最大で 2 つだけです。



上記の例の 2 つのソケットは、プライマリ サーバとセカンダリ サーバとの間で 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。

クラスタ化されていない RMI オブジェクトを特定のサーバに固定する場合、サーバ インスタンスが固定されたオブジェクト（リモート サーバが固定されたオブジェクトを実際にルック アップした場合にのみ解放されます）にアクセスするために別のソケットをオープンすることがあるので、ソケットの最大数は増える可能性があります。以下の図は、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。



この例では、各サーバは、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 上で固定された 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： 5-8 ページの「推奨多層アーキテクチャ」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。

リーダー スレッド数の設定方法については、6-26 ページの「サーバ インスタンスのホストである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。

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 バージョン 6.1 の Java クライアント アプリケーションは、ファイアウォール経由で接続した場合でも、以前の WebLogic Server バージョン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。ファイアウォール経由でクラスタに接続するバージョン 4.5 および 5.1 の Java クライアントが 1 つのソケットを利用していましたが、WebLogic Server バージョン 6.1 にはこうした制約がありません。(明示的に、または「固定された」オブジェクトにアクセスすることで) クライアントが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送信する場合、クライアントは各サーバに対して個々にソケットをオープンします。

最適なパフォーマンスを得るためには、クライアントを実行する JVM 内に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。詳細については、6-26 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。

注意： WebLogic Server バージョン 6.1 クラスタに接続するブラウザベースのクライアントとアプレットは、IP ソケットを 1 つだけ使用します。

クラスタワイドの JNDI ネーミング サービス

個々の WebLogic Server のクライアントは、JNDI 準拠のネーミング サービスを通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、サーバが提供するすべての公開サービスのリストが「ツリー」構造の形で含まれています。WebLogic Server が新しいサービスを提供するには、そのサービスを表す名前を JNDI ツリーにバインドします。クライアントがそのサービスを取得するには、そのサーバに接続し、バインドされたサービスの名前をルックアップします。

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 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。

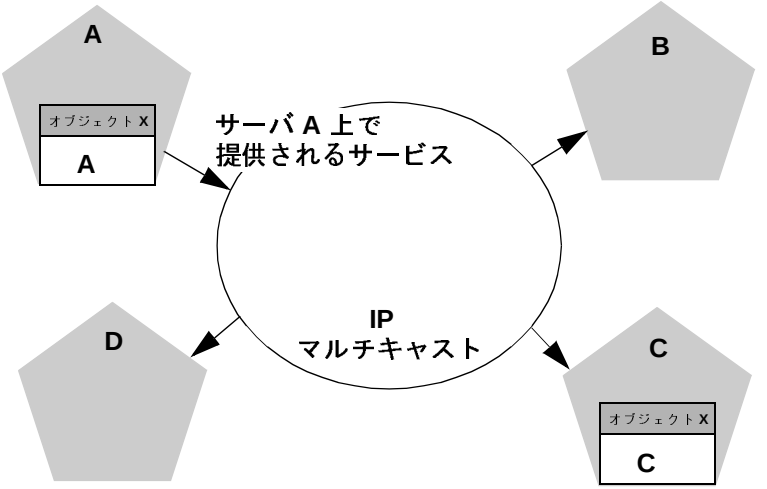
クラスタ内の各 WebLogic Server インスタンスは、クラスタワイドの論理 JNDI ツリーをローカルにコピーして保守します。クラスタワイドのネーミング ツリーの保守方法を理解しておくと、クラスタ環境で発生する可能性のある名前の衝突の原因をより的確に分析できます。

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の JNDI エントリをクラスタ内の他のサーバにレプリケートしますが、元のサーバ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。

クラスタワイドの JNDI ツリーの作成

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 ツリーを作成する場合は、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。サーバが起動する（または新規サービスが動作中のサーバに動的にデプロイされる）と、サーバはまず、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。

サーバ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 サーバが提供するサービスを追加したことを検出します。



上の図は、JNDI をバインドするプロセスを示しています。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 C の実装をバインド中です。

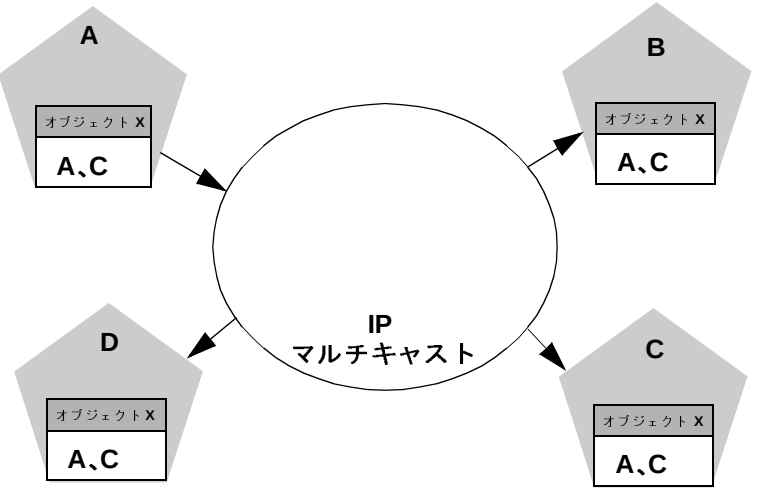
クラスタ内でマルチキャスト アドレスをリスンしている他のサーバは、サーバ A がクラスタ化されたオブジェクト X の新規サービスを提供し始めたことを検出します。これらのサーバは、自身のローカル JNDI ツリーを更新して、新規サービスをツリーに含めます。

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。サーバ B とサーバ D 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、これらのサーバは自身のローカル JNDI ツリーを更新して、これを反映させます。
- サーバがクラスタ対応サービスをすで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は、以下の図のように、サーバ A とサーバ C 上でオブジェクト X が使用可能であることを同じように示します。



注意： 実際のクラスタ システムでは、オブジェクト X は均一にデプロイされ、実装は 4 つのサーバすべてで使用可能になります。

JNDI 名の衝突の処理

単純な JNDI 名の衝突は、サーバが 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に固定された 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、各サーバでは、他のサーバ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 つのサーバのどちらかが終了するか、どちらかのサーバ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、両方のサーバが固定されたオブジェクトを同じ名前でデプロイしようとした場合にも発生します。

均一デプロイメント

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 インスタンスにデプロイする必要があります（均一デプロイメント）。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。

JNDI ツリーの更新

クラスタ化されたオブジェクトを削除（サーバからアンデプロイ）した場合、JNDI ツリーの更新は、新規サービスの追加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 WebLogic Server 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。すでに説明したように、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、「JNDI ツリーの更新」で説明したように、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。

（4-1 ページの「オブジェクトのクラスタ化について」では、EJB および RMI クラスのレプリカ対応スタブについて説明しています。）

クラスタ環境での WebLogic JNDI の実装方法と、独自のオブジェクトを JNDI クライアントから使用できるよう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。

クラスタ化されたサービスのロード バランシング

クラスタがスケーラブルになるには、各サーバを完全に利用できるようにする必要があります。これを実現する標準的な手法がロード バランシングです。ロード バランシングを支える基本的な考え方は、負荷をクラスタ内のすべてのサーバに均等に分散することで、各サーバがフル稼働できるというものです。ロード バランシングを行うには、単純ながらも十分な手法を利用します。クラスタ内のすべてのサーバが同じ能力を持ち、同じサービスを提供する場合は、サーバの知

識を必要としない非常に単純なアルゴリズムを使用することができます。サーバの能力やサーバがデプロイするサービスの種類が異なる場合、アルゴリズムはこれらの相違を考慮する必要があります。

HTTP セッション ステートのロード バランシング

サーブレットと JSP HTTP セッション ステートのロード バランシングは、別途ロード バランシング ハードウェアを使用するか、または WebLogic プロキシ プラグインに組み込まれているロード バランシング機能を使用して実現します。

Web サーバのバンクと WebLogic プロキシ プラグインを利用するクラスタの場合、プロキシ プラグインは、リクエストをクラスタ内のサーブレットと JSP に分散するために、ラウンドロビン アルゴリズムのみを提供します。このロード バランシング方法については、「ラウンドロビン（デフォルト）」で説明します。

ハードウェア 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、ハードウェアがサポート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バランシング方式を採用しているものもあります。

注意： 外部のロード バランサは、HTTP トラフィックを分散させることはできますが、EJB と RMI オブジェクトに対するロード バランシングの機能は備えていません。オブジェクト レベルのロード バランシングを行うには、現在の外部ロード バランサにはない特殊なアルゴリズムとサービスが必要です。WebLogic Server におけるオブジェクト レベルのロード バランシングについては、2-16 ページの「クラスタ化された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。

クラスタ化されたオブジェクトのロード バランシング

WebLogic Server クラスタは、クラスタ化されたオブジェクトをロード バランシングするための複数のアルゴリズムをサポートしています。選択した特定のアルゴリズムは、クラスタ化されたオブジェクトのレプリカ対応スタブ内で維持されます。クラスタ化されたオブジェクトのロード バランシングには、以下のアルゴリズムをコンフィグレーションできます。

- ラウンドロビン
- 重みベース
- ランダム

IIOP プロトコルを介してクライアント上で動作するオブジェクトに対しては、ロード バランシングはサポートされていません。詳細については、4-2 ページの「クラスタ化されたオブジェクトと RMI-IIOP クライアント」を参照してください。

ラウンドロビン（デフォルト）

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。ラウンドロビンは、WebLogic プロキシ プラグインが HTTP セッション ステートのクラスタ化に対して使用する唯一のロード バランシング方式です。

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 からなります。

このアルゴリズムの長所は、単純で軽く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

重みベースのアルゴリズムは、オブジェクトのクラスタ化に対してのみ適用されます。このアルゴリズムは、各サーバに対してあらかじめ割り当てておいた重みを考慮することで、ラウンド ロビン アルゴリズムを改良したものです。クラスタ内の各サーバには、WebLogic Server Administration Console の [クラスタの重み] フィールドを使って、1 ～ 100 の範囲で重みを割り当てます。これは、他のサーバと比較して、負荷をかける比率を宣言するものです。すべてのサーバがデフォルトの重み（100）または、同じ重みであれば、負荷の割合も同じになりま

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重みを決定するには十分な検討が必要です。サーバの重みは、以下の要因を考慮して割り当てます。

- 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（「固定された」）クラスタ化されていないオブジェクトの数

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。

ランダム

このアルゴリズムは、オブジェクトのクラスタ化にのみ適用されます。このアルゴリズムは、次のレプリカをランダムに選択します。結果的には、呼び出しはレプリカ間で均等に分散されます。各サーバが同じ能力を持ち、同じサービスのホストとなっているクラスタでのみ使用することをお勧めします。この方式のメリットは、単純で比較的軽いことです。主なデメリットは、各リクエストに対して乱数を生成するためのコストが多少生じることと、実行回数が少ない場合は負担が均一に分散しない可能性が若干あることです。

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

ロード バランシングをきめ細かく制御することも可能です。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 `CallRouter` クラスの作成の詳細については、付録 B「WebLogic クラスタの API」を参照してください。

ロード バランシングと JDBC 接続

JDBC 接続のロード バランシングを行うには、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 バランシングのサポートは、マルチプールをコンフィグレーションするときに選択できるオプションです。

マルチプールのロード バランシングを行うと、2-23 ページの「フェイルオーバと JDBC 接続」で説明する高可用性の動作が実現されることに加えて、マルチプール内の接続プール間で負荷が調整されます。マルチプールには、その中の接続プールの順番付きリストがあります。ロード バランシングを使用するようにマルチプールをコンフィグレーションしない場合、マルチプールは常に、リスト内の最初の接続プールから接続を取得しようと試みます。ロード バランシングされるマルチプールでは、その中の接続プールへのアクセスはラウンドロビン方式で行われます。クライアントからのその後のマルチプール接続リクエストのたびに、リストが循環し、選択された最初のプールがリストの最後尾に回ります。

JDBC 接続をクラスタ化する方法については、6-20 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。

ロード バランシングと JMS

クラスタ内の複数の管理対象サーバ間で JMS 送り先のロード バランシングを行うことができます。これは、複数の JMS サーバをコンフィグレーションし、ターゲットを使用してそれらを定義済みの WebLogic Server に割り当てることによって行います。各 JMS サーバはただ 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。

ロード バランシングは動的ではありません。コンフィグレーションの間、システム管理者は JMS サーバに対してターゲットを指定することによってロード バランシングを定義します。クラスタに対して JMS を設定する手順については、6-22 ページの「JMS 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ビスのフェイルオーバ サポート

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。この節では、WebLogic Server がクラスタ内の障害を検出する方法と、レプリケートされた HTTP セッション ステートとクラスタ化されたオブジェクトに対してフェイルオーバが機能する仕組みの概要について説明します。

WebLogic Server の障害検出方法

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
- 定期的なサーバ「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 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」を使って検出できます。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。各サーバ ハートビートには、メッセージの送信元のサーバをユニーク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。

クラスタ化されたサーブレットと JSP のフェイルオーバ

WebLogic プロキシ プラグインを持つ Web サーバを利用しているクラスタの場合、プロキシ プラグインはクライアントからは透過的にフェイルオーバを処理します。特定のサーバに障害が発生した場合、プラグインはセカンダリ サーバ上でレプリケート済みの HTTP セッション ステートを探して、クライアントのリクエストをそのまま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。

レプリケートされた HTTP セッション ステートのフェイルオーバ手順の詳細については、6-22 ページの「JMS をコンフィグレーションする」を参照してください。

クラスタ化されたオブジェクトのフェイルオーバ

クラスタ化されたオブジェクトの場合、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行されます。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。

IIOP プロトコルを介してクライアント上で動作するオブジェクトに対しては、フェイルオーバはサポートされていません。詳細については、4-2 ページの「クラスタ化されたオブジェクトと RMI-IIOP クライアント」を参照してください。

多重呼び出し不変オブジェクト

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、特に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。

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。いずれの例外もオブジェクトが変更されない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起こる危険性はありません。

フェイルオーバと JDBC 接続

JDBC はクライアントと DBMS の間の非常にステートフルなプロトコルであり、DBMS 接続とトランザクション状態が、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結び付けられます。この理由から、接続のフェイルオーバはサポートされていません。WebLogic Server インスタンスが停止すると、そのインスタンスによって管理されていた JDBC 接続もすべて切断され、DBMS は進行中であったすべてのトランザクションをロールバックします。影響を受けるアプリケーションは、その時点でのトランザクションを最初から再開しなければなりません。切断された接続と関連付けられていたすべての JDBC オブジェクトも機能を停止します。JDBC をクラスタ化すると、再接続のプロセスが容易になります。外部クライアント アプリケーション内の WebLogic データ ソースはクラスタに対応しているため、以前の接続のホストであったサーバ インスタンスが停止した場合、クライアントは WebLogic データ ソースから別の接続を要求することができます。

同期されるデータベース インスタンスを複製済みである場合、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実現できます。そのような環境では、プールが存在しないか、プールからのデータベース接続が機能していないことによってクライアントがプール内のある接続プールが接続を取得できない場合、WebLogic Server はプール リスト内の次の接続プールから接続の取得を試みます。

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すると、例外が生成され、WebLogic Server は別のプールからの接続取得を試みません。この問題には、接続プール内の接続の数を増やすことによって対処できます。

マルチプールに割り当てられるすべての接続プールは、予約時にその接続をテストするようにコンフィグレーションされる必要があります。これは、接続状態が良好であることをプールが検証するための唯一の方法であり、プール リスト上の次のプールへのフェイルオーバを行う時期をマルチプールが認識するための唯一の方法でもあります。

JDBC 接続をクラスタ化する方法については、6-20 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。

フェイルオーバと JMS

このリリースの WebLogic JMS では、自動フェイルオーバはサポートされていません。『管理者ガイド』の「WebLogic Server の障害からの回復」では、システム障害時に WebLogic Server インスタンスの再起動または入れ替えを行う方法について説明しています。また、手動でのフェイルオーバなどが行われる前に JMS アプリケーションを正しく終了させるためのプログラミング上の考慮事項についても示しています。

3 HTTP セッション ステートのレプリケーションについて

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について説明します。

- 概要
- 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
- レプリケーション グループの使用
- 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
- 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
- 障害後の遅延レプリケーション

概要

サーブレットと JSP の HTTP セッション ステートで自動フェイルオーバをサポートするために、WebLogic Server ではメモリ内のセッション ステート オブジェクトがレプリケートされます。このプロセスでは、クライアントが最初に接続する WebLogic Server でプライマリ セッション ステートが作成され、クラスタ内の別の WebLogic Server インスタンスでそのセッション ステートのセカンダリ レプリカが作成されます。レプリカは常に最新の状態に保たれるので、サーブレットのホスト サーバで障害が起きた場合に使用することができます。インスタンス間で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。インメモリ レプリケーションをセットアップする方法については、6-23 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」を参照してください。

注意： WebLogic Server では、ファイルベースまたは JDBC ベースの永続性を利用して、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。それらの永続性の詳細については、『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。

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には、以下のいずれかを使用して WebLogic Server クラスタにアクセスする必要があります。

- ロード バランシング ハードウェア
- WebLogic プロキシ 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複数の Web サーバ

プロキシの必要条件

WebLogic プロキシ プラグインでは、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が維持され、単純なラウンドロビン方式を使用して HTTP リクエストがそれらのインスタンスに転送されます。プロキシでは、WebLogic Server のインスタンスで障害が起き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のロジックも提供されます。

以下の Web サーバとプロキシ ソフトウェアの組み合わせが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

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は、以下の機能をサポートするハードウェアを使用する必要があります。

- 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
- SSL 永続性

パッシブなクッキーの永続性では、ロード バランサを介して WebLogic Server がクライアントにクッキー（レプリケートされた HTTP セッション ステートの位置情報を格納する）を書き込むことができます。ロード バランサでは、クライアントのクッキーにある識別子文字列を読み取って、クライアントとプライマリ WebLogic Server（HTTP セッション ステートのホスト）の関係を維持します。

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。

SSL 永続性を使用する場合、クライアントと WebLogic Server クラスタ間のデータの暗号化および復号化は、すべてロード バランシング製品が行います。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。

WebLogic Server でサポートされているロード バランシング ソリューションの設定については、「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（省略可能）」を参照してください。Alteon ロード バランシング製品の永続性メカニズムをコンフィグレーションする方法については、「クラスタに関する Alteon™ 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。BIG-IP ロード バランサをコンフィグレーションする方法については、「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。

セッション プログラミングの必要条件

クラスタ環境でデプロイするサーブレットまたは JSP を開発するときには、以下の事項に注意してください。

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サポートするには、すべてのサーブレットと JSP のセッション データがシリアライズ可能でなければなりません。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 でセッション 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、`HttpSession.setAttribute`（廃止された `putValue` に代わるもの）を使用して、セッション オブジェクトの属性を変更する必要があります。`setAttribute` でセッション オブジェクトの属性を設定すると、そのオブジェクトと属性は、インメモリ レプリケーションを使ってクラスタ内で複製されます。他の設定メソッドを使ってセッション内のオブジェクトを変更した場合は、WebLogic Server によってその変更が複製されることはありません。

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、`removeAttribute`（廃止された `removeValue` に代わるもの）を使用する必要があります。

注意： 非推奨の `putValue` メソッドおよび `removeValue` メソッドの使用も、セッション属性の複製を招きます。

セッション オブジェクトのシリアライゼーション オーバーヘッドに注意する

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。セッションで非常に大きなオブジェクトを作成する予定の場合は、あらかじめサーブレットをテストして、パフォーマンスが適切なレベルになるようにください。

フレームを使用するアプリケーションではセッションのアクセスを調整しなければならない

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、フレームを使用してセッション データにアクセスする場合のプランニングに十分な注意を払います。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します。
- 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します（たとえば、最初に訪れる HTML ページでセッションを作成します）。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 グループの使用

デフォルトの WebLogic Server では、プライマリ セッション ステートのホスト マシンとは別のマシンにセッション ステートのレプリカが作成されます。WebLogic Server では、レプリケーション グループを使用してセカンダリ ステートの配置場所をもっと詳しく指定できます。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優先的に使用されるクラスタ化されたインスタンスのリストです。

WebLogic Server Administration Console では、個々のサーバ 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。通常、マシン名はマルチホーム マシン上のサーバを表すために使用します。たとえば、同じマルチホーム マシン上、または同じサーバ 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。

マルチホームのマシンを使用しない場合、または単一のハードウェア上で複数の WebLogic Server を実行しない場合は、WebLogic Server のマシン名を指定する必要はありません。マシン名が関連付けられていないサーバは、自動的に、物理的に異なるハードウェア上にあるものとして扱われます。マシン名の設定の詳細については、「マシン名を定義する（省略可能）」を参照してください。

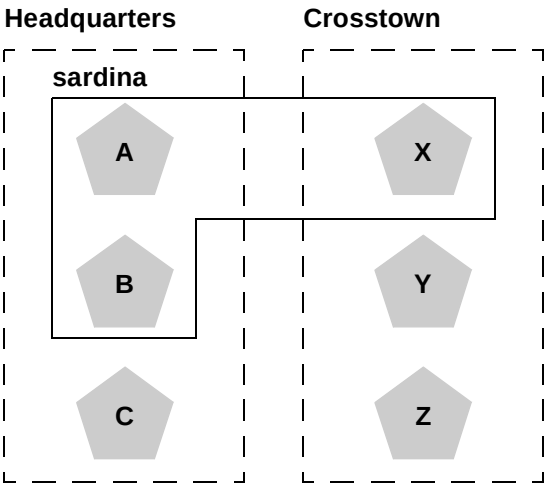
クラスタ化されたサーバ インスタンスをコンフィグレーションするときには、そのサーバで作成されたプライマリ HTTP セッション ステートのレプリカのホストとして検討される優先的なセカンダリ レプリケーション グループだけでなく、レプリケーション グループでのメンバシップも割り当てることができます。

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
3	はい	いいえ
4	いいえ	いいえ

プライマリ WebLogic Server では、前述のランキング ルールを使用してクラスタの他のメンバーがランク付けされ、セカンダリ セッション ステートのホストとなる最高ランクのサーバが選択されます。たとえば、次の図は地理的な区別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。



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 に接続して HTTP セッション ステートを作成する場合、そのステートのレプリカのホストとして最も有力なのはサーバ Y とサーバ Z です。なぜなら、それらが別のマシン上にあり、サーバ A の優先的なセカンダリ グループに属しているからです。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。

クラスタ化された WebLogic Server インスタンスのマシン名を定義する手順については、「マシン名を定義する（省略可能）」を参照してください。レプリケーション グループでのサーバのメンバ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的なセカンダリ 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、「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（省略可能）」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、WebLogic プロキシ プラグインを含むコンフィグレーションにおいて、クラスタ化されたサーブレットおよび JSP に WebLogic Server がどのようにしてアクセスするかについて説明します。

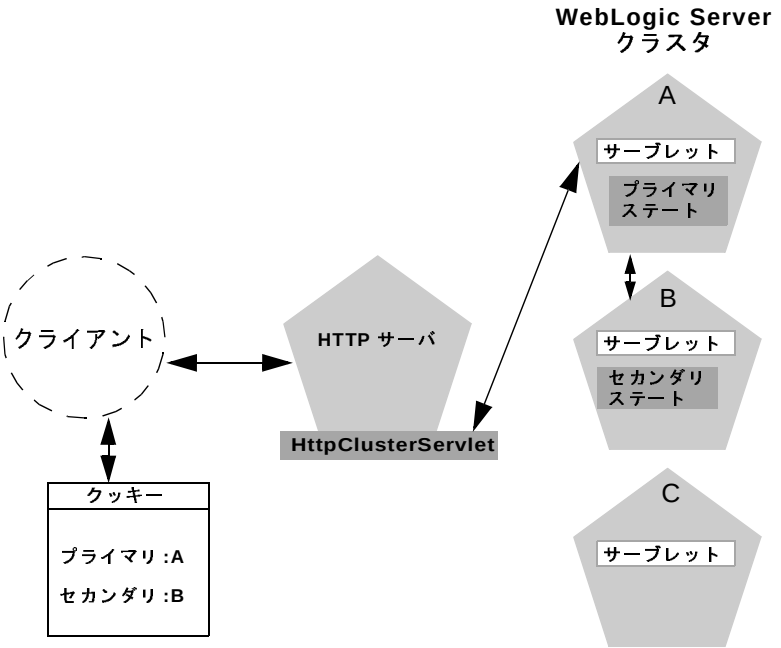
WebLogic プロキシ プラグインは、クラスタ化されたサーブレットまたは JSP のホストである WebLogic Server インスタンスのリストを管理し、ラウンドロビン方式を使用してそれらのインスタンスに HTTP リクエストを転送します。プラグインはまた、WebLogic Server インスタンスで障害が発生した場合に、クライアントの HTTP セッション ステートのレプリカの位置を特定するために必要なロジックも備えています。

WebLogic Server では、Web サーバとプロキシ プラグインの以下の組み合わせが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

プロキシのセットアップ手順については、『管理者ガイド』を参照してください。

次の図は、2 層クラスタ アーキテクチャ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。



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。

HTTP クライアントがサーブレットを要求すると、そのリクエストは `HttpClusterServlet` によって WebLogic Server クラスタに転送されます。`HttpClusterServlet` では、クラスタへのアクセスで使用するロード バランシング ロジックだけでなく、クラスタ内のすべてのサーバのリストが維持されます。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 書き換えを利用したセッション レプリカの追跡

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。詳細については、「セッション クッキーのコンフィグレーション」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 サーバで障害が起きると、`HttpClusterServlet` で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 の位置が確認されます。`HttpClusterServlet` では、クライアントの次の HTTP リクエストが自動的にセカンダリ サーバにリダイレクトされます。フェイルオーバは、クライアントには意識されません。

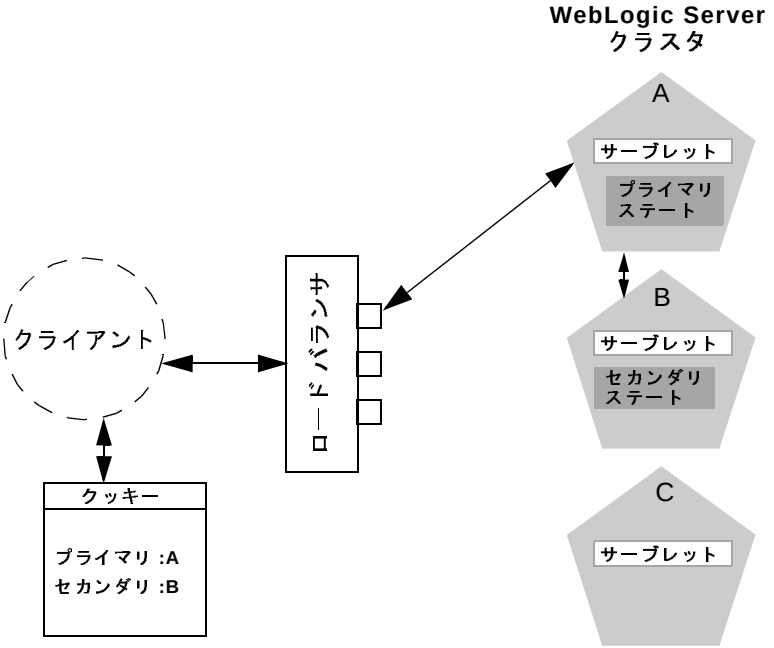
障害の後は、WebLogic Server B がサーブレット セッション ステートのプライマリ サーバになり、新しいセカンダリが作成されます（上の例ではサーバ C）。HTTP 応答では、今後のフェイルオーバに備えて、新しいプライマリ サーバとセカンダリ サーバを反映するためにプロキシによってクライアントのクッキーが更新されます。

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

この節では、ロード バランシング ハードウェアを含むコンフィグレーションにおいて、クラスタ化されたサーブレットおよび JSP に WebLogic Server がどのようにしてアクセスするかについて説明します。外部ロード バランサのセットアップ手順については、6-15 ページの「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（省略可能）」を参照してください。

ロード バランシング ハードウェアを通じたダイレクトなクライアント アクセスをサポートするために、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 バージョン 6.1 でも、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 サイド クッキーまたは URL 書き換えが継続して使用されます。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。以下の手順は、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 ハードウェアと共に使用する場合の接続とフェイルオーバのプロセスを示しています。



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。

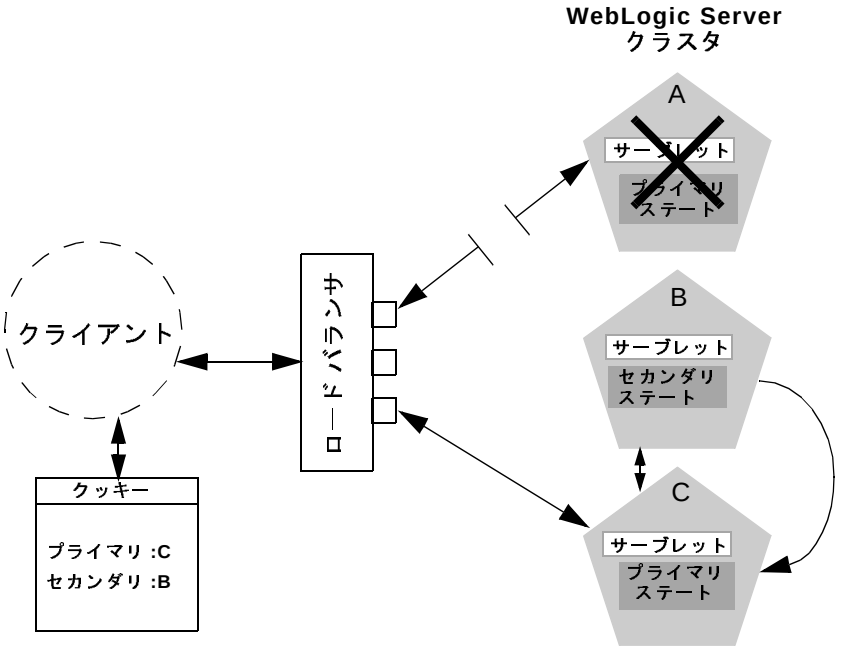
1. クライアントの接続要求は、ロード バランシング ハードウェアを通じて WebLogic Server クラスタに転送されます。ロード バランサは、コンフィグレーションされているポリシーを使用して、クライアントのリクエストを WebLogic Server A に転送します。
2. WebLogic Server A は、クライアントのサーブレット セッション ステートのプライマリ ホストとして機能します。プライマリ ホストは、「レプリケーション グループの使用」で説明されているランキング システムを使用して、セッション ステートのレプリカのホストとなるサーバを選択します。上の例では、WebLogic 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 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： クッキーが無効のクライアントをサポートするには、WebLogic Server の URL 書き換え機能を有効にする必要があります。詳細については、「URL 書き換えの使用」を参照してください。

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サイドのクッキーにある識別子を利用して、リクエストが引き続き WebLogic Server A 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 に障害が起きると、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。



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。
3. 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。

障害後の遅延レプリケーション

プロキシとロード バランシングの例はどちらも、WebLogic Server のセッション ステート レプリケーションの重要な性質を示しています。セッション ステートのホストであるサーバで障害が発生した後、そのサーバのセッション ステートのレプリケーションは、クラスタ内の別のサーバ上にあるセッション ステートへのアクセスをクライアントが試みるのに呼応して遅延方式で実行されます。

たとえば、3-15 ページの「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」で説明したシナリオでは、サーバ A で障害が発生しても、クラスタで直ちにサーバ C 上にセッション ステートの新しいコピーが生成されるわけではありません。実際には、クライアントがサーバ C に接続してセッション ステートの使用を試みた時点ではじめて、セッション ステートがサーバ C にコピーされます（クライアントがサーバ C ではなくサーバ B に再接続した場合、クラスタはやはりセッション ステートをサーバ C にコピーして、アクティブなセッション ステートのコピーがクラスタ内に少なくとも 1 つは存在することを保証します）。

障害後の遅延レプリケーションには、2 つの重要な利点があります。

- サーバでの障害発生直後のレプリケーション要求によってクラスタが過負荷状態になることを防ぎます。
- クライアントに使用されることのない無用なセッション ステートのレプリケーションを防ぎます。

遅延レプリケーション動作には、クラスタに WebLogic Server インスタンスが 2 つしかない特殊な状況でレプリケーションが失敗するわずかなリスクが伴います。次のシナリオは、サーバ A とサーバ B の 2 つのノードで構成されるクラスタで起こりうるレプリケーション失敗の例を示します。

1. クライアントがサーバ A にアクセスし、新しいセッション ステートを作成します。クラスタはセッション ステートをサーバ B にレプリケートします。
2. サーバ A で障害が発生し、サーバ B にはセッション ステートのレプリカが引き続き保持されています。
3. クライアントはクラスタへの接続を再確立し、サーバ B にアクセスして、レプリケートされたセッション ステートを取得します。この時点でサーバ B はクラスタ内の唯一のインスタンスであるため、ローカル セッション ステートがプライマリ セッション ステートとして認識され、レプリカの作成ができなくなります。
4. サーバ A が再起動されますが、遅延レプリケーションの動作が原因で、サーバ A はセッション ステートのレプリカを保持しません。

   この時点の状態では、サーバ B に障害が発生するとクライアントはセッション ステートを失います。

このような状況でセッション ステートが失われる危険性を回避するためには、3 つ以上の WebLogic Server インスタンスを使用してクラスタを構成するようにします。

4　オブジェクトのクラスタ化について

以下の節では、オブジェクトのクラスタ化について説明します。

- 概要
- レプリカ対応スタブ
- クラスタ化された EJB
- クラスタ化された RMI オブジェクト
- ステートフル セッション Bean のレプリケーション
- 連結されたオブジェクトの最適化
- オブジェクト デプロイメントの必要条件

概要

オブジェクトをクラスタ化すると、その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` のコマンドライン 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 コンパイラ」を参照してください。

レプリカ対応スタブ

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。「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。

クラスタ化されたオブジェクトと RMI-IIOP クライアント

IIOP プロトコルを介して動作する RMI オブジェクトに対するクラスタ化のサポートは、サーバサイド オブジェクトに限られます。JDK ORB を使用するクライアントは、WebLogic のクラスにアクセスできず、ロード バランシングやフェイルオーバなど、WebLogic 固有の機能を利用することはできません。IIOP を介して動作するクラスタ化されたオブジェクトに対するロード バランシングとフェイルオーバは、オブジェクトが WebLogic Server の実行時環境内で動作している場合にのみサポートされます。クライアントサイドの RMI オブジェクトに対してロード バランシングとフェイルオーバを行うには、T3 プロトコルを使用する必要があります。

クラスタ化された EJB

EJB は、2 つの異なるレプリカ対応スタブ（EJBHome インタフェースのスタブと EJBObject インタフェースのスタブ）を生成できるという点で、単純な RMI オブジェクトとは異なります。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、さまざまな EJB の機能を概説します。さまざまな EJB タイプでのクラスタ化の動作に関する詳細については、「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。詳細については、ステートフル セッション Bean のレプリケーションを参照してください。

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。

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。

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 EJB を使用する方法の詳細については、「WebLogic Server EJB コンテナ」を参照してください。

エンティティ Bean と EJB ハンドルに対するフェイルオーバ

エンティティ Bean および EJB ハンドルに対する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ではないサーバ インスタンスにマップされてはなりません。

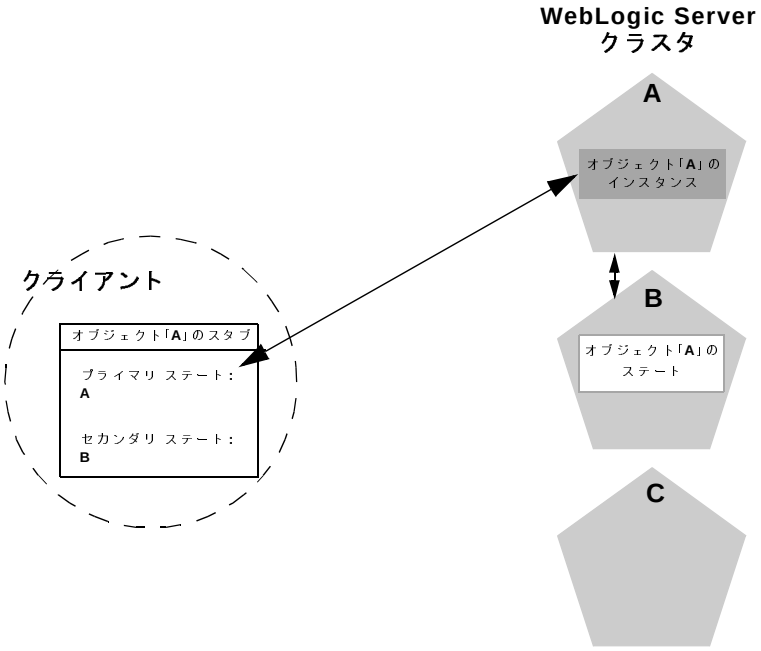
クラスタ化された RMI オブジェクト

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。WebLogic Server クラスタで RMI を使用する方法の詳細については、『WebLogic RMI プログラマーズ ガイド』を参照してください。

ステートフル セッション Bean のレプリケーション

WebLogic Server では、HTTP セッション ステートの場合と同じようにステートフル セッション EJB のステートがレプリケートされます。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、「HTTP セッション ステートのレプリケーションについて」で定義されている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。



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 ステートの変更のレプリケート

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。

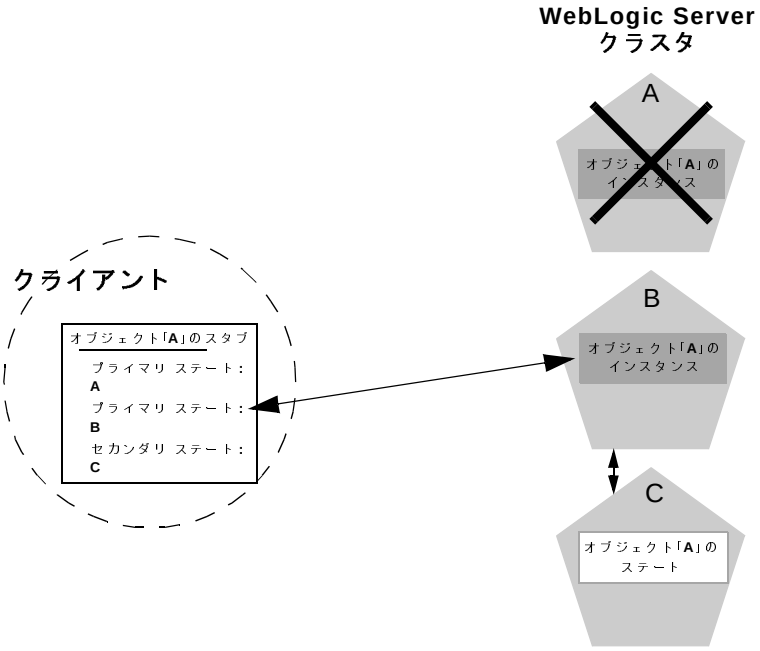
注意： EJB 仕様で説明されているように、ステートフル EJB の実際のステートはトランザクション非対応です。可能性は低いですが、EJB の現在のステートが失われることもあり得ます。たとえば、EJB の関与しているトランザクションがクライアントによってコミットされ、ステートの変更がレプリケートされる前にプライマリ サーバで障害が起きると、クライアントは以前に格納されていた EJB のステートにフェイルオーバされます。

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

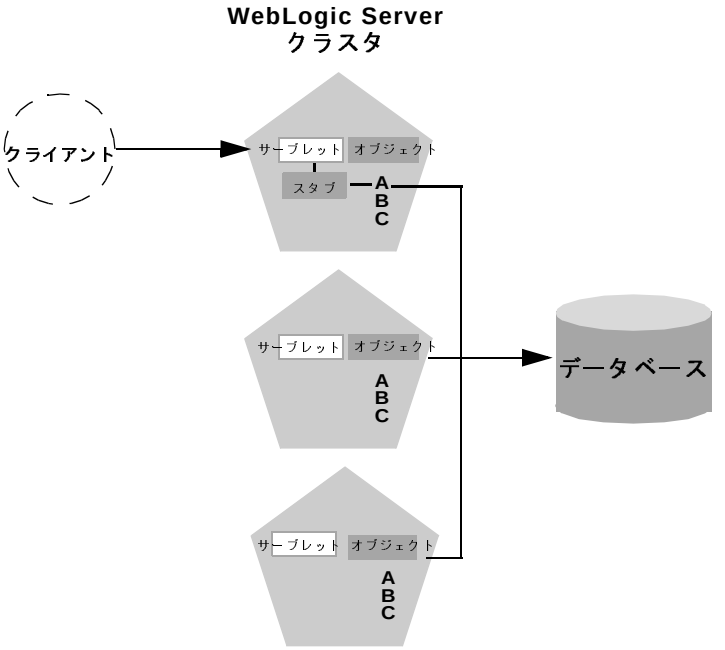
プライマリ サーバで障害が起きると、以降のリクエストはクライアントの EJB スタブによって自動的にセカンダリ WebLogic Server インスタンスにリダイレクトされます。この時点で、レプリケートされたステート データを使用してセカンダリ サーバで新しい EJB インスタンスが作成され、セカンダリ サーバで処理が継続されます。

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。



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が呼び出されるときに、WebLogic Server によって常にロード バランシングが実行されるわけではありません。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。次の図は、そのような状況を詳しく説明しています。



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。

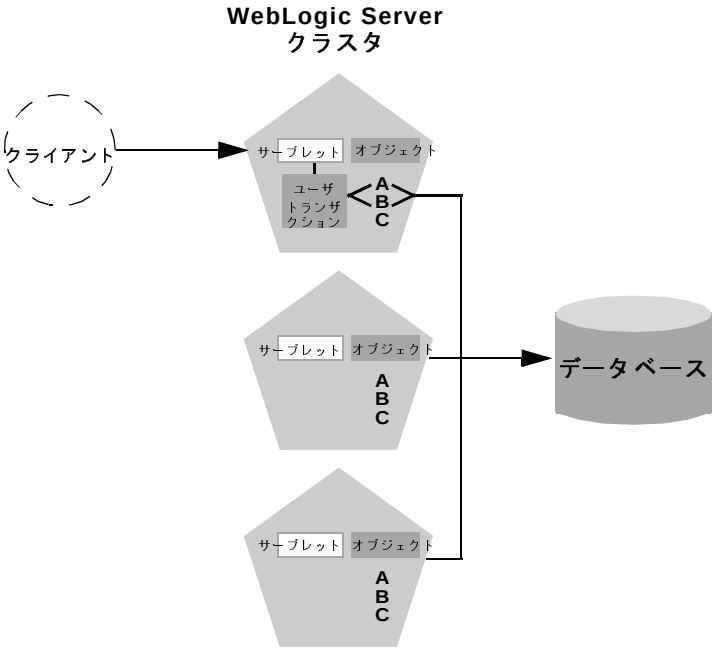
WebLogic Server では、クラスタ内のオブジェクト A の他のレプリカにクライアントの呼び出しを分散しないで、常に、ローカルにある連結されたオブジェクト A のコピーを使用します。クラスタ内の他のサーバとのピア接続を確立するネットワーク オーバーヘッドが避けられるので、ローカル コピーを使用した方が効率的です。

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。クラスタが 1 つの Web アーキテクチャでは、この最適化によってレプリカ対応スタブに固有のロード バランシング ロジックが無効になります。

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、「WebLogic Server クラスタのプランニング」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に関わっているクラスタ化されたオブジェクトも連結されます。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。



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、「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。

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、クラスタ内の各種の EJB の機能について、この章の指示と「WebLogic Server EJB コンテナ」を参照してください。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。weblogic-ejb-jar.xml デプロイメント記述子は、クラスタ化に対応する XML デプロイメント要素を示します。

EJB またはカスタム RMI オブジェクトを開発する場合は、クラスタ化されたオブジェクトの実装を JNDI ツリーでバインディングする方法について「クラスタ環境での WebLogic JNDI の使い方」も参照してください。

5 WebLogic Server クラスタのプランニング

以下の節では、1 つまたは複数の WebLogic Server クラスタをデプロイする前に考慮すべき一般的な問題について説明します。

- 概要
- 推奨基本クラスタ
- アプリケーション層を分割したプランニング
- 推奨多層アーキテクチャ
- 推奨プロキシ アーキテクチャ
- 管理サーバに関する考慮事項
- クラスタ アーキテクチャのセキュリティ オプション
- クラスタに関するファイアウォールの考慮事項

概要

「WebLogic Server のクラスタ化の概要」および「WebLogic クラスタの管理」の各章も併せて読み、WebLogic Server クラスタの操作方法についてよく理解してください。

この節では、WebLogic Server 向けの推奨クラスタ アーキテクチャについても説明します。各推奨アーキテクチャをよく検討した上で、Web アプリケーションに最適なコンフィグレーションを決定してください。

キャパシティ プランニング

この節は、クラスタ化されたシステムのネットワーク トポロジのプランニングに重点を置いています。Web アプリケーションのロード バランシング機能およびフェイルオーバ機能を最大限に利用できるよう、ここでは、1 つまたは複数の WebLogic Server クラスタを、ロード バランサ、ファイアウォール、および Web サーバに関連付けてプランニングする方法について説明します。この種類のプランニングはクラスタ システムのキャパシティに直接影響しますが、ここでは従来のキャパシティ プランニングについては説明しません。クラスタ システムのレイアウトを決定したら、クライアントによる頻繁な使用をシミュレートするソフトウェア（Mercury Interactive の LoadRunner など）を使用して綿密なテストを実行する必要があります。負荷のかかった状況でシステムをテストすることにより、実際の環境でのクライアント負荷に対応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のある場所を特定できます。

マルチ CPU マシン上の WebLogic Server

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 サーバは、大規模なクラスタまたは複数のクラスタのホストとなることができます。

ほとんどの場合、WebLogic Server クラスタは、1 つの WebLogic Server インスタンスを 2 つの CPU ごとにデプロイするのが最適です。ただし、すべてのキャパシティ プランニングと同じように、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、「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。

用語の定義

この節では、クラスタ化されたシステムの各部分の説明に以下の用語を使用します。

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を示します。すべての Web アプリケーションが同じような「層」に分けられるとは限りません。そのため、アプリケーションによっては、以下で説明する層の一部しか利用されない場合もあります。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。

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。

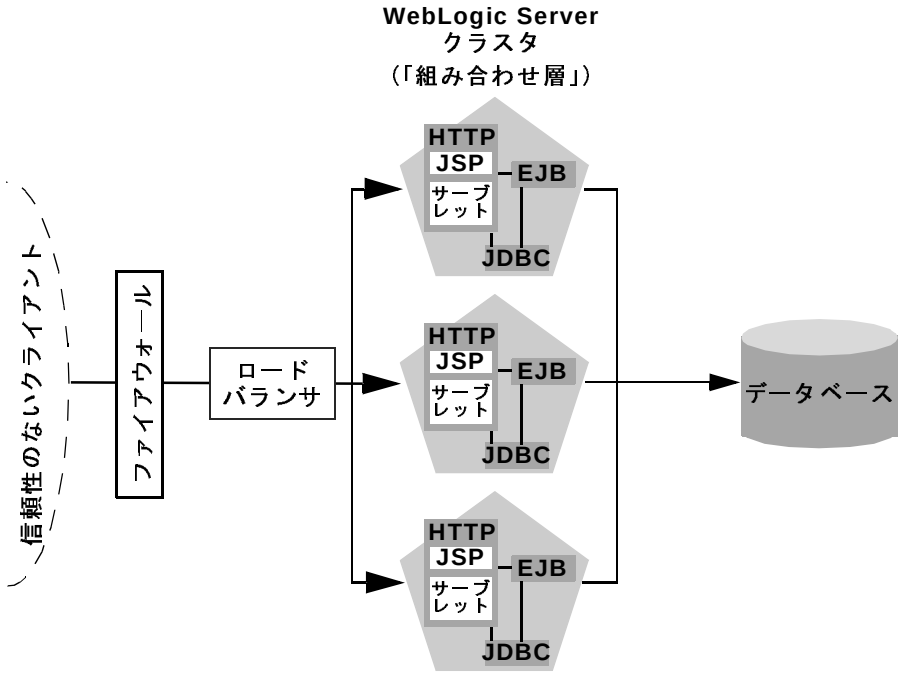
ロード バランサには、クライアント接続をクラスタ内の特定のサーバに関連付ける機能があります。この機能は、クライアント セッション情報のインメモリ レプリケーションを使用する場合に必要です。一部のロード バランシング製品では、インメモリ レプリケーションで使用されるプライマリ サーバおよびセカンダリ サーバを追跡する WebLogic Server クッキーを上書きしないように、クッキーの永続性メカニズムをコンフィグレーションする必要があります。詳細については、ロード バランシング ハードウェアをコンフィグレーションする（省略可能）を参照してください。

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。

推奨基本クラスタ

推奨基本クラスタ アーキテクチャでは、すべての Web アプリケーション層を組み合わせ、関連するサービス（静的 HTTP、プレゼンテーション ロジック、およびオブジェクト）を WebLogic Server インスタンスの 1 つのクラスタに配置します。次の図に、このアーキテクチャを示します。



基本アーキテクチャには、以下のような利点があります。

- **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、Administration Console 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。
- **最適のパフォーマンス：**組み合わせ層アーキテクチャは、サーブレットまたは JSP のほとんどまたはすべてがプレゼンテーション層にあって、それらが EJB または JDBC オブジェクトなどオブジェクト層にあるオブジェクトにアクセスするようなアプリケーションに対して、最高の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、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（省略可能）を参照してください。

アプリケーション層を分割したプランニング

基本クラスタ アーキテクチャでは、Web アプリケーションのすべての層（Web 層、プレゼンテーション層、およびオブジェクト層）を提供する WebLogic Server インスタンスの 1 つのクラスタが使用されます。この「組み合わせ層」クラスタでは、信頼性のない接続（HTTP および Java クライアント）の WebLogic Server クラスタへのインタフェースはロード バランシング ハードウェアを経由する 1 つだけです。基本アーキテクチャは簡略化されていますが、数多くの Web アプリケーションのニーズを満たしています。

しかし、クラスタ化された Web アプリケーションの 2 つの主要な機能（ロード バランシング機能とフェイルオーバ機能）は、Web アプリケーション層間のインタフェースにしか導入できません。基本クラスタ アーキテクチャのように、1 つのハードウェアとソフトウェアのプラットフォームに Web アプリケーション層が組み合わされている場合は、ロード バランシング機能およびフェイルオーバ機能をシステムに導入する機会が少なくなります。

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、基本クラスタ アーキテクチャでも、ほとんどの Web アプリケーションのクラスタ化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、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。これについては、以下の「推奨多層アーキテクチャ」で説明します。

推奨多層アーキテクチャ

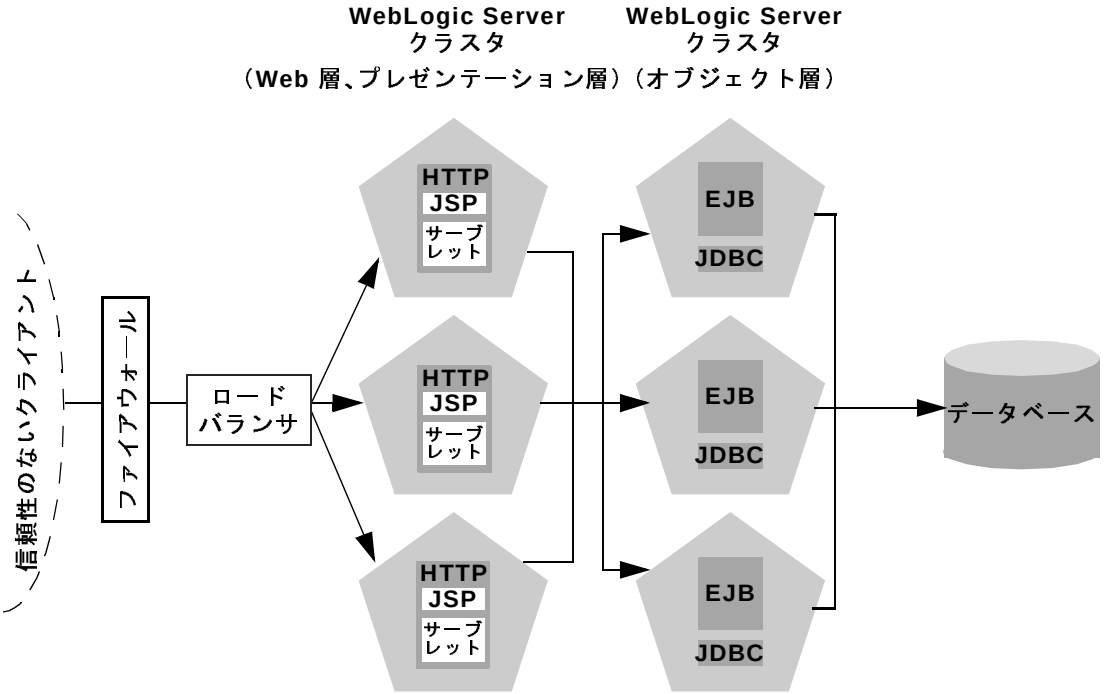
推奨多層アーキテクチャでは、2 つの異なる WebLogic Server クラスタを使用します。1 つは静的な HTTP コンテンツを提供するクラスタであり、もう 1 つはクラスタ化された EJB を提供するクラスタです。多層クラスタは、以下のような Web アプリケーションにお勧めします。

- 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には、プレゼンテーション層からオブジェクト層の呼び出し頻度を考慮します。通常、プレゼンテーション オブジェクトがオブジェクト層を呼び出す場合には、組み合わせ層アーキテクチャの方が多層アーキテクチャに比べてパフォーマンスが良くなります。

次の図に、推奨多層アーキテクチャを示します。



ハードウェアとソフトウェアの物理レイヤ

高度な推奨コンフィグレーションには、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。物理レイヤはアプリケーション自身の論理層（Web 層、プレゼンテーション層、およびオブジェクト層）を構成します。

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（「オブジェクトのクラスタ化について」参照）が失われます。ただし、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、推奨基本クラスタにある利点に加えて、以下のような利点があります。

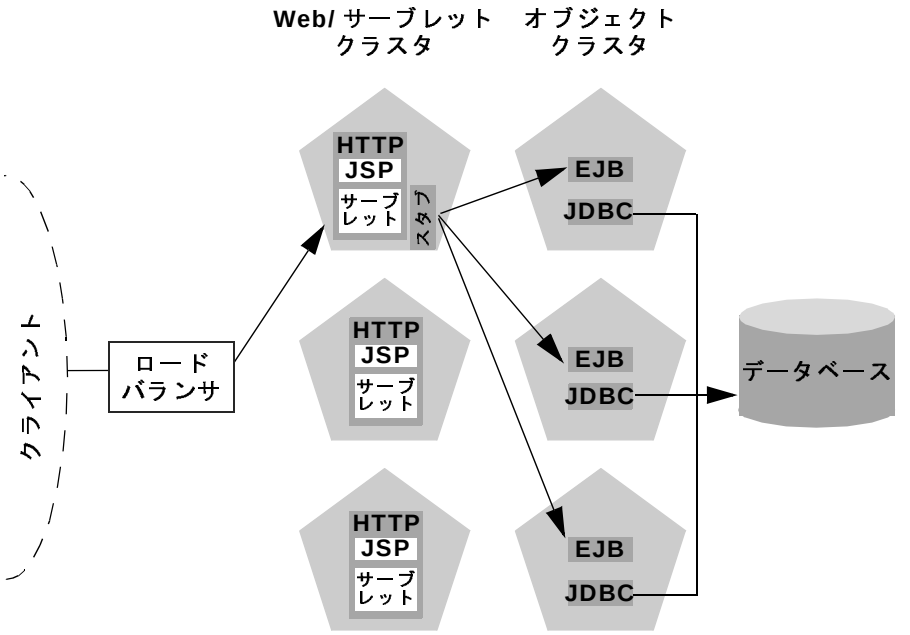
- **EJB メソッドのロード バランシング :** サーブレットと EJB のホストを異なるクラスタに配置することで、サーブレットの EJB へのメソッド呼び出しで、複数のサーバにわたるロード バランシングが可能になります。これについては、以下の「クラスタ化されたオブジェクト呼び出しに対するロード バランシング」で説明します。

- **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、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。

クラスタ化されたオブジェクト呼び出しに対するロード バランシング

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、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。



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-ejb-jar.xml デプロイメント記述子」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（「推奨基本クラスタ」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、「推奨多層アーキテクチャ」で示された図では、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、「Java ソケット実装用のリーダー スレッドのコンフィグレーション」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、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（省略可能）を参照してください。

多層アーキテクチャに関する制限

高度なコンフィグレーションでは、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、Web アプリケーションで「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。

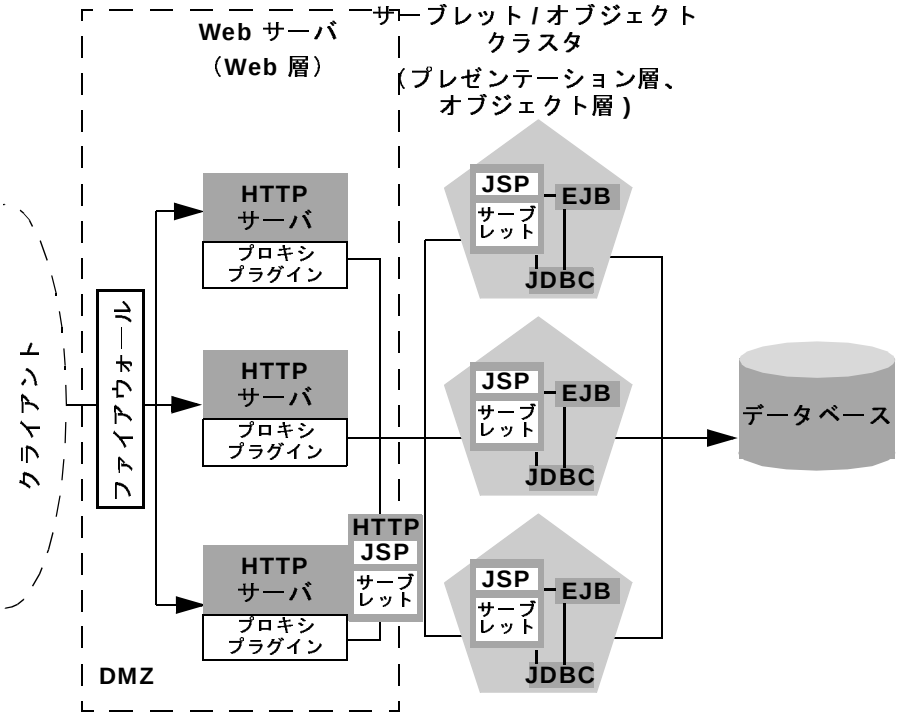
WebLogic Server インスタンスの内部 DNS 名と外部 DNS 名が同一でない場合、サーバ インスタンスの `ExternalDNSName` 属性を使用して、サーバの外部 DNS 名を定義します。ファイアウォールの外側では、`ExternalDNSName` がサーバの外部 IP アドレスに変換されます。Administration Console の [サーバ | コンフィグレーション | 一般] タブを使用してこの属性を設定します。『Administration Console オンライン ヘルプ』の「[サーバ] --> [コンフィグレーション] --> [一般]」を参照してください。

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もできます。この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。

2 層プロキシ アーキテクチャ

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推奨基本クラスタとほぼ同じです。



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。

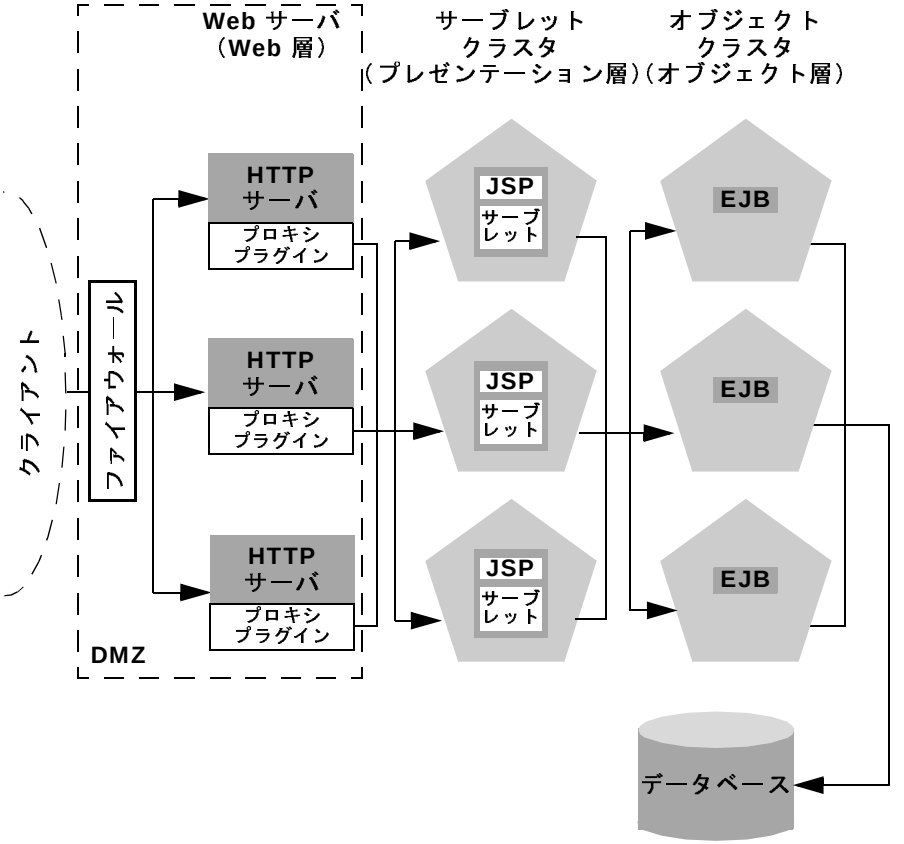
サーブレット / オブジェクト レイヤ

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、マルチホーム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（「推奨基本クラスタ」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。次の図に、このアーキテクチャを示します。



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 アーキテクチャにおけるトレードオフ

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。

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は、ロード バランシング アルゴリズムが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。

管理サーバに関する考慮事項

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、各サーバは、クラスタ自身のコンフィグレーション情報を格納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。管理サーバにある `config.xml` ファイルには、管理サーバのドメイン内のすべてのクラスタ化されたインスタンス（およびその他の管理されたインスタンス）のリポジトリが 1 つあります。WebLogic Server の以前のバージョンのように、クラスタ内のサーバごとに異なるコンフィグレーション ファイルを作成しないでください。

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ります。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ことの利点はありません。管理オブジェクトはクラスタ化できず、管理サーバに障害が発生した場合に別のクラスタ メンバーへのフェイルオーバを行いません。管理サーバ上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する管理機能の安定性が損なわれるおそれがあります。管理サーバ上にデプロイしたアプリケーションが予期しない動作をすると、管理サーバの稼働の妨げとなることがあります。

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ます。

管理サーバの障害時に発生すること

あるドメインの管理サーバの障害は、そのドメイン内の管理対象サーバの処理には影響しません。ドメインの管理サーバが管理するサーバ インスタンスが（クラスタ化されていてもいなくても）起動されて実行中の間は、その管理サーバが使用不能になっても、その管理対象サーバは実行を続けます。そのドメインにクラスタ化されたサーバ インスタンスが含まれている場合、管理サーバが障害となった場合でも、そのドメインのコンフィグレーションでサポートするロードバランシングおよびフェイルオーバ機能はそのまま有効です。

注意： 管理サーバの障害の理由が、ホスト マシン上で発生したハードウェアまたはソフトウェアの障害ならば、同じマシン上のほかのサーバ インスタンスも同様の影響を受けるでしょう。しかし、1 つの管理サーバの生涯それ自体は、そのドメイン内の管理対象サーバの処理を中断することはありません。

管理サーバの再起動の方法については、『管理者ガイド』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。

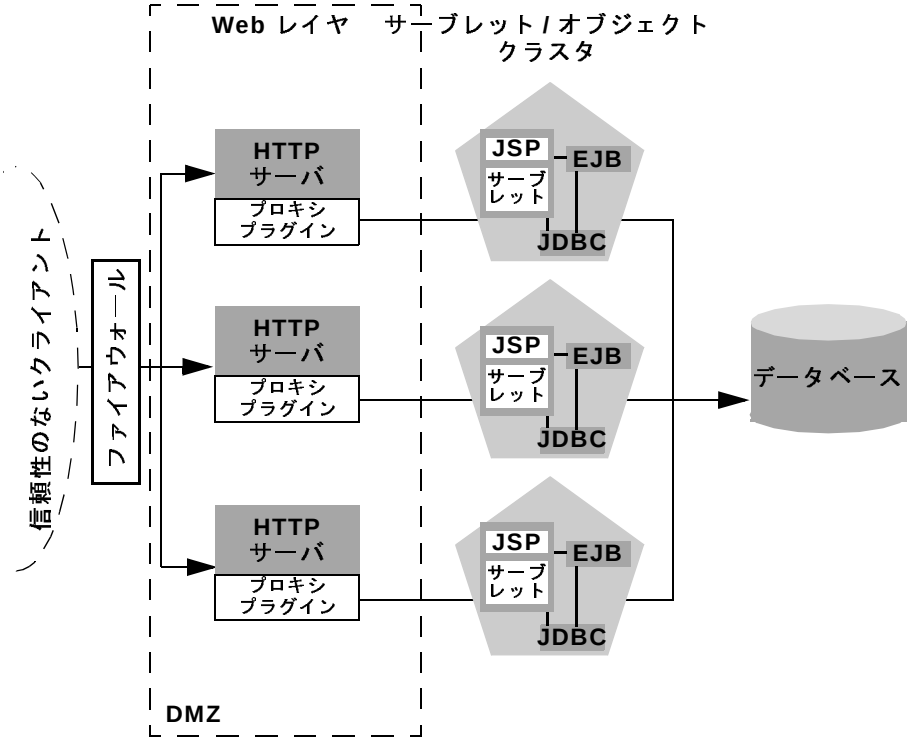
クラスタ アーキテクチャのセキュリティ オプション

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、組み合わせ層クラスタ アーキテクチャでも、多層クラスタ アーキテクチャでも使用できます。



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、「クラスタに関するファイアウォールの考慮事項」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 名が同一でない場合、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側では、`ExternalDNSName` がサーバの外部 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] タブを使用して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。

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。

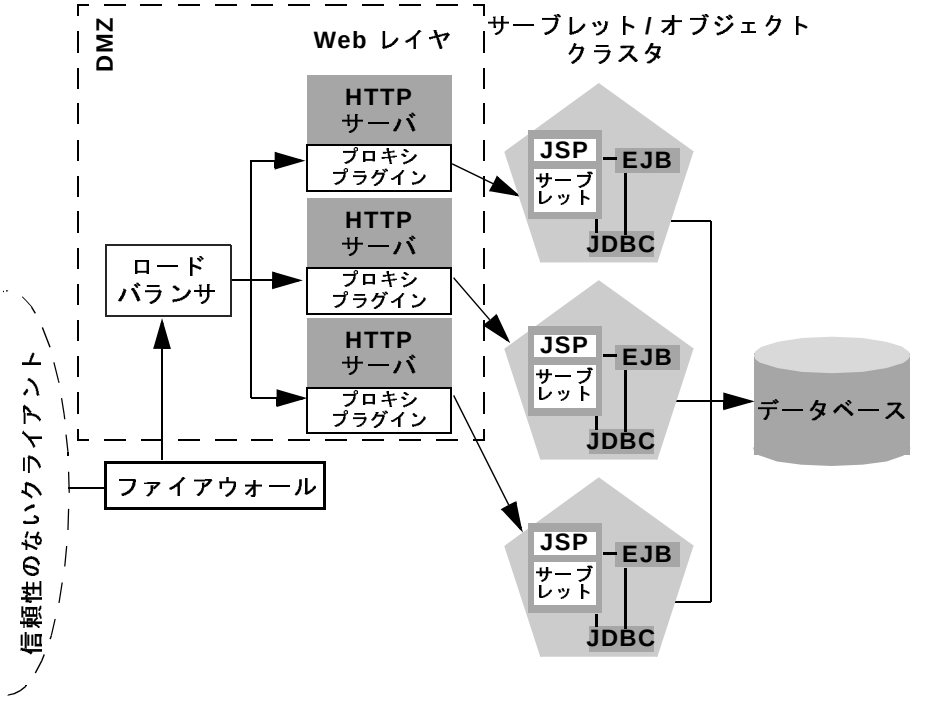
たとえば、ハッカーが Web サーバのホストになっているマシンの 1 つにアクセスしたと仮定します。アクセスのレベルによっては、動的なコンテンツを求めて Web サーバがアクセスする、プロキシされたサーバに関する情報を、ハッカーが入手できる場合もあります。

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（「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 バランサの組み合わせ

推奨クラスタ コンフィグレーションでロード 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。

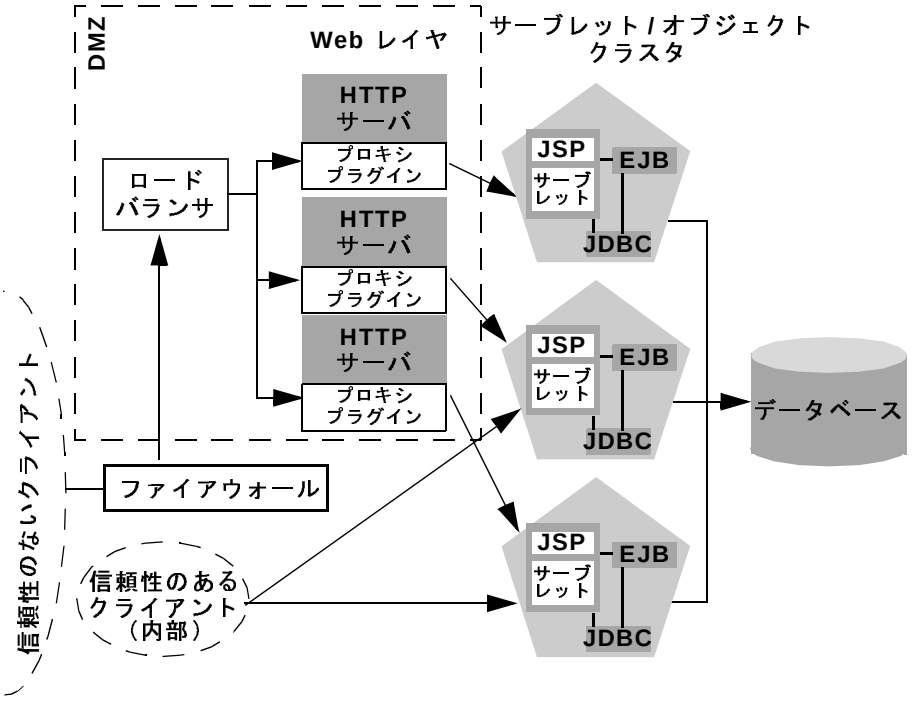


上記の設定では、Web 層を含む DMZ 内にロード バランサが配置されています。このコンフィグレーションでファイアウォールを使用すると、ファイアウォールはロード バランサへのアクセスを制限するだけで済むので、セキュリティ ポリシーの管理を簡素化できます。また、この設定では、以下で説明するような、Web アプリケーションにアクセスする内部クライアントをサポートするサイトの管理を簡素化することもできます。

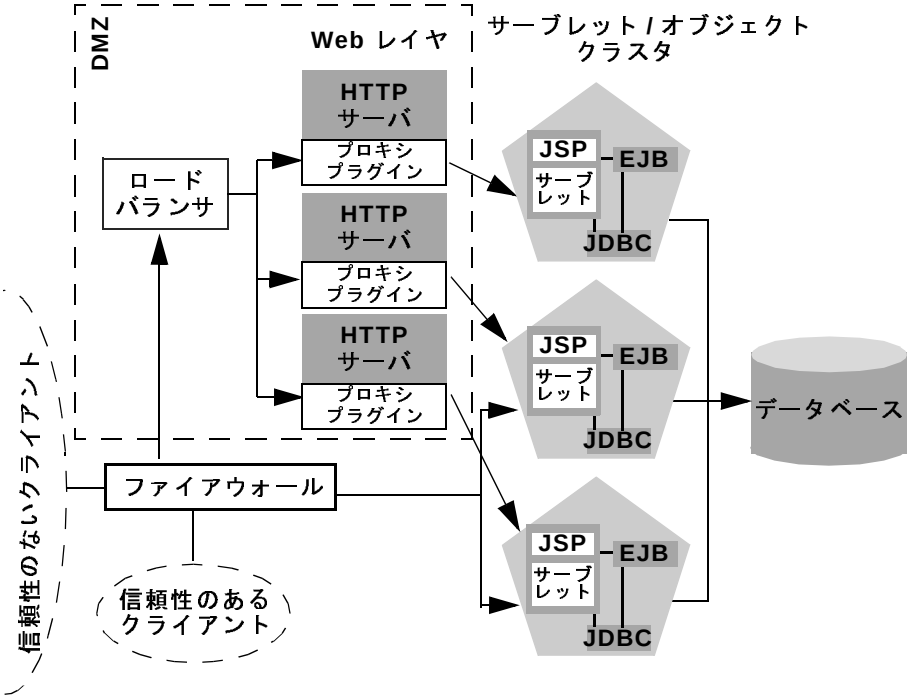
内部クライアントに対するファイアウォールの拡張

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。



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、次の図のように、プレゼンテーション層のホストになっている WebLogic Server 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。



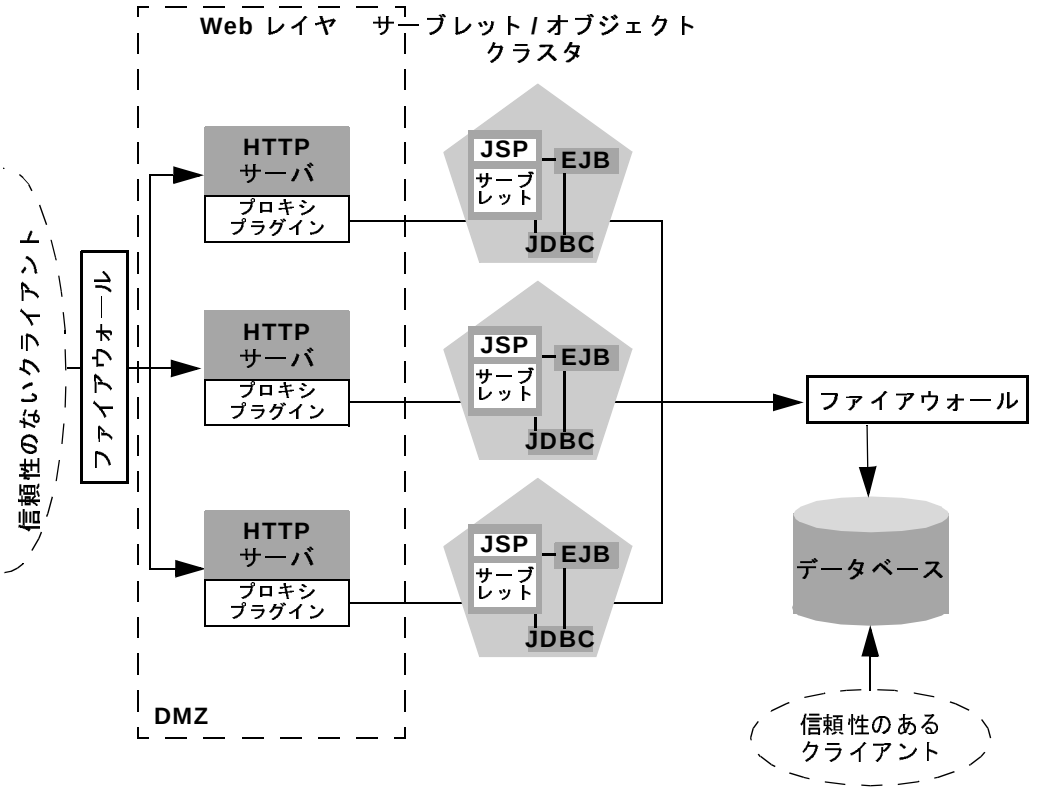
共有データベースに対するセキュリティの追加

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、「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。

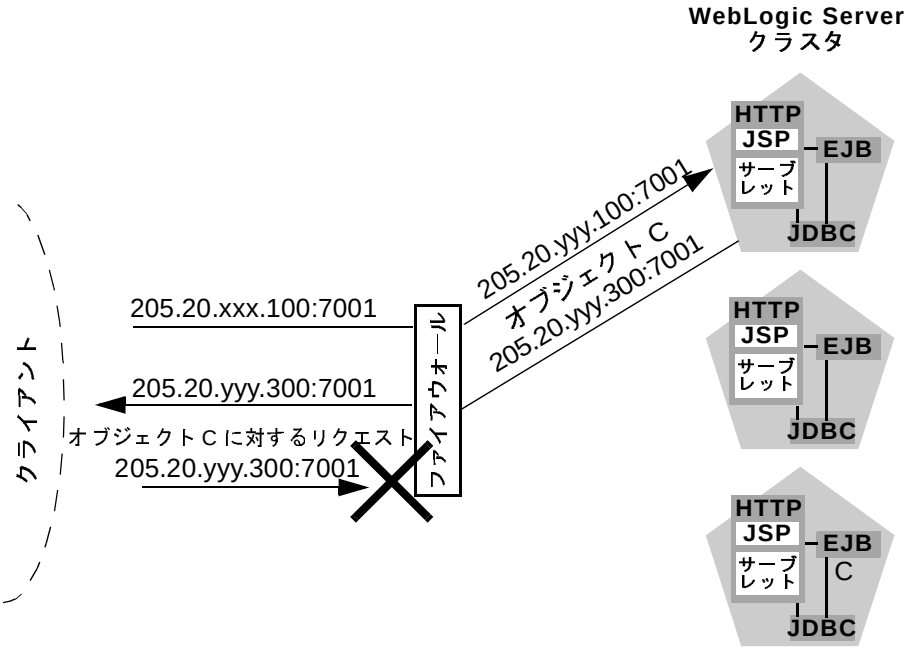


上記のコンフィグレーションでは、オブジェクト層とデータベースの間の境界は追加のファイアウォールによって強固になっています。ファイアウォールは、オブジェクト層のホストになっている WebLogic Server からの JDBC 接続以外のすべての接続を拒否する厳密なアプリケーション レベルのポリシーを保持します。

クラスタに関するファイアウォールの考慮事項

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。

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。



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、固定されているオブジェクト 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。

どの場合でもこの問題を回避するには、WebLogic Server インスタンスを DNS 名にバインドさせ、ファイアウォールの内側と外側で同じ DNS 名を使用します。WebLogic Server インスタンスの内部 DNS 名と外部 DNS 名が同一でない場合、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側では、`ExternalDNSName` がサーバの外部 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] タブを使用して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。

6　WebLogic クラスタの管理

以下の節では、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。

- 6-1 ページの「始める前に」
- 6-8 ページの「クラスタの実装手順」

始める前に

この節では、WebLogic Server クラスタをセットアップするための情報および前提となる作業の概要を示します。

クラスタ ライセンスを取得する

クラスタ構成で WebLogic Server インスタンスをインストールするには、正規のクラスタ ライセンスが必要です。クラスタ ライセンスをお持ちでない場合は、BEA の販売担当者にお問い合わせください。

コンフィグレーション プロセスを理解する

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作業の実施要領についての基本を理解していれば、この節で示す情報を最も有効に活用できます。

WebLogic Server 6.1 クラスタのすべてのコンフィグレーションは Administration Console を使用して行います。クラスタのコンフィグレーション情報は、クラスタを含むドメインの `config.xml` ファイルに格納されます。Administration Console では、クラスタを構成するインスタンスへのオブジェクトと Web アプリケーションのデプロイメントも管理します。

WebLogic Server 6.1 のコンフィグレーションに関する全般的な情報は、『WebLogic Server 管理者ガイド』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 アーキテクチャが最もニーズに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事項を以下に示します。

- 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層を複数のクラスタに分割するか
- クラスタ内のサーバ インスタンス間でどのようにして負荷の分散を図るか。以下のような選択肢がある
  - WebLogic Server の基本的なロード 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 台以上のセカンダリ HTTP サーバにデプロイし、プロキシを使用してアプリケーションへのリクエストをそれらのサーバに転送する
- 1 台以上のファイアウォールを使用して、Web アプリケーションの非武装地帯（DMZ）を定義するか

これらの決定にあたっては、第 5 章「WebLogic Server クラスタのプランニング」を参考にしてください。

どのアーキテクチャを選択するかによって、クラスタのセットアップ方法も変わります。選択したクラスタ 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、そ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。

ネットワークとセキュリティのトポロジを考慮する

適切なセキュリティ トポロジを設定するための基礎を形成するのはセキュリティ上の要件です。さまざまなレベルのアプリケーション セキュリティを実現する各種の代替アーキテクチャについては、5-22 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可能性があります。WAN 間でクラスタをデプロイしている場合の対処については、2-3 ページの「WAN クラスタのマルチキャスト要件」を参照してください。

クラスタ内の複数のサーバ インスタンスを、ファイアウォールを間に挟んでデプロイすることは避けてください。ファイアウォールを越えてマルチキャスト 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、2-4 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」で説明しています。

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マシン（この節全体を通じて「ホスト」と呼びます）を特定し、必要なリソースを各マシンが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事前要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： 動的に IP アドレスが割り当てられるマシンには WebLogic Server をインストールしないでください。

複数 CPU を備えるマシン上の WebLogic Server インスタンス

BEA WebLogic Server では、1 つのクラスタを構成するサーバ インスタンスの数について仕様上の制限はありません。Sun Microsystems, Inc. の Sun Enterprise 10000 など、複数のプロセッサを備える大規模なサーバには、非常に大規模なクラスタや複数のクラスタを収容できます。

ほとんどの場合、WebLogic Server クラスタの規模は、2 基の CPU につき 1 つの WebLogic Server インスタンスの割合でデプロイされるときが最適です。ただし、サーバ インスタンスの最適数および分散構成を決定するために、ターゲットの Web アプリケーションを使用して実際のデプロイメントをテストすることをお勧めします。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

ソケットのパフォーマンスを最大限に高めるには、pure-Java 実装でなく、オペレーティング システムのネイティブ ソケット リーダー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します。ネイティブ ソケットのコンフィグレーションまたは pure-Java ソケット通信の最適化を行う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。

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のアドレス情報（IP アドレスまたは DNS 名）を指定します。

クラスタ内通信の詳細と、クラスタ内通信によってロード バランシングおよびフェイルオーバが実現されるしくみについては、2-2 ページの「クラスタ内のサーバの通信」を参照してください。

クラスタをセットアップするときは、以下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、指定が必要な情報と、リソースの識別に用いる手法に影響する各種の要因について説明します。

リスン アドレスの問題の回避

クラスタをコンフィグレーションするとき、クラスタおよび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は、IP アドレス /DNS 名のどちらで指定してもかまいません。

DNS 名と IP アドレス

DNS 名と IP アドレスのどちらを使うかを決定するときは、クラスタの目的を考慮します。プロダクション環境では、一般に DNS 名の使用が推奨されます。IP アドレスを使用すると、以下の場合に変換エラーが発生する可能性があります。

- 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がある場合。推奨多層アーキテクチャについての説明にあるように、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配置する場合などが相当する。

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、変換エラーを回避できます。1 つのサーバ インスタンスの DNS 名が環境におけるファイアウォールの両側で必ず一致するようにし、ネットワーク上の Windows NT システムの名前と同じ DNS 名は使用しないでください。

IP アドレスの代わりに DNS 名を使用する方法の詳細については、5-14 ページの「ファイアウォールに関する制限」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 インスタンスの外部 DNS 名を定義します。ファイアウォールの外側では、`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] タブを使用して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。

ローカル ホストの検討事項

サーバ インスタンスのリスン アドレスがローカルホストのアドレスとして識別されると、非ローカル プロセスからそのサーバ インスタンスに接続することはできません。そのサーバ インスタンスをホスティングしているマシンのプロセスのみが、サーバ インスタンスに接続できます。サーバ インスタンスにローカルホストとしてアクセスしなければならない場合 (ローカルホストに接続する管理スクリプトがある場合など) で、リモート プロセスからもアクセスできるようにしなければならないときは、リスン アドレスを空白にしておきます。空白にしておくと、サーバ インスタンスはマシンのアドレスを判断し、そのアドレス上でリスンします。

WebLogic Server リソースへの名前の割り当て

WebLogic Server 環境内にあるコンフィグレーション可能な各リソースに、固有の名前が付いていることを確認します。個々のドメイン、サーバ、マシン、クラスタ、JDBC 接続プール、仮想ホストなどのリソースは、固有の名前を持っていなければなりません。

管理サーバのアドレスとポート

クラスタで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 WebLogic Server インスタンスをコンフィグレーションおよび管理するために使用される WebLogic Server インスタンスです。クラスタに属するサーバ インスタンスを起動するときは、その管理サーバのホストおよび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

クラスタに参加させる予定の各管理対象サーバの DNS 名または 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は一意な IP アドレスが必要であり、またリスン ポート番号は同じでなければなりません。1 台のマシンが、クラスタ化される複数のサーバ インスタンスのホストとなる場合、マルチホーム環境を設定して各サーバ インスタンスに別々の IP アドレスを供給する必要があります。

クラスタのマルチキャスト アドレスとポート

クラスタでのマルチキャスト通信専用に使用するアドレスおよび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送信します。

クラスタのマルチキャスト アドレスは、クラスタの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となるか、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は必ず、マルチキャスト通信でクラスタのマルチキャスト ポートとは異なるポートが使用されるようにしてください。

マルチキャストと複数のクラスタ

必要な場合には、ネットワーク上の複数のクラスタで、マルチキャスト アドレスとマルチキャスト ポートの 1 つ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 5 章「WebLogic Server クラスタのプランニング」で説明している推奨多層アーキテクチャを、クラスタ間にファイアウォールを挟む構成でセットアップしている場合、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、ファイアウォールがクラスタの通信に干渉しないことが保証されます。

クラスタ アドレス

Administration Console を使ってクラスタを構成する際に、そのクラスタ内の管理対象サーバを識別するクラスタ アドレスを付けます。

エンティティ Bean およびステートレス Bean 内では、クラスタ アドレスを使って、URL のホスト名部分を構成します。クラスタ アドレスが設定されていないと、EJB ハンドルが正しく機能しません。

クラスタ アドレスは、以下のものとして指定できます。

- クラスタ内の管理対象サーバの DNS 名または IP アドレスにマップされる DNS 名。
- 以下の例に示すような、クラスタ内の管理対象サーバの DNS 名または IP アドレスを含むリスト。

```
DNSName1,DNSName2,DNSName3
```

```
IPaddress1,IPaddress2;IPaddress3
```

プロダクション環境の場合、クラスタ アドレスには、クラスタ内の管理対象サーバのホスト名またはアドレスを含む DNS 名を設定することをお勧めします。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い場合、エンティティ Bean と EJB ハンドルに対してフェイルオーバが機能しません。4-5 ページの「エンティティ Bean と EJB ハンドルに対するフェイルオーバ」を参照してください。

プロダクション環境では、アドレスのカンマ区切りリストとしてクラスタ アドレスを定義することは推奨されません。

注意： 管理サーバはクラスタに参加させないことをお勧めします。管理サーバの IP アドレスが、クラスタ全体の DNS 名に含まれないようにしてください。詳細については、5-20 ページの「管理サーバに関する考慮事項」を参照してください。

クラスタの実装手順

この節では、クラスタ構成のアプリケーションをセットアップして実行する方法を、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

以下、クラスタを実装する作業の一般的な流れを示し、コンフィグレーション時の選択に影響する重要な考慮事項について説明します。実際の作業でのプロセスは、環境ごとに異なる特性やアプリケーションの性質によって異なる場合があります。この節では以下の作業について説明します。

- 6-10 ページの「WebLogic Server をインストールする」
- 6-10 ページの「マシン名を定義する（省略可能）」
- 6-11 ページの「WebLogic Server インスタンスを作成する」
- 6-12 ページの「新しいクラスタを作成する」
- 6-15 ページの「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（省略可能）」
- 6-19 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（省略可能）」
- 6-19 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（省略可能）」
- 6-20 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」
- 6-22 ページの「JMS をコンフィグレーションする」
- 6-24 ページの「Web アプリケーションと EJB をデプロイする」
- 6-23 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」
- 6-25 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」

すべてのクラスタ実装ですべての手順を経る必要があるとは限りません。また場合によっては、追加の手順が必要になることがあります。

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は、WebLogic Server をインストールします。マルチホームのクラスタにインストールする場合は、\bea ディレクトリにある WebLogic Server 配布キットを 1 つインストールして、すべてのクラスタ化されたインスタンスで使用します。リモートのネットワーク接続されたマシンにインストールする場合は、各 WebLogic Server ホストにインストールします。

クラスタ化された WebLogic Server インスタンスに対してインストールを行う場合にも、有効なクラスタ ライセンスが必要になります。詳細については、6-1 ページの「クラスタ ライセンスを取得する」を参照してください。

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。共有ファイルシステムにアクセスする場合（たとえば、個々のログ ファイルに書き込みを行う場合など）に、すべてのサーバが競合することになります。さらに、共有ファイルシステムに障害が発生した場合には、クラスタ化されたサーバを起動できなくなることもあります。

マシン名を定義する（省略可能）

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。通常、マシン名は WebLogic Server インスタンスのホストとなるマルチホーム 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、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の選択に悪影響を与えることがあります（3-6 ページの「レプリケーション グループの使用」を参照）。

新しい WebLogic Server インスタンスを作成する前に、以下の手順を実行して、サーバ インスタンスのホストになる個々のマシンの名前を定義します。

1. システムの管理サーバを起動します。手順については、『WebLogic Server 管理者ガイド』の「WebLogic 管理サーバの起動」を参照してください。
2. 『WebLogic Server 管理者ガイド』の「Administration Console の起動」にある手順に従って、Administration Console を起動します。
3. [マシン] ノードを選択します。
4. [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。
5. [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。
6. [作成] をクリックして、新しいマシンの定義を作成します。
7. 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場合は、Administration Console オンライン ヘルプを参照してください。
8. 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。

WebLogic Server インスタンスを作成する

サーバでクラスタを構成する前に、WebLogic Server Administration Console を使用して各サーバ インスタンスの新しい定義を作成する必要があります。以下の手順を実行します。

1. システムの管理サーバを起動します。詳細については、『WebLogic Server 管理者ガイド』の「WebLogic 管理サーバの起動」を参照してください。
2. 『WebLogic Server 管理者ガイド』の「Administration Console の起動」にある手順に従って、Administration Console を起動します。
3. [サーバ] ノードを選択します。
4. [新しい Server のコンフィグレーション] 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。

- [名前] : Administration Console で使用する、このサーバ インスタンスの名前を入力します。startup コマンドでこの名前を使用して、起動するサーバを指定します。

  6-6 ページの「WebLogic Server リソースへの名前の割り当て」を参照してください。

- [リスン ポート] : このサーバに接続する場合に使用するポート番号を入力します。指定したクラスタ内のすべてのサーバは、同じポート番号を使用する必要があります。
- [リスン アドレス] : このサーバにバインドする DNS 名または IP アドレスを入力します。

  6-5 ページの「リスン アドレスの問題の回避」を参照してください。

6. [マシン] 属性については、新しいサーバが存在するマシンを選択します。[マシン] 属性には、6-10 ページの「マシン名を定義する（省略可能）」で作成したすべてのマシンの名前が表示されます。
7. [作成] をクリックして、新しいサーバのコンフィグレーションを作成します。
8. 新しいサーバのその他の属性をコンフィグレーションする場合は、『WebLogic Server 管理者ガイド』の「サーバ コンフィグレーションの作業」を参照してください。
9. クラスタを構成する WebLogic Server ごとに、上記の手順を繰り返します。

新しいクラスタを作成する

個々の WebLogic Server インスタンスを作成したら、以下の手順を実行して、新しいクラスタをコンフィグレーションします。

1. Administration Console を起動します。
2. [クラスタ] ノードを選択します。
3. [新しい Cluster のコンフィグレーション] を選択します。
4. 以下の属性フィールドに値を入力します。

- [名前] : Administration Console で使用する、このクラスタの名前を入力します。この名前を使用して、クラスタにメンバシップを割り当てたり、別のクラスタ属性をコンフィグレーションしたりします。
- [クラスタ アドレス] : クラスタで使用する DNS 名（クラスタ内の個々の WebLogic Server インスタンスすべての IP アドレスまたは DNS 名を含む）を入力します。詳細については、6-7 ページの「クラスタ アドレス」を参照してください。
- [デフォルトのロード バランス アルゴリズム] : このクラスタで使用するデフォルトのロード アルゴリズムを入力するか、またはデフォルト値をそのまま使用します。

5. [作成] をクリックして、新しいクラスタのコンフィグレーションを作成します。
6. [マルチキャスト] タブを選択します。
7. [マルチキャスト アドレス] 属性フィールドにクラスタのマルチキャスト アドレスを入力します。
8. 変更を適用します。

WebLogic Server クラスタの起動

クラスタに参加する WebLogic Server インスタンスを起動するには、管理対象サーバの起動と同じ手順に従います。サーバ インスタンスが使用する管理サーバを特定します。サーバ インスタンスのコンフィグレーション情報は、管理サーバと関連付けられた `config.xml` ファイルから取得されます。

クラスタを起動するための基本的なプロセスは以下のとおりです。

9. クラスタが位置しているドメインの Administration Console を起動します。手順については、『WebLogic Server 管理者ガイド』の「Administration Console の起動」を参照してください。
10. . クラスタを構成する個別のサーバ インスタンスを管理対象サーバとして起動します。次に例を示します。

```
% java -ms64m -mx64m -classpath $CLASSPATH
-Dweblogic.Domain=mydomain -Dweblogic.Name=clusterServer1
-Djava.security.policy==$WL_HOME/lib/weblogic.policy
-Dweblogic.management.server=192.168.0.101:7001
```

```
-Dweblogic.management.username=system
-Dweblogic.management.password=systemPassword weblogic.Server
```

サーバ インスタンスのクラスタ コンフィグレーションは管理サーバによって格納されるため、アドレスのバインドまたはマルチキャストの情報をコマンドラインで明示的に指定する必要はありません。ただし、以下の要素は指定が必要です。

- `weblogic.Name` : 起動するクラスタ内のインスタンスを識別する
- `weblogic.management.server` : クラスタ化されるインスタンスのコンフィグレーションを格納する管理サーバのホストとポート番号を識別する
- `weblogic.management.username` : 管理サーバに接続するためのユーザ名を指定する
- `weblogic.management.password` : ユーザのパスワードを指定する

詳細については、『WebLogic Server 管理者ガイド』の「WebLogic 管理対象サーバの起動」を参照してください。

11. クラスタ化されたインスタンスの起動中、インスタンスがクラスタに参加していることをログ メッセージで確認します。インスタンスはクラスタのマルチキャスト アドレスと共通ポート番号をバインドすることによって開始します。

```
Starting Cluster Service ....

<Jul 25, 2001 6:35:17 PM PDT> <Notice> <WebLogicServer> <ListenThread listening
on port 7001, ip address 172.17.13.25>

<Jul 25, 2001 6:35:18 PM PDT> <Notice> <Cluster> <Listening for multicast messages
(cluster MyCluster) on port 7001 at address 239.0.0.84>

<Jul 25, 2001 6:35:18 PM PDT> <Notice> <WebLogicServer> <Started WebLogic Managed
Server "MyServer-1" for domain "mydomain" running in Production Mode>
```

12. クラスタ化されたすべてのインスタンスがクラスタに参加していることを確認するために、Administration Console を開きます。クラスタを選択して [モニタ] タブを選択し、[このクラスタを構成するサーバをモニタ] を選択します。起動したサーバ インスタンスのうち表示されていないものがある場合、サーバ インスタンスのコンフィグレーションでマルチキャスト アドレスとポート番号が正しいかどうか確認します。

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（省略可能）

この節では、外部ロード バランサのセットアップに関する一般的なガイドラインを示します。Alteon ロード バランサのセットアップ手順については、「クラスタに関する Alteon™ 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。BIG-IP ロード バランサのセットアップ手順については、「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。

ロード バランシング ハードウェアを含むコンフィグレーションで WebLogic Server がクラスタ化されたサーブレットおよび JSP にアクセスするしくみについては、3-12 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。

HTTP セッション ステートのレプリケーションでハードウェアによるロード バランシング ソリューションを使用している場合は、WebLogic Server セッション クッキーをサポートするよう、ロード バランサをコンフィグレーションする必要があります。コンフィグレーションの手順は、ロード バランシング ハードウェアで使用される、クッキーの永続性メカニズムのタイプによって異なります。次の表に、使用できるコンフィグレーションを示します。

永続性タイプ		
アクティブなクッキーの永続性		**パッシブなクッキーの永続性**
ロード バランサによってクッキーが上書きされる	**ロード バランサによって追加のクッキーが挿入される**	
サポートされていない	コンフィグレーションは不要	オフセットと文字列定数を指定する

アクティブなクッキーの永続性の使用

WebLogic Server クラスタでは、WebLogic HTTP セッション クッキーを上書き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。

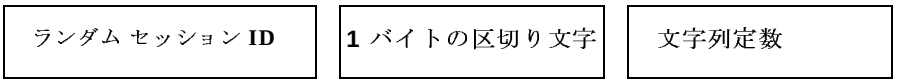
ロード バランサのアクティブなクッキーの永続性メカニズムが、クライアント セッションに独自のクッキーを追加するものである場合は、WebLogic Server クラスタでロード バランサを使用するにあたって、新たなコンフィグレーションは不要です。

パッシブなクッキーの永続性の使用

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 クッキーに文字列定数を使用することで、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 インスタンスに関連付けることができます。文字列定数は、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 バランサのコンフィグレーションには、オフセットと文字列定数の長さも必要です。セッション クッキーの形式に従ったオフセットと長さの有効な値は、次のようになります。

セッション クッキーについて

セッション クッキーの基本形式は以下のとおりです。



各値は次のとおりです。

- *ランダム セッション ID* は、ランダムに生成された HTTP セッション ID。値の長さは、そのアプリケーション用の `weblogic.xml` ファイルのなかの <session-descriptor> 要素の `IDLength` パラメータを設定することにより、コンフィグレーションが可能です。デフォルトでは、ランダム セッション ID の長さは 52 バイトです。

- *文字列定数* は、セッションをホストする WebLogic Server インスタンスを、そのホスト マシンとポートも含めて識別します。

  クラスタ化されていない環境では、文字列定数は、以下の形式で最大 60 バイトです。

  ```
  Primary_JVMID_Differentiator!HOST!PORT!SSLPORT
  ```

  クラスタ環境では、文字列定数はそのセッションに対してプライマリとセカンダリ両方の WebLogic Server インスタンスを指定し、つまり、以下の形式で最大 120 バイトです。

  ```
  Primary_JVMID_Differentiator!Host!Port!SSLPort!Secondary_JVMID_
  Differentiator!Host!Port!SSLPort
  ```

  WebLogic Server バージョン 6.1 の初期リリースでは、文字列定数の最初の 19 バイトがプライマリ WebLogic Server インスタンスを一意に識別します。WebLogic Server バージョン 6.1 サービス パック 1 およびそれ以降では、最初の 10 バイトだけでサーバ インスタンスが識別されます。

WAP- 対応アプリケーションのためのセッション クッキー

無線デバイスがクッキーをサポートしないため、それに代わるセッション追跡方法として、WAP アプリケーションに対して、URL 書き換えを 使用できます。無線デバイスでは、限られた長さの URL だけをサポートすることが多いので、WAP 対応アプリケーションに対する URL 書き換えの使用には、セッション パラメータの長さを短くすることが必要です。無線アプリケーション用の URL 書き換えに関する詳細は、『WebLogic Server Wireless Application 開発プログラマーズ ガイド』の「セッション トラッキングの代替手法 － URL 書き換え」を参照してください。

2 通りの方法で、ロード バランサをどうコンフィグレーションするかに影響するセッション パラメータ の長さを制限します。

- アプリケーション用の `weblogic.xml` ファイル内にある <session-descriptor> 要素の `IDLength` パラメータの値を編集して、ランダム セッション ID の長さを制限できます。ランダム セッション ID は、最短 8 バイトです。
- そのドメイン用の `config.xml` ファイル内にある `WAPEnabled` パラメータを「true」に設定することにより、プライマリ サーバ インスタンスとセカンダリ サーバ インスタンスを指定するセッション パラメータの長さを制限できます。`WAPEnabled` が「true」になっていると、プライマリ サーバ インスタ

ンスとセカンダリ サーバ インスタンスは、短いハッシュ コードによってクッキー内で識別されます。

サーバ インスタンスを指定するランダム セッション ID とセッション パラメータを短くすることにより、文字列定数は以下の形式になります。

```
Rand_Sess_ID!Primary_JVMID_HASH!Secondary_JVMID_HASH
```

ここで、それぞれの長さは以下のとおりです。

`Rand_Sess_ID` は 8 バイト。

`Primary_JVMID_HASH` は 8 ～ 10 バイト。

`SECONDARY_JVMID_HASH` は 8 ～ 10 バイト。

ロード バランサのコンフィグレーション

ランダム セッション ID の長さを、デフォルト長である 52 バイトから変更していない場合、以下のようにロード バランサの機能を使って , 文字列定数のオフセットと長さを設定します。

- 文字列定数オフセット —53 バイトに設定。デフォルトのランダム セッション ID 長＋区切り文字用の 1 バイト。
- 文字列定数の長さ —BEA は、文字列定数長は、少なくとも 19 バイトに設定することをお勧めします。19 バイトあれば、プライマリ サーバ インスタンスをユニークに識別するのに十分な長さです。

使用するアプリケーションまたは環境による要件によって、ランダム セッション ID の長さをデフォルト値の 52 バイトから変更する必要がある場合、それに伴ってロード バランサ上で文字列定数オフセットを設定します。文字列定数オフセットは、「ランダム セッション ID 長＋区切り文字用の 1 バイト」に等しくします。

使用するクラスタが、WAP 対応アプリケーションをホストする場合、ロード バランサのコンフィグレーションに影響すると思われるセッション パラメータを考慮します。詳細については、「セッション パラメータ長短縮のためのロード バランサのコンフィグレーション（WAP 対応）」を参照してください。

セッション パラメータ長短縮のためのロード バランサのコンフィグレーション（WAP 対応）

使用するドメインについて、`WAPEnabled` が「true」、`IDLength` が 8 バイトに設定されている場合、文字列定数のオフセットと長さを以下のように設定します。

- 文字列定数オフセット — 9 バイトに設定。ランダム セッション ID 長＋区切り文字用の 1 バイト。
- 文字列定数の長さ — 文字列定数の長さを 8 に設定。プライマリ サーバ インスタンスを識別するハッシュ コードの最短の長さ。

プロキシ プラグインをコンフィグレーションする（省略可能）

WebLogic プロキシ プラグイン（または `HttpClusterServlet`）を使用する Web サーバからクラスタにアクセスする場合は、『WebLogic Server 管理者ガイド』にある手順に従って、プロキシ ソフトウェアをコンフィグレーションします。リクエストをクラスタにプロキシするすべての Web サーバを同じようにコンフィグレーションする必要があります。プロキシ プラグインを使用した接続とフェイルオーバの詳細については、3-8 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。

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（省略可能）

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 のホストとなる場合、セッション ステートのレプリカを保持するために、WebLogic Server インスタンスのレプリケーション グループを作成できます。そのためには、3-6 ページの「レプリケーション グループの使用」の指示に従って、各レプリケーション グループに参加するサーバ インスタンスを決定し、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、次の手順に従います。

1. Administration Console を開きます。
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. [クラスタ] タブを選択します。
5. 以下の属性フィールドの値を入力します。
   - [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 グループの名前を入力します。
   - [セカンダリ プリファレンス グループ]: このサーバ インスタンスで、レプリケートされた HTTP セッション ステートのホストとして使用するレプリケーション グループの名前を入力します。
6. 変更内容を適用します。

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 を使用して JDBC コンポーネントをコンフィグレーションする手順を示します。JDBC コンポーネントをコンフィグレーションする過程で行う選択は、クラスタが位置している WebLogic Server ドメインの `config.xml` ファイルに反映されます。

まず接続プールと、必要であればマルチプールを作成します。次にデータ ソースを作成します。データ ソース オブジェクトを作成するときは、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。これにより、データ ソースが 1 つの特定の接続プールまたはマルチプールと関連付けられます。

- 1-4 ページの「JDBC 接続」では、JDBC オブジェクトが WebLogic Server クラスタ内で機能するしくみの概要を示しています。
- 2-23 ページの「フェイルオーバと JDBC 接続」では、JDBC のクラスタ化によってアプリケーションの可用性を改善できるしくみについて説明しています。

- 2-19 ページの「ロード バランシングと JDBC 接続」では、JDBC のクラスタ化におけるロード バランシングのサポートについて説明しています。

接続プールのクラスタ化

クラスタ内で基本接続プールをセットアップするには、次の手順に従います。

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「JDBC 接続プールのコンフィグレーション」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、「サーバ、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。`PoolName` 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、「JDBC データ ソースの割り当て」を参照してください。

マルチプールのクラスタ化

可用性を高めるためにマルチプールのクラスタを作成し、必要に応じてロード バランシングを行うには、次の手順に従います。

注意： マルチプールは一般に、可用性の向上と、レプリケートおよび同期の対象となるデータベース インスタンスへの接続のロード バランシングを目的として使用されます。詳細については、1-4 ページの「JDBC 接続」を参照してください。

1. 2 つ以上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、「サーバ、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、「サーバ、クラスタへの 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。`Pool Name` 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、「JDBC データ ソースの割り当て」を参照してください。

JMS をコンフィグレーションする

クラスタに対して JMS をコンフィグレーションするには、『管理者ガイド』の「JMS の管理」の指示に従います。以下のガイドラインを守るようにしてください。

- JMS サーバ —1 つの JMS サーバを複数の管理対象サーバにデプロイすることはできません。
- 送り先 — 複数の JMS サーバに同じ送り先をデプロイすることはできません。
- 接続ファクトリ — 個々の接続ファクトリを複数の WebLogic Server にデプロイすることができます。
- 管理対象サーバ — サーバが複数の異なったドメインに分散している場合でも、クラスタ化された WebLogic Server に JMS クライアントがアクセスするためには、一意な WebLogic Server 名が必要です。必ず、JMS クライアントがアクセスするすべての管理対象サーバに一意なサーバ名を付けるようにしてください。

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

サーブレットと JSP の自動フェイルオーバに対応するために、WebLogic Server は HTTP セッション ステートをメモリ内でレプリケートします。

注意： WebLogic ServerWebLogic Server では、サーブレットまたは JSP の HTTP セッション ステートを、ファイルベースまたは JDBC ベースの永続性を使用して管理することもできます。これら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「セッションの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 ステートのインメモリ 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。このレプリケーションを制御する `PersistentStoreType` パラメータは、アプリケーションの WebLogic デプロイメント記述子ファイル（`weblogic.xml`）の `session-descriptor` 要素の内部に位置します。

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 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するには、`PersistentStoreType` を `replicated` に設定します。`weblogic.xml` での正しい XML 記述を次に示しています。

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

Web アプリケーションと EJB をデプロイする

「Web アプリケーションのパッケージ化とデプロイ」にある手順に従って、Web アプリケーションおよび EJB をクラスタにデプロイします。アプリケーションまたは EJB をデプロイする対象を選択するときには、クラスタ内の個々の WebLogic Server インスタンスではなく、6-12 ページの「新しいクラスタを作成する」で指定したクラスタ名を選択します。クラスタ名を使用することで、アプリケーションまたは EJB がクラスタ全体に均一にデプロイされます。

WebLogic Server では、クラスタ化されたオブジェクトは均一にデプロイされなければなりません。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。それ以外の場合は、レプリカ非対応の（「ピン固定された」）オブジェクトを個々のサーバ インスタンスに対してのみデプロイします。

Administration Console では、レプリカ対応オブジェクトのクラスタへのデプロイが自動化されています。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）。

Administration Console を使用してアプリケーションをデプロイするとき、管理サーバはターゲット サーバ インスタンスにアプリケーション ファイルのコピーを送り、その後、ターゲット サーバ インスタンスがアプリケーションをロードします。

注意： あるサーバ インスタンスへのデプロイメントが失敗すると、ターゲット サーバ インスタンス間でデプロイメント状態の整合性が失われる場合があります。

クラスタ化されるオブジェクトがクラスタワイドの JNDI ツリーにバインドされるしくみについては、2-11 ページの「クラスタワイドの JNDI ツリーの作成」および 2-14 ページの「JNDI ツリーの更新」を参照してください。

コンフィグレーションに関するその他のトピック

この節では、特定のクラスタ コンフィグレーションで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

BEA では、最適なパフォーマンスが得られるように、WebLogic Server インスタンスのホストとなるマシン上で、pure-Java 実装ではなくネイティブ ソケット リーダー実装を使用することを推奨しています。

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、個々のサーバ インスタンスとクライアント マシンについてソケット リーダー スレッド数を適切に調整することによって、ソケット通信のパフォーマンスを改善することができます。

- クラスタで IP ソケットが使用されるしくみの詳細と、ネイティブ ソケット リーダー スレッドによって最適なパフォーマンスが得られる理由については、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 2-15 ページの「クライアントとクラスタワイドの JNDI ツリーとの対話」を参照してください。
- クラスタでのソケット リーダー スレッドの必要数を決定する方法については、2-8 ページの「ソケットの使用数の確定」を参照してください。多層クラスタ アーキテクチャでサーブレット クラスタをデプロイしている場合、5-13 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、このこと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手順と、ホストおよびクライアント マシンに対してリーダー スレッド数を設定する手順を示します。

サーバ インスタンスのホストであるマシンに対してネイティブ 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 Administration Console を開きます。

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] チェック ボックスをオンにします。
6. 変更内容を適用します。

サーバ インスタンスのホストであるマシン上のリーダー スレッド数を設定する

デフォルトでは、WebLogic Server インスタンスは起動時に 3 個のソケット リーダー スレッドを作成します。負荷のピーク時にクラスタ システムで 3 個以上のソケットを利用できると判断した場合、次の手順でソケット リーダー スレッドの数を増やすことができます。

1. WebLogic Server Administration Console を開きます。
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 スレッドの比率を変更します。Java ソケット リーダーの数は、総実行スレッド数（[実行スレッド数] 属性フィールドの値）にこの比率を乗じて算出されます。
6. 変更内容を適用します。

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、クライアントを実行する Java 仮想マシン（JVM）内のソケット リーダー スレッドの数を設定できます。クライアントの Java コマンドラインで、`-Dweblogic.ThreadPoolSize=`*`value`* および `-Dweblogic.ThreadPoolPercentSocketReaders=`*`value`* オプションを定義することによってソケット リーダーを指定します。

マルチキャスト生存時間（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生存時間（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。マルチキャスト生存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。マルチキャスト生存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 トポロジを設定する方法については、2-3 ページの「WAN クラスタのマルチキャスト要件」を参照してください。

クラスタのマルチキャスト生存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 [マルチキャスト生存時間] の値を変更します。`config.xml` の以下の抜粋部分は、マルチキャスト生存時間の値に 3 を指定したクラスタを示しています。この値により、クラスタのマルチキャスト メッセージが破棄されるまでに 3 つのルータを通過できることが保証されます。

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

マルチキャスト バッファ 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切なタイミングで処理していないことが理由でマルチキャスト ストームが発生する場合、マルチキャスト バッファのサイズを大きくすることで対処できます。マルチキャスト ストームの詳細については、2-5 ページの「マルチキャスト ストームが発生する場合」を参照してください。

UNIX の ndd ユーティリティを使用して、TCP/IP カーネル パラメータを調整することができます。udp_max_buf パラメータは、UDP ソケットの送信バッファおよび受信バッファのサイズをバイト単位で設定します。udp_max_buf の適切な値はデプロイメント環境によって異なります。マルチキャスト ストームが発生している場合、udp_max_buf の値を 32K ずつ大きくして調整の効果を確認してください。

udp_max_buf パラメータは必要な場合以外は変更しないでください。udp_max_buf パラメータを変更する場合は、事前に『*Solaris Tunable Parameters Reference Manual*』（http://docs.sun.com/?p=/doc/806-6779/6jfmsfr7o&）の「TCP/IP Tunable Parameters」の章の「UDP Parameters with Additional Cautions」に記されている Sun からの警告を確認してください。

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事項

クラスタで多層アーキテクチャを採用している場合、5-13 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」に示されているコンフィグレーション上のガイドライ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 では代わりに URL 書き換えを利用してプライマリ 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 サーバの間で受け渡しされる URL に埋め込まれます。この機能を使用するには、WebLogic Server クラスタ上で URL 書き換えを有効にしておく必要があります。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。

A　一般的な問題のトラブルシューティング

以下の節では、一般的な問題のトラブルシューティング方法について説明します。

- 診断情報の収集
- 一般的な問題の解決

診断情報の収集

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、この節の手順に従って、システムの診断情報を収集してください。クラスタ関連の問題についての主な診断情報は、（可能な場合）クラスタ化されたサーバからの複数のスレッド ダンプを格納するログ ファイルです。ログ ファイルはクラスタ関連のさまざまな問題の診断に役立ちますが、クラスタの「フリーズ」やデッドロックに関する問題の解決に特に重要です。

注意： サーバ インスタンス間のデッドロックを伴うか、クラスタが「ハング」する原因となったクラスタ関連の問題が発生した場合、複数のスレッド ダンプを格納するログ ファイルは問題の診断に欠かせません。

必要なログ ファイルを作成するには次の手順に従います。

1. 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。実際には、これまでのログ ファイルに新規セッションを追加するのではなく、WebLogic Server インスタンスを起動するたびに新規のログ ファイルを作成する必要があります。
2. WebLogic Server を起動したときに、Java VM の verbose ガベージ コレクション（GC）出力をオンにします。サンプルのコマンドラインについては、次の手順を参照してください。

3. 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。リダイレクトすると、スレッド ダンプ情報が WebLogic Server の情報メッセージとエラー メッセージに関連してログに記録され、診断する際にログ ファイルがさらに役立ちます。次に例を示します。

```
% 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
-Dweblogic.management.username=system
-Dweblogic.management.password=systemPassword weblogic.Server >>
logfile.txt 2>&1
```

4. 問題が再発生するまで、WebLogic Server 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` 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。各サーバの複数のスレッド ダンプがスレッド ダンプどうしで明確な間隔をおいてログ ファイルに入っていることを確認します。

注意： Linux 環境で JRockit JVM を実行している場合の対処については、A-2 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。

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 ダンプを生成します。

- `weblogic.admin THREAD_DUMP` コマンドを使用します。指示と制限事項については、『*WebLogic Server 管理者ガイド*』の「THREAD_DUMP」を参照してください。
- JVM の管理サーバが（`-Xmanagement` オプションを指定して JVM を起動することによって）有効になっている場合、JRockit Management Console を使用してスレッド ダンプを生成できます。
- `kill -3 PID` コマンドを使用します。`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' **. **
  ```

  サーバの起動コマンドと一致する文字列を `grep` 引数に指定して、プロセス スタック内を検索します。返される最初の PID はルート プロセスになりま

す。これは、ps コマンドが別のルーチンにパイプされていないことを想定しています。

注意： Linux 環境では、各実行スレッドは Linux プロセス スタック下の独立したプロセスであるかのように動作します。Linux 上で kill -3 コマンドを使用するには、メインの WebLogic 実行スレッドの PID と一致する PID を指定しなければなりません。正しく指定しないと、スレッド ダンプは生成されません。

BEA テクニカル サポートへの診断情報の提供

（可能であれば複数のスレッド ダンプが入った）診断ログ ファイルを作成したら、次のガイドラインを参考にして、BEA テクニカル サポート担当者に診断情報を提供します。

1. オペレーティング システムの圧縮ユーティリティを使ってログ ファイルを圧縮します。

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

2. テクニカル サポート担当者宛の E メールに圧縮済みログ ファイルを追加します。

   注意： 圧縮済みファイルは必ずメッセージに添付する形で追加します。E メールの本文にログ ファイルを切り取って貼り付けないでください。

3. 圧縮済みログ ファイルのサイズが大きすぎて E メールに添付できない場合は、BEA カスタマ サポートの FTP サイトを利用します。

一般的な問題の解決

次の節では、クラスタ関連の一般的な問題を解決する方法について説明します。また、クラスタのパフォーマンスが悪いなど、具体的な問題以外について診断する方法についても説明します。

サーバがクラスタを構成できない場合

WebLogic Server が起動時にクラスタを構成できない理由には、一般的なネットワークの可用性や WebLogic 固有のコンフィグレーションの問題などがあります。次のチェックリストを使用してクラスタの構成、および起動プロセスを確認してください。

1. コマンドライン パラメータの入力の間違い、つづりの誤りなどがないか確認します。
2. ネットワーク接続などの物理的な問題が無いことを確認します。ネットワーク接続の確認には、「接続のテスト」で説明した `dbping` ユーティリティを使用します。
3. ほかのアプリケーションがクラスタ マルチキャスト アドレスを使用していないことを確認します。
4. utils.MulticastTest ユーティリティを実行して、マルチキャストが正常に動作していることを確認します。

これ以外にも、トラブルシューティングが必要なエラーとして、次のような一般的なコンフィグレーション エラー、および通信エラーなどがあります。

- **Incompatible version numbers.** クラスタ内のすべての WebLogic Server は同じバージョンでなければなりません。クラスタ内のほかのサーバとは異なるバージョンの WebLogic Server でクラスタを構成しようとしても、エラーメッセージが表示されます。
- **Unable to find a license for clustering.**WebLogic のライセンスにクラスタリング機能が含まれていません。営業担当者までお問い合わせください。
- **Unable to send service announcement.** これは、ネットワークに関する一般的な問題、または DNS の構成が誤っていることを意味します。クラスタ構成のサーバは、、マルチキャストを経由して相互に通信するため、同一の（排他的な）マルチキャスト アドレスを共有する必要があります。
- **Cannot set default clusterAddress properties value.** これは、同じ IP アドレスを持つ、別のサーバがすでにクラスタ内に存在することを意味します。複数のマシンに同じ IP アドレスを割り当てていないかどうか確認してください。

- **Unable to create a multicast socket for clustering, Multicast socket send error, or Multicast socket receive error.** これらの通信エラーは、主にマルチキャスト アドレスが間違っているか不正かのどちらかの原因により発生します。

  各オペレーティングシステム システムには、マルチキャストを構成するための固有のコンフィグレーション要件があります。したがって、このエラーの修復については、使用しているオペレーティング システムのマニュアルを参照してください。

B　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

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 の Administration 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

レプリカを呼び出すオブジェクトと同じサーバ インスタンス上にレプリカが存在する場合は、ローカル レプリカを使う方が効率的なので、呼び出しのロードバランシングは行われません。詳細については、4-8 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。

C　クラスタに関する Alteon™ ハードウェアのコンフィグレーション

この節の内容は以下のとおりです。

- 概要
- 要件
- サンプル コンフィグレーション
- WebLogic Server クラスタに関する Alteon のコンフィグレーション
- WebLogic Server クラスタに関する Alteon SSL アクセラレータのコンフィグレーション

概要

この節では、Alteon WebSystems の Web スイッチおよびサーバ ロード バランシング ソフトウェアを WebLogic Server クラスタで動作するように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 Web OS 管理ツールの使い方やハードウェアと Web スイッチの接続方法といった Alteon 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。

Alteon 製品のコンフィグレーションを順を追って説明している部分もあります。設定および管理の詳細な手順については、Alteon 製品のマニュアルを参照してください。

要件

ここで説明するコンフィグレーション手順では、以下の製品を使用する必要があります。

- WebLogic Server バージョン 6.0 クラスタ
- Alteon ACEdirector シリーズ AD3 またはそれ以上の Web スイッチ
- Alteon Web OS とサーバ ロード バランシング（SLB）およびアプリケーション リダイレクション（AR）

サンプル コンフィグレーション

ここで説明する手順では、以下のコンポーネントで構成されたサンプルの WebLogic Server クラスタを基準とします。

- IP アドレス、192.168.0.10、192.168.0.11、および 192.168.0.12 にバインドされた 3 つの Weblogic Server インスタンス。
- ポート 6、7、および 8 を WebLogic Server インスタンス用に設定した Alteon Web スイッチ。ポート 1 はクライアント接続用です。
- 予約済み 仮想 IP アドレス 172.17.10.100。外部クライアントがクラスタとの接続に使用します。
- 予約済み仮想 IP アドレス。Web スイッチ自体に割り当てます。

実際のコンフィグレーションでは、IP アドレスも Web スイッチのポートもこのサンプルとは異なる可能性があります。システムに iSD-SSL アクセラレータが含まれている場合は、「WebLogic Server クラスタに関する Alteon のコンフィグレーション」の手順に従って基本的なロード バランシング機能をコンフィグレーションしてから、「WebLogic Server クラスタに関する Alteon SSL アクセラレータのコンフィグレーション」を参照してください。

WebLogic Server クラスタに関する Alteon のコンフィグレーション

WebLogic Server クラスタで Alteon 製品をコンフィグレーションする手順を簡単に説明します。

1. WebLogic Server クラスタをインストールし、サーバ インスタンスをそれぞれの内部 IP アドレス（192.168.0.10、192.168.0.11、192.168.0.12）にバインドします。すべての IP アドレスがイントラネットの同じサブネット内にあることを確認します。すべてのサーバ インスタンスがマルチキャスト トラフィックを受け取れるようにするため、サブネットは 1 つにする必要があります。
2. すべてのコンポーネントを Alteon Web スイッチに物理的に接続します。
3. Alteon Web スイッチ用と WebLogic Server 用の仮想 LAN（VLAN）をそれぞれ作成します。Web スイッチの VLAN はデフォルトで VLAN 1 としてコンフィグレーションされます。クラスタ用の新しい VLAN は次のようにコンフィグレーションします。

   ```
   >> # /cfg/vlan 2
   >> VLAN 2# add 6
   >> VLAN 2# add 7
   >> VLAN 2# add 8
   >> VLAN 2#ena
   ```

4. Web スイッチの VLAN および WebLogic Server の VLAN で IP インタフェースをコンフィグレーションして有効にします。
   a. Web スイッチの VLAN の IP インタフェースは次のようにコンフィグレーションします。

      ```
      >> # /cfg/ip/if 1
      >> IP Interface 1# addr 192.168.0.20
      >> IP Interface 1# ena
      ```

   b. WebLogic Server クラスタの VLAN の IP インタフェースは次のようにコンフィグレーションします。

```
>> # /cfg/ip/if 2
>> IP Interface 2# addr 172.17.10.100
>> IP Interface 2# vlan 2
>> IP Interface 2# ena
```

5. Web スイッチにクラスタ内の各 WebLogic Server の実 IP アドレスを追加します。

```
>> # /cfg/slb/real 1
>> Real Server 1# rip 192.168.0.10
>> Real Server 1# ena
>> Real Server 1# /cfg/slb/real 2
>> Real Server 2# rip 192.168.0.11
>> Real Server 2# ena
>> Real Server 2# /cfg/slb/real 3
>> Real Server 3# rip 192.168.0.12
>> Real Server 3# ena
```

6. Alteon 管理ソフトウェアを使用して、WebLogic Server インスタンスの IP アドレス グループを定義します。このグループ（と仮想 IP アドレス）を使用して、クラスタ化されたシステム間のトラフィックのルーティング ポリシーを定義します。

```
>> # /cfg/slb/group 1
>> Real server group 1# add 1
>> Real server group 1# add 2
>> Real server group 1# add 3
```

7. グループのハッシュ ロード バランシング メトリックを定義します。

```
>> # /cfg/slb/group 1
>> Real Server group 1# metrc hash
```

8. WebLogic Server クラスタの仮想 IP アドレスを定義します。

```
>> # /cfg/slb/virt 1
>> Virtual server 1# vip 172.17.10.100
>> Virtual server 1# service http
>> Virtual server 1 http Service# group 1
```

```
>> Virtual server 1 http Service# ..
>> Virtual server 1# ena
```

9. クッキーベースの永続性をコンフィグレーションし、WebLogic Server 識別子を指定します。次の例では、デフォルトの WebLogic Server クッキーおよびサイズを使用しています。

```
>> # /cfg/slb/virt 1/service 80
>> Virtual server 1 http service# pbind
Enter clientip|cookie|sslid persistence mode: cookie
Enter passive|rewrite cookie persistence mode: passive
Enter Cookie Name:JSESSIONID
Enter the starting point of the cookie value: 53
Enter the number of bytes to extract: 19
Look for cookie in URI [e|d]: dis
```

10. クラスタおよびクライアントの接続先となる Web スイッチのポートをコンフィグレーションします。

```
>> # /cfg/slb/port 6
>> Port 6# server ena
>> # /cfg/slb/port 7
>> Port 7# server ena
>> # /cfg/slb/port 8
>> Port 8# server ena
>> Port 8# /cfg/slb/port 1
>> Port 1# client ena
```

11. 新しいロード バランシング設定を有効にして保存します。

```
>> # /cfg/slb
>> Server Load Balancing# on
>> Server Load Balancing# apply
>> Server Load Balancing# save
```

WebLogic Server クラスタに関する Alteon SSL アクセラレータのコンフィグレーション

WebLogic Server クラスタで Alteon SSL アクセラレータを使用している場合、アクセラレータ用の仮想 LAN もコンフィグレーションし、クラスタとアクセラレータ間のトラフィックの方向を決めるルーティング ポリシーを作成する必要があります。次の手順では、Alteon iSD100-SSL アクセラレータを使用するために「WebLogic Server クラスタに関する Alteon のコンフィグレーション」で説明したサンプル システムを拡張します。アクセラレータは、Alteon Web スイッチのポート 2 に接続します。

WebLogic Server クラスタで SSL アクセラレータを使用するには次の手順に従います。

1. Alteon 製品のマニュアルを参照して、iSD100-SSL アクセラレータをインストールして初期化します。
2. Web スイッチの SSL アクセラレータ用と WebLogic Server 用の仮想 LAN（VLAN）を作成します。Web スイッチ用の仮想 LAN である VLAN 1 はデフォルトの VLAN なので、明示的に作成する必要はありません。クラスタ用の仮想 LAN である VLAN 2 は、「WebLogic Server クラスタに関する Alteon のコンフィグレーション」で説明したように設定します。Web SSL アクセラレータ用の VLAN は次のように作成して有効にします。

   ```
   >> # /cfg/vlan 2
   >> VLAN 2# add 2
   >> VLAN 2# ena
   ```

3. SSL アクセラレータおよび WebLogic Server クラスタの接続先ポートの Web スイッチのスパニング ツリー プロトコルを無効にします。

   ```
   >> # /cfg/stp/port 2
   >> STP PORT 2# off
   >> # /cfg/stp/port 6
   >> STP PORT 6# off
   ```

```
>> # /cfg/stp/port 7
>> STP PORT 7# off
>> # /cfg/stp/port 8
>> STP PORT 8# off
```

4. SSL アクセラレータの VLAN の IP インタフェースをコンフィグレーションして有効にします。この例では、WebLogic Server クラスタと同じサブネットに SSL アクセラレータを配置しています。アクセラレータは、独自のプライベート サブネットに配置することもできます。

```
>> # /cfg/ip/if 2
>> IP Interface 2# addr 192.168.0.21
>> IP Interface 2# vlan 2
>> IP Interface 2# ena
```

5. まだコンフィグレーションしていない場合は、「WebLogic Server クラスタに関する Alteon のコンフィグレーション」の手順に従って、クラスタの実 IP アドレス、サーバ グループ、ロード バランシング メトリック、およびクッキーベースの永続性をコンフィグレーションします。

6. SSL アクセラレータの SSL トラフィック用の TCP ポート番号をコンフィグレーションします。

```
>> # /cfg/isd/ssl/setport 81
```

7. TCP ポート 443 のクライアントの HTTPS トラフィックを SSL アクセラレータにリダイレクトするためのフィルタをコンフィグレーションします。次のコマンドでは、フィルタ 100 を HTTPS フィルタとして使用します。次の手順のグループ 256 は SSL アクセラレータ用に予約されているので、明示的に作成する必要はありません。

```
>> # /cfg/slb/filt 100
>> FILTER 100# proto tcp
>> FILTER 100# dport https
>> FILTER 100# action redir
>> FILTER 100# group 256
>> FILTER 100# rport https
>> FILTER 100# ena
```

8. その他の TCP とトラフィック用のデフォルト フィルタをコンフィグレーションします。

```
>> # /cfg/slb/filt 101
>> FILTER 101# sip any
>> FILTER 101# dip any
>> FILTER 101# proto any
>> FILTER 101# action allow
>> FILTER 101# ena
```

9. 新しいフィルタを Web スイッチのクライアント ポートに追加します。

```
>> # /cfg/slb/port 1
>> PORT 1# add 100
>> PORT 1# add 101
>> PORT 1# filt ena
```

10. TCP トラフィックを WebLogic Server クラスタから SSL アクセラレータに送るフィルタをコンフィグレーションします。

```
>> # /cfg/slb/filt 102
>> FILTER 102# proto tcp
>> FILTER 102# sport 81
>> FILTER 102# action redir
>> FILTER 101# group 256
>> FILTER 101# ena
```

11. 新しいフィルタをデフォルト フィルタと一緒に Web スイッチの WebLogic Server インスタンスのポートに追加します。

```
>> # /cfg/slb/port 6
>> PORT 6# add 101
>> PORT 6# add 102
>> PORT 6# filt ena
>> # /cfg/slb/port 7
>> PORT 7# add 101
>> PORT 7# add 102
>> PORT 7# filt ena
```

```
>> # /cfg/slb/port 8
>> PORT 8# add 101
>> PORT 8# add 102
>> PORT 8# filt ena
```

12. 新しい設定を有効にして保存します。

```
>> # /cfg/slb
>> Server Load Balancing# on
>> Server Load Balancing# apply
>> Server Load Balancing# save
```

D 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

この節の内容は以下のとおりです。

- 概要
- ロード バランシングと URL 書き換えについて
- WebLogic Server クラスタに関するセッションの永続性のコンフィグレーション

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を順を追って説明している部分もあります。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。

ロード バランシングと URL 書き換えについて

場合によっては、URL の書き換えによるロード バランシングが必要になることがあります。たとえば、`http://www.paradiso.com` という URL のアプリケーションがあり、そのトラフィックを `www.paradiso[0].com` から `www.paradiso[4].com` までの 5 台のサーバにロード バランシングする必要があるような場合です。

一般に、URL の書き換えによるロード バランシングには、複数のソリューションがあります。ただし、ロード バランサとして BIG-IP を使用する場合は、URL を書き換えることはできません。

WebLogic Server クラスタに関するセッションの永続性のコンフィグレーション

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。
8. 変更を適用して、ユーティリティを終了します。