BEA WebLogic Server™

BEA WebLogic Express™

BEA WebLogic Server 6.1 の紹介

BEA WebLogic Server バージョン 6.1
マニュアルの日付：2002 年 6 月 24 日

著作権

限定的権利条項

商標または登録商標

BEA WebLogic Server の紹介

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
860-001002-008	2002 年 6 月 24 日	BEA WebLogic Server 6.1

目次

このマニュアルの内容

1. WebLogic Server とは

2. WebLogic Server のサービス

索引

このマニュアルの内容

このマニュアルでは、Sun Microsystems の Java 2 Enterprise Edition（J2EE）に関連する基本概念を紹介します。また、BEA WebLogic Server の機能について概説し、WebLogic Server プラットフォーム上で動作する J2EE 準拠アプリケーションのアーキテクチャについて説明します。

このマニュアルの構成は次のとおりです。

- 第 1 章「WebLogic Server とは」では、WebLogic Server を紹介し、BEA WebLogic Server 製品について説明します。
- 第 2 章「WebLogic Server のサービス」では、Web、セキュリティ、クラスタ化、および管理サービスなど、WebLogic Server が提供する基本的なサービスについて概説します。

対象読者

このマニュアルは、J2EE プラットフォームを使用して e- コマース アプリケーションを構築するアプリケーション開発者を主な対象としています。Web 技術、オブジェクト指向プログラミング技術、および Java プログラミング言語に読者が精通していることを前提として書かれてます。

このマニュアルは、エンタープライズ ソフトウェア システムでのアプリケーションの位置、J2EE アプリケーションの基本的な要件とアーキテクチャ、およびそれらの要件を WebLgoic Server でどのように実現しているかについて説明しているため、開発者以外の読者にも参考になります。

e-docs Web サイト

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

このマニュアルの印刷方法

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

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

Adobe Acrobat Reader 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。

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
〔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：
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

表記法	適用
斜体の等幅テキスト	コード内の変数を示す。 例 : `String *CustomerName*;`
すべて大文字のテキスト	デバイス名、環境変数、および論理演算子を示す。 例 : LPT1 BEA_HOME OR
{ }	構文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 :
	`java utils.MulticastTest -n *name* -a *address*`
	`[-p *portnumber*] [-t *timeout*] [-s *send*]`

|	構文の中で相互に排他的な選択肢を区切る。実際には、この記号は入力しない。
	例 :
	`java weblogic.deploy [list|deploy|undeploy|update]`
	`password {application} {source}`

...	コマンドラインで以下のいずれかを示す。 ♦ 引数を複数回繰り返すことができる ♦ 任意指定の引数が省略されている ♦ パラメータや値などの情報を追加入力できる 実際には、この省略符号は入力しない。
. . .	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 とは

以下の節では、WebLogic Server の e- コマース プラットフォームについて概説します。

- はじめに
- WebLogic Express とは
- WebLogic Server のアプリケーション アーキテクチャ
- ソフトウェア コンポーネントの層
- アプリケーション ロジックのレイヤ

はじめに

今日のビジネス環境では、新市場への参入を促進し、顧客の確保を手助けし、また新製品や新サービスをいち早く紹介することを可能にする Web および e- コマース アプリケーションが求められています。こうした新しいソリューションを構築してデプロイするためには、あらゆるタイプのユーザを接続して利用できるようにすると同時に、企業データ、メインフレーム アプリケーション、およびその他の企業アプリケーションを統合して強力でフレキシブルなエンド ツーエンドの e- コマース ソリューションを構築できる、実績があり信頼性の高い e- コマース プラットフォームが必要です。さらに、重要度の高い企業規模のコンピューティングを処理するためには、パフォーマンス、スケーラビリティ、および高可用性を提供しなくてはなりません。

産業界をリードする e- コマース トランザクション プラットフォームである WebLogic Server は、信頼性が高く、セキュリティが確保され、スケーラブルでかつ管理の容易なアプリケーションの開発とデプロイメントを可能にします。システム レベルでの詳細は WebLogic Server で管理するため、ユーザはビジネス ロジックとプレゼンテーションに集中することができます。

J2EE プラットフォーム

WebLogic Server には、Java 2 Platform、Enterprise Edition（J2EE）技術が組み込まれています。J2EE は、Java プログラミング言語に基づいた多層エンタープライズ アプリケーションを開発するための標準プラットフォームです。J2EE を構成する技術は、BEA Systems をはじめとするソフトウェア ベンダと Sun Microsystems によって共同開発されました。

J2EE アプリケーションは、標準化され、モジュール化されたコンポーネントに基づいています。WebLogic Server では、これらのコンポーネント用にあらゆるサービスが用意され、細かなアプリケーションの動作を、プログラミングを必要とせずに自動的に処理します。

WebLogic Server 6.1 と J2EE 1.2 および J2EE 1.3

BEA WebLogic Server 6.1 は、高度な J2EE 1.3 の機能を実装する最初の e- コマース トランザクション プラットフォームです。J2EE のルールに準拠するために、BEA Systems では 2 つの別個のダウンロードを用意しています。1 つは J2EE 1.3 の機能が有効になっているもの、1 つは J2EE 1.2 の機能に制限されているものです。いずれのダウンロードもコンテナは同じですが、利用可能な API だけ異なります。

J2EE 1.2 の機能に加えて J2EE 1.3 の機能を備える WebLogic Server 6.1

このダウンロードでは、WebLogic Server はデフォルトで J2EE 1.3 の機能を使用して動作します。それらの機能には、EJB 2.0、JSP 1.2、サーブレット 2.3、および J2EE コネクタ アーキテクチャ 1.0 が含まれます。J2EE 1.3 の機能を有効にして WebLogic Server 6.1 を実行しても、J2EE 1.2 アプリケーションはそのままフルサポートされます。J2EE 1.3 機能の実装では、適切な API 仕様の最終ではないバージョンが使用されます。したがって、J2EE 1.3 の新機能を使用する BEA WebLogic Server 6.1 用に開発されたアプリケーション コードは、BEA WebLogic Server の今後のリリースでサポートされる J2EE 1.3 プラットフォームとは互換性を持たない場合があります。

J2EE 1.2 認定の WebLogic Server 6.1

このダウンロードでは、WebLogic Server はデフォルトで J2EE 1.3 機能が無効な状態で動作し、J2EE 1.2 の仕様と規定に完全に準拠します。

J2EE 1.2 および 1.3 の製品 CD インストーラ

配布キットは両方とも、http://www.beasys.co.jp/evaluation/index.html からダウンロードできるほか、WebLogic Server 6.1 の製品 CD でも提供されます。Windows マシンでは、J2EE 1.3 機能の有効な WebLogic Server のインストーラは CD を挿入すると自動的に開始されます。

XML 実装

WebLogic Server では、WebLogic Server に適用可能な Extensible Markup Language（XML）テクノロジと、WebLogic Server に基づく XML アプリケーションが統合されています。Standard Generalized Markup Language（SGML）の簡略化されたバージョンである XML は、文書内のデータの内容と構造を記述するものであり、インターネット上でコンテンツを配信する際の業界標準となっています。通常、XML は J2EE アプリケーションとクライアント アプリケーション間、または J2EE アプリケーションのコンポーネント間におけるデータ交換フォーマットとして使用されます。WebLogic Server XML サブシステムでは、XML ファイルを処理および変換するために、標準パーサ、WebLogic FastParser、XSLT トランスフォーマ、DTD、および XML スキーマの使用をサポートしています。

分散異種環境におけるアプリケーションのデプロイメント

WebLogic Server は、分散異種コンピューティング環境にまたがる、ミッション クリティカルな e- コマース アプリケーションを開発しデプロイする上で欠くことのできない機能を提供します。以下のような機能が含まれています。

- 標準のリーダーシップ ― エンタープライズ Java の包括的なサポートにより、アプリケーション コンポーネントの実装やデプロイメントを容易にします。WebLogic Server は、独自に開発された J2EE 公認の Java アプリケーション サーバとしては最初のものです。
- 豊富なクライアント オプション ― WebLogic Server は、HTTP を使用する Web ブラウザおよびその他のクライアント、RMI（Remote Method Invocation）または IIOP（Internet Inter-ORB Protocol）を使用する Java クライアント、および WAP（Wireless Access Protocol）を使用するモバイル デバイスをサポートしています。BEA および他社製のコネクタを使うと、任意のクライアントやレガシー アプリケーションを WebLogic Server アプリケーションと実質上連携させることができます。
- 柔軟な Web サービス ― WebLogic Server は、異種分散アプリケーションのコンポーネントとして Web サービスをデプロイするための、強力なプラットフォームを提供します。Web サービスはクロスプラットフォーム、クラス言語データ モデル（XML）を使用して、多様なハードウェアおよびソフトウェア プラットフォームにおけるアプリケーション コンポーネント間の相互運用性をもたらします。

  WebLogic Server 6.1 は、XML ベースの仕様である Web Services Description Language（WSDL）1.1 を使用して Web サービスを 記述しています。WebLogic Web サービスでは、Simple Object Access Protocol（SOAP）1.1 をメッセージ フォーマットとして使用し、HTTP を接続プロトコルとして使用します。
- エンタープライズ e- ビジネス スケーラビリティ ― エンタープライズ JavaBean のビジネス コンポーネント、および動的な Web ページに対応する WebLogic Server クラスタ化、バックエンドのリソース プーリング、接続共有といったメカニズムを採用することにより、重要なリソースを効率的に使用し、高可用性を実現します。

- 堅牢な管理 ― WebLogic Server では、WebLogic Server サービスのコンフィグレーションやモニタを行うために、Web ベースの Administration Console を提供しています。スクリプトを使って WebLogic Server を管理する場合に便利な、コンフィグレーション用のコマンドライン インタフェースも用意されています。
- e- コマースに対応したセキュリティ ― WebLogic Server では、WebLogic Server、クライアント、および他のサーバとの間で転送されるデータを暗号化するために、セキュア ソケット レイヤ（SSL）をサポートしています。WebLogic セキュリティ レルムでは、すべての WebLogic Server サービスに対してユーザ認証および認可を行えます。Lightweight Directory Access Protocol（LDAP）サーバなどの外部セキュリティ ストアを WebLogic レルムに適合させて、エンタープライズ用に 1 つのサインオンを使用することができます。Security サービス プロバイダ インタフェースを使用すると、WebLogic Security サービスを拡張したり、アプリケーションに WebLogic Security 機能を実装したりできます。
- 開発およびデプロイメントにおける最大限の柔軟性 ― WebLogic Server は、データベース、開発ツール、およびその他の環境を強固に統合し、サポートします。

WebLogic Express とは

BEA WebLogic Express™ は、Web および無線アプリケーションに動的コンテンツとデータを提供する、スケーラブルなプラットフォームです。WebLogic Express は、プレゼンテーション サービス、および WebLogic Server からのデータ ベース アクセス サービスを備えており、これにより開発者は対話型のアプリケーションやトランザクション対応の e- ビジネス アプリケーションを迅速に開発したり、既存のアプリケーションにプレゼンテーション サービスを提供したりできます。

WebLogic Express には、WebLogic JDBC 機能、JavaServer Pages （JSP）、Remote Method Invocation（RMI）、および Web サーバ機能といった、WebLogic Server で使用できるサービスや API が多く用意されています。

WebLogic Express は、WebLogic Server とは異なり、エンタープライズ JavaBeans（EJB）、Java Message Services（JMS）、またはトランザクション用の 2 フェーズ コミット プロトコルは提供していません。

WebLogic Server のアプリケーション アーキテクチャ

WebLogic Server は、アプリケーションサーバ、すなわち多層分散エンタープライズ アプリケーションを開発しデプロイするためのプラットフォームです。WebLogic Server は、Web サーバ機能、ビジネス コンポーネント、およびバックエンドのエンタープライズ システムへのアクセスといった、アプリケーション サービスを一元化します。キャッシングや接続プーリングなどの技術を使用して、リソースを有効に利用し、アプリケーションの性能を改善します。また、WebLogic Server は、エンタープライズ レベルのセキュリティ、および強力な管理機能を備えています。

WebLogic Server は、多層（あるいは *n* 層）アーキテクチャの中間層で動作します。多層アーキテクチャとは、コンピューティング システムを構成するソフトウェア コンポーネントが、相互に、かつハードウェア、ネットワーク、およびユーザに関連して動作する場所を決定するものです。各ソフトウェア コンポーネントの最適な位置を選択すると、アプリケーションをより速く開発でき、デプロイメントや管理が容易になります。また、パフォーマンス、使用率、セキュリティ、スケーラビリティ、および信頼性をより自在に制御することができます。

WebLogic Server は、Java Enterprise 標準である J2EE を実装しています。Java は、ネットワークと相性のよいオブジェクト指向プログラミング言語であり、J2EE には、分散オブジェクトを開発するためのコンポーネント技術が含まれています。この機能により、WebLogic Server のアプリケーション アーキテクチャに、アプリケーション ロジックのレイヤリングという第 2 の次元が追加され、各レイヤは WebLogic Server の J2EE 技術の中に選択的にデプロイされます。

以降の 2 つの節では、WebLogic Server のアーキテクチャを表す 2 つの概念、ソフトウェアの層とアプリケーション ロジックのレイヤについて説明します。

ソフトウェア コンポーネントの層

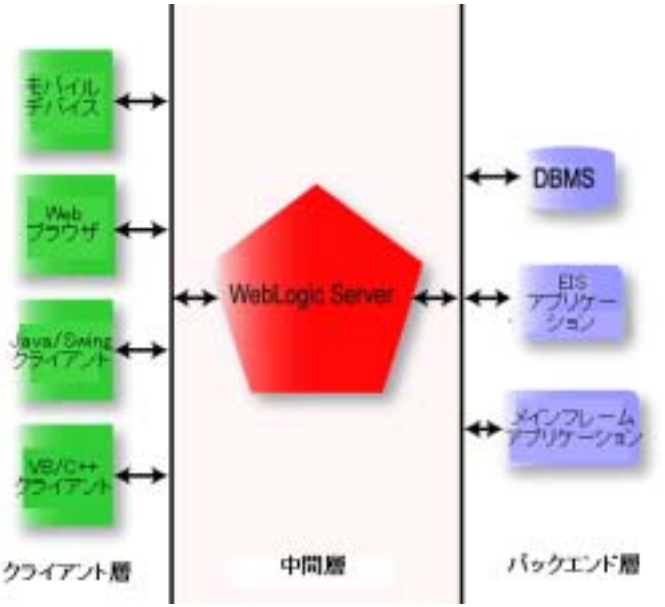
多層アーキテクチャのソフトウェア コンポーネントは、以下の 3 層で構成されています。

- クライアント層には、Web ブラウザやネットワーク対応アプリケーション プログラムなどの、ユーザが実行するプログラムが含まれます。これらのプログラムは、実質的にはどのプログラミング言語でも記述できます。
- 中間層には、WebLogic Server、および既存の Web サーバやプロキシ サーバといったクライアントが直接アクセスするその他のサーバが含まれます。
- バックエンド層には、データベース システム、メインフレーム、およびレガシー アプリケーションなどのエンタープライズ リソースと、パッケージ化されたエンタープライズ リソース プランニング（ERP）アプリケーションが含まれます。

クライアント アプリケーションは直接 WebLogic Server にアクセスするか、別の Web サーバまたはプロキシ サーバを経由してアクセスします。一般には、WebLogic Server がクライアントの代わりにバックエンド サービスと接続します。しかし、クライアントが多層 JDBC ドライバを使用して、バックエンド サービスに直接アクセスしてもかまいません。

図 1-1 に、WebLogic Server アーキテクチャの 3 つの層を示します。

図 1-1 3 層アーキテクチャ



クライアント層のコンポーネント

WebLogic Server のクライアントは、標準のインタフェースを使用して WebLogic Server のサービスにアクセスします。WebLogic Server は Web サーバ機能を完全装備しているため、Web ブラウザは、Web の標準 HTTP プロトコルを使用して、WebLogic Server のページをリクエストできます。WebLogic Server サーブレットと JavaServer Pages (JSP) は、高度な e- コマース Web アプリケーションで求められる、動的でパーソナライズされた Web ページを生成します。

Java で記述するクライアント プログラムには、Java Swing クラスを使用して構築される高度な対話型のグラフィカル ユーザ インタフェースを備えることができます。標準の J2EE API を使用して WebLogic Server のサービスにアクセスすることもできます。

WebLogic Server にサーブレットや JSP ページをデプロイすることにより、Web ブラウザ クライアントではこれらのサービスをすべて利用できます。

Visual Basic、C++、Java、またはその他のプログラミング言語で記述された CORBA 対応のクライアント プログラムは、WebLogic RMI-IIOP を使用して、WebLogic Server エンタープライズ JavaBean および RMI (Remote Method Invocation) クラスを実行することができます。どのプログラミング言語で記述されたクライアント アプリケーションでも、HTTP プロトコルをサポートしていれば、サーブレットを介して任意の WebLogic Server サービスにアクセスできます。

中間層のコンポーネント

中間層には、WebLogic Server、その他の Web サーバ、ファイアウォール、およびクライアントと WebLogic Server の間のトラフィックを仲介するプロキシ サーバが含まれます。BEA の モバイル コマース ソリューションの一部である Nokia WAP サーバは、無線デバイスと WebLogic Server との間の接続を提供する中間層サーバの一例です。

多層アーキテクチャに基づいたアプリケーションでは、中間層で、信頼性、スケーラビリティ、および高いパフォーマンスが要求されます。そのため、中間層に選択するアプリケーション サーバは、システムの成功にとってきわめて重要です。

WebLogic Server クラスタを使用すると、連携している複数の WebLogic Server に、クライアントのリクエストやバックエンド サービスを分散できます。クライアント層のプログラムは、単一の WebLogic Server のようにクラスタにアクセスします。負荷が大きくなる場合は、WebLogic Server をクラスタに追加して、処理を共有させることができます。クラスタは、選択可能なロードバランシング アルゴリズムを使用して、そのリクエストを処理できる WebLogic Server をクラスタ内で選択します。

リクエストが失敗すると、リクエストされたサービスを提供する別の WebLogic Server が引き継ぐことができます。フェイルオーバは可能な限り透過的であり、障害から回復するために記述しなければならないコードの量を最小限に抑えます。たとえば、WebLogic Server でリクエスト処理が失敗した場合、クライアントのセッションがセカンダリ サーバ上でその処理を再開できるように、サーブレット セッション ステートをセカンダリ WebLogic Server にレプリケートしておくことができます。WebLogic EJB、JMS、JDBC、および RMI サービスは、すべて、クラスタ化機能を使用して実装されます。

バックエンド層のコンポーネント

バックエンド層には、クライアントから WebLogic Server を介してのみアクセスできるサービスが含まれます。バックエンド層のアプリケーションは、最も価値が高く、ミッションクリティカルなエンタープライズ リソースである場合が多いため、WebLogic Server では、エンド ユーザによる直接的なアクセスを制限することによって、これらを保護します。接続プールやキャッシングなどの技術によって、WebLogic Server は、バックエンドのリソースを効率的に使用し、アプリケーションの応答を向上させます。

バックエンド サービスには、データベース、エンタープライズ リソース プランニング（ERP）システム、メインフレーム アプリケーション、レガシー エンタープライズ アプリケーション、およびトランザクション モニタが含まれます。Sun Microsystems の Java コネクタ アーキテクチャ（JCA）仕様を使用して、既存のエンタープライズ アプリケーションをバックエンド層に統合できます。WebLogic Server では、統合されたバックエンド アプリケーションに Web インタフェースを簡単に追加できます。

データベース管理システムは、ほとんどすべての WebLogic Server アプリケーションが必要とする、最も代表的なバックエンド サービスです。WebLogic EJB および WebLogic JMS は、通常、永続的なデータをバックエンド層のデータベースに格納します。

WebLogic Server で定義される JDBC 接続プールは、あらかじめ指定された数のデータベース接続を開きます。データベース接続は、一度開かれると、データベースへのアクセスが必要なすべての WebLogic Server アプリケーションによって共有されます。接続の確立に関連する負荷の大きなオーバーヘッドは、クライアントのリクエストごとに 1 回ではなく、プールの各接続ごとに 1 回だけ発生します。WebLogic Server は、データベース接続を監視し、必要に応じて接続をリフレッシュし、アプリケーションに対して信頼できるデータベース サービスを保証します。

BEA WebLogic Enterprise™ システムへのアクセスを提供している WebLogic Enterprise Connectivity および BEA Tuxedo システムへのアクセスを提供している JOLT for WebLogic Server でも、システムのパフォーマンスを強化するために接続プールを使用しています。

アプリケーション ロジックのレイヤ

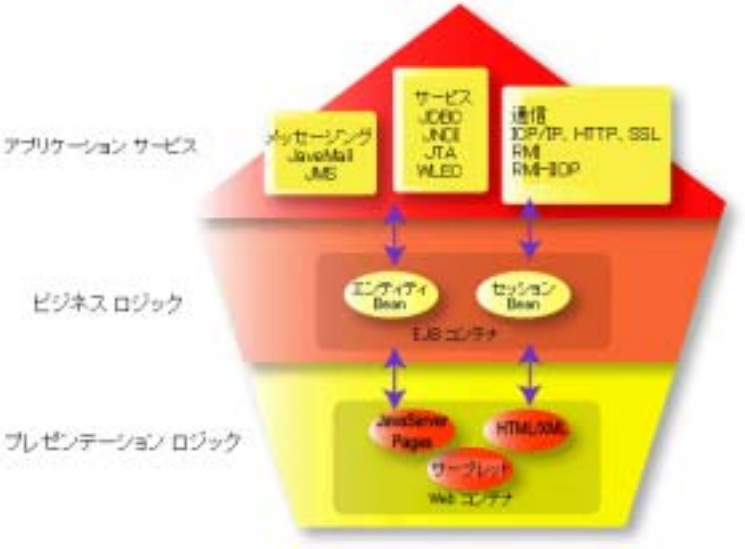
WebLogic Server は、J2EE のコンポーネント技術およびサービスを実装しています。J2EE コンポーネント技術には、サーブレット、JSP ページ、およびエンタープライズ JavaBean（EJB）が含まれます。J2EE サービスには、標準のネットワーク プロトコル、データベース システム、およびメッセージング システムへのアクセスが含まれます。WebLogic Server のアプリケーションを構築するには、必要に応じてこれらのサービス API を使用して、コンポーネントを作成し組み立てる必要があります。

コンポーネントは、WebLogic Server の Web コンテナまたは EJB コンテナ内で実行されます。コンテナでは、J2EE 仕様で定義されたライフサイクルのサポートやサービスを提供しているため、構築するコンポーネントでは、下層部の詳細を扱う必要はありません。

Web コンポーネントは、ブラウザ ベースの J2EE アプリケーションに対してプレゼンテーション ロジックを提供します。EJB コンポーネントは、ビジネスのオブジェクトやプロセスをカプセル化します。Web アプリケーションと EJB は、JDBC、JMS（Java Messaging Service）、および JTA（Java Transaction API) などの J2EE アプリケーション サービスの上に構築されています。

図 1-2 に、WebLogic Server のコンポーネント コンテナおよびアプリケーション サービスを示します。

図 1-2 アプリケーション ロジックのレイヤ



以下の節では、プレゼンテーション レイヤ、ビジネス ロジック、アプリケーション サービスについて説明します。

プレゼンテーション ロジックのレイヤ

プレゼンテーション レイヤには、アプリケーションのユーザ インタフェースや表示ロジックが含まれます。ほとんどの J2EE アプリケーションは、クライアント マシン上の Web ブラウザを使用します。クライアント プログラムをすべてのユーザのコンピュータにデプロイするよりも容易なためです。この場合、プレゼンテーション ロジックは WebLogic Server の Web コンテナです。ただし、どのプログラミング言語で記述されたクライアント プログラムでも、HTML を表示するためのロジックまたは独自のプレゼンテーション ロジックを保持している必要があります。Web サービスにアクセスするクライアントは、呼び出そうとしている Web サービスを示す SOAP メッセージをアセンブルし、その本文の中に、必要なデータを含める必要があります。

Web ブラウザ クライアント

標準の Web 技術を使って構築される Web ベースのアプリケーションは、アクセスや保守が容易で、ポータブルです。Web ブラウザ クライアントは、e- コマース アプリケーションのための標準仕様です。

Web ベースのアプリケーションでは、HTML ドキュメント、JavaServer Pages（JSP）、およびサーブレットによってユーザ インタフェースが表現されます。Web ブラウザには、HTML の記述から Web ページをユーザのコンピュータに表示するためのロジックが含まれています。

JavaServer Pages（JSP）とサーブレットは、密接に関連しています。どちらも、呼び出されるたびに、WebLogic Server 上で Java コードを実行して、動的な Web コンテンツを作成します。2 つの相違点は、JSP が拡張 HTML で記述されるのに対し、サーブレットは Java プログラミング言語を使用して記述される点です。

HTML の知識があり HTML エディタまたはデザイナを使用して作業することに慣れている Web デザイナにとって、JSP は便利です。サーブレットはすべて Java で記述されるため、Web デザイナよりも Java プログラマに適しています。サーブレットを記述するには、HTTP プロトコルおよび Java プログラミングに関する知識が多少必要になります。サーブレットは、リクエスト オブジェクト内の HTTP リクエストを受け取って、通常はその応答オブジェクトに HTML または XML を記述します。

JSP ページは、WebLogic Server 上で実行される前に、サーブレットに変換されます。つまり JSP ページとサーブレットでは、同じものに対する表現方法が異なっています。JSP ページは、HTML がデプロイされる場合と同じ方法で WebLogic Server にデプロイされます。`.jsp` ファイルは、WebLogic Server が提供するディレクトリにコピーされます。クライアントが `.jsp` ファイルを要求すると、WebLogic Server は、そのページがコンパイル済みかどうか、また最後にコンパイルされてから変更されているかどうかを確認します。必要に応じて、`jsp` ファイルから Java サーブレット コードを生成する WebLogic JSP コンパイラを呼び出して、Java コードを Java クラス ファイルにコンパイルします。

非ブラウザ クライアント

Web ブラウザでないクライアント プログラムでは、ユーザ インタフェースを表示するために独自のコードを用意する必要があります。非ブラウザ クライアントには、通常、独自のプレゼンテーション ロジックと表示ロジックが含まれて

いて、ビジネス ロジックとバック エンド サービスへのアクセスに関してのみ、WebLogic Server に依存します。このため非ブラウザ クライアントは、ブラウザ ベースのクライアントよりも開発やデプロイが難しく、インターネットベースの e- コマース アプリケーションに対して適合しにくくなります。

Java で記述されたクライアント プログラムは、Java RMI（Remote Method Invocation) を介して任意の WebLogic Server サービスを使用できます。RMI を使用すると、クライアント プログラムでは、クライアント内のローカル オブジェクトと同じように、WebLogic Server オブジェクトを操作できます。RMI はネットワーク経由の呼び出しの詳細を隠蔽するため、J2EE クライアント コードとサーバ サイド コードはよく似ています。

Java のプログラムでは、Java Swing クラスを使用して、強力で移植性の高いユーザ インタフェースを作成することができます。Java を使用することで移植性の問題を回避することはできますが、クライアント上に WebLogic Server クラスがインストールされない限り、RMI 経由で WebLogic Server のサービスを使用することはできません。このため、Java RMI クライアントは、e- コマース アプリケーションには適していません。ただし、内部ネットワークを使用してインストールと保守が行えるエンタープライズ アプリケーションでは、Java RMI クライアントを効率的に使用できます。

Java 以外の言語で記述されたクライアント プログラム、および RMI 経由で WebLogic Server オブジェクトを使用しない Java クライアント プログラムは、HTTP または RMI-IIOP を使用して WebLogic Server にアクセスできます。

HTTP は Web の標準プロトコルです。HTTP により、クライアントは、サーバにさまざまなタイプのリクエストを送り、パラメータを渡すことができます。WebLogic Server 上のサーブレットは、任意の WebLogic Server サービスを使用して、クライアントのリクエストを調べたり、リクエストからパラメータを取得したり、クライアントへの応答を用意したりできます。たとえば、あるサーブレットが XML ビジネス ドキュメントを伴うクライアントプログラムに応答するとします。そのような場合、アプリケーションはサーブレットを他の WebLogic Server サービスへのゲートウェイとして使用できます。

WebLogic RMI-IIOP を使うと、CORBA 対応のプログラムは、WebLogic Server エンタープライズ Bean および RMI クラスを CORBA オブジェクトとして実行できます。WebLogic Server の RMI および EJB コンパイラは RMI クラスおよびエンタープライズ Bean 用の IDL（インタフェース定義言語）を生成することができます。こうして生成された IDL がコンパイルされて、ORB（Object Request

Broker）用のスケルトンとクライアント プログラム用のスタブが作成されます。WebLogic Server は、受信した IIOP リクエストを解析して、RMI 実行時システムにディスパッチします。

Web サービス クライアント

WebLogic Web サービスを呼び出すクライアント アプリケーションは、Java、Microsoft SOAP Toolkit などの任意のテクノロジを使用して記述できます。クライアント アプリケーションは、呼び出す Web サービスに関する SOAP メッセージをアセンブルして必要なすべてのデータを SOAP メッセージの本文に含めます。次にクライアントは SOAP メッセージを HTTP/HTTPS を介して WebLogic Server に送信します。そこで Web サービスが実行され、SOAP メッセージが HTTP/HTTPS を介してクライアントに送り返されます。

WebLogic Server は、Web サービスの作成に必要なサーブレットと関連インフラストラクチャのセットである Web サービス実行時コンポーネントを提供します。実行時の要素の 1 つは、クライアントからの SOAP リクエストを処理するサーブレットのセットです。これらのサーブレットは、WebLogic Server 配布キットに含まれているため、記述する必要はありません。

Java ベースの Web サービス クライアントの場合、WebLogic Server はオプションの Java クライアント JAR ファイルも提供します。JAR ファイルには、WebLogic Web サービス クライアント API や WebLogic FastParser など、クライアント アプリケーションが WebLogic Web サービスを呼び出すために必要なものがすべて含まれています。他の Java WebLogic Server クライアントとは異なり、Web サービス クライアントには `weblogic.jar` ファイルを含める必要がないため、シン クライアント アプリケーションを作成できます。

ビジネス ロジックのレイヤ

エンタープライズ JavaBean は、J2EE アプリケーションのビジネス ロジック コンポーネントです。WebLogic Server の EJB コンテナは、エンタープライズ Bean のホストになり、ライフサイクル管理、およびキャッシング、永続性、トランザクション管理といったサービスを提供します。

エンタープライズ Bean には、エンティティ Bean、セッション Bean、およびメッセージ駆動型 Bean の 3 タイプがあります。次の節では、それぞれのタイプについて詳しく説明します。

エンティティ Bean

エンティティ Bean は、顧客、口座、または倉庫のアイテムといったデータを含むオブジェクトを表します。エンティティ Bean には、データ値およびそれらの値に対して呼び出されるメソッドが含まれます。これらの値は、（JDBC を使用して）データベース、または、何らかのデータ ストアに保存されます。エンティティ Bean は、他のエンタープライズ Bean が関連するトランザクションやトランザクション対応のサービスに参加できます。

エンティティ Bean は、たいていデータベース内のオブジェクトにマップされます。エンティティ Bean は、テーブル内の 1 行、行内の 1 カラム、または、テーブル全体やクエリ結果を表すことができます。各エンティティ Bean には、Bean の検索、取得、および保存に使用されるユニークな主キーが関連付けられています。

エンティティ Bean では、次のいずれかを使用できます。

- Bean 管理による永続性 ― Bean に、永続的な値を取得および保存するためのコードが含まれます。
- コンテナ管理による永続性 ― EJB コンテナが Bean に代わって値のロードおよび保存を行います。

コンテナ管理による永続性を使用すると、WebLogic EJB コンパイラで JDBC サポート クラスを生成して、エンティティ Bean をデータベース内の行にマップできます。コンテナ管理による永続性では、その他のメカニズムも利用できます。たとえば、WebGain の TOPLink for BEA WebLogic Server は、オブジェクト リレーショナル データベースに対し永続性を提供します。

エンティティ Bean は、多くのクライアントおよびアプリケーションで共有できます。エンティティ Bean のインスタンスは、任意のクライアントのリクエストで作成されますが、そのクライアントの接続が解除されても消滅しません。エンティティ Bean のインスタンスは、クライアントがアクティブに使用している限り生存します。EJB コンテナは、使用されなくなった Bean に対してパッシベーションを行う場合があります。つまり、生存しているインスタンスであっても、サーバから削除される可能性があります。

セッション Bean

セッション Bean は、1 つのクライアントを提供する、一時的な EJB インスタンスです。セッション Bean は、データよりもアクションを具体化するため、手続き的なロジックを実装する傾向があります。

EJB コンテナは、クライアントからのリクエストを受けて、セッション Bean を作成します。そして、そのクライアントがその Bean との接続を維持している間のみ、その Bean を維持します。セッション Bean は永続的ではありませんが、必要に応じて、永続ストレージにデータを保存できます。

セッション Bean は、ステートレスにもステートフルにもなります。ステートレス セッション Bean は、呼び出しの間クライアント固有の状態を維持しないため、どのクライアントでも使用できます。ドキュメントをプリンタに送信したり、読み込み専用のデータをアプリケーション内に取得したりといった、セッションのコンテキストに依存しないサービスへのアクセスを提供できます。

ステートフル セッション Bean は、特定のクライアントの代わりに状態を維持します。ステートフル セッション Bean を使うと、ワークフロー プロセスにおける順序の組み立てやドキュメントのルーティングといった、プロセスの管理ができます。クライアントとの複数の対話の間、状態を蓄積して維持することができるため、セッション Bean は、アプリケーション内の制御オブジェクトとしてよく使用されます。セッション Bean は永続的ではないため、単一のセッション内で処理を完了し、JDBC、JMS、またはエンティティ Bean を使用してその処理を永久に記録しておく必要があります。

メッセージ駆動型 Bean

EJB 2.0 仕様で導入されたメッセージ駆動型 Bean は、JMS メッセージ キューから受け取った非同期メッセージを処理するエンタープライズ Bean です。JMS からメッセージが転送されると、メッセージ駆動型 Bean は、プールからインスタンスを選択してメッセージを処理します。

メッセージ駆動型 Bean は、WebLogic Server の EJB コンテナで管理されます。メッセージ駆動型 Bean は、ユーザ駆動型アプリケーションから直接呼び出されることはありません。そのため、EJB ホームを使用してアプリケーションからメッセージ駆動型 Bean にアクセスすることはできません。ただし、ユーザ駆動型アプリケーションでも、その Bean の JMS キューにメッセージを送信することにより、メッセージ駆動型 Bean を間接的にインスタンス化できます。

アプリケーション サービスのレイヤ

WebLogic Server では、コンポーネントが低レベルな実装の詳細に関わることなくビジネス ロジックに集中できるようにする、基盤となるサービスを提供します。WebLogic Server は、ネットワーク、認証、認可、永続性、および EJB やサーブレット用のリモート オブジェクトへのアクセスを扱います。標準 Java API は、データベースやメッセージング サービスといった、アプリケーションが使用できるその他のサービスへの移植性の高いアクセスを提供します。

ネットワーク通信技術

クライアント アプリケーションは、TCP/IP に基づいた標準のネットワーキング プロトコルを使用して、WebLogic Server に接続します。WebLogic Server は、Uniform Resource Identifier (URI) の一部として指定できるネットワーク アドレスで、接続リクエストをリスンします。

URI は、標準化された文字列で、インターネットを含むネットワーク上のリソースを指定します。URI には、プロトコルを指定する方式、サーバのネットワーク アドレス、要求されたリソース名、およびパラメータ（省略可能）が含まれます。Web ブラウザに入力する URL、たとえば `http://www.bea.com/index.html` は、最も一般的な URI 形式です。

Web ベースのクライアントは、HTTP プロトコルを使用して WebLogic Server と通信します。Java クライアントは、Java RMI（Remote Method Invocation）を使用して接続します。Java RMI を使用すると、Java クライアントは WebLogic Server のオブジェクトを実行できます。CORBA 対応クライアントは、RMI-IIOP を使用して WebLogic Server の RMI オブジェクトにアクセスします。RMI-IIOP を使用すると、CORBA 対応クライアントは、標準の CORBA プロトコルを使用して WebLogic Server のオブジェクトを実行できます。

以下の表のように、URI の方式は、クライアントと WebLogic Server がネットワーク経由でやりとりするためのプロトコルを決定します。

表 1-1 ネットワーク プロトコル

方式	プロトコル
HTTP	HyperText Transfer Protocol。Web ブラウザおよび HTTP 対応プログラムで使用される。
HTTPS	Hypertext Transfer Protocol 上のセキュア ソケット レイヤ（SSL）。Web ブラウザおよび HTTPS 対応のクライアント プログラムで使用される。
T3	Java-to-Java 接続のための WebLogic T3 プロトコル。JNDI、RMI、EJB、JDBC、およびネットワーク接続でのその他の WebLogic のサービス を多重化する。
T3S	セキュア ソケット レイヤ（SSL) での WebLogic T3 プロトコル。
RMI	分散アプリケーションのための標準的な Java 機能である Remote Method Invocation（RMI）。
IIOP	Internet Inter-ORB プロトコル。CORBA 対応 Java クライアントが IIOP を介して WebLogic RMI オブジェクトを実行するのに使用される。その他の CORBA クライアントは、WebLogic Server の URI の代わりに CORBA のネーミング コンテキスト を使用して、WebLogic Server に接続する。
IIOPS	セキュア ソケット レイヤ（SSL) での Internet Inter-ORB プロトコル。
SOAP	WebLogic Web サービスでは、Simple Object Access Protocol（SOAP）1.1 をメッセージ フォーマットとして使用し、HTTP を接続プロトコルとして使用する。

以下の節では、これらのプロトコルの詳細について説明します。

HTTP

World Wide Web の標準プロトコルである HTTP は、要求 / 応答プロトコルです。クライアントは URI が含まれた要求を発行します。URI は、`http://` で始まり、WebLogic Server のアドレス、次に HTML ページ、サーブレット、または JSP ページといった WebLogic Server 上のリソースの名前が続きます。リソース名が省略された場合、WebLogic Server はデフォルトの Web ページ（通常は `index.html`）を返します。HTTP リクエストのヘッダには、コマンド（通常は `GET` または `POST`）が含まれます。HTTP リクエストには、データ パラメータおよびメッセージ コンテンツを含めることができます。

WebLogic Server は、クライアントに結果を返すサーブレットを実行することにより、HTTP リクエストに常に応答します。HTTP サーブレットは、ネットワーク上で受け取った HTTP リクエストのコンテンツにアクセスして、HTTP 準拠の結果をクライアントに返すことができる Java クラスです。

WebLogic Server は、HTML ページに対するリクエストを、組み込みの `File` サーブレットに転送します。`File` サーブレットは、WebLogic Server のファイル システムのドキュメント ディレクトリ内で HTML ファイルを探します。カスタム コード化されたサーブレットへのリクエストが、WebLogic Server 上の対応する Java クラスに対して実行されます。JSP ページへのリクエストが発生すると、WebLogic Server はその JSP ページがまだコンパイルされていない場合はコンパイルして、サーブレットを実行します。サーブレットは結果をクライアントに返します。

T3

T3 は最適化されたプロトコルで、クライアントとそれ以外の WebLogic Server 間も含む、WebLogic Server と他の Java プログラムとの間のデータ転送に使用されます。WebLogic Server は、接続された個々の Java 仮想マシン（JVM）を追跡して、JVM に対するすべてのトラフィックを実行できる単一の T3 接続を作成します。

たとえば、Java クライアントが WebLogic Server 上のエンタープライズ Bean および JDBC 接続プールにアクセスすると、1 つのネットワーク接続が WebLogic Server の JVM とクライアントの JVM との間に確立されます。T3 プロトコルは 1 つの接続上のパケットを見えない形で多重化するため、EJB および JDBC のサービスでは、専用のネットワーク接続を単独で使用しているかのように記述することができます。

T3 は不要なネットワーク接続イベントを回避し、OS のリソースをほとんど使用しないため、Java-to-Java アプリケーションにとって効率的なプロトコルです。また、このプロトコルにはパケット サイズを最小限に抑えるす内部の機能があります。

RMI

Remote Method Invocation（RMI）は、分散アプリケーションのための標準的な Java 機能です。RMI を使うと、「サーバ」と呼ばれる Java プログラムが Java オブジェクトをパブリッシュし、「クライアント」と呼ばれるもう 1 つの Java プログラムがそのオブジェクトを実行できます。ほとんどのアプリケーションでは、WebLogic Server が RMI サーバで、Java クライアント アプリケーションがクライアントになります。ただし、RMI では任意の Java プログラムがサーバの役割を果たせるため、この役割を逆にすることもできます。

RMI アーキテクチャは、CORBA アーキテクチャと似ています。リモート オブジェクトを作成するには、リモート クライアントが実行するメソッドを定義した Java クラス用のインタフェースをプログラマが記述します。WebLogic Server の RMI コンパイラ `rmic` を使用してこのインタフェースを処理し、RMI スタブおよびスケルトン クラスを生成します。リモート クラス、スタブ、およびスケルトンが WebLogic Server にインストールされます。

Java クライアントは、この節で後述する Java Naming and Directory Interface (JNDI) を使用して、WebLogic Server のリモート オブジェクトをルックアップします。JNDI は、WebLogic Server への接続を確立し、リモート クラスをルックアップして、スタブをクライアントへ返します。

クライアントは、リモート クラス上で直接メソッドを実行しているように、スタブ メソッドを実行します。スタブ メソッドは呼び出しを用意し、それをネットワーク経由で WebLogic Server 内のスケルトン クラスに転送します。

WebLogic Server では、スケルトン クラスがリクエストを復元し、サーバ サイド オブジェクト上でメソッドを実行します。スケルトン クラスは、その結果をパッケージ化してクライアント サイドのスタブに返します。

WebLogic EJB および Java クライアントから利用できるその他のサービスが RMI 上に構築されます。EJB はビジネス オブジェクトをよりよく抽象化できるため、ほとんどのアプリケーションでは、RMI を直接使用するよりも、EJB を使用するほうがよいでしょう。さらに、WebLogic Server の EJB コンテナには、キャッシング、永続性、ライフサイクル管理といった、リモートクラスでは自動的に利用できない拡張機能があります。

RMI-IIOP

Remote Method Invocation over Internet Inter-ORB Protocol（RMI-IIOP）は、CORBA クライアント プログラムでエンタープライズ Bean を含む WebLogic RMI のオブジェクトを実行できるようにするプロトコルです。RMI-IIOP は、Object Management Group（`http://www.omg.com`）が策定した 2 つの仕様に基づいています。

- Java-to-IDL マッピング
- Objects-by-Value

Java-to-IDL 仕様は、Interface Definition Language（IDL）が Java インタフェースからどのように派生するかについて定義しています。WebLogic Server の RMI および EJB コンパイラでは、RMI および EJB オブジェクトをコンパイルする際に IDL を生成するオプションがあります。IDL API コンパイラを使用して IDL をコンパイルすると、CORBA クライアントに必要なスタブを生成できます。

Objects-by-Value 仕様は、Java と CORBA の間で複合データ型をどのようにマップするかを定義します。Objects-by-Value を使用するには、CORBA クライアントは、CORBA 2.3 でサポートされている Object Request Broker（ORB) を使用する必要があります。CORBA 2.3 仕様 の ORB がない場合は、CORBA クライアントは、Java プリミティブ データ型のみ使用できます。

SSL

HTTP および T3 プロトコルとの間で交換されるデータは、セキュア ソケット レイヤ（SSL）プロトコルを使用して暗号化できます。SSL を使用することで、認証されたサーバに接続されたこと、およびネットワーク経由で転送されるデータが非公開であることを、クライアントに対し保証します。

SSL では公開鍵暗号化を使用しています。公開鍵暗号化を使用するには、WebLogic Server の証明書である Server ID を、VeriSign などの認証局から購入する必要があります。クライアントが WebLogic Server の SSL ポートに接続すると、サーバとクライアントはプロトコルを実行します。ここで、サーバの Server ID の認証、暗号化アルゴリズムとセッション パラメータのネゴシエーションが行われます。クライアントに対して証明書の提示を要求するように、WebLogic Server をコンフィグレーションすることもできます。これを、相互認証と呼びます。

SOAP

SOAP（Simple Object Access Protocol）は、分散型環境で情報を交換するために使用する軽量 XML ベースのプロトコルです。このプロトコルは、SOAP メッセージを記述するエンベロープ、エンコーディング ルール、およびリモート プロシージャ呼び出しと応答を表す規則で構成されます。

すべての情報は、HTTP またはその他の Web プロトコル上で転送可能な Multipurpose Internet Mail Extensions（MIME）エンコード パッケージ内に埋め込まれています。MIME は、非 ASCII メッセージをインターネット上で送信できるようにフォーマットするための仕様です。

データ サービスとアクセス サービス

WebLogic Server は、アプリケーションおよびコンポーネントにデータ サービスとアクセス サービスを提供する、標準の J2EE 技術を実装しています。これらのサービスには、以下の API が含まれています。

- Java Naming and Directory Interface （JNDI）
- Java Database Connectivity （JDBC）
- Java Transaction API （JTA）
- J2EE コネクタ アーキテクチャ

以下の節では、これらのサービスについて詳しく説明します。

JNDI

Java Naming and Directory Interface（JNDI）は、アプリケーションでオブジェクトを名前からルックアップできるようにする、標準 Java API です。WebLogic Server またはユーザ アプリケーションは、提供する Java オブジェクトをネーミング ツリー内の名前にバインドします。WebLogic Server から JNDI コンテキストを取得し、オブジェクトの名前で JNDI のルックアップ メソッドを呼び出すことにより、RMI オブジェクト、エンタープライズ JavaBean、JMS キューおよびトピックといったオブジェクトや JDBC データソースをアプリケーションでルックアップできます。ルックアップは、WebLogic Server オブジェクトへの参照を返します。

WebLogic JNDI は、WebLogic Server クラスタのロード バランシングやフェイルオーバ機能をサポートします。クラスタ内の各 WebLogic Server は、レプリケートされたクラスタワイドのネーミング ツリー内で提供するオブジェクトをパブリッシュします。アプリケーションは、クラスタ内の任意の WebLogic Server から JNDI の初期コンテキストを取得したり、ルックアップを実行したり、そのオブジェクトを提供するクラスタ内の任意の WebLogic Server からオブジェクト参照を受け取ったりできます。コンフィグレーション可能なロード バランシング アルゴリズムが、クラスタ内のサーバ間での負荷を分散するために使用されています。

JDBC

Java Database Connectivity（JDBC）は、バックエンドのデータベース リソースへのアクセスを提供します。Java アプリケーションは、JDBC ドライバを使用して JDBC にアクセスします。JDBC ドライバは、データベース ベンダ固有の、データベース サーバ用インタフェースです。すべての Java アプリケーションでは、ベンダの JDBC ドライバをロードしたり、データベースに接続したり、データベース操作を実行したりできますが、WebLogic Server では、JDBC 接続プールを使用することにより、性能上の大きなメリットを提供します。

JDBC 接続プールとは、WebLogic Server で管理される JDBC 接続のグループに名前を付けたものです。WebLogic Server は、起動時に JDBC 接続を開いて、それらをプールに追加します。JDBC 接続が必要になると、アプリケーションはプールから接続を取得し、使用してから、他のアプリケーションが使用できるようにプールに返します。データベース接続を確立する処理は、多くの場合時間がかかり、リソースを大量に消費するため、接続プールでは、接続処理の数を制限することでパフォーマンスを改善します。

WebLogic Server はまた、単一サーバ コンフィグレーションにおけるデータベース接続でロード バランシングまたは高可用性の機能を実現するための JDBC マルチプールを提供します。マルチプールとは「プールのプール」で、特定の要求に対してどの接続を提供するかを選択するためのコンフィグレーション可能なアルゴリズムを備えています。現在、WebLogic Server はデータベース接続用に高可用性またはロード バランシングのどちらかをサポートするアルゴリズムを提供しています。

接続プールを JNDI ネーミング ツリーに登録するには、そのための DataSource オブジェクト定義します。Java クライアント アプリケーションは、DataSource 名で JNDI ルックアップを行うことにより、プールから接続を取得できます。

サーバサイド Java クラスでは、WebLogic JDBC プール ドライバを使用します。これは、ベンダ固有の JDBC ドライバを呼び出す汎用 JDBC ドライバです。このメカニズムにより、アプリケーション コードの移植性を高めることができ、バックエンド層で使用するデータベースのブランドを変更するような場合にも対応できます。

クライアントサイドの JDBC ドライバは、WebLogic JDBC/RMI ドライバです。これは、プール ドライバへの RMI インタフェースです。このドライバは、標準 JDBC ドライバを使う場合と同様に使用できます。JDBC/RMI ドライバを使うと、Java プログラムは、他の WebLogic Server の分散オブジェクトと同じように JDBC にアクセスでき、また中間層のデータベースのデータ構造を維持できます。

WebLogic EJB および WebLogic JMS では、永続的なオブジェクトをロードおよび保存するために、JDBC 接続プールから取得した接続を使用します。EJB および JMS を使用すると、アプリケーションで直接 JDBC を使用するよりも便利な抽象化が提供されます。たとえば、エンタープライズ Bean を使用してデータの多いオブジェクトを表現すると、JDBC コードを修正せずに、後から基底のストアを変更することができます。JDBC を使用してデータベース操作をコーディングする代わりに、永続的な JMS メッセージを使用すると、サードパーティ製のメッセージング システムにアプリケーションを後から適応させることが簡単になります。

JTA

Java Transaction API（JTA）は、Java アプリケーションでトランザクションを管理するための、標準インタフェースです。トランザクションを使用すると、データベース内のデータの整合性を保護し、アプリケーションまたはアプリケーション インスタンスによるデータへの同時アクセスを管理できます。トランザクションを開始したら、すべてのトランザクション処理を正常にコミットするか、すべてをロールバックする必要があります。

WebLogic Server では、EJB、JMS、JCA、および JDBC 処理を含むトランザクションをサポートしています。2 フェーズ コミットにより調整される分散トランザクションは、BEA WebLogic jDriver for Oracle/XA のような XA 準拠 の JDBC ドライバを使用してアクセスされる複数のデータベースにまたがることができます。

EJB 仕様では、Bean 管理によるトランザクションおよびコンテナ管理によるトランザクションが定義されています。エンタープライズ Bean がコンテナ管理のトランザクションと共にデプロイされると、WebLogic Server は自動的にトラン

ザクションを調整します。エンタープライズ Bean が Bean 管理によるトランザクションと共にデプロイされる場合は、EJB プログラマはトランザクション コードを用意する必要があります。

JMS または JDBC API に基づいたアプリケーション コードを使用すると、トランザクションを開始したり、それ以前に開始されたトランザクションに参加したりできます。アプリケーションを実行する WebLogic Server スレッドには、1 つのトランザクション コンテキストが関連付けられています。スレッド上で実行されるすべてのトランザクション処理は現在のトランザクションに参加します。

J2EE コネクタ アーキテクチャ

J2EE コネクタ アーキテクチャは、エンタープライズ情報システム（EIS）を簡単に J2EE プラットフォームに統合します。数多くのアプリケーション サーバと EIS との間を接続するという問題を Java により解決します。コネクタ アーキテクチャを使用することで、EIS ベンダはアプリケーション サーバごとに自社製品をカスタマイズする必要がなくなります。J2EE コネクタ アーキテクチャに準拠することで、BEA WebLogic Server ではカスタム コードを追加しなくても、新しい EIS への接続機能を追加できるようになります。

J2EE コネクタ アーキテクチャは、WebLogic Server および EIS 固有のリソース アダプタのどちらでも実装されます。リソース アダプタとは EIS に固有のシステム ライブラリのことで、EIS への接続を提供するものです。リソース アダプタは JDBC ドライバと同様の機能を持っています。リソース アダプタと EIS 間のインタフェースは基底となる EIS に固有なので、ネイティブ インタフェースの場合もあります。

J2EE コネクタ アーキテクチャには、WebLogic Server と特定のリソース アダプタの間のシステムレベル規約、クライアントがアダプタにアクセスするための共通のインタフェース、およびリソース アダプタを J2EE アプリケーションに対してパッケージ化およびデプロイするためのインタフェースで構成されています。詳細については『WebLogic J2EE コネクタ アーキテクチャ』を参照してください。

メッセージング技術

J2EE のメッセージング技術は、WebLogic Server のアプリケーション同士の通信や非 WebLogic Server アプリケーションとの通信に使用できる、標準 API を提供します。メッセージング サービスには、以下の API が含まれています。

- Java Message Service（JMS）
- JavaMail

次の節では、これらの API について詳しく説明します。

JMS

Java Messaging Service（JMS）を使用すると、メッセージを交換してアプリケーション同士で通信できます。メッセージとは、異なるアプリケーション間の通信を調整するために必要な情報が含まれている、要求、レポート、およびイベントです。メッセージで提供される抽象化のレベルにより、送り先システムについての詳細情報をアプリケーション コードから切り離すことができます。

WebLogic JMS は、ポイント ツー ポイント（PTP）およびパブリッシュ / サブスクライブ（pub/sub）という 2 つのメッセージング モデルを実装しています。PTP モデルでは、任意の数の送信者がキューにメッセージを送信できます。キューの中の各メッセージは、1 つのリーダーに送信されます。pub/sub モデルでは、任意の数の送信者がトピックにメッセージを送信できます。トピック上の各メッセージは、そのトピックに対するサブスクリプションを持つすべてのリーダーに送信されます。メッセージは、同期的または非同期的に、リーダーに配信できます。特定のメッセージング モードは、Administration Console を使用するか、または JMS でメッセージ送信に使用されるメソッドによって制御できます。

JMS メッセージは永続的にも非永続的にもできます。永続的なメッセージはデータベースに保存され、WebLogic Server を再起動しても失われません。非永続的なメッセージは WebLogic Server を再起動すると失われます。トピックに送信される永続的なメッセージは、そのメッセージに関心のあるすべてのサブスクライバが受信するまで保持できます。

JMS では、異なるタイプのアプリケーションに役立つ複数のメッセージ タイプをサポートしています。メッセージ本体には、任意のテキスト、バイト ストリーム、Java プリミティブ データ型、名前と値の組み合わせ、シリアライズ可能な Java オブジェクト、または XML コンテンツを含めることができます。

JavaMail

WebLogic Server には、Sun の JavaMail 参照実装が含まれています。JavaMail を使用すると、アプリケーションで電子メール メッセージを作成して、ネットワーク上の SMTP サーバ経由で送信できます。

2　WebLogic Server のサービス

以下の節では、WebLogic Server のサービスについて説明します。

- Web サーバとしての WebLogic Server
- セキュリティ サービス
- WebLogic Server クラスタ
- サーバの管理とモニタ

Web サーバとしての WebLogic Server

WebLogic Server は、高度な Web 対応アプリケーションのためのプライマリ Web サーバとして使用することができます。J2EE Web アプリケーションとは、HTML または XML ページ、JavaServer Pages、サーブレット、Java クラス、アプレット、画像、マルチメディア ファイル、およびその他の種類のファイルの集まりです。

Web サーバとしての WebLogic Server の機能

Web アプリケーションは、Web サーバの Web コンテナ内で動作します。WebLogic Server 環境では、Web サーバは論理的なエンティティであり、クラスタ内の 1 つまたは複数の WebLogic Server 上でデプロイされます。

Web アプリケーション内のファイルは、ディレクトリ構造に格納されます。Java の `jar` ユーティリティを使用して 1 つの `.war`（Web アーカイブ）ファイルにまとめることもできます。一連の XML デプロイメント記述子では、アプリケーションのコンポーネント、およびセキュリティ設定などの実行時パラメータ

が定義されます。デプロイメント記述子を使用すると、Web アプリケーション コンポーネントの内容を変えることなく実行時の動作を変更でき、また同じアプリケーションを簡単に複数の Web サーバ上にデプロイできます。

Web サーバの機能

Web サーバとしての WebLogic Server には、次のような機能がサポートされています。

- 仮想ホスティング
- プロキシ サーバのコンフィグレーションのサポート
- ロード バランシング
- フェイルオーバ

この節では、これらの機能に関する WebLogic Server のサポートについて説明します。

仮想ホスティング

WebLogic Server は、仮想ホスティングをサポートしています。これにより、1 つの WebLogic Server または WebLogic Server クラスタで複数の Web サイトをホストすることができます。論理上の Web サーバには、それぞれ独自のホスト名が付けられていますが、すべての Web サーバは、DNS で同じクラスタの IP アドレスにマップされます。クライアントが HTTP リクエストをクラスタ アドレスに送信すると、リクエストを提供する WebLogic Server が選択されます。Web サーバ名が HTTP リクエストのヘッダから抽出されて、仮想ホスト名がクライアント側から見て一定になるように、以降のクライアントとのやりとりの間維持されます。

複数の Web アプリケーションを WebLogic Server にデプロイできます。各 Web アプリケーションを仮想ホストにマップできます。

プロキシ サーバのコンフィグレーションの使用

WebLogic Server は既存の Web サーバと統合できます。リクエストを WebLogic Server から別の Web サーバへプロキシしたり、WebLogic Server に付属のネイティブ プラグインを使用して、リクエストを別の Web サーバから WebLogic Server へプロキシしたりできます。BEA では、Apache Web Server、Netscape Enterprise Server、および Microsoft Internet Information Server 用のプラグインを提供しています。

クライアントと、一連の独立した WebLogic Server または WebLogic Server クラスタとの間でプロキシ Web サーバを使用すると、Web リクエストのロードバランシングおよびフェイルオーバを実行できます。クライアントにとっては、それらは単に 1 つの Web サーバに見えます。

ロード バランシング

プロキシ サーバの背後に複数の WebLogic Server をセットアップすると、大量のリクエストに対応できます。プロキシ サーバはその背後の層にある複数のサーバ間にリクエストを分散することによってロードバランシングを行います。

プロキシ サーバは、WebLogic Server Web サーバでも、Apache、Netscape、または Microsoft Web サーバでもかまいません。WebLogic Server には、いくつかのプラットフォーム用のネイティブ コード プラグインが用意されています。このプラグインを使用すると、これらのサードパーティ製 Web サーバは、リクエストを WebLogic Server にプロキシできます。

プロキシ サーバは、特定のタイプのリクエストをその背後にあるサーバにリダイレクトするようセットアップされています。たとえば、一般的な使い方として、静的な HTML ページへのリクエストはプロキシ サーバで処理し、サーブレットおよび JavaServer Pages へのリクエストはプロキシの背後の WebLogic Server クラスタにリダイレクトするようにプロキシ サーバをコンフィグレーションします。

フェイルオーバ

Web クライアントがサーブレット セッションを開始すると、プロキシ サーバは、同じセッションの一部である後続のリクエストを別の WebLogic Server に送信できます。WebLogic Server では、クライアントのセッション ステートが利用可能のままであることを保証するために、セッション レプリケーションを提供しています。

セッション レプリケーションには、2 つのタイプがあります。

- JDBC セッション レプリケーションは、WebLogic Server クラスタまたは一連の独立した WebLogic Server で使用されます。WebLogic Server のクラスタ化オプションを選択する必要はありません。
- インメモリ セッション レプリケーションでは、WebLogic Server のクラスタ化オプションを選択する必要があります。

JDBC セッション レプリケーションでは、データベースへセッション データが記述されます。セッションが開始すると、プロキシ サーバが選択した任意の WebLogic Server は、データベースからセッション データを取得して、セッションを続行できます。

WebLogic Server クラスタをプロキシ サーバの背後にデプロイする場合に、サーブレット セッションをクラスタの選択したセカンダリ WebLogic Server にネットワーク経由でレプリケートすると、データベースにアクセスする必要がなくなります。インメモリ レプリケーションは、消費するリソースが少なく JDBC セッション レプリケーションよりも高速なため、WebLogic Server クラスタを使用する場合は、インメモリ レプリケーションが、サーブレットにフェイルオーバを提供するための最良の方法です。

セキュリティ サービス

WebLogic Server は、セキュリティ レルムと呼ばれるサービスを通じてアプリケーションにセキュリティを提供します。セキュリティ レルムは、以下の 2 つのサービスへのアクセスを提供します。

- 認証サービスは、WebLogic Server でユーザの ID を照合できるようにします。

- 認証サービスは、ユーザによるアプリケーションへのアクセスを制御します。

注意： WebLogic Server は JNDI 認証メカニズムの代わりとして JAAS も提供しています。JAAS を使用するには、Java SDK バージョン 1.3 をインストールする必要があります。JAAS の認可コンポーネントは WebLogic Server では提供されていません。

認証

レルムは、ユーザおよびグループのストアへアクセスでき、ユーザが入力した資格（通常はパスワード）をセキュリティ ストア内のユーザ名および資格と照合することによって、ユーザを認証できます。Web ブラウザは、Web クライアントが保護された WebLogic Server サービスにアクセスしようとしたときにユーザ名とパスワードを要求することによって、認証をサポートしています。他の WebLogic Server クライアントが、WebLogic Server との接続を確立する場合は、プログラム的にユーザ名と資格を提供します。

認可

WebLogic Server のサービスは、アクセス制御リスト（ACL）によって保護されています。ACL は、サービスへのアクセスを認可されたユーザおよびグループのリストです。ACL で一度ユーザが認可されたら、WebLogic Server は、ACL をチェックしてから、ユーザにそのサービスへのアクセスを許可します。

代替レルムとカスタム レルム

セキュリティ レルムは、プラグイン可能です。WebLogic Server に付属の代替レルムであるデフォルトのファイル レルムを使用することも、独自のレルムを作成することもできます。デフォルトのファイル レルムは、ユーザ、グループ、および ACL を、暗号化されたファイルに格納します。そのファイルは、Administration Console で管理します。

WebLogic Server では、ユーザ、グループ、場合により ACL の外部ストアにアクセスする代替レルムが用意されています。代替レルムは、以下の外部のセキュリティ システムに対して提供されます。

- Netscape Directory Server、Microsoft Site Server、Novell NDS などの Lightweight Directory Access Protocol（LDAP) サービス
- UNIX ログイン サービス
- Windows NT ドメイン
- RDBMS。ユーザ、グループ、および ACL を格納するためにデータベース システムを使用するレルム

WebLogic Server で使用するカスタム レルムは、WebLogic レルム インタフェースを実装することによって開発できます。WebLogic Server では、キャッシング システムが組み込まれたレルムをサポートして、外部ストアへの呼び出しを最小限に抑えます。カスタム レルムで実装されていない機能は、デフォルトではファイル レルムに設定されます。たとえば、一般的に、ユーザおよびグループの外部ストアはカスタム レルムで使用するよう開発しますが、ACL はデフォルトのファイル レルムで保持します。

暗号化

WebLogic Server は、ネットワーク経由で転送されるデータを保護するために、セキュア ソケット レイヤ（SSL）プロトコルをサポートしています。SSL はセキュアな Web 接続のための標準プロトコルです。WebLogic Server は、Web（HTTPS）接続、および RMI ベースのサービスを利用するスタンドアロンの Java クライアントでの SSL をサポートします。

SSL を提供するには、最初に VeriSign などの認証局（CA）から、WebLogic Server のサーバ ID を購入する必要があります。クライアントがセキュアな接続を確立する場合、WebLogic Server はクライアントへ証明書を送信し、クライアントでは接続が正しいことを確認できます。クライアントは証明書を調べて、期限が切れていないこと、送信元のサーバと一致していること、および認められた認証局による署名があることを確認します。その後、サーバとクライアントは暗号化キーを交換し、セキュアな接続を確立します。

WebLogic Server では、相互認証を要求するようにコンフィグレーションすることもできます。相互認証では、クライアントは証明書の提出を要求され、サーバはその証明書を検証してから接続を受け付けます。

WebLogic Server クラスタ

WebLogic Server クラスタは、複数の WebLogic Server インスタンスのグループで、互いに連携して強力で信頼性の高い Web アプリケーション プラットフォームを提供します。クラスタは、クライアントにとっては 1 つのサーバに見えますが、実際には、1 つのものとして機能するサーバのグループです。クラスタには、スケーラビリティと可用性という、単一のサーバでは提供できない 2 つの主要なメリットがあります。

『WebLogic Server Clusters ユーザーズ ガイド』では、クラスタのプランニングとコンフィグレーションに関する詳細を説明しています。

クラスタを使用するメリット

WebLogic Server クラスタは、アプリケーション開発者からは見えないように、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します。スケーラビリティのメリットは、WebLogic Server の単一のインスタンスまたは単一のコンピュータよりも、中間層の能力を拡大できることです。クラスタのメンバーシップにおける唯一の制限は、すべての WebLogic Server のインスタンスが IP マルチキャストで通信できなければならない点です。新しい WebLogic Server 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。

WebLogic Server クラスタはまた、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することで高可用性を保証します。クラスタ内の複数のサーバで、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。

クラスタ アーキテクチャ

WebLogic Server クラスタは、ネットワークにデプロイされた複数の WebLogic Server インスタンスから構成され、Domain Name Service（DNS）、JNDI ネーミング ツリーのレプリケーション、セッション データのレプリケーション、および WebLogic RMI を組み合わせて調整されます。

Web クライアントと WebLogic Server クラスタの間にある Web プロキシ サーバでは、サーブレットおよび JavaServer Pages 用にクラスタ化されたサービスを調整します。Web プロキシ サーバになるのは、WebLogic Server、または WebLogic Server 付属のプラグインと共に使用される Netscape、Microsoft、または Apache のサードパーティ製 Web サーバです。

Web クライアントは、プロキシ サーバにリクエストを転送することにより、WebLogic Server クラスタに接続します。Java RMI ベースのクライアントは、ネットワーク上で定義されたクラスタ アドレスを使用して、WebLogic Server クラスタに接続します。

サーバ サイド コードも、WebLogic クラスタの提供するロード バランシングおよびフェイルオーバ サービスの恩恵を受けています。J2EE アプリケーションでは、ほとんどのアプリケーション コードは中間層で動作するため、複数の WebLogic Server 間に分散されたサービスを使用できます。たとえば、WebLogic Server「A」で動作中のサーブレットから、WebLogic Server「B」のエンタープライズ Bean を使用して、WebLogic Server「C」の JMS キューからのメッセージを読み込むことができます。

WebLogic Server クラスタをネットワークで定義する方法

WebLogic Server のサービスへは、DNS を介してアクセスします。DNS とは、インターネットを含むネットワーク上のリソースのための標準ネーミング サービスです。DNS は、170.0.20.1 などの IP アドレスを、「mycomputer.mydomain.com」または「www.bea.com」などの名前にマップします。各 WebLogic Server インスタンスは、ネットワーク上でユニークな IP アドレスで動作します。クライアントは、名前、および接続をリスンしているポートの番号を URL にエンコードすることにより、WebLogic Server に接続します。

たとえば、「onyx」という名前で、リスン ポートが 7701 に設定されたコンピュータ上で動作する WebLogic Server へは、「http://onyx:7701」という URL を使用して、Web ブラウザからアクセスできます。この接続が成功するには、ネットワーク上のネーム サーバで、ローカル ドメインの「onyx」という名前を解決する必要があります。送り先のサーバがインターネット上の別のドメインにある場合は、「http://onyx.bea.com:7701」のように、完全なドメイン名を入力する必要があります。

追加の DNS エントリでは、クラスタに参加するすべての WebLogic Server インスタンスの名前が 1 つのクラスタ名にマップされます。クライアントは、クラスタ名を使用するか、Web プロキシ サーバを介してリクエストをクラスタに転送することにより、クラスタに接続します。DNS は、クラスタ名のルックアップを実行すると、そのクラスタに属するすべてのサーバのリストを返します。クライアントは通常リストの最初のサーバを選択し、応答がなければ 2 番目のサーバを選択して、応答が得られるまでリスト内のサーバを順に選択してゆきます。

DNS は、リクエストをクラスタ内のサーバ間に分散する初期ロードバランシング サービスを提供します。各 DNS は、最終的に各サーバがリストを一巡するようにサーバのリストを 1 つずつローテーションして、クラスタ名のルックアップに応答します。

ネットワーク上で動作するインテリジェント ルータ、プロキシ サーバ、ファイアウォール、またはその他のソフトウェアは、DNS をオーバーライドし、マシンの負荷、ネットワーク トラフィック、またはその他の動的なロードバランシングの基準に基づいて、最初のサーバを選択します。

最初の WebLogic Server の接続は、クライアントに対してネーミング サービスを提供します。クライアントから要求されたサービスをルックアップし、WebLogic Server でコンフィグレーションされたロードバランシング アルゴリズムを使用して、要求を処理するサーバをクラスタから選択します。

クラスタ内の WebLogic Server の通信方法

クラスタ内の WebLogic Server は、特定のクラスの情報をクラスタ内のすべてのサーバにレプリケートするために、IP マルチキャストを使用して互いに通信します。クラスタ内の各サーバ インスタンスには、共通のマルチキャスト アドレスがコンフィグレーションされます。1 つのサーバがメッセージをクラスタのマルチキャスト アドレスに送信すると、すべてのサーバがそのメッセージを受信

します。これは、サーバがポイント ツー ポイント メッセージを送信するよりも効率的です。ただし、この場合はクラスタ内のすべてのサーバがネットワーク上にあり、マルチ キャストをサポートしている必要があります。マルチキャストはインターネット上では動作しないため、クラスタがインターネットをまたがることはできません。

サービスによっては、クラスタはプライマリおよびセカンダリの WebLogic Server を選択します。プライマリ WebLogic Server がリクエストの処理を開始してから使用できなくなると、セカンダリ サーバがリクエストの処理を停止せずに引き継ぐことができます。プライマリ サーバは、サーバ ツー サーバ接続を使用してセカンダリ サーバにステートをレプリケートします。

ほとんどのサービスは、クラスタ内の任意の数の WebLogic Server にデプロイできます。各サービスがデプロイされると、WebLogic Server は IP マルチキャストを使用して、サービスをクラスタワイドのネーミング ツリーに追加します。クラスタ内のいずれのサーバでも、クラスタワイドのネーミング ツリーでサービスをルックアップすることにより、サービスを提供する WebLogic Server を見つけることができます。複数のサーバがそのサービスを提供できる場合、クラスタはコンフィグレーション可能なロードバランシング アルゴリズムを使用してサーバを選択します。

クラスタ化されたサービス

ほとんどの WebLogic Server サービスはクラスタ化できます。したがって、これらのサービスはクラスタ内の任意の数のサーバにデプロイできます。クラスタは、サービスを提供する WebLogic Server インスタンスを選択します。サーバが選択され、ステートフル オブジェクトがサーバ上でインスタンス化されると、クライアントはサービスが終了するまでその WebLogic Server に「固定」されます。固定されたオブジェクトのホストとなっている WebLogic Server で障害が発生した場合、クライアントはその障害を検出し、クラスタ内の他のサーバ上にもう 1 つのインスタンスを作成する必要があります。

より回復力のあるフェイルオーバを提供するために、WebLogic Server クラスタでは、特別に必要な場合以外は、オブジェクトをサーバに固定しません。場合により、クラスタはステートフル オブジェクトをバックアップ サーバにレプリケートして、サービスのフェイルオーバを有効にします。

2-1 ページの「Web サーバとしての WebLogic Server」で説明したように、Web アプリケーションはクラスタ化できます。サーブレット セッションがセカンダリ サーバにレプリケートされることにより、クラスタは透過的に障害から回復できます。

すべてのエンタープライズ JavaBean はクラスタ化できます。エンタープライズ JavaBean は、WebLogic Server クラスタ内の任意の数のサーバ上にデプロイできます。ただし、すべての EJB インスタンスがクラスタ化できるわけではありません。アプリケーションは、Bean がデプロイされている任意のサーバから EJB のホーム インタフェースを取得し、そのホーム インタフェースを使用して、Bean インスタンスを作成できます。ホーム インタフェースを提供するサーバに障害が発生すると、アプリケーションを停止せずに、別のサーバからホーム インタフェースを取得できます。

ステートレス セッション Bean や読み取り専用エンティティ Bean など、EJB インスタンスのタイプによっては、常にクラスタ化できるものもあります。ステートフル セッション Bean は、インメモリ レプリケーションを使用してフェイルオーバを提供することにより、クラスタ化できます。読み書き対応 エンティティ Bean は、Bean をインスタンス化するサーバに常に固定されます。読み書き対応エンティティ Bean のホストになっているサーバに障害が発生した場合、エンティティ Bean はフェイルオーバしても安全であれば、自動的にフェイルオーバします。そうしなかった場合、フェイルオーバは次のトランザクションで発生し、そのクラスタ内の別のサーバ上に、リモート スタブによってエンティティ Bean の インスタンスが再作成されます。

JDBC メタプールは、WebLogic Server クラスタ内の複数サーバにデプロイされた JDBC 接続プールにクラスタ化を提供します。クライアントがメタプールからの接続を要求すると、クラスタは接続を提供するサーバを選択し、ロードバランシングおよびサーバ障害に対する保護を可能にします。クライアントが接続を確立すると、JDBC ドライバで維持されているステートにより、クライアントをホストである WebLogic Server に固定する必要が生じます。

JMS オブジェクトは、クラスタ内のサーバ間に分散できます。それぞれの送り先（メッセージ キューまたはトピック) は、クラスタ内の単一の WebLogic Server によって管理されます。ただし、クライアントが送り先への接続を確立するために使用する接続ファクトリは、クラスタ内の複数のサーバにデプロイできます。送り先および接続ファクトリをクラスタ全体に分散すると、管理者は、JMS サービスの負荷を手動で調整できます。

サーバの管理とモニタ

WebLogic Server では、Administration Console またはコマンドライン インタフェースを使用して、ドメイン内のサーバの属性を設定することにより、管理を行います。Administration Console は Web ブラウザ アプリケーションです。Administration Console を使用すると、WebLogic Server サービスのコンフィグレーション、セキュリティの管理、アプリケーションのデプロイメント、およびサービスの動的なモニタを行えます。

Administration Console とコマンドライン インターフェースは、いずれも管理サーバに接続します。

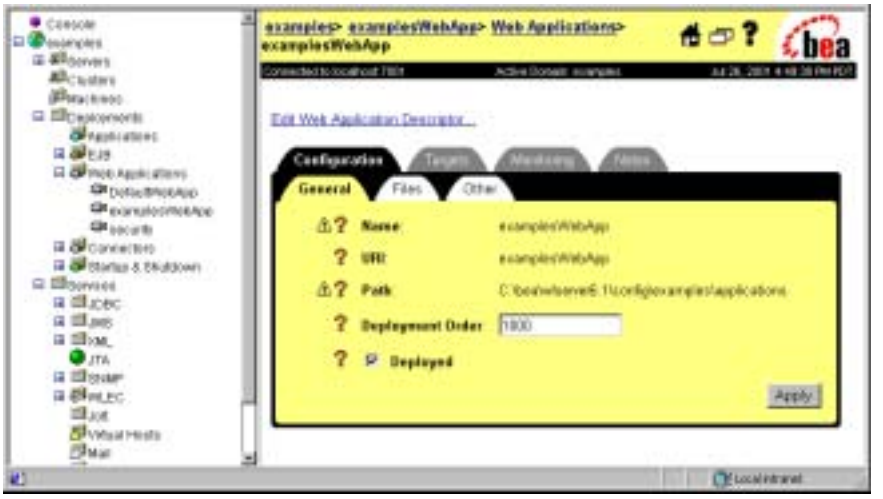
管理サーバ

管理サーバは、ドメイン内のすべての WebLogic Server をコンフィグレーションおよび管理するために使用する WebLogic Server です。ドメインには複数の WebLogic Server クラスタおよび複数の独立した WebLogic Server インスタンスが含まれます。ドメインに 1 つの WebLogic Server しか含まれない場合は、そのサーバが管理サーバになります。ドメインに複数の WebLogic Server インスタンスがある場合、最初に起動するインスタンスは管理サーバでなければなりません。

Administration Console

WebLogic Server Administration Console は Web ブラウザで動作します。クラスタや個々の WebLogic Server など、管理するドメインのコンポーネントが、左ペインのグラフィカル ツリーに表示されます。右ペインには、左ペインで選択したオブジェクトの詳細が表示されます。図 2-1 は、Administration Console の画面例です。

図 2-1 Administration Console



Administration Console を使用してサービスをコンフィグレーションするには、左ペインでアイテムを選択し、右ペインで [コンフィグレーション] タブを選択します。Administration Console の右ペインに、コンフィグレーション可能な属性が表示されます。表示される属性の詳細については、オンライン ヘルプを参照してください。

Administration Console を使用してサービスをコンフィグレーションする通常の手順では、サービスをコンフィグレーションしてから、そのサービスをデプロイする対象（WebLogic Server）を選択します。

デプロイされた各サービスでは実行時の統計が維持されます。統計は、Administration Console の右ペインの [モニタ] タブで表示できます。

索引