BEA WebLogic Integration™

WebLogic Integration ソリューションのデプロイメント

リリース 7.0
マニュアルの日付 : 2002 年 6 月

著作権

限定的権利条項

商標または登録商標

WebLogic Integration ソリューションのデプロイメント

パート番号	日付	ソフトウェアのバージョン
なし	2002 年 6 月	7.0

目次

このマニュアルの内容

1. はじめに

2. WebLogic Integration クラスタについて

3. クラスタ デプロイメントのコンフィグレーション

4. WebLogic Integration の高可用性

5. WebLogic Integration セキュリティの使い方

6. パフォーマンスのチューニング

索引

このマニュアルの内容

このマニュアルでは、BEA WebLogic Integration を使用してプロダクション環境に統合ソリューションをデプロイする方法について説明します。具体的には、高可用性、パフォーマンス、スケーラビリティ、およびセキュリティの目標を達成するための統合ソリューションをデプロイする方法について説明します。主要なデプロイメントの概念を定義し、WebLogic Integration クラスタ上に統合ソリューションをデプロイする方法、WebLogic Integration セキュリティの概要、プロダクション環境でのパフォーマンスをチューニングする方法について説明します。

WebLogic Integration の概要についてのマニュアル

このマニュアルは、WebLogic Integration の概要、および WebLogic Integration の機能を統合ソリューションの設計、開発、デプロイメントのさまざまな段階でどのように使用するかについて説明した 4 冊のマニュアル シリーズの 1 つです。まずこれらのマニュアルで WebLogic Integration の機能を包括的に理解してください。他の 3 冊のマニュアルの内容は、以下の通りです。

- 『*WebLogic Integration 入門*』 — WebLogic Integration の概要について説明したマニュアルです。このマニュアルでは、異種のフラグメント化されたビジネス システムの集合を駆使した業務展開が追求されるなかで、今日の E ビジネスが直面している統合問題について概説します。次に、WebLogic Integration が E ビジネスの統合問題を解決するために提供する Application Integration、B2B Integration、Business Process Management、および Data Integration の各機能について説明します。
- 『*WebLogic Integration チュートリアル*』 —サンプル統合アプリケーションについて説明したマニュアルです。このサンプル アプリケーションでは、サプライチェーン ハブをデプロイして、ビジネス パートナを接続し、複数のビジネス プロセスを自動化し、バックエンドのエンタープライズ情報システム

を統合します。このマニュアルでは、サンプル アプリケーションの設定および実行方法と、WebLogic Integration を使用して統合ソリューションを構築および開発する方法について学習します。

- 『*WebLogic Integration ソリューションの設計*』 ― BEA WebLogic Integration 環境で統合ソリューションを設計する方法について説明したマニュアルです。このマニュアルでは、主要な設計概念を定義し、包括的に解析されたビジネスおよび技術的な要件に基づいて統合要件を決定する方法、および高可用性、スケーラビリティ、パフォーマンスなどの設計目標を達成するための統合アーキテクチャを設計する方法について説明します。

上記のマニュアルをはじめ、各種の WebLogic Integration マニュアルを、次の URL で入手できます。

`http://edocs.beasys.co.jp/e-docs/wli/docs70/index.html`

これらのマニュアルの内容を理解したら、WebLogic Integration の機能に関する詳細マニュアルに進んでください。

このマニュアルの内容は以下のとおりです。

- 第 1 章「はじめに」では、デプロイメントのリソース、概念、タスク、デプロイメント チーム メンバーのロールを含めた、WebLogic Integration のデプロイメントのアーキテクチャについて説明します。
- 第 2 章「WebLogic Integration クラスタについて」では、1 つのまとまりで管理されるサーバの集まりであるクラスタ上に、統合ソリューションをデプロイする方法について説明します。主要なクラスタ化の概念、設計タスク、およびクラスタ デプロイメントのコンフィグレーション方法について説明します。
- 第 3 章「クラスタ デプロイメントのコンフィグレーション」では、クラスタ環境で WebLogic Integration を設定およびコンフィグレーションする上で必要な手順について説明します。
- 第 4 章「WebLogic Integration の高可用性」では、WebLogic Integration アプリケーションの高可用性の実現方法について説明します。
- 第 5 章「WebLogic Integration セキュリティの使い方」では、セキュアな WebLogic Integration デプロイメントを設定する方法について説明します。
- 第 6 章「パフォーマンスのチューニング」では、WebLogic Integration デプロイメントでのパフォーマンスに関する主要な考慮事項と、システムのパフォーマンスをモニタする方法について説明します。WebLogic Integration のリソース、ハードウェア、オペレーティング システム、ネットワーク接

続、およびデータベースのパフォーマンスをチューニングする方法について説明します。

対象読者

このマニュアルは主に、次のユーザを対象としています。

- 統合ソリューションのためのデプロイメント トポロジの設計や、1 つまたは複数のサーバへのさまざまな WebLogic Integration 機能のコンフィグレーションといった、デプロイメント作業を調整するデプロイメント スペシャリスト
- プロダクション環境で WebLogic Integration を設定、デプロイ、および管理するシステム管理者
- プロダクション環境で WebLogic Integration のデータベース管理システムを設定、デプロイ、および管理するデータベース管理者

WebLogic Integration のアーキテクチャの概要については、『*WebLogic Integration 入門*』を参照してください。

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、WebLogic Integration のマニュアル CD 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。

Adobe Acrobat Reader がない場合は、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 Integration のインストールおよびコンフィグレーション ウィザードの実行の詳細については、『*WebLogic Platform インストール ガイド*』および『*コンフィグレーション ウィザードの使い方*』を参照してください。次の URL にあります。

`http://edocs.beasys.co.jp/e-docs/platform/docs70/index.html`

WebLogic Integration のマニュアルは次の URL にあります。

`http://edocs.beasys.co.jp/e-docs/wli/docs70/index.html`

WebLogic Server のマニュアルは次の URL にあります。

`http://edocs.beasys.co.jp/e-docs/wls/docs70/index.html`

サポート情報

WebLogic Integration 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 Integration 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、ご使用の製品のバージョンをお書き添えください。

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
	`#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*`
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

表記法	適用
{ }	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、この省略記号は入力しない。
	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、この省略記号は入力しない。

1　はじめに

このマニュアルでは、BEA WebLogic Integration ソリューションをプロダクション環境にデプロイする方法について説明します。以下の節では、WebLogic Integration を組織にデプロイする際の主要な概念およびタスクを紹介します。

- デプロイメントの目標
- 主要なデプロイメント タスク
- 統合ソリューション デプロイメントのロール
- デプロイメントのアーキテクチャ
- 主要なデプロイメント リソース

デプロイメントの目標

WebLogic Integration は、企業が新しいアプリケーションを開発し、それらを既存システムに統合し、ビジネス プロセスを効率化し、トレーディング パートナに接続するための機能を提供する、単一の統合化されたプラットフォームです。WebLogic Integration ソリューションをデプロイする場合は、以下の目標について検討してください。

- *高可用性*。デプロイメントは、ハードウェアやネットワークで障害が発生したときにはフェイルオーバーが機能し、可用性が十分に確保されていなければなりません。
- *パフォーマンス*。デプロイメントでは、負荷のピーク時でもそれ以外のときでも十分なパフォーマンスが発揮されなければなりません。
- *スケーラビリティ*。デプロイメントでは、コードの変更ではなくハードウェア リソースを追加するだけで、予期される負荷の増加に対応できる必要があります。
- *セキュリティ*。デプロイメントでは、無認可アクセスや改ざんからデータが十分に保護されなければなりません。

WebLogic Integration デプロイメントでは、これらの目標やそれ以外の目標も達成することができます。

主要なデプロイメント タスク

WebLogic Integration のデプロイには、以下のタスクの一部またはすべてを実行する必要があります。

1. WebLogic Integration デプロイメントの目標の定義。1-1 ページの「デプロイメントの目標」を参照してください。
2. WebLogic Integration アプリケーションのクラスタへのデプロイ。そのためには、まず、クラスタを設計する必要があり、設計に着手するにあたっては、WebLogic Integration デプロイメントのコンポーネントを理解する必要があります。第 2 章「WebLogic Integration クラスタについて」で、用途に最適の環境を設計する上で役立つこれらのコンポーネントについて説明します。
3. 高可用性を実現するための、クラスタ環境への WebLogic Integration アプリケーションのデプロイ。そのためには、アプリケーションを第 3 章「クラスタ デプロイメントのコンフィグレーション」の説明に従ってコンフィグレーションする必要があります。
4. WebLogic Integration デプロイメントのセキュリティ設定。第 5 章「WebLogic Integration セキュリティの使い方」を参照してください。
5. 第 6 章「パフォーマンスのチューニング」の説明に従ったシステム全体のパフォーマンスの最適化（WebLogic Integration デプロイメントの起動後）。

統合ソリューション デプロイメントのロール

統合ソリューションのデプロイメントを成功させるには、デプロイメント チームに、以下のロールを遂行するメンバーが参加している必要があります。

- デプロイメント スペシャリスト
- WebLogic Server 管理者
- データベース管理者

一人で複数のロールを担当することができ、また、すべてのロールがあらゆるデプロイメント シナリオで等程度の関連性を持つわけではありませんが、デプロイメントを成功させるには、各ロールを担当するメンバーによる協力が必要です。

デプロイメント スペシャリスト

デプロイメント スペシャリストは、デプロイメント作業を調整します。デプロイメント スペシャリストは、WebLogic Integration 製品の機能について十分な知識を持っています。1 つまたは複数のサーバ上で WebLogic Integration の各種の機能をコンフィグレーションする方法についての知識に基づいて、統合ソリューションのデプロイメント トポロジの設計に際して専門的な判断を下します。デプロイメント スペシャリストは、以下の分野の経験を持っています。

- リソース要件の分析
- デプロイメント トポロジの設計
- プロジェクトの管理

WebLogic Server 管理者

WebLogic Server 管理者は、組織にデプロイされている WebLogic Server に関する深い技術知識と運用知識を提供します。ハードウェアおよびプラットフォームに関する知識があり、インストール、コンフィグレーション、モニタ、セキュリティ、パフォーマンスのチューニング、トラブルシューティングなどの管理作業をはじめとする WebLogic Server デプロイメントのあらゆる面を管理した経験があります。

データベース管理者

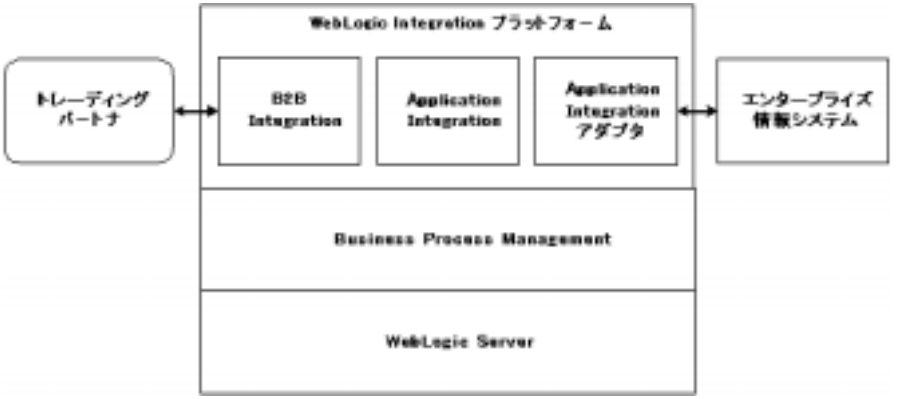
データベース管理者は、組織にデプロイされているデータベース システムに関する深い技術知識と運用知識を提供します。WebLogic Server 管理者は、以下の分野の経験を持っています。

- ハードウェアおよびプラットフォームに関する知識
- リレーショナル データベース（RDBMS）のあらゆる側面を管理した経験。インストール、コンフィグレーション、モニタ、セキュリティ、パフォーマンスのチューニング、トラブルシューティングなどの管理作業が含まれます。

デプロイメントのアーキテクチャ

次の図は、WebLogic Integration デプロイメントのアーキテクチャの概要を示しています。

図 1-1　WebLogic Integration デプロイメントのアーキテクチャ



次の節では、図で説明したこれらのリソースについて説明します。

主要なデプロイメント リソース

ここでは、デプロイメント時に変更可能なリソースの概要を説明します。

- WebLogic Server リソース
- Business Process Management リソース
- B2B Integration リソース
- Application Integration リソース
- リレーショナル データベース管理システム リソース
- ハードウェア、オペレーティング システム、およびネットワークのリソース

WebLogic Server リソース

この節では、WebLogic Integration ソリューションのデプロイメントに最も関連性の高い BEA WebLogic Server リソースの概要を説明します。これらのリソースをコンフィグレーションするには、WebLogic Server Administration Console または EJB デプロイメント記述子を使用します。

WebLogic Server には、WebLogic Integration ソリューションをサポートされている環境に合わせてデプロイするためのコンフィグレーション オプションおよび調整可能な設定が用意されています。以下の節では、WebLogic Integration デプロイメントに最も関連性の高い WebLogic Server でコンフィグレーションできる機能について説明します。

- クラスタ化
- Java Message Service
- EJB プールおよびキャッシュ
- JDBC 接続プール
- 実行スレッド プール
- J2EE コネクタ アーキテクチャ

クラスタ化

負荷処理能力を拡大するため、WebLogic Server をクラスタ、つまり 1 つの単位として管理可能な一群のサーバ上で実行することができます。クラスタ化すると、1 つのサーバよりもスケーラブルなデプロイメント プラットフォームを実現できます。クラスタ化の詳細については、第 2 章「WebLogic Integration クラスタについて」を参照してください。

Java Message Service

WebLogic Java Message Service (JMS) を使用すると、メッセージを交換（作成、送信、および受信）するメッセージング システムを Java アプリケーション間で共有できます。WebLogic JMS は、Sun Microsystems, Inc. の *Java Message Service 仕様*バージョン 1.0.2 に基づいています。

JMS サーバはクラスタ化できるので、接続ファクトリを WebLogic Server の複数のインスタンス上にデプロイすることができます。また、JMS イベントの送り先をコンフィグレーションして、ワークフローの通知およびメッセージを処理することもできます。1-10 ページの「Business Process Management リソース」を参照してください。

WebLogic JMS の詳細については、以下のトピックを参照してください。

- 次の URL にある『WebLogic JMS プログラマーズ ガイド』の「WebLogic JMS の概要」

  `http://edocs.beasys.co.jp/e-docs/wls/docs70/jms/intro.html`

- JMS のコンフィグレーションおよびモニタの詳細については、次の URL にある『*BEA WebLogic Server 管理者ガイド*』の「JMS の管理」を参照してください。

  `http://edocs.beasys.co.jp/e-docs/wls/docs70/adminguide/jms.html`

EJB プールおよびキャッシュ

WebLogic Integration デプロイメントでは、EJB の数がシステムのスループットに影響します。システム内の EJB の数を調整するには、EJB の型に応じて、EJB プールまたは EJB キャッシュを使用します（プールおよびキャッシュのサイズのコンフィグレーションについては、6-4 ページの「その他の EJB のプール サイズおよびキャッシュ サイズをコンフィグレーションする」を参照）。次の表では、EJB のタイプとそれらに関連付けられた調整可能なパラメータについて説明します。

表 1-1 EJB の調整に使用するパラメータ

グループ名	説明	リソース グループの種類
イベント リスナ メッセージ駆動型 Bean	`max-beans-in-free-pool`[1]	キューから作業を取り出すリスナの最大数。
ステートレス セッション Bean	`max-beans-in-free-pool`[1]	作業要求に利用できる Bean の最大数

表 1-1 EJB の調整に使用するパラメータ (続き)

グループ名	説明	リソース グループの種類
ステートフル セッション Bean	max-beans-in-cache	同時にアクティブにできる Bean の数設定値が小さすぎると、CacheFullExceptions が発生します。設定値が大きすぎると、メモリ使用量が非常に大きくなります。
エンティティ Bean		

1. WebLogic Server マニュアルでは、max-beans-in-free-pool ではなく、実行スレッドの数を設定することを推奨しています。ただし、WebLogic Integration 環境では、実行スレッドの数を設定するよりも、イベント リスナ メッセージ駆動型 Bean の max-beans-in-free-pool 設定を指定することによって負荷を制御する方が効率的です。

JDBC 接続プール

Java Database Connectivity (JDBC) を使用すると、Java アプリケーションから SQL データベース内のデータにアクセスできます。データベース接続の確立に関連するオーバーヘッドを抑えるため、WebLogic JDBC には、DBMS との接続ですぐに利用できる接続プールが用意されています。

JDBC 接続プールは、DBMS 接続を最適化するために使用されます。JDBC 接続プールのサイズをコンフィグレーションすることで、WebLogic Integration のパフォーマンスをチューニングできます。WebLogic Integration クラスタにおける JDBC 接続プールのサイズの決定に関する詳細は、6-6 ページの「JDBC 接続プールのサイズをコンフィグレーションする」を参照してください。設定値が小さすぎると、接続が利用可能になるまでの WebLogic Integration の待機時間が長くなります。設定値が大きすぎると、DBMS のパフォーマンスが低下します。

WebLogic JDBC 接続プールの詳細については、以下の節を参照してください。

- 次の URL の『WebLogic JDBC プログラマース ガイド』、「WebLogic JDBC の概要」の「接続プールの概要」

  http://edocs.beasys.co.jp/e-docs/wls/docs70/jdbc/intro.html

- 次の URL の『*BEA WebLogic Server 管理者ガイド*』の「JDBC 接続の管理」

  http://edocs.beasys.co.jp/e-docs/wls/docs70/adminguide/jdbc.html

実行スレッド プール

*実行スレッド プール*によって、WebLogic Server 上で同時に実行可能なスレッド数が決まります。設定値が小さすぎると、順次処理が行われるようになるので、デッドロックが発生する可能性があります。設定値が大きすぎると、メモリ使用量が非常に大きくなり、スラッシングが発生する可能性があります。

実行スレッド プールは、実行する可能性のあるすべてのスレッドを実行できるだけの大きさに設定しますが、システム内でのコンテキストの過剰な切り替えによりパフォーマンスが低下するほどには大きくしないでください。実行スレッド数によって、着信ソケット メッセージを読み出すスレッド（socket-reader スレッド）の数も決まります。この数は、デフォルトでは、実行スレッド数の 3 分の 1 です。数が小さすぎると、ソケットを読み出すスレッドの競合が発生し、デッドロックが発生する場合があります。稼働システムを監視して、スレッド プールの最適の値を経験的に決定してください。

実行スレッド プールのコンフィグレーションについては、6-7 ページの「実行スレッド プールをコンフィグレーションする」を参照してください。

実行スレッド プールの調整に関するこれらの推奨事項は、WebLogic Integration のパフォーマンスの最適化に役立ちます。ただし、WebLogic Integration 環境では、メッセージ駆動型 Bean の数を制御することが負荷を抑える方法として最適です（1-7 ページの「EJB プールおよびキャッシュ」参照）。

J2EE コネクタ アーキテクチャ

WebLogic J2EE コネクタ アーキテクチャ（JCA）は、J2EE プラットフォームと 1 つまたは複数の異種エンタープライズ情報システム（EIS）を統合します。WebLogic JCA は、Sun Microsystems, Inc. の *J2EE コネクタ仕様*、バージョン 1.0、最終草案 2 に基づいています。

WebLogic J2EE-CA の詳細については、次の URL にある『*BEA WebLogic Server 管理者ガイド*』の「WebLogic J2EE コネクタ アーキテクチャの管理」を参照してください。

```
http://edocs.beasys.co.jp/e-docs/wls/docs70/adminguide/jconnector
.html
```

Business Process Management リソース

WebLogic Integration では、Business Process Management (BPM) 機能を使用して、ビジネス プロセスを定義および実行します。BPM 機能の概要については、『*WebLogic Integration 入門*』の「Business Process Management」を参照してください。

以下の節では、WebLogic Integration ソリューションをデプロイするための BPM 機能について説明します。

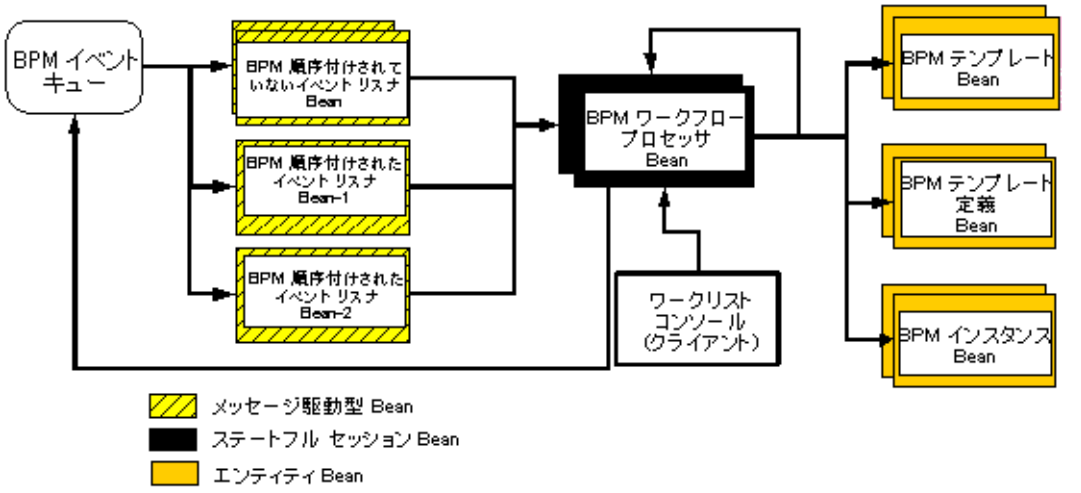
- BPM リソースの概要
- BPM リソースの種類
- BPM 作業シーケンス

BPM リソースは、クラスタ、つまり 1 つの単位として管理可能な一群のサーバ上で動作するようにコンフィグレーションできます。クラスタ化および BPM の詳細については、第 2 章「WebLogic Integration クラスタについて」を参照してください。

BPM リソースの概要

次の図は、クラスタ内の 1 つのノード用の BPM リソースを示しています。

図 1-2 BPM EJB リソース



次の「BPM リソースの種類」の節では、これらのリソースについて説明します。

BPM リソースの種類

BPM では、WebLogic JMS（1-6 ページの「Java Message Service」を参照）を使用して、エラーや監査メッセージだけでなく、ワークリスト、時刻、およびイベント通知もやり取りします。BPM クライアント アプリケーションは、これらのメッセージを *XML イベント*として JMS イベント キューに送信します。BPM では、イベント リスナ メッセージ駆動型 Bean を使用して、イベント キューに送られた XML イベントを処理し、BPM エンジンの実行中のインスタンスに配信します。

WebLogic Server Administration Console を使用してカスタム メッセージ キューを作成し、MDB Generator ユーティリティを実行して、キューでイベントをリスンするイベント リスナ Bean を生成し、新しいイベント リスナ Bean を認識するよう BPM のコンフィグレーションを更新します。詳細については、2-19 ページの「新しいプールを作成する」を参照してください。

以下の節では、BPM をクラスタ環境に合わせてコンフィグレーションしたり、BPM のパフォーマンスをチューニングしたりする際に使用できるリソースの種類について説明します。

- ワークフロー プロセッサ Bean
- イベント リスナ メッセージ駆動型 Bean
- テンプレート Bean
- テンプレート定義 Bean
- インスタンス Bean
- イベント キュー
- Worklist コンソール（非推奨）

ワークフロー プロセッサ Bean

ワークフロー プロセッサ Bean は、開始 / イベント ノードから停止 / イベント ノード（静止状態から静止状態）まで進むワークフローを実行するステートフル セッション Bean です。ワークフロー プロセッサ Bean は、イベント リスナ Bean、Worklist クライアント、および他のワークフロー プロセッサ Bean からの作業を受け入れます（サブワークフロー使用時）。

ワークフロー プロセッサ Bean はシステムの負荷に基づいて実行時に開始されるため、実行時のワークフロー プロセッサ Bean の正確な数は絶えず変化します。一度にアクティブになるワークフロー プロセッサ Bean の数は、ワークフロー プロセッサ Bean プールのサイズによって決まります。Bean の数がプールのサイズを超えると、プール内の Bean が利用可能になるまで、超過分の Bean のパッシベーションが行われます。一般に、プールのサイズは、小さすぎるよりも大きすぎる方が問題は少なくなります。詳細については、6-4 ページの「その他の EJB のプール サイズおよびキャッシュ サイズをコンフィグレーションする」を参照してください。

ワークフロー プロセッサ Bean はクラスタにデプロイされます。WebLogic Server は、クラスタ内の各ノードがワークフロー プロセッサ Bean のローカル コピーを使用するように、クラスタ化システムを最適化します。

イベント リスナ メッセージ駆動型 Bean

イベント リスナ メッセージ駆動型 Bean は、イベント キューから作業を取り出し、ワークフロー プロセッサ Bean に送信します。イベント リスナ Bean は、ワークフロー プロセッサ Bean が実行を完了するか、新しい作業をキューから取り出す前に静止状態に達するまで、待機します。

イベント リスナ Bean は、順序付けされていないメッセージ用のプール サイズがコンフィグレーションされており、順序付けされているメッセージには一連の単一 Bean（使用可能なプール サイズ 1 を設定した名前付き Bean）からなるプールを使用します。『*BPM クライアント アプリケーション プログラミング*』の「JMS 接続の確立」にある「複数のイベント キューに対するメッセージ駆動型 Bean の生成」を参照してください。

これらのプールの組み合わせによって、イベントから開始された場合に同時に実行できるワークフロー数が決定されます。

テンプレート Bean

テンプレート Bean は、実行するワークフロー テンプレートを格納するエンティティ Bean です。通常、テンプレート エンティティ Bean プールのサイズは、同時に実行するワークフロー テンプレート（インスタンス数ではなくワークフロー数）の最大数と等しくする必要があります。一般に、プールのサイズは、小さすぎるよりも大きすぎる方が問題は少なくなります。テンプレート エンティティ Bean はクラスタ化できる（クラスタ対応スタブ）ので、クラスタ内の他のノードのワークフロー プロセッサ Bean で使用できます。

テンプレート定義 Bean

テンプレート定義 Bean は、実行するワークフロー テンプレート定義を格納するするエンティティ Bean です。

ビジネス プロセスは、データベースにワークフロー テンプレートとして保存されます。これらのテンプレートは、異なるバージョンのワークフローを保存できるように、基本的には空のコンテナです。また、複数の組織に関連付けることができます。テンプレートには、テンプレート定義が格納されますが、それらは、

同じワークフローの異なるバージョンで、有効期限によって区別されます。ビジネス プロセスおよびワークフローの詳細については、『*WebLogic Integration Studio ユーザーズ ガイド*』を参照してください。

通常、テンプレート定義エンティティ Bean プールのサイズは、同時に実行する *ワークフロー テンプレート*（インスタンス数ではなくテンプレート数）の最大数と等しくする必要があります。一般に、プールのサイズは、小さすぎるよりも大きすぎる方が問題は少なくなります。テンプレート定義エンティティ Bean はクラスタ化できる（クラスタ対応スタブ）ので、クラスタ内の他のノードのワークフロー プロセッサ Bean で使用できます。

インスタンス Bean

インスタンス Bean は、実行中のワークフロー インスタンスを格納するエンティティ Bean です。通常、インスタンス エンティティ Bean プールのサイズは、ワークフロー プロセッサ Bean プールのサイズと等しくする必要があります。インスタンス エンティティ Bean プールのサイズをワークフロー プロセッサ Bean の数より大きくしても利点はありません。一般に、プールのサイズは、小さすぎるよりも大きすぎる方が問題は少なくなります。インスタンス エンティティ Bean はクラスタ化できる（クラスタ対応スタブ）ので、クラスタ内の他のノードのワークフロー プロセッサ Bean で使用できます。

イベント キュー

1 つの JAR ファイルには、特定のキューに関して順序付けされたイベント リスナ メッセージ駆動型 Bean と順序付けされていない イベント リスナ メッセージ駆動型 Bean が格納されています。WebLogic Integration をインストールすると、デフォルトの EventQueue からのメッセージを消費する message-driven bean が入った `wli-mdb-ejb.jar` ファイルが提供されます。この JAR ファイルは、クラスタを対象とする必要があります。また、2-19 ページの「新しいプールを作成する」で説明するように、新しいイベント キューを作成することもできます。クラスタ内の BPM イベント キューの詳細については、2-17 ページの「クラスタにおける BPM 機能のロード バランシング」を参照してください。

注意： クラスタでの BPM 機能の規模を拡大するには、新しいイベント キューを作成する必要があります。

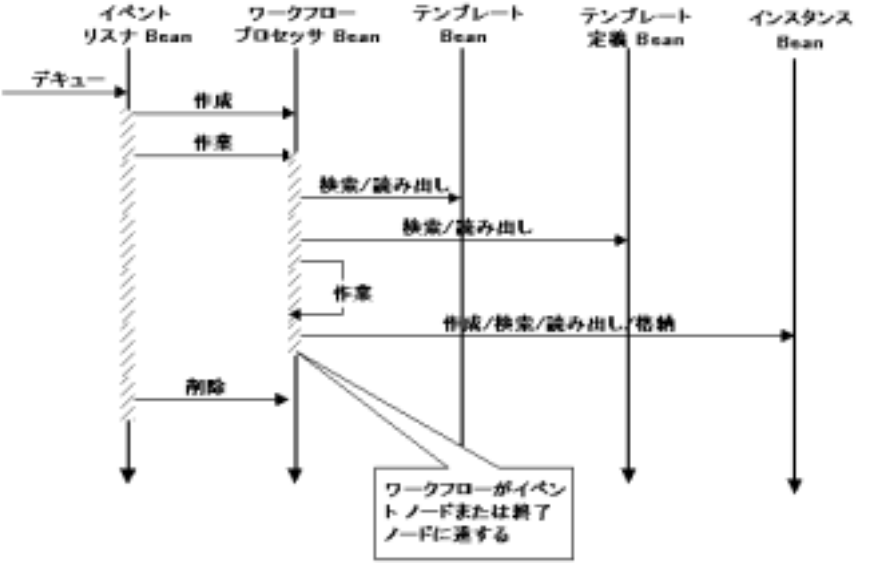
Worklist コンソール（非推奨）

Worklist クライアントには、BPM API からワークフローを作成するユーザ コードだけでなく、Swing ベースの WebLogic Integration Worklist コンソールも含まれています。図 1-2 には、コンテキストのみに関して示されています。Worklist コンソールは実行時にコンフィグレーション可能なリソースではないからです。

BPM 作業シーケンス

次の図は、イベントを処理する場合の BPM EJB 間の会話を示しています。

図 1-3 イベント処理時の BPM EJB 間の会話



BPM *イベント リスナ Bean* は、デフォルトまたはユーザ定義のイベント キューから作業要求を受け取ると、*ワークフロー プロセッサ Bean* を作成してその要求を処理します。ワークフロー プロセッサ Bean は、ワークフローが停止ノードまたはイベント ノードに達するまで、ワークフローを実行します。ワークフローが別のワークフローを呼び出した場合、新しいワークフロー プロセッサ Bean が作成され、呼び出し元ワークフローはそのワークフロー プロセッサ Bean を終了しません。

テンプレート Bean とテンプレート定義 Bean は、ワークフローの実行開始時に読み出されます。インスタンス Bean はワークフロー実行の開始時に読み出され、ワークフローがトランザクション境界（イベント ノードや終了ノードなど）で静止したときに書き込まれます。

イベント駆動型ワークフローの場合、ワークフロー プロセッサ Bean を追加作成してもデプロイメントの作業量は変わりません。同時に処理可能なワークフロー インスタンスの数は、イベント リスナ Bean の数によって制限されます。

B2B Integration リソース

WebLogic Integration をクラスタ化ドメインにデプロイする際、すべての B2B Integration リソースは、管理サーバ用リソースを除いて、クラスタ内の同種サーバに均一にデプロイする必要があります。つまり、高可用性、スケーラビリティ、パフォーマンスの向上を実現するためには、B2B integration リソースは、1 つのドメイン内でクラスタ化されたすべてのサーバを対象とする必要があります。クラスタ化および B2B Integration の詳細については、2-3 ページの「クラスタ デプロイメントの設計」を参照してください。

B2B Integration リソースは必要に応じて動的に割り当てられる場合が多く、デプロイメントを事前にコンフィグレーションすることはできません。B2B の負荷に対応してにコンフィグレーションできるリソースの詳細については、『*B2B Integration 管理ガイド*』の「コンフィグレーション要件」を参照してください。

B2B Integration の機能を使用するクラスタには、共用ファイル システムが必要です。Storage Area Network (SAN) またはマルチポート型のディスク システムをお勧めします。

注意： XOCP ビジネス プロトコルに基づく WebLogic Integration アプリケーションは、クラスタ環境ではサポートされません。

Application Integration リソース

以下の節では、WebLogic Integration がサポートしている Application Integration リソースの種類について説明します。

- 同期サービス呼び出し

- 非同期サービス呼び出し
- イベント
- アプリケーション ビューと接続ファクトリ

クラスタ化および Application Integration の詳細については、第 2 章「WebLogic Integration クラスタについて」を参照してください。

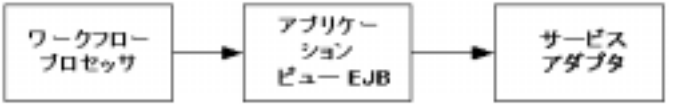
Application Integration 機能は、WebLogic Integration 製品に統合されていますが、単独の独立した J2EEear ファイルとしても入手できます。したがって、任意の有効な WebLogic ドメイン上に Application Integration をデプロイすることができます。たとえば、Web サービス開発者や WebLogic Portal 開発者は、アプリケーション ビューを使用して、EIS アプリケーションと会話することができます。WebLogic Integration 環境の外部への Application Integration のデプロイに関する詳細については、『*Application Integration ユーザーズ ガイド*』の「Application Integration のモジュラ デプロイメント」を参照してください。

同期サービス呼び出し

同期呼び出しを使用するのは、基盤となる EIS が要求に迅速に応答できる場合、またはクライアント アプリケーションが待機できる場合です。

次の図は、同期サービス呼び出しのフローを示しています。

図 1-4 同期サービス呼び出し



同期サービス呼び出しでは、クライアント（図中では *ワークフロー プロセッサ*）が *アプリケーション ビュー EJB*（ステートレス セッション Bean）を呼び出します。アプリケーション ビューは、同期 Common Client Interface (CCI) 呼び出しを使用して *サービス アダプタ*を呼び出します。サービス アダプタは、実際に要求を処理する J2EE-CA サービス アダプタです。

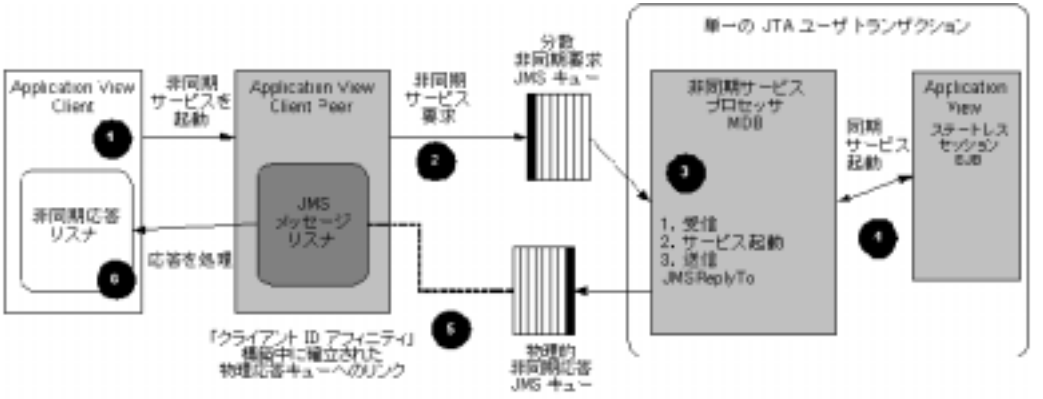
注意： ワークフローが EIS のクライアントとしての役割を果たす場合、ワークフロー プロセッサは、ワークフロー プロセッサ Bean と場合によってはイベント リスナ Bean も一緒に、要求が完了するまでストールします。

スループットを最適化するため、基盤となる EIS システムが要求に迅速に応答できない限り、非同期呼び出しを使用することを検討してください。

非同期サービス呼び出し

次の図は、WebLogic Integration の非同期サービス処理を示しています。

図 1-5 非同期サービス呼び出し



注意： WebLogic Integration クラスタ内の分散送り先として、`WLAI_ASYNC_REQUEST_QUEUE` キューおよび `WLAI_ASYNC_RESPONSE_QUEUE` キューがデプロイされています。非同期サービス要求プロセッサは message-driven bean (`wlai-asyncprocessor-ejb.jar`) で、これもクラスタにデプロイされています。高可用性を実現するための Application Integration リソースのデプロイ方法の詳細については、2-23 ページの「高可用性を備えた JMS」を参照してください。

上の図は、Application Integration の非同期サービスの以下に説明するプロセスフローを示したものです。

1. アプリケーション ビューのクライアントがアプリケーション ビューのインスタンスを開始します。

クライアントには、構築時に恒久 ID を定義するオプションが与えられています。恒久クライアント ID は、非同期応答メッセージに対する相関 ID として使用されます。

クライアントは `invokeServiceAsync` メソッドを呼び出して、要求 IDocument を非同期サービス応答（AsyncServiceResponse）リスナに渡して、応答を処理させます。

2. アプリケーション ビューのインスタンスは `AsyncServiceRequest` オブジェクトを作成して、`WLAI_ASYNC_REQUEST_QUEUE` に送信します。

   `AsyncServiceRequest` オブジェクトには、応答リスナが対象とする送り先の名前が格納されています。The `AsyncServiceProcessor` message-driven bean は、この情報を使用して、応答の物理送り先を決定します。

   応答オブジェクトに、応答の送り先の名前が入っていない場合は、`AsyncServiceProcessor` message-driven bean は、JMS メッセージ用に指定されている送り先を使用します（`JMSReplyTo()` メソッドへの呼び出しにより）。

   そのクライアントのみが `AsyncServiceResponseListener` をアプリケーション ビューに提供しているとします。

   ```
   invokeServiceAsync(String serviceName, IDocument request,
   AsyncServiceResponseListener listener);
   ```

   このシナリオでは、アプリケーション ビューは、アプリケーション ビュー EJB メソッドの `getAsyncResponseQueue`*JNDIName*`()` によって定義される JNDI ロケーションに結び付けられた JMS キューに受信者を確立します。アプリケーション ビューのインスタンスは、`QueueReceiver.getQueue()` を使用して、要求メッセージに対して `ReplyTo` 送り先を設定します。

3. クラスタでは、`WLAI_ASYNC_REQUEST_QUEUE` キューは分散 JMS キューとしてデプロイされます。ただし、各メッセージは、1 つの物理キューに送信されるので、そのキューからのみ取り出すことができます。その物理キューが、メッセージがそこから取り出される前に使用できなくなると、物理キューが手動 JMS の移行またはサーバの再起動によって再び使用できるようになるまで、そのメッセージ（非同期サービス要求）は、使用できないままになります。

   分散キューにメッセージを送信しただけでは、そのキューの QueueReceiver によって必ず受信されるとはかぎりません。メッセージは 1 つの物理キューにのみ送信されるため、その物理キューでリスンしている QueueReciever が必要です。この要件を満足するため、AsyncServiceProcessor

(wlai-asyncprocessor-ejb.jar) をクラスタ内のすべてのノードにデプロイする必要があります。

AsyncServiceProcessor message-driven bean は、FIFO (先入れ先出し) 方式でメッセージを受信します。

AsyncServiceProcessor は、JMS ObjectMessage にある AsyncServiceRequest オブジェクトを使用して、アプリケーション ビューの完全修飾名、サービス名、要求ドキュメントおよび応答送り先を決定します。

4. AsyncServiceProcessor は、アプリケーション ビュー EJB を使用して、サービスを同期的に呼び出します。このサービスは、リソース アダプタに対する同期 CCI ベースの要求または応答メッセージに変換されます。

5. AsyncServiceProcessor はその応答を受け取ります。その後、この応答は、AsyncServiceResponse オブジェクトとしてカプセル化され、AsyncServiceRequest オブジェクトで指定された応答の送り先に送信されます（この場合は、WLAI_ASYNC_RESPONSE_QUEUE_myserver1）。

   AsyncServiceProcessor は、分散送り先（WLAI_ASYNC_RESPONSE_QUEUE）ではなく指定された物理送り先（WLAI_ASYNC_RESPONSE_QUEUE_myserver1）に応答を送信する必要があります。物理送り先のキューは、アプリケーション ビュー EJB の getAsyncResponseQueueJNDIName() メソッドが呼び出されたときにクライアント上で実行されているアプリケーション ビューのインスタンスによって確立されています（詳細は、手順 2. 参照）。

   注意： 受け取るべきすべての応答メッセージを受信する前に、クライアント アプリケーションに障害が発生することもあります。障害からの回復後に、クライアントと JMS 応答キューの関連付けが障害発生前のとおりに回復されていることを確認するには、回復後も、以前と同じクライアント ID を使用する必要があります。次のリストは、回復コードの例です。障害発生前と回復後で同じユニークなクライアント ID を使用することにより、クライアントと JMS 応答キューとのこのような関連付けをサポートしています。

```
String uniqueClientID = "uniqueClientID";

ApplicationView myAppView = new ApplicationView(jndiContext,
"MyAppView", uniqueClientID);

myAppView.recoverAsyncServiceResponses(new
MyAsyncResponseListener());
```

6. アプリケーション ビューのインスタンス開始時に作成されたアプリケーション ビュー メッセージ リスナのインスタンスは、AsyncServiceResponse メッセージを JMS ObjectMessage として受信し、手順 2. で示した `invokeServiceAsync()` 呼び出しで指定された AsyncServiceResponseListener に渡します。

イベント

Application Integration アダプタは、BPM または WebLogic Workshop によって消費されるイベントを生成します。イベントは、アプリケーション ビューから JMS キュー（`WLAI_EVENT_QUEUE`）へ転送されます。このキューは、複数の物理送り先が格納された分散送り先です。message-driven bean (WLI-AI Event プロセッサ) は `WLAI_EVENT_QUEUE` 分散送り先でリスンします。

WLI-AI イベント プロセッサは以下のタスクを実行します。

- イベントのコピーを、BPM イベント プロセッサに配信（BPM がインストールされていて、サーバ インスタンスで実行されている場合）またはアプリケーション ビューの WebLogic Workshop Control イベント プロセッサに配信（このコントロールを使用している場合）します。

  各イベントのコピーが正確に 1 つずつ BPM または WebLogic Workshop に配信されます。

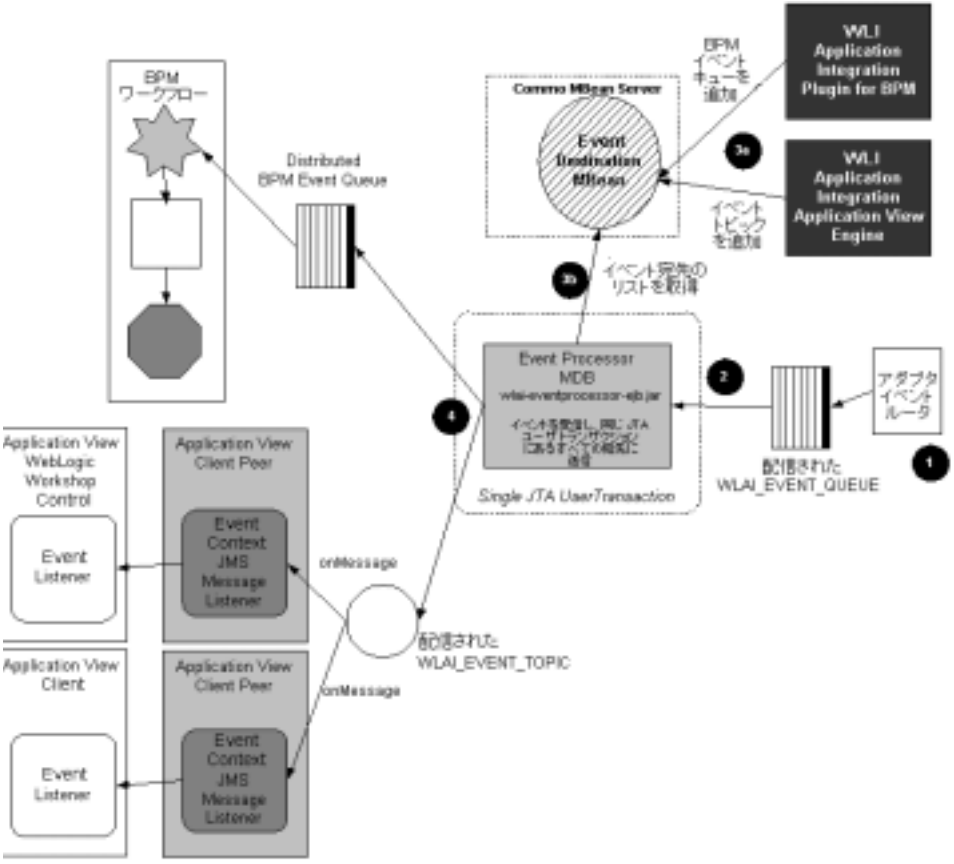
- イベントのコピーを `WLAI_EVENT_TOPIC` に公開する。

  `WLAI_EVENT_TOPIC` は、分散 JMS トピックで、リモートのアプリケーション ビュー クライアントへのイベントの配信を処理します。アプリケーション ビュー クラスが `WLAI_EVENT_TOPIC` に EventContext を作成します。EventContext クラスによって、アプリケーション ビューの名前に基づいてメッセージのフィルタリングが行われます。アプリケーションビューの名前は `SourceKey` JMS ヘッダ プロパティに格納されています。`SourceType` は `ApplicationView` です。

- 処理に失敗した場合にキューにメッセージがロール バックできるように、トランザクションを通じてイベントをキューから取り出す。

次の図は、WebLogic Integration のイベント処理を示しています。

図 1-6 イベント



上の図は、次に説明するイベント処理手順のフローを示しています。

1. イベントはエンタープライズ情報システム（EIS）で発生し、次のような仕組みで JMS キューに送信されます。

   a. イベントが EIS で発生します。

b. イベント データがリソース アダプタのイベント ジェネレータに送信されます。イベント ジェネレータは、EIS 固有データを XML ドキュメントに変換し、IEvent オブジェクトをイベント ルータ（アダプタ イベント ルータ）にポストします。

c. イベント ルータは、特定のイベント タイプに関心のある各 Application Integration サーバに対して、IEvent オブジェクトを EventContext オブジェクトに渡します。

d. EventContext は IEvent オブジェクトを JMSObjectMessage としてカプセル化し、JMS QueueSender を使用して、`com.bea.wlai.EVENT_QUEUE` という JNDI コンテキストにバインドされている JMS キューに送信します。

2. ObjectMessage `WLAI_EVENT_QUEUE` に格納されて、WLI-AI Event プロセッサ message-driven bean (`wlai-eventprocessor-ejb.jar`) によって FIFO で処理されます。

   クラスタでは、`WLAI_EVENT_QUEUE` キューは分散 JMS キューとしてデプロイされます。ただし、各メッセージは、1 つの物理キューに送信され、送り先の物理キューからのみ使用できます。その物理キューが、メッセージがそこから取り出される前に使用できなくなると、物理キューが再びオンラインに戻るまでは、そのメッセージ（すなわち、イベント）は使用できないままになります。

   分散キューにメッセージを送信しただけでは、その分散キューの QueueReceiver によって必ず受信されるとはかぎりません。メッセージは 1 つの物理キューにのみ送信されるので、その物理キューでリスンしている QueueReciever が必要です。この要件を満足するため、WLI-AI Event プロセッサ message-driven bean (`wlai-eventprocessor-ejb.jar`) をクラスタ内のすべてのノードにデプロイする必要があります。

3. WLI-AI Event プロセッサ message-driven bean (`wlai-eventprocessor-ejb.jar`) がイベントの送り先のリストを決定する。

   a. イベントの送り先が AIDestinationMBean に追加されます。イベント送り先の同一リストが各管理対象サーバのイベントプロセッサ message-driven bean に渡されるように、MBean はクラスタ全体で複製されます。WLI-AI BPM プラグイン（`wlai-plugin-ejb.jar`）は、デプロイ時に BPM イベント キューをイベントの送り先として追加します。この送り先を追加することにより、EIS イベントを BPM プロセス エンジンに送信することが可能になります。また、アプリケーション ビューのイベント リスナが登録されていると、イベントは `WLAI_EVENT_TOPIC` に送信されます。

b. WLI-AI イベント プロセッサ message-driven bean が、イベントを送信するためのイベント送り先のリストを MBean から読み込みます。

4. ObjectMessage イベントが 1 つの JTA ユーザ トランザクション内のすべての登録済みイベント送り先に配信される。メッセージがどのイベント送り先へも配信されない場合は、`WLAI_EVENT_QUEUE` にロール バックされます。`WLAI_EVENT_QUEUE` は、不良メッセージは WebLogic Integration エラー送り先（`com.bea.wli.FailedEventQueue`）に転送するようにコンフィグレーションされています。`FailedEventQueue` の詳細については、2-30 ページの「エラー送り先」を参照してください。

注意： イベントの送り先は通常、JMS の送り先なので、システムがイベントの転送に失敗する可能性はほとんどありません。

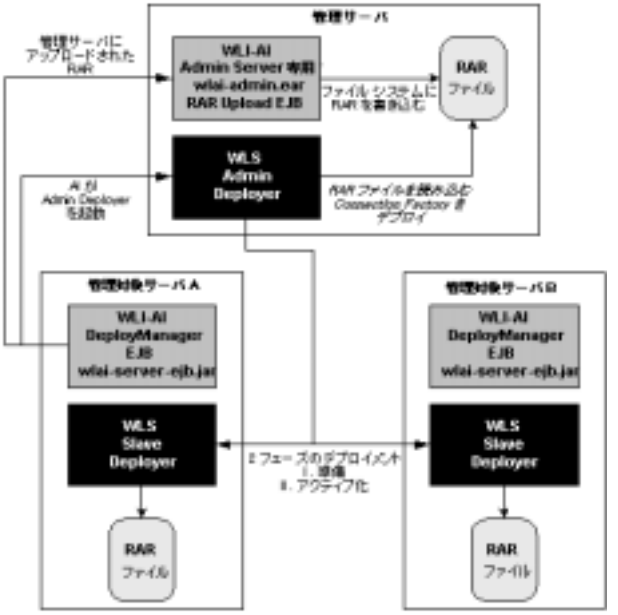
アプリケーションビューと接続ファクトリ

これまでの節で説明した実行時 Application Integration 機能（同期サービス呼び出し、非同期サービス呼び出し、およびイベント）をクラスタ化することにより、スケーラビリティと可用性が向上します。設計時 Application Integration 機能（アプリケーション ビューおよび接続ファクトリ）をクラスタ化することにより、スケーラビリティが向上しますが、可用性を高めることはできません。つまり、アプリケーション ビューは、クラスタ内のサーバが実行されていない場合はデプロイもアンデプロイ（編集）も不可能です。言い換えれば、デプロイおよびアンデプロイ（編集）は、健全なクラスタにおいてのみ可能です。

リソース アダプタ（RAR）は、`wlai-admin.ear` アーカイブファイルを、クラスタ化された管理対象サーバにではなく、管理サーバにデプロイすることによりアップロードされます。2 段階デプロイメントが行われます。管理サーバ上の WebLogic Deployer ユーティリティが、管理サーバへのデプロイメントをコントロールします。

次の図は、設計時の接続ファクトリ デプロイメントを示しています。

図 1-7 接続ファクトリ デプロイメント



アプリケーション ビューのデプロイメントの成否は、接続ファクトリのデプロイメントが正常に行われるかどうかによります。クラスタ化環境への Application Integration アダプタのデプロイメントの詳細については、2-21 ページの「クラスタにおける Application Integration 機能のロード バランシング」および 2-32 ページの「アダプタのデプロイ」を参照してください。

リレーショナル データベース管理システム リソース

WebLogic Integration は、実行時の処理を行うためやアプリケーション データの永続性を保証するためにデータベース リソースを利用します。データベースのパフォーマンスは、WebLogic Integration 全体のパフォーマンスに大きく影響します。詳細については、6-38 ページの「データベースのチューニング」を参照してください。

ハードウェア、オペレーティング システム、およびネットワークのリソース

ハードウェア、オペレーティング システム、およびネットワークのリソースは、WebLogic Integration のパフォーマンスに重要な影響を与えます。したがって、デプロイメントは、『*WebLogic Integration リリース ノート*』で説明されているハードウェアおよびソフトウェア要件に準拠している必要があります。プロダクション環境で最大限のパフォーマンスを引き出すためのリソースのコンフィグレーションの詳細については、4-2 ページの「推奨ハードウェアおよびソフトウェア」および 6-34 ページの「ハードウェア、オペレーティング システム、およびネットワークのリソースのチューニング」を参照してください。

2 WebLogic Integration クラスタについて

以下の節では、WebLogic Integration をクラスタ環境でコンフィグレーションおよびデプロイメントする方法について説明します。内容は以下のとおりです。

WebLogic Integration クラスタについて

クラスタ化すると、1 つの単位として管理可能なサーバ群で WebLogic Integration を実行できます。クラスタ環境では、複数のマシンが負荷を分担します。WebLogic Integration は、リソース要求がすべてのマシン間で均等に分散されるようにロード バランシング機能を提供します。WebLogic Integration デプロイメントでは、クラスタ化とロード バランシングを使用してノード間で負荷を分散することで、スケーラビリティを向上できます。クラスタ化すると、1 つのサーバよりもスケーラブルなデプロイメント プラットフォームを実現できます。

WebLogic Server ドメインは、ただ 1 つの管理サーバと 1 つまたは複数の管理対象サーバで構成されます。WebLogic Integration ドメイン内の管理対象サーバをクラスタにまとめることができます。WebLogic Integration のクラスタ対応リソースをコンフィグレーションする場合、そのリソースのデプロイ対象とするクラスタを指名します。クラスタをリソース デプロイメントの対象として指定すると、管理対象サーバをクラスタに追加することによって能力を動的に増大できるという利点が得られます。

ここでは、クラスタ環境で WebLogic Integration をコンフィグレーションする場合に必要な情報について説明します。WebLogic Server がクラスタをサポートする方法についても背景知識として取り上げますが、クラスタ環境に合わせた WebLogic Integration のコンフィグレーションに固有の手順が中心です。

読み進む前に、WebLogic Server マニュアルの以下の節の内容を確認し、クラスタ化について十分に理解しておくことをお勧めします。

- 次の URL にある『*WebLogic Server クラスタ ユーザーズ ガイド*』

  `http://edocs.beasys.co.jp/e-docs/wls/docs70/cluster/index.html`

- 次の URL にある『*WebLogic Server クラスタ ユーザーズ ガイド*』の「クラスタのコンフィグレーションとアプリケーションのデプロイメント」

  `http://edocs.beasys.co.jp/e-docs/wls/docs70/cluster/config.html`

- 次の URL にある『*BEA WebLogic Server パフォーマンス チューニング ガイド*』、「WebLogic Server のチューニング」の「WebLogic Server クラスタとスケーラビリティ」

  `http://edocs.beasys.co.jp/e-docs/wls/docs70/perform/WLSTuning.html`

クラスタ デプロイメントの設計

以下の節では、クラスタ デプロイメントの設計に必要な情報について説明します。

- WebLogic Integration ドメイン入門
- WebLogic Integration リソースのデプロイメント
- WebLogic Integration クラスタにおけるロード バランシング

WebLogic Integration ドメイン入門

クラスタ化したドメインに合わせてアーキテクチャを設計する前に、WebLogic Server クラスタがどのように運用されるかを知っておく必要があります。

ドメインを作成する

ドメインおよびクラスタの作成は、WebLogic Integration、Business Process Management (BPM)、または Enterprise Application Integration (EAI) の機能に基づいて、ドメイン テンプレートからドメインが生成ができる Configuration Wizard により簡素化されています。ユーザのクエリに基づいて、Configuration Wizard によって、ドメイン、サーバ、エンタープライズ アプリケーションなどが、コンフィグレーション済みの適切なコンポーネントおよび資産を含めて生成されます。さまざまなドメインに利用できるテンプレートの詳細については、次の URL にある『*コンフィグレーション ウィザード テンプレート リファレンス*』を参照してください。

http://edocs.beasys.co.jp/e-docs/platform/docs70/template/index.htm

Configuration Wizard による WebLogic Integration ドメイン作成の詳細については、第 3 章「クラスタ デプロイメントのコンフィグレーション」 を参照してください。

クラスタ サーバ

サーバには管理対象サーバと管理サーバがあります。管理サービスを実行する WebLogic Server は管理サーバと呼ばれ、Administration Console のホストとなります。複数の WebLogic Server が稼働しているドメインでは、そのうち 1 つのみが管理サーバで、他のサーバは管理対象サーバと呼ばれます。各管理対象サーバは、起動時に管理サーバからコンフィグレーションを取得します。

概要については、次の URL にある WebLogic Server マニュアルの『*WebLogic Server クラスタ ユーザーズ ガイド*』を参照してください。

http://edocs.beasys.co.jp/e-docs/wls/docs70/cluster/index.html

推奨する基本的な多層プロキシ アーキテクチャの詳細については、『*WebLogic Server クラスタ ユーザーズ ガイド*』の「クラスタ アーキテクチャ」を参照してください。

クラスタおよび管理ドメインに関する注意

WebLogic Server の管理ドメインがクラスタ ドメインとは別のものである場合、（WebLogic Server クラスタが別の管理ドメインに属するノードを持つ場合）もありますが、WebLogic Integration デプロイメントの場合は、クラスタ ドメインが管理ドメインとなるように設計する必要があります。

WebLogic Integration リソースのデプロイメント

クラスタ化されたドメインの各サーバに対して、そのドメイン内のサーバの機能を定義するさまざまな属性をコンフィグレーションできます。これらの属性は Administration Console の Servers ノードを使用してコンフィグレーションされます。

この節では、WebLogic Integration リソースおよびクラスタ内でのそれらのパーティショニングと分散化について説明します。内容は以下のとおりです。

- クラスタ対応リソース
- WebLogic Integration の 2 段階デプロイメント

- 分散に関するガイドライン
- WebLogic Integration アプリケーションにおけるデプロイメントの順序
- デフォルト Web アプリケーションをデプロイする
- 管理サーバに関する注意

クラスタ対応リソース

表 2-1 では、WebLogic Integration デプロイメントのリソースについて説明します。内容は以下のとおりです。

- リソース グループ―クラスタ化するために分類された、互いに関連性を持つデプロイメント リソースの任意に指定したセットです。リソース グループ内のすべてのリソースは、同じマシンを対象としたものでなければなりません。

  リソース グループには 2 種類あります。

  - *クラスタ対応*リソース― 1 つまたは複数のサーバを対象とすることができますが、グループのすべてのメンバは同じサーバまたはサーバのセットを対象としていなければなりません。
  - *シングル ノード*―クラスタ内の 1 つかつ唯一のサーバを対象とします。シングル ノード リソースをクラスタ化することはできません。

- リソース名― WebLogic Server Console に表示される個々のパッケージまたは（あるリソース グループ内の）サービスの名前です。

  一部のリソース名には、以前のリリースの WebLogic Integration から受け継いだ略語が使用されているので注意してください。

  - `wlc` は B2B Integration に対応します。
  - `wlpi` は BPM に対応します。
  - `wlai` は Application Integration に対応します。

- Administration Console のナビゲーション― WebLogic Server Administration Console のナビゲーション ツリーをたどって、指定されたパッケージまたはサービスまでのルートを表します。これらのリソースはすべて、Administration Console で表示したり、変更したりすることができます。

次の表では、WebLogic Integration デプロイメントのリソースについて説明します。

表 2-1 WebLogic Integration デプロイメントのリソース

リソース グループ	説明 (シングル ノード / クラスタ対応)	リソース名	Administration Console のナビゲーション
bpm-singleNode	BPM マスタ コンポーネント シングル ノード	[WLI-BPM Plugin Manager] (wlpi-master-ejb.jar)	[デプロイメント	EJB]
bpm-clusterable	BPM コンポーネント クラスタ対応	[WLI-BPM initialization] (bpm-init-ejb.jar)	[デプロイメント	EJB]
		[WLI-BPM Server] (wlpi-ejb.jar)	[デプロイメント	EJB]
		[WLI-BPM Event Processor MDBs] (wlpi-master-ejb.jar)	[デプロイメント	EJB]
		[User-defined Event Processor MDBs] wlpi-mdb-xxx.jar[1]	[デプロイメント	EJB]
		[wlpiFactory] (com.bea.wlpi.TopicConnectionFactory)	[サービス	JMS	Connection Factories]
		[wlpiQueueFactory] (com.bea.wlpi.QueueConnectionFactory)	[サービス	JMS	Connection Factories]
		[TXDataSource]	[サービス	JDBC	Tx Data Sources]

表 2-1 WebLogic Integration デプロイメントのリソース (続き)

リソース グループ	説明 (シングル ノード / クラスタ対応)	リソース名	Administration Console のナビゲーション
B2B-singleNode	B2B Integration 管理 シングル ノード : 管理サーバ)	[B2B console] (b2bconsole.war)	[デプロイメント	WebApplications]
		[WLI-B2B Startup] (b2b-startup.jar) **注意 :** 管理サーバおよびクラスタ化管理対象サーバにデプロイされます。	[デプロイメント	EJB]

表 2-1 WebLogic Integration デプロイメントのリソース (続き)

リソース グループ	説明 (シングル ノード / クラスタ対応)	リソース名	Administration Console のナビゲーション
B2B-clusterable	B2B Integration コンポーネント クラスタ対応	[WLI-B2B Startup] (b2b-startup.jar)	[デプロイメント	EJB]
		[WLCShutdown]	[デプロイメント	起動とシャットダウン]
		[WLCHub.DS]	[サービス	JDBC	Tx Data Sources]
		[TransportServlet] (b2b.war)	[デプロイメント	WebApplications]
		[WLI-B2B RN MDB] (b2b-rosettanet.jar)	[デプロイメント	EJB]
		[WLI-B2B RN BPM Plug-in] (wlc-wlpi-plugin.jar)	[デプロイメント	EJB]
		[WLI-B2B ebXML BPM Plug-in] (ebxml-bpm-plugin.jar)	[デプロイメント	EJB]
		[RNQueueFactory] (com.bea.wli.b2b.rosettanet.QueueConnectionFactory)	[サービス	JMS	Connection Factories]
		[B2BTopicFactory] (com.bea.wli.b2b.rosettanet.QueueConnectionFactory)	[サービス	JMS	Connection Factories]

表 2-1 WebLogic Integration デプロイメントのリソース (続き)

リソース グループ	説明 (シングル ノード / クラスタ対応)	リソース名	Administration Console のナビゲーション
AI-admin	Application Integration 管理 (シングル ノード管理サーバ[2]	[WLI-AI RAR Upload] (wlai-admin.ear)	[デプロイメント	アプリケーション	WLI-AI RAR Upload]
AI-clusterable	Application Integration コンポーネント クラスタ対応	[WLI-AI Server] (wlai-server-ejb.jar)	[デプロイメント	EJB]
		[ApplicationViewManagementConsole] (wlai.war)	[デプロイメント	Web Applications	wlai]
		[WLI-AI Event Processor] (wlai-eventprocessor-ejb.jar)	[デプロイメント	EJB]
		[WLI-AI Async Processor] (wlai-asyncprocessor-ejb.jar)	[デプロイメント	EJB]
		[WLI-AI BPM Plug-in] (wlai-plugin-ejb.jar)	[デプロイメント	EJB]
		[WLI-AI BPM Plug-in Help] (wlai-plugin.war)	[デプロイメント	WebApplications]
		[WLAI_JMSConnection Factory]	[サービス	JMS	Connection Factories]
wlai-event-yyy[1]	Application Integration イベント アダプタ アダプタの種類によって異なる。[3]	*yyyEventRouter*[1]	[デプロイメント	アプリケーション] *yyy*EventRouter[1,4]

表 2-1 WebLogic Integration デプロイメントのリソース (続き)

リソース グループ	説明 (シングル ノード / クラスタ対応)	リソース名	Administration Console のナビゲーション
wlai-service-yyy[1]	Application Integration サービス アダプタ **アダプタの種類によって異なる。**[3]	*BEA . yyy . ADK_RAR*[1]	[デプロイメント	アプリケーション	BEA . *yyy* . ADK_RAR][1,4]
		[BEA . yyy . ADK_WEB][1]	[デプロイメント	アプリケーション	*BEA. . yyy . ADK_WEB*]
DI-clusterable	Data Integration コンポーネント **クラスタ対応**	[WLI-DI BPM Plug-in] (`wlxtpi.jar`)	[デプロイメント	EJB]
		[WLI-DI BPM Plug-in Help] (`wlxtpi.war`)	[デプロイメント	Web Applications]
wli-clusterable	ドメイン内のすべてのサーバに配置しなければならないリソース **クラスタ対応**	[WLI-Repository] (`respository-ejb.jar`)	[デプロイメント	EJB]
		[WLI Error Listener] (`wli-errorlistener-mdb.jar`)	[デプロイメント	EJB]
		[Mailsession] (`wlpiMailSession`)	[サービス	メール] BPM Send E-mail アクション用の Java メール セッション
		[JDBCConnectionPool] (`wliPool`)	[サービス	JDBC	接続プール] WebLogic Integration でのすべてのデータベース接続用

1. リソース名はユーザ定義パッケージまたはリソース グループを表しています。

2. `wlai-admin.ear` は、WebLogic Integration をクラスタにデプロイする場合にのみデプロイする必要があります。シングル ノード環境にデプロイする場合はデプロイしないでください。Application Integration コンポーネントの詳細については、2-21 ページの「クラスタにおける Application Integration 機能のロードバランシング」を参照してください。
3. たとえば、DBMS サンプル イベント アダプタはシングル ノードにデプロイされます。詳細については、アダプタのマニュアルを参照。
4. イベントおよびサービス アダプタは 1 つの EAR ファイルに格納されますが、別々にデプロイされ、WebLogic Server Administration Console では別個のリソースとして示されます。詳細については、「WebLogic Integration の 2 段階デプロイメント」の節を参照してください。

WebLogic Integration の 2 段階デプロイメント

システムのメッセージ処理が開始される前にすべての WebLogic Integration アプリケーション コンポーネントをデプロイすることが大切です。それを保証するため、WebLogic Integration デプロイするときはを TwoPhase 属性を指定します。サンプルの `config.xml` ファイルからの次の抜粋は、WebLogic Integration のデプロイメントを指定する application 要素を示しています。

コード リスト 2-1 WebLogic Integration アプリケーションのデプロイメント

```
<Domain Name="MyCluster">
...
    <Application Name="WebLogic Integration" Path="WLI_HOME/lib"
TwoPhase="true">
...
```

分散に関するガイドライン

WebLogic Integration クラスタ デプロイメントは以下のガイドラインに従って行われます。

- ほとんどのリソースは、ドメイン内のすべてのサーバにデプロイする必要があります。WebLogic Integration リソースをデプロイする対象が、クラスタ内のすべての管理対象サーバなのか、1 つの管理対象サーバなのか、あるクラスタの管理サーバなのか、などの情報については、2-4 ページの「WebLogic Integration リソースのデプロイメント」を参照してください。
- 同じリソース グループのメンバーとして識別されるリソースは、同じサーバ、または同じサーバのセット（クラスタ対応の場合）を対象としていなけ

ればなりません。2-4 ページの「WebLogic Integration リソースのデプロイメント」を参照してください。

- 管理サーバは、すべての WebLogic Integration リソースを必要とするわけではありませんが、次のリソースはデプロイしてください。
  - [B2B Console](`b2bconsole.war`)
  - [WLI-B2B Startup] (`b2b-startup.jar`)
  - [WLI-AI RAR Upload](`wlai-admin.ear`)
  - [B2BTopic JMS Destination] (`com.bea.wli.b2b.server.B2BTopic`)
- あるノード上の JMS キューの数は、2-17 ページの「クラスタにおける BPM 機能のロード バランシング」と 2-21 ページの「クラスタにおける Application Integration 機能のロード バランシング」で説明したガイドラインに従って決定する必要があります。WebLogic Integration デプロイメントの JMS リソースの使用法の詳細については、2-25 ページの「JMS リソース」を参照してください。

リソースの対象としてクラスタを指定する

2-4 ページの「WebLogic Integration リソースのデプロイメント」で示したとおり、ほとんどの WebLogic Integration リソースは、クラスタ内のすべてのサーバにデプロイされます。このデプロイメントは、使用ドメインのコンフィグレーション ファイル（`config.xml`）で指定されます。

WebLogic Server Administration Console を使用して、コンポーネントの対象としてクラスタ内のノードを指定することができます。詳細については、第 3 章「クラスタ デプロイメントのコンフィグレーション」を参照してください。

次のリストは、クラスタ ドメイン用のコンフィグレーション ファイルの抜粋で、BPM コンポーネントが指定されています。このリストは、これらのコンポーネントの対象として、MyCluster というクラスタが指定されていることを示しています。

コード リスト 2-2 WebLogic Integration コンポーネントの対象としてクラスタを指定する

```
<Application Deployed="true" Name="WebLogic Integration"
    Path="C:/bea/weblogic700/integration/lib" TwoPhase="true">

<!--Repository-->
```

```
    <EJBComponent Name="WLI Repository" Targets="MyCluster"
     URI="repository-ejb.jar" />
<!--BPM-->
    <EJBComponent Name="WLI-BPM Server" Targets="MyCluster"
     URI="wlpi-ejb.jar" />
    <EJBComponent Name="WLI-BPM Event Processor"
     Targets="MyCluster" URI="wlpi-mdb-ejb.jar" />
    <EJBComponent Name="WLI-BPM Master Components"
     Targets="MyServer-1" URI="wlpi-master-ejb.jar" />
    <EJBComponent Name="WLI-BPM Initialization"
     Targets="MyCluster" URI="bpm-init-ejb.jar"/>
...
</Application>
```

上のリストでは、WLI-BPM マスタ コンポーネント（wlpi-master-ejb.jar）以外のすべての BPM コンポーネントの対象として、このクラスタが指定されています。表 2-1 で指定されているとおり、クラスタ内の 1 つのサーバ（この場合は MyServer-1）に、WLI-BPM マスタ コンポーネントをデプロイする必要があります。

WebLogic Integration アプリケーションにおけるデプロイメントの順序

次のファイルでは、WebLogic Integration のすべてのコンポーネントが指定されています。

WLI_HOME\lib\META-INF\application.xml

コンポーネントは、application.xml にリストされている順序でデプロイされるので、ファイルにリストする順序は変更しないでください。指定された順序は、コンポーネント間の依存関係を表しているので重要です。このアプリケーションには、BPM 機能がアクセスする必要のある EJB プラグインおよび BPM プラグインが組み込まれています。

カスタム リソース（カスタムのプラグインや EJB、message-driven bean など）を WebLogic Integration アプリケーションにデプロイする場合は、application.xml ファイルを編集して新しいコンポーネントを指定する必要があります。

警告： カスタムの新しいリソースが BPM へのプラグインでない場合は、カスタムリソースは、`application.xml` ファイルの最後のエントリとして指定することができます。それが BPM へのプラグインである場合は、新しいコンポーネントは最後から 2 番目のエントリとして指定します。すなわち、`bpm-init-ejb.jar` モジュールの直前、アプリケーション内の他のすべてのモジュールの後に指定してください。

`bpm-init-ejb.jar` モジュールは、`application.xml` の最後に指定されます。

```
<module>
    <ejb>bpm-init-ejb.jar</ejb>
</module>
```

デフォルト Web アプリケーションをデプロイする

デフォルトでは、WebLogic Integration ドメイン テンプレートのいずれかに基づいてドメインを作成すると、そのドメインには、管理サーバにデプロイされる Web サーバのコンフィグレーションが格納されます。すると、この Web サーバのコンフィグレーションによって、デフォルトの Web アプリケーション（`DefaultWebApp`）が指定されます。

このデフォルトの Web アプリケーションに対するデプロイメント記述子（`web.xml`）は、次の場所にあります。

`DOMAIN_HOME``\applications\DefaultWebApp_myserver\WEB-INF\`

上の行で、*`DOMAIN_HOME`* は作成したドメインのパス名を表しています。

Web アプリケーションには、サーブレット、Java Server Pages (JSP)、JSP タグライブラリなどのアプリケーションのリソースや HTML ページおよびイメージファイルなどの静的リソースが格納されます。

カスタム JSP および HTML ページをデプロイする

カスタムの JSP または HTML ページを WebLogic Integration アプリケーションの一環としてデプロイする場合は、それらの JSP や HTML ページは、次のディレクトリに配置する必要があります。

`DOMAIN_HOME``\applications\DefaultWebApp_`*`node`*

上のパスで、*DOMAIN_HOME* は、Configuration Wizard を使用して作成したカスタム ドメインのルート ディレクトリを表し（3-7 ページの「手順 2. WebLogic Integration ドメインの作成」参照）、*node* はクラスタ内の WebLogic Server インスタンスの名前を表しています。

Web サーバは、クラスタ内のノードごとにコンフィグレーションする必要があります。config.xml ファイルからの以下の抜粋は、次の要素を示しています。

- managedserver1 という名前の管理対象サーバに対してコンフィグレーションされた WebServer 要素
- デフォルト Web アプリケーションに対してコンフィグレーションされた Application 要素

（関係のある情報は強調するため、以下のリストでは太字で表記されている）。

コード リスト 2-3　config.xml ファイルに格納された管理対象サーバ用 WebServer 要素

```
<Server Name="managedserver1" ...
...
<WebServer Name="managedserver1" DefaultWebApp="DefaultWebApp_node"
    HttpsKeepAliveSecs="120" KeepAliveSecs="60"
    LogFileName="C:/bea/user_projects/mydomain/logs/access.log"
    LoggingEnabled="true"/>
...
</Server>

    :

<Application Deployed="true" Name="DefaultWebApp_node"
   Path="C:/bea/weblogic700/samples/integration/config/samples/RN2Security/
config/peer2/applications"
    StagedTargets="" TwoPhase="false">
    <WebAppComponent IndexDirectoryEnabled="true"
    Name="DefaultWebApp_node" Targets="managedserver1"
    URI="DefaultWebApp_node"/>

</Application>
```

上のリストでは、WebServer 要素の DefaultWebApp 属性は、デフォルトの Web アプリケーション コンポーネントを参照します。このリストには、デフォルト Web アプリケーションのコンフィグレーションも示されています。このデフォルト Web アプリケーションは、JSP および HTML ページが格納されているディレクトリを参照します（*node* はクラスタ内のサーバ名を表している）。

Web アプリケーションのデプロイメントに関する詳細は、次の URL にある『*Web アプリケーションのアセンブルとコンフィグレーション*』を参照してください。

http://edocs.beasys.co.jp/e-docs/wls/docs70/webapp/index.html

管理サーバに関する注意

あるクラスタの管理サーバが使用できなくなると、デプロイ要求やアンデプロイ要求は中断されますが、管理対象サーバは要求の処理を続行します。管理対象サーバは、既存のコンフィグレーションを使用して起動および再起動することができます。ただし、管理サーバが復帰するまで、クラスタのコンフィグレーションを変更することはできません（たとえば、クラスタへの新しいノードの追加などは不可）。詳細については、4-19 ページの「Administration Server に対するバックアップとフェイルオーバ」を参照してください。

WebLogic Integration クラスタにおけるロード バランシング

WebLogic Integration アプリケーションをクラスタ化する目的の 1 つは、スケーアラビリティの実現です。クラスタをスケーラブルなものにするには、各サーバを十分に活用する必要があります。ロード バランシングにより、クラスタ内のすべてのサーバ間で負荷が均等に分散され、各サーバがそれぞれ最大能力で動作できるようになります。以下の節では、さまざまな機能分野についての、WebLogic Integration クラスタでのロード バランシングについて説明します。

- クラスタにおける WebLogic Server 機能のロード バランシング
- クラスタにおける BPM 機能のロード バランシング

- クラスタにおける Application Integration 機能のロード バランシング
- クラスタにおける B2B Integration 機能のロード バランシング

クラスタにおける WebLogic Server 機能のロード バランシング

WebLogic Server は、HTTP セッション ステートおよびクラスタ オブジェクトのロード バランシングをサポートしています。詳細は、次の URL にある『*WebLogic Server クラスタ ユーザーズ ガイド*』の「クラスタでの通信」を参照してください。

`http://edocs.beasys.co.jp/e-docs/wls/docs70/cluster/features.html`

クラスタにおける BPM 機能のロード バランシング

BPM ワークフローには、イベントベースのワークフローを処理するためのイベント キューが必要です。詳細については、1-10 ページの「Business Process Management リソース」を参照してください。

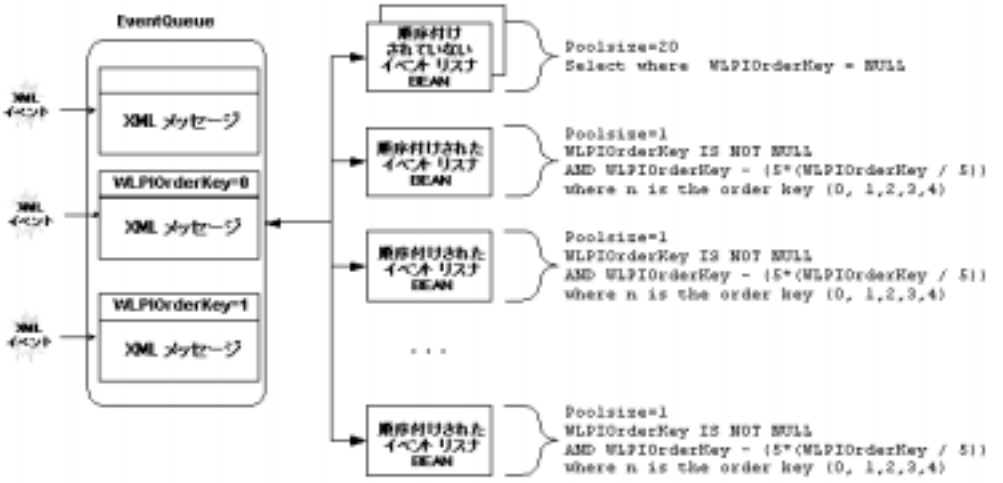
イベント キューと関連プール

各 BPM イベント キューには次の 2 種類のプールが関連付けられます。

- *順序付けされていないイベント リスナ メッセージ駆動型 Bean のプール*
- JMS キューから順序キーを選択する*順序付けされたイベント リスナ メッセージ駆動型 Bean のセット*

次の図は、イベント キューとその関連プールを示しています。

図 2-1 イベント キューと関連プール



順序付けされていないメッセージ駆動型 Bean は、非決定的な順序でメッセージを処理します。メッセージの読み出しは先入れ先出し（FIFO）方式ですが、読み出し後は、スレッドのスケジューリングおよび処理時点の負荷に応じて、任意の順序でメッセージを処理できます。

順序付けされたイベント リスナ メッセージ駆動型 Bean では、特定の順序キー（`WLPIOrderKey`）に従ってメッセージが処理されます。このために、クラスタ内で 1 つのイベント リスナ メッセージ駆動型 Bean を、`WLPIOrderKey` に関するメッセージ処理用にコンフィグレーションする必要があります。

順序キーは整数値でなければならず、その値は受け取った順序で処理させたいすべてのイベントで同じでなければなりません。また、順序指定メッセージは、同じ JMS キューに送信する必要があります。メッセージ プロデューサは、正しい順序でメッセージをキューに転送する必要があります。

`WLPIOrderKey` は、BPM で使用するカスタム JMS プロパティです。このプロパティは、WebLogic Integration Studio で設定するか、次のように、プログラム的に設定します。

- ワークフロー間でメッセージを送信するときに [XML イベントをポスト] ダイアログボックスで `WLPIOrderKey` を設定する。詳細は、『*WebLogic*

Integration Studio ユーザーズ ガイド』の「アクションの定義」にある「JMS トピックや JMS キューへの XML メッセージの通知」を参照してください。

- メッセージに対する [WLPIOrderKey JMS] ヘッダ フィールドをプログラム的に定義する。『*BPM クライアント アプリケーション プログラミング*』、「JMS 接続の確立」の「メッセージの順次処理の保証」を参照してください。

1 つの JAR ファイル（wlpi-mdb-ejb.jar）には、特定のキューに関して順序付けされたイベント リスナ メッセージ駆動型 Bean と順序付けされていない イベント リスナ メッセージ駆動型 Bean が格納されています。wlpi-mdb-ejb.jar ファイルで定義されている message-driven bean は、デフォルトの EventQueue からのメッセージを消費します。この JAR ファイルは、クラスタを対象とする必要があります。

BPM のロード バランシングは、wlpi-mdb-ejb.jar をクラスタにデプロイすることによって達成されます。この JAR ファイルには、5 つの順序付きイベント リスナ メッセージ駆動型 Bean と 5 つの順序なしイベント リスナ メッセージ起動型 Bean が格納されています。message-driven bean は、分散送り先からのメッセージを消費してイベントキューを検証するか、検証を拒否します。分散送り先には、JMS サーバごとに物理送り先が 1 つ、WebLogic Server のインスタンスごとに JMS サーバが 1 つ指定されています。分散キューの各メッセージ プロデューサは、それぞれ 1 つの物理送り先にバインドされています。メッセージ駆動型 Bean は、デプロイ先のサーバ内の物理送り先にバインドされています（サーバ親和性がある、という）。サーバ親和性の利用とは、メッセージが、その処理中に、同じ JVM および WebLogic Server インスタンス内に保持されることになります。したがって、あるプロデューサから分散送り先に送信された順序付きメッセージは、同じ順序付き message-driven bean によって消費されることが保証されます。このプロセスによって、メッセージの順序付き配信が保証されます。

新しいプールを作成する

1 つのサーバに十分な処理能力がある場合は、6-22 ページの「メッセージ駆動型 Bean 数の確認する」で説明するように、wli-mdb-ejb.jar ファイルのイベント リスナ メッセージ駆動型 Bean のサイズおよび範囲を大きくすることができます。

カスタムの JMS キューおよびイベント リスナの作成方法の詳細については、『*WebLogic Integration の起動、停止およびカスタマイズ*』、「WebLogic Integration のカスタマイズ」の「カスタム Java Message Service キューのコンフィグレーション」を参照してください。

BPM 機能のロード バランシングの要件

WebLogic Integration クラスタで BPM 機能をロード バランシングする場合、以下の要件を考慮してください。

- 特定の JMS キュー用の順序付きおよび順序なしのイベント リスナ メッセージ起動型 Bean を含む JAR ファイルは、クラスタを対象とする必要がある。言い換えれば、クラスタ全体を通じた均一なデプロイメントが必須です。WebLogic Integration には、デフォルトのイベント キュー（`com.bea.wli.bpm.EventQueue`）から作業を取り出す `wlpi-mdb-ejb.jar` ファイルが用意されています。

  注意： この記述は、XML 検証実行中の検証キューにも当てはまります、デフォルトの検証 イベント キューは `com.bea.wlpi.ValidatingEventQueue` です。

- MDB Generator ユーティリティを使用して、新しい JAR ファイルを作成できます。新しい JAR ファイルには、新しいユニークな JMS キューを関連付ける必要があります。
- アプリケーションでは、新しいイベント リスナ メッセージ駆動型 Bean の作業を呼び出すために、新しい JMS キューを認識している必要があります。

時限イベント

イベントおよび検証イベントキュー用の message-driven bean 同様、時限イベント リスナも、`wlpi-mdb-ejb.jar` ファイルで指定されたクラスタにデプロイされます。これらの message-driven bean は、`com.bea.wli.bpm.TimerQueue` から作業を取り出します。

時限イベントは、JMS 配信時刻を使用して実装されます。時限イベントは、次の 2 種類のプールによって実行されます。

- 順序付けされていない処理（時限ワークフロー開始）用のプールに類似した、時限イベント リスナ message-driven bean のプール

- ワークフロー インスタンス（タスクの締め切り、時限イベント アクション）が関連付けられている時限イベントを消費するための message-driven bean のセット（クラスタ内の 1 つの時限イベント リスナのメッセージ駆動型 Bean は、特定の順序キーに対してメッセージを処理するようにコンフィグレーションされている）。

クラスタにおける Application Integration 機能のロード バランシング

すべて同種のクラスタ、つまりすべてのリソースが同じ管理対象サーバを対象としているクラスタをコンフィグレーションすることができます。ただし、アダプタ自身の制約があります。

BPM 機能とは対照的に、JMS キューおよびサーバを使用して、クラスタ内で Application Integration の機能をロード バランシングできます。

クラスタ デプロイメントでは、1 つの EJB（`wlai-admin-ejb.jar`）を管理サーバのみにデプロイする必要があります。この EJB は `wlai-admin.ear` アーカイブファイルからデプロイされます（2-4 ページの「WebLogic Integration リソースのデプロイメント」の表 2-1 を参照）。

注意： `wlai-admin-ejb.jar` はクラスタ デプロイメントのみに必要です。WebLogic Integration を単独のサーバにデプロイする場合、`wlai-admin.ear` はデプロイしないでください。

以下のコードは、クラスタ ドメイン用の `config.xml` ファイルの抜粋です。クラスタにおける `wlai-admin.ear` ファイルのデプロイメント指定を示しています。

コード リスト 2-4 config.xml ファイルにおける wlai-admin EJB のデプロイメント

```
<Application Name="WLI-AI Admin Server Only"
Path="WLI_HOME/lib/wlai-admin.ear" TwoPhase="true">

<EJBComponent Name="WLI-AI RAR Upload" Targets="admin_server_name"
URI="wlai-admin-ejb.jar"/>

</Application>
```

クラスタにおける B2B Integration 機能のロード バランシング

B2B Integration 機能は、クラスタ内での作業のパーティショニングを要求しません。そのような機能をサポートするためには、完全に同質的なクラスタをコンフィグレーションする必要があります。言い換えれば、すべての B2B リソース（JMS のコンシューマ、送り先、およびプロデューサ）を、クラスタ内のすべてのノードで利用できます。

クラスタにおける B2B Integration 機能のロード バランシングを、デフォルトの JMS キューおよびサーバを使用して行うことができます。

B2B Integration リソースは、クラスタ内のすべてのノードに対して均一にデプロイされます。したがって、B2B エンジンをシャット ダウンする必要が生じた場合、クラスタ内のすべてのノードの B2B エンジンがシャット ダウンされます。クラスタ内の 1 つのノードの B2B エンジンのみをシャット ダウンすることはできません。その場合は、まず、そのノードをクラスタから削除する必要があります。

WebLogic JMS は、分散送り先を使用することによって、複数の物理送り先にわたるメッセージング負荷を均衡することができ、その結果、リソースがより効果的に活用され、応答時間が短縮されます。WebLogic JMS のロード バランシング アルゴリズムは、メッセージの送信先となる物理送り先（分散送り先のセット内の）およびコンシューマが割り当てられる物理送り先を決定します。メッセージ駆動型 Bean は、デプロイ先のサーバ内の物理送り先にバインドされています（サーバ親和性）。メッセージが、特定のサーバ上の特定の物理送り先（キュー）に送信されると、そのサーバがメッセージを処理します。

B2B Integration 機能は、クラスタ環境におけるサーバ親和性を備えたヒューリスティクなインメモリ キャッシュ機能を利用しています。B2B のメッセージ処理を実行する際は、B2B デコーダがその B2B メッセージのメッセージ エンベロープを、BPM JMS イベント キューに追加します。BPM message-driven bean は、そのメッセージをキューから取り出し、その後の処理を実行するために B2B 固有プラグインが呼び出されます。B2B 固有プラグインは、メッセージ ID、トレーディング パートナ、配信チャネル（URI）を使用して、MessageStore インメ

モリ キャッシュからメッセージ ペイロードを取り出します。このように、B2B Integration 機能はインメモリ キャッシュ機能を活用することができ、その結果、パフォーマンスが向上します。

WebLogic Integration クラスタの高可用性

メッセージ駆動型 Bean は、JMS の送り先からのメッセージを消費します。各 WebLogic Integration 送り先にはいくつかの message-driven bean が各送り先にデプロイされています。WebLogic Integration 送り先（JMS キューとトピック）の詳細なリストについては、2-27 ページの「JMS サーバと JMS 送り先」を参照してください。

高可用性を備えた JMS

WebLogic JMS の実装により、複数の物理送り先を単一の分散送り先のメンバーとしてコンフィグレーションできる能力を通じて、きわめて高い可用性がもたらされます。具体的には、管理者は、クラスタ内の各ノードに対して、各分散送り先に対する物理送り先を 1 つ、コンフィグレーションする必要があります。クラスタ内のあるノードに障害が発生して、そのノードの物理送り先が使用できなくなった場合、分散送り先のメンバーとしてコンフィグレーションされた他の物理送り先が、JMS のプロデューサとコンシューマにサービスを提供することができます。

JMS サーバおよびそのすべての送り先は、クラスタ内の別の WeLogic Server に移行できるため、クラスタ環境で単独の送り先としてデプロイされる必要のある送り先についても、高可用性を実現することができます。ただし、そのための移行は手動で行われるので、送り先の単独デプロイはお勧めできません。

メッセージ駆動型 Bean は、分散送り先からのメッセージを消費します。分散送り先には、WebLogic Server のインスタンスごとに、物理送り先が 1 つ格納されています。分散キューの各メッセージ プロデューサは、それぞれ 1 つの物理送り先にバインドされています。メッセージ駆動型 Bean は、デプロイ先のサーバ内の物理送り先にバインドされています（サーバ親和性）。したがって、あるプロデューサから分散送り先に送信された順序付きメッセージは、同じ順序付き message-driven bean によって消費されることが保証されます。このプロセスによ

り、メッセージの順序付けられた配信、および 2-22 ページの「クラスタにおける B2B Integration 機能のロード バランシング」で説明した B2B キャッシュ機能が保証されます。

クラスタ内の管理対象サーバに障害が発生した場合、そのサーバの message-driven bean はアトミックに移行されますが、複数のメッセージ処理を防止するため、自動的移行は行われません。

次の節では、WebLogic Integration がクラスタ デプロイメントにおいて、分散送り先およびサーバ親和性を利用して高可用性を実現する例を説明します。

- 非同期サービス要求に対する高可用性
- イベント転送に対する高可用性

非同期サービス要求に対する高可用性

WebLogic Integration クラスタでは、`WLAI_ASYNC_REQUEST_QUEUE` キューおよび `WLAI_ASYNC_RESPONSE_QUEUE` キューが、分散送り先としてデプロイされます。非同期サービス要求プロセッサは、関連付けられたメッセージ駆動型 EJB で、クラスタ内のすべてのサーバにデプロイされます。非同期の要求および応答は、それらを受け入れた JMS サーバがクラッシュした後でも、処理されます。

message-driven bean が同期サービス要求を受信する前に物理キューに障害が発生すると、その要求は、この物理キューが再びオンラインに復帰するまで使用できなくなります。非同期サービス応答についても同様です。

同期呼び出しおよび非同期呼び出しの処理に関する詳細については、1-16 ページの「Application Integration リソース」を参照してください。

イベント転送に対する高可用性

Application Integration アダプタは、BPM 機能または WebLogic Workshop によって消費されるイベントを生成します。イベントは、アプリケーション ビューから JMS キュー（`WLAI_EVENT_QUEUE`）へ転送されます。

イベントについてのメタデータを取得するには、イベント ルータが HTTP を使用して WebLogic Integration インスタンスと会話する必要があります。イベント ルータのコールバック交信に対してロード バランシングと高可用性を実現したいが、クラスタ アドレスに DNS 名は使用していないという場合は、`wlai.clusterFrontEndHostAndPort` プロパティを設定する必要があります。

このプロパティの詳細については、3-5 ページの「wlai.clusterFrontEndHostAndPort プロパティの設定（オプション）」を参照してください。

WLAI_EVENT_QUEUE は、複数の物理送り先が格納された分散送り先です。message-driven bean (AI Event プロセッサ) は WLAI_EVENT_QUEUE 分散送り先でリスンします。このキューのメッセージ処理には複数のサーバが関与するので、1 つのサーバに障害が発生しても対応できます。アダプタ イベントの WebLogic Integration による処理については、1-21 ページの「イベント」を参照してください。

JMS リソース

この節では、クラスタ環境で WebLogic Integration アプリケーション用の JMS リソースをコンフィグレーションする方法について説明します。具体的には、以下のリソースのコンフィグレーション方法について説明します。

- JMS 接続ファクトリ
- JMS JDBC ストア
- JMS サーバと JMS 送り先
- ストアを作成して接続プールと関連付ける
- JMS サーバを作成しストアを関連付ける

JMS リソースは、WebLogic Server Administration Console でコンフィグレーションされます。コンソールを起動するには、『*WebLogic Integration の起動、停止およびカスタマイズ*』、「WebLogic Integration 管理ツールと設計ツール」の「WebLogic Server Administration Console の起動」を参照してください。

JMS 接続ファクトリ

以下の JMS 接続ファクトリは、管理サーバおよびクラスタ化された管理対象サーバのある WebLogic Integration ドメインでコンフィグレーションされます。

- BPM トピック用ファクトリ（クラスタにデプロイ）

- BPM キュー用ファクトリ（クラスタにデプロイ）
- Application Integration 用ファクトリ（クラスタにデプロイ）
- B2B Integration 用ファクトリ（管理サーバにのみデプロイ）
- B2B RosettaNet キュー用ファクトリ（クラスタにデプロイ）

次のリストは、WebLogic Integration クラスタにおける JMA 接続ファクトリのサンプル仕様を示しています。接続ファクトリの Target と JNDI name は太字で表記されています。

コード リスト 2-5 config.xml ファイルの JMSConnectionFactory 要素

```
...
<!--Application Integration Connection Factories>

<JMSConnectionFactory AllowCloseInOnMessage="false"
    DefaultDeliveryMode="Persistent" DefaultPriority="4"
    DefaultTimeToLive="0"
    JNDIName="com.bea.wlai.JMSConnectionFactory"
    MessagesMaximum="10" Name="WLIJMSConnectionFactory"
    OverrunPolicy="KeepOld" Targets="MyCluster"
    UserTransactionsEnabled="true"/>

<!--B2B Integration Connection Factories>

<JMSConnectionFactory AllowCloseInOnMessage="true"
    JNDIName="com.bea.wli.b2b.server.TopicConnectionFactory"
    Name="B2BTopicFactory" Targets="MyServer-1"
    UserTransactionsEnabled="true"/>

<JMSConnectionFactory AllowCloseInOnMessage="true"
    JNDIName="com.bea.wli.b2b.rosettanet.QueueConnectionFactory"
    Name="RNQueueFactory" Targets="MyCluster"
    UserTransactionsEnabled="true"/>

<!--BPM Connection Factories>

<JMSConnectionFactory AllowCloseInOnMessage="true"
    JNDIName="com.bea.wlpi.TopicConnectionFactory"
    Name="wlpiFactory" Targets="MyCluster"
    UserTransactionsEnabled="true"/>

<JMSConnectionFactory AllowCloseInOnMessage="true"
    JNDIName="com.bea.wlpi.QueueConnectionFactory"
    Name="wlpiQueueFactory" Targets="MyCluster"
    UserTransactionsEnabled="true"/>

...
```

JMS JDBC ストア

JMS JDBC ストアは、デプロイメント内の JMS サーバごとに定義する必要があります。

次のリストは、config.xml ファイルの抜粋で、myserver 管理サーバの管理下にある 2 つの管理対象サーバ（MyServer-1 および MyServer-2）を含むクラスタ（MyCluster）の JMS JDBC ストアを定義しています。接続プールの対象（Target）として、クラスタと管理サーバの両方がリストされている点に注意してください。

コード リスト 2-6 config.xml ファイルの JMSJDBCStore 要素

```
<JMSJDBCStore ConnectionPool="wliPool"
    Name="JMSWLCStore-MyServer-1" PrefixName="MyServer_1"/>

<JMSJDBCStore ConnectionPool="wliPool"
    Name="JMSWLCStore-MyServer-2" PrefixName="MyServer_2"/>

<JMSJDBCStore ConnectionPool="wliPool" Name="JMSWLCStore-myserver"
    PrefixName="myserver"/>

...

<JDBCConnectionPool CapacityIncrement="1"
DriverName="oracle.jdbc.driver.OracleDriver" InitialCapacity="1"
LoginDelaySeconds="1" MaxCapacity="15" Name="wliPool"
Properties="user=scott;password=tiger;dll=ocijdbc8;protocol=thin
RefreshMinutes="0" ShrinkPeriodMinutes="15"
ShrinkingEnabled="true" Targets="myserver,MyCluster"
URL="jdbc:oracle:thin:@machine:port:name"/>
```

JMS サーバと JMS 送り先

JMS サーバは、クラスタ内の各管理対象サーバに対して 1 つずつ、管理サーバに対して 1 つコンフィグレーションする必要があります（管理サーバとしてコンフィグレーションされた JMS サーバには、表 2-2 で示されているように、送り先（B2B トピック）は 1 つのみデプロイされる）。JMS サーバについては、以下の命名規約の使用をお勧めします。WLI_JMSServer_*node*。ここで、*node* は JMS サーバがデプロイされているサーバの名前を表します。

次の表では、WebLogic Integration が使用する送り先（JMS キューとトピック）について説明し、それらが単独の送り先として指定されているのか、分散送り先として指定されているのか示します。

表 2-2 JMS 送り先

送り先	分散 / 単独
com.bea.wli.bpm.TimerQueue	分散
com.bea.wli.bpm.EventQueue	分散
com.bea.wli.bpm.ValidatingEventQueue	分散
com.bea.wli.bpm.ErrorTopic	分散
om.bea.wli.bpm.AuditTopic	分散
com.bea.wli.bpm.NotifyTopic	分散
com.bea.wlpi.EventTopic	単独の管理対象サーバ
com.bea.wli.b2b.server.B2BTopic	管理サーバのみ
com.bea.b2b.OutboundQueue	分散
com.bea.b2b.rosettanet.EncoderQueue	分散
com.bea.wlai.ASYNC_REQUEST_QUEUE	分散
com.bea.wlai.ASYNC_RESPONSE_QUEUE	分散
com.bea.wlai.EVENT_QUEUE	分散
com.bea.wlai.EVENT_TOPIC	分散
com.bea.wli.FailedEventQueue[1]	分散

1. com.bea.wli.FailedEventQueue 送り先は、WebLogic Integration のすべてのコンポーネントによって使用されます。この送り先は、JTA UserTransaction のメッセージを消費する JMS 送り先のエラー送り先として使用します。エラー キューに関する詳細については、2-30 ページの 「エラー送り先」 を参照してください。

次のリストは、config.xml ファイルの抜粋です。管理サーバ（myserver）の管理下にある 2 つの管理対象サーバ（MyServer-1 および MyServer-2）を含むクラスタ コンフィグレーションとして選択された要素を示しています。

コード リスト 2-7 config.xml ファイルの JMSServer 要素

```
<!--Distributed Destinations-->

<JMSDistributedQueue JNDIName="com.bea.wli.bpm.EventQueue"
    Name="WLI_BPM_Event" Targets="MyCluster">

    <JMSDistributedQueueMember JMSQueue="WLI_BPM_Event_MyServer-1"
    Name="WLI_BPM_Event_MyServer-1"/>
    <JMSDistributedQueueMember JMSQueue="WLI_BPM_Event_MyServer-2"
    Name="WLI_BPM_Event_MyServer-2"/>

</JMSDistributedQueue>

<!--Administration Server-->

<JMSServer Name="WLI_JMSServer_myserver"
    Store="JMSWLCStore-myserver" Targets="myserver"
    TemporaryTemplate="TemporaryTemplate">
    <JMSTemplate Name="TemporaryTemplate"/>
   <JMSTopic JNDIName="com.bea.wli.b2b.server.B2BTopic"
    Name="B2BTopic"/>
</JMSServer>

<!--Managed Server-->

<JMSServer Name="WLI_JMSServer_MyServer-1"
Store="WLI_JMSJDBCStore_MyServer-1" Targets="MyServer-1
(migratable)"

    <JMSQueue JNDIName="com.bea.wli.bpm.Event.MyServer-1"
    Name="WLI_BPM_Event_MyServer-1" StoreEnabled="true"
    Template="WLI_JMSTemplate-1"/>

...
    <JMSTopic JNDIName="com.bea.wli.bpm.EventTopic"
    Name="wlpiEvent" StoreEnabled="false"/>

...
</JMSServer>
```

このリストの以下の点に注意してください。

- JMS 分散送り先が 1 つ（JMSDistributedQueue）示しされている。これは、WLI_BPM_Event キューを次のように指定します。

- クラスタ内のすべてのサーバにデプロイされる（Targets="MyCluster" で指定）。
- JMSDistributedQueue 要素には、各物理送り先に対して 1 つずつ、すなわち、2 つの JMSDistributedQueueMember 要素が含まれる。つまり、各管理対象サーバには送り先が 1 つずつ関連付けられています。

■ 2 つの JMSServer 要素が示されている（管理サーバと管理対象サーバに対して 1 つずつ）。

■ 管理サーバに対する JMSServer 要素には、JMSTopic 要素が 1 つ含まれていて、管理サーバには送り先が 1 つのみ（B2B トピック送り先）デプロイされることを指定している。

■ 各管理対象サーバに対する JMSServer には、表 2-2 で説明されているすべての JMS 分散送り先に対する JMSTopic 要素と JMSQueue 要素を格納する必要がある。このリストの config.xml ファイルの抜粋には、次の例が示されています。

- WLI_BPM_Event queue for MyServer-1 (WLI_BPM_Event_MyServer-1) に対して WLI_BPM_Event queue for MyServer-1 を指定する JMSQueue 要素
- com.bea.wli.bpm.EventTopic を指定する JMSTopic 要素。これは、クラスタ内の 1 つのサーバ（この場合は MyServer-1）にのみデプロイする必要があります。

エラー送り先

com.bea.wli.FailedEventQueue は、JTA User Transaction に格納されているメッセージを消費する JMS 送り先に対するエラー送り先です。たとえば、EventQueue、ValidatingEventQueue、TimerQueue などのエラー送り先となります。対象とする BPM ワークフロー インスタンスが見つからないメッセージや、1 分間に何回も再試行に失敗するメッセージは、FailedEventQueue に送信されます（デフォルトの再試行回数は 10 回ですが、別の回数に設定することもできる）。JMS メッセージが FailedEventQueue に到着すると、そのキューでリスンする message-driven bean (com.bea.wli.common.errorlistener.ErrorListenerBean) が、WebLogic Server ログにログ エントリを書き込みます。

エラー キューが使用する JMS テンプレートである WLI_JMSTemplate-*node* で再配信のための属性をコンフィグレーションして、再試行の回数を指定することができます。エラー キューは、分散送り先で、再配信属性はノード固有の物理送り先に対してコンフィグレーションされ、WLI-FailedEvent-*node* と名付けられます（この名前については、*node* は名前このクラスタ内の WebLogic Server インスタンスを表している）。

1. WebLogic Server Administration Console のナビゲーション ツリーで、[サービス | JMS | テンプレート | WLI_JMSTemplate-*node*] を選択します。
2. [コンフィグレーション] タブ、[再配信] タブの順に選択します。
3. 再配信属性と、それらの属性をコンフィグレーションする対象となる適切な WLI-FailedEvent-*node* を指定します。
4. [適用] をクリックします。

JMS テンプレートの再配信属性をコンフィグレーションする方法の詳細については、次の URL にある『*Adminstration Console オンライン ヘルプ*』の「JMS」で、「[JMS テンプレート] → [コンフィグレーション] → [再配信]」を参照してください。

http://edocs.beasys.co.jp/e-docs/wls/docs70/ConsoleHelp/domain_jmstemplate_config_redelivery.html

さらに、カスタム メッセージ リスナを作成するオプションもあり、それをクラスパスに追加して、FailedEventQueue message-driven bean のデプロイメント記述子でリフレッシュできます。そうすることで、エラー メッセージを永続化するようにシステムをコンフィグレーションできます。

ストアを作成して接続プールと関連付ける

ストアを作成し、接続プールに関連付けるには、次の手順を実行します。

1. Administration Console のナビゲーション ツリーで [サービス | JMS | ストア] を選択し、[新しい JMSJDBCStore のコンフィグレーション] を選択します。[コンフィグレーション] タブがデフォルトで表示されます。
2. [名前] フィールドに、このストアを識別するための名前を入力します。

   各 JMS サーバには、独自の JMSJDBCStore があります。各管理対象サーバには、独自の JMS サーバがあります。サーバの作成手順については、2-32 ページの「JMS サーバを作成しストアを関連付ける」を参照してください。

3. [接続プール] フィールドで、使用する接続プールを選択します。
4. [プレフィックス名] フィールドに、名前の先頭に追加する文字（WL-AI など）を入力します。
5. [作成] をクリックします。

JMS サーバを作成しストアを関連付ける

JMS サーバを作成し、JMSJDBCStore に関連付けるには、次の手順を実行します。

1. Administration Console のナビゲーション ツリーで [サービス | JMS | ストア] を選択し、[新しい JMSServer の作成] を選択します。
2. [名前] フィールドに、この JMS サーバを識別するための名前を入力します。
3. [ストア] フィールドで、JMS サーバの関連先の JMSJDBCStore を選択します。
4. [Temporary Template] フィールドで、次の使用可能なテンプレートのうち、いずれかを選択します。
   - [Temporary Template]
   - [WLIJMSTemplate]

   注意 : これらの JMS テンプレートのプロパティは、Administration Console の [サービス | JMS | テンプレート] からアクセスすることができます。
5. [作成] をクリックします。

アダプタのデプロイ

1-16 ページの「Application Integration リソース」で説明した実行時 Application Integration 機能（同期サービス呼び出し、非同期サービス呼び出し、およびイベント）をクラスタ化することにより、スケーラビリティと可用性を高めることができます。設計時に利用できる Application Integration 機能（アプリケーション ビューおよび接続ファクトリ）をクラスタ化することにより、スケーラビリティを高めることができますが、可用性を高めることはできません。つまり、クラス

タ内に実行されていないサーバがあれば、アプリケーション ビューをデプロイまたはアンデプロイ（編集）することができません。言い換えれば、デプロイおよびアンデプロイ（編集）は、健全なクラスタにおいてのみ可能です。

Application Integration アダプタは通常、次の 3 つのコンポーネントで構成されています。

- RAR ファイルからデプロイされたリソース アダプタ
- WAR ファイルからデプロイされた設計時 Web アプリケーション
- WAR ファイルからデプロイされたイベント ジェネレータ Web アプリケーション

リソース アダプタ ファイル（RAR）と設計時 Web アプリケーション（WAR）ファイルは、クラスタにデプロイする必要があります。イベント ジェネレータ Web アプリケーション（WAR）ファイルは、ほとんどの場合、クラスタ内の 1 つのサーバにデプロイします（具体的な説明は、アダプタのマニュアルを参照）。

たとえば、DBMS アダプタが使用されている場合、DbmsEventRouter Web アプリケーションは、クラスタ内の 1 つのサーバを対象としている必要があります。次のリストは、config.xml ファイルの抜粋です。ここでは、DBMS アダプタをデプロイするためのコンフィグレーションを指定する application 要素が示されています。

コード リスト 2-8 Application Integration アダプタをデプロイするためのコンフィグレーション

```
<Application Deployed="true" Name="BEA_WLS_DBMS_ADK"
Path="/bea/weblogic700/integration/adapters/dbms/lib/
BEA_WLS_DBMS_ADK.ear" StagingMode="stage" TwoPhase="true">

    <ConnectorComponent Name="BEA_WLS_DBMS_ADK"
Targets="MyCluster" URI="BEA_WLS_DBMS_ADK.rar"/>

    <WebAppComponent Name="BEA_WLS_DBMS_ADK_Web"
Targets="MyCluster" URI="BEA_WLS_DBMS_ADK_Web.war"/>

    <WebAppComponent Name="DbmsEventRouter" Targets="MyServer-1"
URI="BEA_WLS_DBMS_ADK_EventRouter.war"/>
</Application>
```

このリストの以下の点に注意してください。関係のある情報は強調するため、太字で表記されています。

- 管理サーバが各サーバの起動に先立って、アダプタの EAR ファイルを StagedTargets リストにあるすべての管理対象サーバにコピーすることを保証するため、StagingMode の値は `stage` に設定する必要がある。
- アダプタは、TwoPhase="true" 属性を使用してデプロイする必要がある。この設定により、メッセージ処理が開始される前にすべてのアダプタ コンポーネントがデプロイされることが保証されます。
- リソース アダプタの設計時 Web アプリケーションはクラスタにデプロイされるが、イベント ジェネレータ Web アプリケーションは、WAR ファイルからデプロイされ、ただ 1 つの管理対象サーバ（`MyServer-1`）にのみデプロイされる。

デプロイメントに備えたアダプタのコンフィグレーション

リソース アダプタのコンフィグレーションは、Configuration Wizard によって作成された WebLogic Integration ドメインで定義されます。Configuration Wizard によって作成されたコンフィグレーションでは、アダプタの 3 つのコンポーネントは、クラスタへのデプロイメントを目標としています。前の節（特に コード リスト 2-8）で説明したとおり、イベント ジェネレータ Web アプリケーション（WAR）ファイルは、ほとんどの場合、クラスタ内の 1 つのサーバにデプロイします。この要件を満足するため、ドメインのコンフィグレーションを変更する必要があります。ドメイン コンフィグレーションの変更方法の詳細については、3-16 ページの「手順 5. アダプタ用イベント ルータ WAR ファイルのコンフィグレーション」 を参照してください。

また、リソース アダプタをクラスタ内のサーバを起動した後でデプロイすることもできます。クラスタ デプロイメントの設定および起動の詳細については、第 3 章「クラスタ デプロイメントのコンフィグレーション」 を参照してください。`weblogic.Deployer` コマンドライン ユーティリティまたは WebLogic Server Administration Console を使用して稼働中のクラスタにアダプタをデプロイする方法については、付録 B「リソース アダプタのデプロイ」 を参照してください。

WebLogic Integration 環境でのアダプタのデプロイに関する詳細は、『*アダプタの開発*』の「アダプタのデプロイ」を参照してください。

3 クラスタ デプロイメントのコンフィグレーション

この章では、クラスタ環境で WebLogic Integration をデプロイするためのコンフィグレーションに必要なタスクについて説明します。

単独のサーバへの WebLogic Integration のデプロイメントの詳細は、『*WebLogic Integration の起動、停止およびカスタマイズ*』、「WebLogic Integration のカスタマイズ」の「新しいドメインの作成とカスタマイズ」を参照してください。

クラスタ ドメインのアーキテクチャは、2-3 ページの「クラスタ デプロイメントの設計」の説明に従ってプランニングし、WebLogic Integration をクラスタ環境に設定します。そのためには、ルータ（ハードウェアおよびソフトウェア）、管理サーバ、および管理対象サーバをコンフィグレーションして、WebLogic Integration リソースをそれらのサーバにデプロイする必要があります。WebLogic Server インスタンスとクラスタで構成されるドメインに対する永続的コンフィグレーションは、管理サーバの XML コンフィグレーション ファイル（`config.xml`）にあります。

クラスタ ドメインで WebLogic Integration を設定およびデプロイするには、次の手順を実行します。

- 手順 1. コンフィグレーションの前提条件への準拠
- 手順 2. WebLogic Integration ドメインの作成
- 手順 3. ドメイン用データベースのコンフィグレーション
- 手順 4. 1 つの管理対象サーバ用 BPM リソースのコンフィグレーション
- 手順 5. アダプタ用イベント ルータ WAR ファイルのコンフィグレーション
- 手順 6. RDBMS レルムのコンフィグレーション
- 手順 7. ルータのコンフィグレーション
- 手順 8. startWeblogic コマンド ファイルの編集
- 手順 9. ドメインの管理対象サーバの設定

手順 1. コンフィグレーションの前提条件への準拠

この節では、クラスタ環境で実行される WebLogic Integration をコンフィグレーションするための以下の前提条件について説明します。

- 各インストールの WebLogic Server クラスタ ライセンスの取得。

  WebLogic Server をクラスタ コンフィグレーションで使用するには、専用のクラスタ ライセンスが必要です。クラスタ ライセンスの取得については、BEA 販売代理店にお問い合わせください。

- そのクラスタで使用する管理サーバの IP アドレスの取得。

  1 つのクラスタ内のすべての WebLogic Server インスタンスは、コンフィグレーションおよびモニタに同じ管理サーバを使用します。サーバをクラスタに追加する場合、各サーバが使用する管理サーバを指定する必要があります。

- 各クラスタへのマルチキャスト アドレスの割り当て。

  注意： Configuration Wizard による WebLogic Integration ドメインの作成時に、マルチキャスト アドレスの割り当てが要求されます（3-7 ページの「手順 2. WebLogic Integration ドメインの作成」を参照）。

  マルチキャスト アドレスは、クラスタ メンバー間の通信に使用されます。クラスタ化されたサーバは、1 つの専用マルチキャスト アドレスを共有する必要があります。ネットワーク上の各クラスタに対して、一意のマルチキャスト アドレスとポート番号の組み合わせを割り当てる必要があります。ネットワーク上の 2 つのクラスタが同じマルチキャスト アドレスを使用する場合、異なるポートを使用する必要があります。クラスタのマルチキャスト ア

ドレスが異なる場合は、同じポートを使用するか、またはデフォルトのポート（7001）をそのまま使用することもできます。マルチキャスト メッセージをサポートするため、1 つのクラスタ内の管理サーバおよび管理対象サーバを同じサブネット上に配置する必要があります。

- クラスタ内サーバに対する IP アドレスの定義。アドレス定義には、以下のように、いくつかの方法があります。

  注意： Configuration Wizard による WebLogic Integration ドメインの作成時に、サーバに対するリスン アドレスの割り当てが要求されます（3-7 ページの「手順 2. WebLogic Integration ドメインの作成」を参照）。

  - クラスタ内のサーバに対して 1 つの IP アドレスと異なるリスン ポート番号を割り当てる方法。

    1 つの IP アドレスとサーバごとに異なるポート番号をクラスタ サーバに割り当てることにより、1 つのマシンに、そのマシンをマルチホーム サーバ化することなく、クラスタ環境を設定することができます。

    クライアントからこのような IP アドレスにアクセスできるようにするには、以下のいずれかの方法で、IP アドレスとポート番号で URL を構成します。

ipAddress:portNumber-portNumber	ポート番号が連番になっている場合。 例： `127.0.0.1:7003-7005`
ipAddress:portNumber+...+portNumber	ポート番号が連番ではない場合。 例： `127.0.0.1:7003+7006+7008`
ipAddress:portNumber,ipAddress:portNumber, ...	冗長で明示的な指定。 例： `127.0.0.1:7003,127.0.0.1:7004,127.0.0.1:7005`

  - クラスタ内の各マシン上で起動する WebLogic Server インスタンスごとに静的 IP アドレスを割り当てる方法。

    この方法では、複数のサーバが 1 つのマシン上で実行されている場合、そのマシンはマルチホーム サーバとしてコンフィグレーションする必要があります。つまり、複数の IP アドレスが 1 つのコンピュータに割り当てられます。

この場合は、クラスタ アドレスは、カンマ区切りの IP アドレスのリストの形にします。たとえば、次のリストは、`config.xml` ファイルで指定されているクラスタ アドレスの例です。`MyCluster` という名前のクラスタ内の、4 つのサーバのそれぞれに対して、静的 IP アドレスが指定されています。

```
<Cluster
ClusterAddress="127.0.0.1:7001,127.0.0.2:7001,127.0.0.3,127.
0.0.4:7001" Name="MyCluster"/>
```

注意： 開発およびテストのためには、カンマ区切りリストを使用することができます。プロダクション環境用には、クラスタ アドレスは DNS 名または 1 つの IP アドレスとしてのみ指定することをお勧めします。ClusterAddress に DNS 名を使用しない場合は、ロード バランシングおよびイベント ルータのコールバック通信（Application Integration 機能）に対する高可用性をサポートするため、`wlai.clusterFrontEndHostAndPort` プロパティを設定してください。このプロパティの設定方法の詳細については、3-5 ページの「wlai.clusterFrontEndHostAndPort プロパティの設定（オプション）」を参照してください。

- クラスタ ドメイン用の Oracle、Microsoft SQL、Sybase、または DB2 のデータベースのコンフィグレーション。
- 共用ファイル システムの組み込み。B2B Integration 機能が使用されるあらゆるクラスタに対して、共用ファイル システムをお勧めします。共用ファイル システムは、高可用性が求められるあらゆるクラスタに不可欠です。Storage Area Network (SAN) またはマルチポート型のディスク システムをお勧めします。
- システム用のルータ（ハードウェアまたはソフトウェア）のコンフィグレーション。サーブレットと JSP のロード バランシングは、組み込みロード バランシング機能か WebLogic プロキシ プラグインのいずれか、または別個のロード バランシング用ハードウェアを使用して達成されます。

  WebLogic Integration ドメイン用のソフトウェア ルータのコンフィグレーションの詳細については、3-19 ページの「手順 7. ルータのコンフィグレーション」を参照してください。

  ハードウェア ルータおよびソフトウェア ルータに関する詳細は、次の URL にある『*WebLogic Server クラスタ ユーザーズ ガイド*』を参照してください。

```
http://edocs.beasys.co.jp/e-docs/wls/docs70/cluster/index.html
```

クラスタ化された WebLogic Server インスタンスの設定に関する詳細については、次の URL にある、『*WebLogic Server クラスタ ユーザーズ ガイド*』の「WebLogic クラスタのセットアップ」を参照してください。

```
http://edocs.beasys.co.jp/e-docs/wls/docs70/cluster/setup.html
```

注意： 1 つまたは複数のファイアウォールを含めるようにドメインを設計する場合は、この他にも要件があります。詳細は、次の URL にある『*WebLogic Server クラスタ ユーザーズ ガイド*』の「クラスタでの通信」を参照してください。

```
http://edocs.beasys.co.jp/e-docs/wls/docs70/cluster/features.html
```

wlai.clusterFrontEndHostAndPort プロパティの設定（オプション）

Application Integration アダプタは、BPM エンジンによって消費されるイベントを生成します。WebLogic Integration のイベントおよびイベント処理の詳細については、1-21 ページの「イベント」を参照してください。

クラスタ アドレスに DNS 名を使用しない場合は、ロード バランシングおよびイベント ルータのコールバック通信に対する高可用性を達成するため、`wlai.clusterFrontEndHostAndPort` プロパティを設定してください。

wlai.clusterFrontEndHostAndPort プロパティを設定する理由

次の表では、次のアドレスをクラスタ アドレスとするサンプル クラスタのクラスタ コンフィグレーションについて説明します。

```
<Cluster ClusterAddress="127.0.0.1:7001,127.0.0.1:7002"
Name="MyCluster"/>
```

サーバ名	サーバの種類	リスン アドレス : ポート
MyAdmin	管理サーバ	127.0.0.5:7005
MyServer1	管理対象サーバ	127.0.0.1:7001
MyServer2	管理対象サーバ	127.0.0.1:7002
MyRouter	ルータ	127.0.0.1:7003

イベント ルータは、HTTP を介して WebLogic Integration のインスタンスと通信を行うことによって、イベントに関するメタデータを取得します。このような通信の実際の方法は、`wlai.clusterFrontEndHostAndPort` プロパティが設定されているかどうかで決まります。

- `wlai.clusterFrontEndHostAndPort` プロパティが設定されていない場合

  イベント ルータとの通信の確立時に、WebLogic Integration によって、`ClusterAddress` に最初にリストされているアドレスがコールバック アドレスとして渡されます。この例では、コールバック アドレスは `127.0.0.1:7001` です。このシナリオでは、MyServer-1 に障害が発生すると、MyServer-2 は実行されているにもかかわらず、イベント ルータは WebLogic Integration アプリケーションにアクセスできません。

- `wlai.clusterFrontEndHostAndPort` プロパティが設定されている場合

  `wlai.clusterFrontEndHostAndPort` プロパティは、クラスタのフロントエンド（この例では MyRouter サーバ）のアドレスに設定され、そこに HttpClusterServlet がホストされています。

  その結果、イベント ルータとの通信の確立時に、WebLogic Integration はイベント ルータに、`127.0.0.1:7003` というアドレスを渡します。このシナリオでは、クラスタ内の管理対象サーバに障害が発生した場合でも、イベント ルータは WebLogic Integration アプリケーションにアクセスできます。

wlai.clusterFrontEndHostAndPort プロパティの設定方法

各管理対象サーバの WLAIStartup EJB 環境プロパティ内に `wlai.clusterFrontEndHostAndPort` プロパティを作成する必要があります。たとえば、`wlai.clusterFrontEndHostAndPort=127.0.0.1:7003` と設定するには、次の手順を実行します。

1. Administration Console のナビゲーション ツリーで、[*Domain_Name* | デプロイメント | EJB | WLI-AI Server] を選択します。
2. [Edit EJB Descriptor] をクリックして、EJB 記述子の編集ができる新しいウィンドウを表示します。
3. 新しいウィンドウの左のナビゲーションペインで、[EJB Jar | Enterprise Beans | セッション | WLAIStartup | Env Entries] を選択して、コンフィグレーションウィンドウを開きます。
4. [新しい Environment Entry のコンフィグレーション] をクリックします。
5. 次の情報を入力します。
   - [Env Entry Name] : `wlai.clusterFrontEndHostAndPort`
   - [Env Entry Value] : *`127.0.0.1:7003`*

     `127.0.0.1:7003` は、HttpClusterServlet をホストするクラスタのフロントエンドのリスン アドレスとポートを表しています。

手順 2. WebLogic Integration ドメインの作成

この手順を実行するには、各管理対象サーバの定義をドメイン コンフィグレーション ファイル（`config.xml`）に追加し、すべての管理対象サーバをクラスタに割り当て、WebLogic Integration コンポーネントをドメイン上のサーバに指定する必要があります。

クラスタ化された WebLogic Integration デプロイメントの定義は、BEA Configuration Wizard によるドメインの作成から始まります。

注意： この節で説明するドメイン設定手順は、Configuration Wizard が Windows の [スタート] メニューから GUI モードで実行されていることを前提としています。

さまざまなモードで Configuration Wizard を使用する方法の詳細については、次の URL にある『*Configuration Wizard の使い方*』を参照してください。

```
http://www.beasys.co.jp/e-docs/platform/docs70/confgwiz/index.html
```

Configuration Wizard を使用して WebLogic Integration ドメインを作成するには、以下の手順を完了してください。

1. [スタート] から、[プログラム | BEA WebLogic Platform 7.0 | Domain Configuration Wizard] を選択します。

   Configuration Wizard が起動します。ドメインのコンフィグレーションに使用するデータの入力が要求されます。

2. Configuration Wizard のプロンプトに従って、次の表に示す情報を入力します。

ウィンドウ	アクション
[ドメインのタイプと名前を選択]	作成するドメインのベースとなるテンプレートを選択し、ドメインに名前を付ける。 作成するドメインの要件に合わせて、次のテンプレートの中から 1 つ選択する。 ■ [WLI Domain] － BPM、B2B Integration、Application Integration、Data Integration など、すべての WebLogic Integration 機能をサポートするドメインを作成する場合に使用する。 ■ [WLI BPM] － BPM および Data Integration 機能をサポートするドメインを作成する場合に使用する。 ■ [WLI EAI] － Application Integration、BPM および Data Integration 機能をサポートするドメインを作成する場合に使用する。
[サーバ タイプを選択]	サーバの種類の入力を要求されたら次のオプションを選択する。 `Admin Server with Clustered Managed Server(s)`

[ドメインの場所を選択]	作成したドメインをインストールするディレクトリを指定する。 デフォルト ディレクトリをそのまま使用するか、別のディレクトリを選択する。マシン上の有効なディレクトリはどれでもドメイン ディレクトリとして使用できる。
[クラスタ化サーバのコンフィグレーション]	クラスタ内の各管理対象サーバのサーバ名、リスン アドレス、リスン ポートを指定する。[1]
[クラスタのコンフィグレーション]	クラスタ名、クラスタのマルチキャスト アドレスとマルチキャスト ポート、およびクラスタ アドレスを指定する。[1]
[スタンドアロン / 管理サーバのコンフィグレーション]	サーバ名、サーバのリスン アドレスとリスン ポート、管理サーバに対するサーバの SSL リスン ポート（このポートからクラスタ ドメインに対するすべての管理機能が実行される）を指定する。[1] **注意：** 管理サーバのコンフィグレーションでは、Configuration Wizard のプロンプトに従ってデフォルトのサーバ名（`myserver`）をそのまま使用することをお勧めします。デフォルト以外のサーバ名を指定する場合は、作成したドメインの次のディレクトリ名を変更して、`myserver` を新たに指定する名前と交換する必要があります。 *`DOMAIN_HOME`*`/applications/DefaultWebApp_`*`myserver`* このパスで、*`DOMAIN_HOME`* は、Configuration Wizard を使用して作成したカスタム ドメインのルート ディレクトリを表しています。
[管理ユーザを作成]	ユーザ名とパスワードを入力する。
[サーバの [スタート] メニュー エントリを作成]	管理サーバを Windows のスタート メニューにインストールするかどうか指定する。
[コンフィグレーションの概要]	以下のいずれかを行う。 ■ コンフィグレーションのサマリ情報を見直し、[Create] をクリックして定義の完了したドメインを作成する。 ■ コンフィグレーションのサマリ情報を見直し、[Previous] をクリックして表示済みのウィンドウに戻り、ドメインを作成する前に、すでに入力した情報を変更する機会を得る。

1. アドレスおよびポート番号の設定に関する詳細については、3-2 ページの「手順 1. コンフィグレーションの前提条件への準拠」を参照してください。

Configuration Wizard によるドメインのコンフィグレーションが完了すると、指定した場所に新しいドメインが作成されます。コンフィグレーション ファイル（config.xml）が、そのドメイン内に作成されます。このファイルには、クラスタ内の管理サーバと各管理対象サーバの定義が格納され、また、このファイルによって管理対象サーバがクラスタに割り当てられます。

注意： この手順の後半の手順では、config.xml ファイルを編集して、クラスタ ドメインをコンフィグレーションします。したがって、次の手順に進む前に、ここで作成した config.xml ファイルのバックアップ コピーを作成することをお勧めします。

手順 3. ドメイン用データベースのコンフィグレーション

データベース ウィザードは、WebLogic Integration コンフィグレーション ユーティリティで、前の手順で作成したドメイン用のデータベースの設定に役立ちます。

データベース ウィザードを実行する手順は次のとおりです。

1. 手順 2 で作成したドメインで wliconfig スクリプトを実行します。

   たとえば、mydomain というドメインをデフォルトの場所に作成した場合は、以下のコマンド シーケンスのいずれかを実行します（オペレーティング システムに合わせる）。

   - Windows:

     ```
     cd %BEA_HOME%\user_projects\mydomain
     wliconfig
     ```

   - UNIX:

     ```
     cd $BEA_HOME/user_projects/mydomain
     wliconfig
     ```

2. データベース ウィザードには、以下のオプションがあります。

- [データベースの切り替え]
  ドメインで使用するデータベースとして別のデータベースを指定するときに、このオプションを選択します。データベースを初期化するために呼び出されるコマンド（`CreateDB` および `RunSamples` など）で使用される環境変数は更新され、`config.xml` ファイルは、新しい設定を反映するように変更されます。このオプションは、データベースを初期化するわけではありません。データベースの初期化に備えて環境をコンフィグレーションするだけです。
- [データベースの切り替え]
  このオプションを選択すると、現在指定されているデータベースを初期化するか、新しいデータベースに切り替えてから初期化することができます。

3. データベース ウィザードでは、コンフィグレーション対象データベースへの接続に必要な値の入力が必要です。

データベース ウィザードの実行方法の詳細については、『*WebLogic Integration の起動、停止およびカスタマイズ*』、「WebLogic Integration のカスタマイズ」の「データベース コンフィグレーション ウィザードの使用」を参照してください。

手順 4. 1 つの管理対象サーバ用 BPM リソースのコンフィグレーション

2-27 ページの「JMS サーバと JMS 送り先」および 2-4 ページの「WebLogic Integration リソースのデプロイメント」で説明したとおり、以下の BPM リソースをクラスタ内の 1 つのノードにデプロイする必要があります。

- WLI-BPM Plugin Manager（URI は `wlpi-master-ejb.jar`）
- EventTopic JMS トピック（JNDI 名は `vcom.bea.wlpi.EventTopic`）

この要件を満足するため、ドメインのコンフィグレーションを変更する必要があります。これは、以下のいずれかの方法で実現できます。

- コンフィグレーション ファイルを編集する
- WebLogic Server Administration Console を使用する

コンフィグレーション ファイルを編集する

作成したドメインに格納されている config.xml ファイルには、変更すべき要素をすばやく特定して編集するために役立つコメントが入っています。次の手順を実行して、ドメインのコンフィグレーション ファイルに必要な変更を加えます。

1. 作成したドメインのルート ディレクトリで、config.xml ファイルをテキスト エディタで開きます。
2. MODIFY というラベルの付いたコメントを探します。

   次の表では、コンフィグレーション ファイルで MODIFY コメントのラベルの付いている要素を示し、必要な変更について説明します。

表 3-1 1 つの管理対象サーバ用 BPM リソースのコンフィグレーション

見つけるべき要素	要素の変更手順

<!-- MODIFY: In a cluster, the BPM Plugin Manager Targets attribute must specify only ONE cluster server -->
`<EJBComponent Name="WLI-BPM Plugin Manager"`
`Targets="`**`mycluster`**`"`
`URI="wlpi-master-ejb.jar"/>`
	WLI-BPM Plugin Manager EJB コンポーネントをこの管理対象サーバ（この場合は *managedserver-1:*）にデプロイするように、Targets 属性を変更する。
	
	`<EJBComponent Name="WLI-BPM Plugin Manager"`
	`Targets="`*`managedserver-1`*`"`
	`URI="wlpi-master-ejb.jar"/>`

<!-- MODIFY: This JMS Topic must be deployed to only one node in the cluster. Uncomment this section for one node.
`<JMSTopic Name="wlpiEvent"`
`JNDIName="com.bea.wlpi.EventTopic"`
`/>`
-->
	クラスタ内のただ 1 つの管理対象サーバにデプロイされるように、この JMSTopic 要素のコメントを解除する。
	
	`<JMSTopic Name="wlpiEvent"`
	`JNDIName="com.bea.wlpi.EventTopic"/>`

3. 3-16 ページの「手順 5. アダプタ用イベント ルータ WAR ファイルのコンフィグレーション」に進みます。

WebLogic Server Administration Console を使用する

上の表で説明した変更を行う前に、作成したドメインの管理サーバを起動すると、config.xml ファイルに書き込んだコメントは失われます。ただし、コメントが失われても、次の節で説明する手順に従い、システムを適切にコンフィグレーションすることができます。

- 1 つの管理対象サーバ用 BPM マスタ EJB をコンフィグレーションする
- 1 つの管理対象サーバ用 BPM イベント トピック をコンフィグレーションする

1 つの管理対象サーバ用 BPM マスタ EJB をコンフィグレーションする

BPM マスタ EJB を 1 つの管理対象サーバに対してコンフィグレーションする最も簡単な方法は、表 3-1 で示したように、ドメイン コンフィグレーション ファイルを変更する方法です。

注意： 以下の手順は参考情報です。この手順は、たとえば、コンフィグレーションを完了する前に config.xml ファイルのコメントが失われた場合に使用してください。

1 つの管理対象サーバに対して BPM マスタ EJB をコンフィグレーションする手順は以下のとおりです。

1. Administration Console のナビゲーション ツリーで、作成したドメインの [WLI-BPM Plugin Manager EJB] を選択します。

   [*Domain_Name* ｜デプロイメント｜ EJB ｜ WLI-BPM Plugin Manager]

2. [対象] タブを選択し、Administration Console の指示に従って、WLI-BPM Plugin Manager EJB がクラスタ内の 1 つの管理対象サーバのみにデプロイされるように、デプロイメント設定を変更します。

   Administration Console を使用して、EJB のデプロイメントをコンフィグレーションする方法の詳細については、次の URL にある『*Adminstration Console オンライン ヘルプ*』を参照してください。

http://e-docs.beasys.co.jp/e-docs/wls/docs70/ConsoleHelp/index.html

3-7 ページの「手順 2. WebLogic Integration ドメインの作成」の手順に従ってドメインの作成を完了すると、WLI-BPM Plugin Manager はこのクラスタを対象として設定されます。言い換えれば、Targets 属性にそのクラスタ名が格納されます。

必要な変更を行った後は、WLI-BPM Plugin Manager EJB は、作成したクラスタ内の 1 つの管理対象サーバを対象とします。以下の抜粋は、編集済みの 1 `config.xml` ファイルで、Targets 属性のコンフィグレーションの変更が示されています。この例では、EJB は `manageserver-1` という管理対象サーバを対象としています。

コード リスト 3-1 WLI-BPM Plugin Manager を 1 つの管理対象サーバを対象として設定する

```
<Application Name="WebLogic Integration" Deployed="false"
Path="C:/bea/weblogic700/integration/lib" TwoPhase="true">

...
<EJBComponent Name="WLI-BPM Plugin Manager"
Targets="manageserver-1"
URI="wlpi-master-ejb.jar"/>

...
</Application>
```

1 つの管理対象サーバ用 BPM イベント トピック をコンフィグレーションする

ドメインのコンフィグレーションを完了する最も簡単な方法は、3-11 ページの「手順 4. 1 つの管理対象サーバ用 BPM リソースのコンフィグレーション」で説明したように、ドメイン コンフィグレーション ファイルを変更する方法です。

注意： 以下の手順は参考情報です。この手順は、たとえば、コンフィグレーションを完了する前に `config.xml` ファイルのコメントが失われた場合に使用してください。

1 つの管理対象サーバに対して BPM イベント EJB をコンフィグレーションする手順は以下のとおりです。

1. Administration Console のナビゲーション ツリーで、JMS サーバに対して [送り先] を選択します。例を示します。

   [サービス | JMS | サーバ | WLIJMSServer_*manageserver-1* | 送り先]

   WLIJMSServer_*manageserver-1* は、作成したドメイン内のある 1 つの管理対象サーバに対する JMS サーバの名前を表しています。

   注意 : JMS サーバには任意の名前を割り当てることができます。ただし、`WLIJMSServer_`*node* の例で示される命名規約に従うことをお勧めします。この命名規約では、*node* は JMS サーバがデプロイされているサーバの名前を表します。

2. [新しい JMSTopic のコンフィグレーション] をクリックして [コンフィグレーション] タブを表示します。

3. 適切なフィールドに以下の情報を入力します。

   - [名前]: `wlpiEvent`
   - [JNDI 名] `com.bea.wlpi.EventTopic`

4. 各フィールドの残りの部分は、デフォルトをそのまま使用して、[適用] をクリックします。

JMS キューおよびトピック作成方法の詳細については、次の URL にある『*Adminstration Console オンライン ヘルプ*』を参照してください。

`http://e-docs.beasys.co.jp/e-docs/wls/docs70/ConsoleHelp/index.html`

次のリストは `config.xml` ファイルの抜粋で、BPM イベント トピック が、管理対象サーバに対する JMSServer 要素に追加されています（この例では、管理対象サーバには、`WLIJMSServer_manageserver1` という名前が付けられている）。このリストの中で、この節で特に取り上げている部分は太字で表記されています。

コード リスト 3-2 com.bea.wlpi.EventTopic のコンフィグレーション

```
<JMSServer Name="WLIJMSServer_manageserver1"
      Targets="manageserver1 (migratable)"
      TemporaryTemplate="TemporaryTemplate"
      Store="JMSWLIStore_manageserver1">

      <JMSTopic Name="wlpiEvent"
JNDIName="com.bea.wlpi.EventTopic"/>
```

```
...
</JMSServer>
```

手順 5. アダプタ用イベント ルータ WAR ファイルのコンフィグレーション

注意 : この手順は、WebLogic Integration または EAI ドメイン テンプレートに基づくドメインに対してのみ必要です。クラスタ ドメインが BPM ドメイン テンプレートに基づいている場合は 3-18 ページの「手順 6. RDBMS レルムのコンフィグレーション」 に進んでください。

2-4 ページの 「WebLogic Integration リソースのデプロイメント」 で説明したように、サンプル アダプタ用イベント ルータ WAR ファイルは、クラスタ内の 1 つのノードにデプロイする必要があります。この要件を満足するため、WebLogic Integration ドメインまたは EAI ドメインの一環としてコンフィグレーションされた、BEA_WLS_DBMS_ADK アダプタおよび BEA_POWERENTERPRISE_3_0 アダプタのドメイン コンフィグレーションを編集する必要があります。

コンフィグレーションを変更するには、WebLogic Server Administration Console を使用する方法と、ドメインにある `config.xml` ファイルを編集する方法があります。

:Administration Console を使用する方法

1. Administration Console のナビゲーション ツリーで、作成したドメインの次のイベント ルータ WAR ファイルを選択します。
   - [*Domain_Name* | デプロイメント | Web アプリケーション | DbmsEventRouter]
   - [*Domain_Name* | デプロイメント | Web アプリケーション | BEA_POWERENTERPRISE_3_0_EventRouter]
2. 適切な [対象] タブを選択し、Administration Console の指示に従って、各イベント ルータに対するデプロイメント設定を変更します。各イベントルータが、クラスタ内の 1 つの管理対象サーバにのみデプロイされるように指定します。

   Administration Console を使用して、Web アプリケーションのデプロイメントをコンフィグレーションする方法の詳細については、次の URL の『*Adminstration Console オンライン ヘルプ*』を参照してください。
   http://e-docs.beasys.co.jp/e-docs/wls/docs70/ConsoleHelp/index.html

config.xml の使用方法

次のリストは、管理サーバとクラスタ化された管理対象サーバがデプロイされたドメインの場合の、サンプルの `config.xml` ファイルの抜粋です。このリストでは、クラスタ内のある 1 つの管理対象サーバにデプロイするようにコンフィグレーションされた 2 つのアダプタ用のイベント ルータ WAR ファイルが示されています。`WebAppComponent` 要素は太字で表記されています。

コード リスト 3-3 BEA_WLS_DBMS_ADK アダプタおよび BEA_POWERENTERPRISE_3_0 アダプタのコンフィグレーション

```
<Application Deployed="true" Name="BEA_WLS_DBMS_ADK"
Path="<WLI_HOME>/adapters/dbms/lib/BEA_WLS_DBMS_ADK.ear"
TwoPhase="true">

      <ConnectorComponent Name="BEA_WLS_DBMS_ADK"
        Targets="MyCluster" URI="BEA_WLS_DBMS_ADK.rar"/>
```

```
      <WebAppComponent Name="DbmsEventRouter" Targets="MyServer-1"
        URI="BEA_WLS_DBMS_ADK_EventRouter.war"/>

      <WebAppComponent Name="BEA_WLS_DBMS_ADK_Web"
        Targets="MyCluster" URI="BEA_WLS_DBMS_ADK_Web.war"/>

</Application>

 :

<Application Deployed="true" Name="BEA_POWERENTERPRISE_3_0"
Path="<WLI_HOME>/adapters/powerenterprise/lib/
BEA_POWERENTERPRISE_3_0_EAR.ear" TwoPhase="true">

      <ConnectorComponent Description="J2EE CA adapter for
        PowerEnterprise!" Name="BEA_POWERENTERPRISE_3_0"
        Targets="MyCluster" URI="BEA_POWERENTERPRISE_3_0.rar"/>

       <WebAppComponent
Name="BEA_POWERENTERPRISE_3_0_EventRouter"
        Targets="MyServer-1"
        URI="BEA_POWERENTERPRISE_3_0_EventRouter.war"/>

        <WebAppComponent Name="BEA_POWERENTERPRISE_3_0_Web"
        Targets="MyCluster"
URI="BEA_POWERENTERPRISE_3_0_Web.war"/>

</Application>
```

手順 6. RDBMS レルムのコンフィグレーション

作成したドメインで、以前のリリースの WebLogic Integration の RDBMS レルムを使用する場合は、RDBMSRealm 要素を作成したドメイン用の `config.xml` ファイルに格納する必要があります。この要素は、3-7 ページの「手順 2. WebLogic Integration ドメインの作成」の手順を使用してドメインを作成した時点で生成された `config.xml` ファイルではコンフィグレーションはされていますが無効になっています。RDBMSRealm 要素を有効にする手順は次のとおりです。

1. 作成した WebLogic Integration ドメインのルート ディレクトリにある `config.xml` ファイルを開きます。
2. RDBMSRealm 要素を探して、RDBMSRealm 要素のコメントを解除します。

3. config.xml ファイルで定義されている RDBMSRealm 要素は、次のリストに示すように、Pointbase データベース用にコンフィグレーションされています。別のデータベースを使用している場合は、RDBMSRealm 要素の DatabaseDriver、DatabasePassword、DatabaseURL、および DatabaseUserName の各属性のコンフィグレーションをやり直してください。

コード リスト 3-4 RDBMSRealm 要素

```
<RDBMSRealm Name="wlpiRDBMSRealm"
          DatabaseDriver="com.pointbase.jdbc.jdbcUniversalDriver"
        DatabasePassword="none"
        DatabaseURL="jdbc:pointbase://localhost:9094/WLIDB”
        DatabaseUserName="none"
          :
          :
```

セキュリティ レルム データの移行に関する詳細については、『*WebLogic Integration 移行ガイド*』、「WebLogic Integration 2.1 から WebLogic Integration 7.0 への移行」 の 「手順 8. セキュリティ レルム データの移行」 を参照してください。

手順 7. ルータのコンフィグレーション

ソフトウェア ルータは、作成したドメインにある config.xml ファイルの定義済みセクションのコメントを解除することによって、組み込み WebLogic HttpClusterServlet に基づいたコンフィグレーションを行うことができます。

作成したドメインに格納されている config.xml ファイルには、変更すべき要素をすばやく特定して編集するのに役立つコメントが入っています。次の手順を実行して、ドメインのコンフィグレーション ファイルに必要な変更を行います。

1. 作成した WebLogic Integration ドメインのルート ディレクトリにある config.xml ファイルを開きます。

2. ROUTER-OPTION というラベルの付いたコメントを探し、それらのコメントに記述されている指示に従って、作成したドメイン用のルータをコンフィグレーションします。

   ルータのサーバ名、リスン アドレス、リスン ポートなどに対して、適切な値を入力する必要があります。

3. 次の config.xml ファイルから抜粋したコードに示されているように、Web サーバのコンフィグレーションは、ルータのコンフィグレーションに含まれています。

   ```
   <WebServer Name="ROUTER_NAME"
   DefaultWebApp="DefaultWebApp_ROUTER_NAME"
   . . .  />
   ```

   Web サーバ要素は、DefaultWebApp 属性を介してデフォルトの Web アプリケーションを参照する点に注意してください（*ROUTER_NAME* は、ルータに割り当てられた名前）。

   a. 指定された DefaultWebApp の値（この例では、DefaultWebApp_*ROUTER_NAME*）と一致するディレクトリが、次の場所にあることを確認してください。

      DOMAIN_HOME/applications/

      このパスで、*DOMAIN_HOME* は、作成したドメインのルート ディレクトリを表します。

   b. 前の手順で指定した DefaultWebApp ディレクトリに web.xml デプロイメント記述子を作成します。web.xml デプロイメント記述子には、HttpClusterServlet の登録が格納されている必要があります。

      web.xml デプロイメント記述子の作成方法の詳細については、次の URL にある『*WebLogic Server クラスタ ユーザーズ ガイド*』、「WebLogic クラスタのセットアップ」の「プロキシ プラグインのコンフィグレーション」を参照してください。

      http://edocs.beasys.co.jp/e-docs/wls/docs70/cluster/setup.html

注意： クラスタに対してハードウェア ルータまたはソフトウェア ルータをコンフィグレーションするときは、クラスタの外部からのメッセージは、そのルータの URL に送信される必要があります。

ハードウェア ルータおよびソフトウェア ルータに関する詳細は、次の URL にある『*WebLogic Server クラスタ ユーザーズ ガイド*』を参照してください。

http://edocs.beasys.co.jp/e-docs/wls/docs70/cluster/index.html

手順 8. startWeblogic コマンド ファイルの編集

-Dweblogic.management.discover パラメータを true に設定するには、作成したドメインの startWeblogic.cmd ファイルまたは startWeblogic.sh ファイルを編集する必要があります。

1. 作成した WebLogic Integration ドメインのルート ディレクトリにある startWeblogic ファイルを開きます。
2. WebLogic 開始コマンドに対する -Dweblogic.management.discover の引数を見つけます。
3. 指定されている値を false から true に変更します。

   次のコード リストは、サーバ起動コマンドの例を示していて、-Dweblogic.management.discover 引数が変更されています。このコードは、1 つのコマンドを表します。この例では複数の行に分けて読みやすくしてあります。ただし、コマンド ファイルでは 1 行で入力されています。

コード リスト 3-5 WebLogic Integration のクラスタ ドメインに対するサーバ開始コマンド

```
REM Start weblogic

%JAVA_HOME%\bin\java %JAVA_VM% %JAVA_OPTIONS% -Xmx256m
-classpath %SVRCP%
-Dweblogic.servlet.ClasspathServlet.disableStrictCheck=true
-Dwli.bpm.server.evaluator.supportsNull=false
-Dweblogic.management.username= -Dweblogic.management.password=
-Dweblogic.Name=adminserver
-Dweblogic.RootDirectory=%WLI_DOMAIN_HOME%
-Djava.security.policy=%WL_HOME%\lib\weblogic.policy
-Dweblogic.management.discover=true
-Dweblogic.ProductionModeEnabled=%STARTMODE%  weblogic.Server
```

ドメインの管理対象サーバが実行されているときに管理サーバが再起動されるシナリオでは、`-Dweblogic.management.discover` が true に設定されている場合に、管理サーバは、実行されている管理対象サーバを検出することができます。

手順 9. ドメインの管理対象サーバの設定

この手順では、管理対象サーバを追加することにより、作成したドメインを拡張する方法を説明します。管理対象サーバを追加するには、管理対象サーバを作成し、そのサーバの WebLogic Integration コンポーネントをコンフィグレーションする必要があります。

WebLogic Integration ドメインは、次のいずれかの方法で設定することができます。

- 管理サーバとクラスタ化された管理対象サーバを同じマシン上で設定する。
- 管理サーバとクラスタ化された管理対象サーバを別々のマシン上で設定する。
- 上記 2 つのコンフィグレーションの組み合わせ。たとえば、管理サーバといくつかの管理対象サーバをホストする 1 つのマシンと、追加の管理対象サーバをホストする 1 つまたは複数の他のマシンを組み合わせて 1 つのクラスタとする場合もあります。

この節では、クラスタで管理対象サーバを設定する以下の方法について説明します。

- 既存のインストールへの管理対象サーバを追加する
- 新しい場所へ管理対象サーバを追加する

両方とも、Configuration Wizard が提供するテンプレートのいずれかを使用して作成されたドメインに管理対象サーバを追加する手順の説明です。

既存のインストールへの管理対象サーバを追加する

管理対象サーバを WebLogic Integration ドメインに追加するには、次の手順を実行します。

- 手順 1. 管理対象サーバを新しく作成する
- 手順 2. 新しい管理対象サーバのドメイン コンフィグレーションを更新する（オプション）

注意： この節で説明する手順では、ドメイン名を `mydomain` とし、次のデフォルトの場所にあるものとします。
BEA_HOME`\user_projects`

手順 1. 管理対象サーバを新しく作成する

1. 管理サーバと WebLogic Server Administration Console を起動します。
   a. 管理サーバを起動するには、『*WebLogic Integration の起動、停止およびカスタマイズ*』、「はじめに」の「WebLogic Integration の起動」を参照してください。
   b. コンソールを起動するには、『*WebLogic Integration の起動、停止およびカスタマイズ*』、「WebLogic Integration 管理ツールと設計ツール」の「WebLogic Server Administration Console の起動」を参照してください。
2. Administration Console のナビゲーション ツリーで [サーバ] を選択します。
3. [新しい Server のコンフィグレーション] をクリックします。
4. [名前]、[リスン アドレス] (サーバ インスタンスの IP アドレス)、および必要に応じて [外部 DNS 名] フィールドに値を入力します。
   [外部 DNS 名] に指定する値は、マルチホーム マシンのホスト名または仮想ホスト名でもかまいません（マルチホーム マシンは、複数の IP アドレスが割り当てられているマシン）。
5. [マシン] ドロップダウン リストからマシン名を選択します。
6. [作成] をクリックします。
7. [クラスタ] タブを選択します。

8. [クラスタ] ドロップダウン リストから該当するクラスタを選択します。

注意 : WebLogic Server Administration Console によるサーバ、クラスタ、マシン、およびドメインの作成およびコンフィグレーションについての詳細は、次の URL にある『*Adminstration Console オンライン ヘルプ*』を参照してください。

```
http://edocs.beasys.co.jp/e-docs/wls/docs70/ConsoleHelp/index.html
```

手順 2. 新しい管理対象サーバのドメイン コンフィグレーションを更新する（オプション）

管理対象サーバを起動するには、まず、そのサーバをドメインのコンフィグレーションに追加する必要があります。管理対象サーバは、ドメインの作成時にドメイン コンフィグレーションに追加することも、ドメインの作成後に追加することも可能です。したがって、この手順は省略可能です。次のガイドラインに従って、この手順を実行する必要があるかどうか判断してください。

- ドメインを最初に作成した時点で、前の手順（3-23 ページの「手順 1. 管理対象サーバを新しく作成する」）で作成した管理対象サーバをコンフィグレーションに追加しなかった場合は、ドメイン コンフィグレーションを更新する必要があります（言い換えれば、3-7 ページの「手順 2. WebLogic Integration ドメインの作成」を実行したときに Configuration Wizard で管理対象サーバを定義しなかった場合は更新が必要）。
- 3-23 ページの「手順 1. 管理対象サーバを新しく作成する」で作成した管理対象サーバがすでにドメインにコンフィグレーションされている場合は、ドメイン コンフィグレーションを更新する必要はありません（言い換えれば、3-7 ページの「手順 2. WebLogic Integration ドメインの作成」の説明に従ってドメインを作成した時点で Configuration Wizard で管理対象サーバを定義した場合は更新は不要）。

新しい管理対象サーバのドメイン コンフィグレーションを更新するには、次の手順を実行します。

1. JMS JDBC ストアをコンフィグレーションして、接続プールに関連付けます。
   a. Administration Console のナビゲーション ツリーで、[*Domain_Name* ｜ サービス ｜ JMS ｜ ストア] を選択します。

b. [新しい JMSJDBCStore のコンフィグレーション] をクリックして [コンフィグレーション] タブを表示します。

c. 適切なフィールドに以下の情報を入力します。

[名前] : `JMSWLIStore_new`*`manageserver`*

Connection Pool: `wliPool`

[プレフィックス名] : *`newmanageserver`*

d. [作成] をクリックして、新しい管理対象サーバに対する JMSJDBCStore を新たに作成します。

2. JMS サーバをコンフィグレーションし、JMS JDBC ストアと関連付けます。

a. Administration Console のナビゲーション ツリーで、[*Domain_Name* ｜ サービス ｜ JMS ｜ サーバ] を選択します。

b. [新しい JMSServer のコンフィグレーション] をクリックして [コンフィグレーション] タブを表示します。

c. 適切なフィールドに以下の情報を入力します。

[名前] : `WLIJMSServer_new`*`manageserver`*

[ストア] : `JMSWLIStore_new`*`manageserver`*

[一時的なテンプレート] `TemporaryTemplate`

`WLIJMSServer_`*`newmanageserver`* は、新しい JMS サーバの名前を表します。`JMSWLIStore_new`*`manageserver`* は、JMSJDBC ストアの作成時に付けた名前です。

JMS サーバには任意の名前を与えることができます。ただし、`WLIJMSServer_`*`node`* の例に示される命名規約を使用することをお勧めします。このフォーマットでは、*node* は JMS サーバがデプロイされているサーバ（WebLogic Server インスタンス）の名前を表します。

d. 残りのフィールドについては、デフォルトをそのまま使用し、[作成] をクリックして新しい管理対象サーバ用の JMS サーバを新たに作成します。

3. 新たに定義された JMS サーバに対して送り先をコンフィグレーションします。

a. Administration Console ナビゲーション ツリーで、新しい JMS サーバのコンフィグレーションにアクセスします。

[*Domain_Name* ｜サービス｜ JMS ｜サーバ｜ *Server_Name* ｜送り先]

b. [送り先のコンフィグレーション] をクリックします。

c. [新しい JMSTopic のコンフィグレーション] または [新しい JMSQueue のコンフィグレーション] をクリックします。

`WLI_FailedEvent-node` の送り先を先にコンフィグレーションします。そうすることにより、`WLI_JMSTemplate-node` を再配信用にコンフィグレーションすることができます（再配信属性のコンフィグレーションの方法に関する詳細については、2-30 ページの「エラー送り先」を参照）。`WLI_JMSTemplate-node` は、他のいくつかのキュー送り先によっても使用されます。

Administration Console を使用してこのタスクを完了する方法については、次の URL にある『*Adminstration Console オンライン ヘルプ*』の、「JMS」の「JMS 送り先のタスク」を参照してください。

`http://edocs.beasys.co.jp/e-docs/wls/docs70/ConsoleHelp/index.html`

注意： クラススタ内の既存ノードを参照すると、コンフィグレーションを必要とする送り先が確認できます。また、どの送り先が `WLI_JMSTemplate-node` を使用するかもかわかります。必要な送り先は、ドメイン作成に使用したドメイン テンプレートによって異なります。

WLI ドメイン テンプレートに基づいて、ドメインの JMS サーバに対してコンフィグレーションされるすべての JMS キューおよびトピックのリストについては、2-27 ページの「JMS サーバと JMS 送り先」を参照してください。

4. WebLogic Integration の分散送り先をコンフィグレーションします。

各分散送り先に対して複数の JMS 送り先がコンフィグレーションされます。物理送り先は、クラスタ内の各管理対象サーバに対して 1 つずつコンフィグレーションされます。

新しく作成された管理対象サーバに対して JMS 送り先をコンフィグレーションするには、次の手順を実行します。

a. Administration Console のナビゲーション ツリーで次のノードを選択して、作成した WebLogic Integration ドメインの分散送り先にアクセスします。

[*Domain_Name* ｜サービス｜ JMS ｜分散送り先]

注意 : WebLogic Integration デプロイメントに対する送り先は、2-27 ページの「JMS サーバと JMS 送り先」にリストしてあります。

b. 各分散送り先に対して、JMS 分散キュー メンバーを作成します。

分散キュー メンバーの詳細については、次の URL にある『*Adminstration Console オンライン ヘルプ*』の、「JMS」の「JMS 分散送り先のタスク」を参照してください。

`http://edocs.beasys.co.jp/e-docs/wls/docs70/ConsoleHelp/index.html`

注意 : WebLogic Integration 分散送り先のコンフィグレーション例については、2-27 ページの「JMS サーバと JMS 送り先」を参照してください。また、ドメイン作成時に作成された `config.xml` ファイルで、`JMSDistributedQueue` 要素および `JMSDistributedTopic` 要素を調べることもできます。

新しい場所へ管理対象サーバを追加する

管理サーバとクラスタ化された管理対象サーバが別々のマシンに配置されているドメインに管理対象サーバを追加するには、次の手順を実行します。

- 手順 1. コンフィグレーション済みのドメインの内容を新しい場所にコピーする
- 手順 2. コピーしたディレクトリの内容を変更する
- 手順 3. 管理対象サーバを作成する
- 手順 4. 新しい管理対象サーバのドメイン コンフィグレーションを更新する（オプション）

手順 1. コンフィグレーション済みのドメインの内容を新しい場所にコピーする

管理対象サーバを新しい場所に設定するには、作成したドメイン ディレクトリの内容を新しい場所にコピーして修正します。

次の手順を実行してください。

1. WebLogic Integration を新しい場所にインストールします。
2. 作成したドメイン（3-7 ページの「手順 2. WebLogic Integration ドメインの作成」参照）のディレクトリの内容をリモート マシンにコピーします。コピーするディレクトリは、管理対象サーバの起動場所となります。

注意： 混合クラスタ環境（Windows システム上で実行されている WebLogic Integration のインスタンスと UNIX システム上で実行されているインスタンスを含むクラスタ）を設定すると、改行文字にかかわって、ある問題が発生する場合があります。Windows システムで実行されるスクリプトで使用される改行文字は `^M` です。これらの文字が Windows システムから UNIX システムにコピーされたファイルに残っていることがあります。Windows の改行文字が UNIX システム上に残される場合は、そのファイルを開いて、スクリプトを実行する前に `^M` 文字を削除します。この処理は、任意のテキスト エディタで実行できます。また、Solaris システムでは、`dos2unix` コマンドを使用することもできます。`dos2unix` ユーティリティは、DOS の拡張文字セットに含まれる文字を、対応する ISO 規格の文字に変換します。

FTP を使用して ASCII ファイルを Windows システムから UNIX システムへ転送すると、デフォルトの ASCII モードを選択することによって、改行文字に関するこの問題を回避することができます。

手順 2. コピーしたディレクトリの内容を変更する

注意： 以下は、`mydomain` というドメイン ディレクトリを *`BEA_HOME/user_projects`* にコピーしたと仮定した場合の説明です。

ディレクトリの内容を変更するには、*`BEA_HOME/user_projects`* から、次の表にリストするものを除くすべてのファイルおよびディレクトリを削除します。

ファイル	`startWeblogic.cmd` or `startWebLogic.sh`

	startManagedWeblogic.cmd or startManagedWebLogic.sh
	caKeyStore.pks
	privateKeyStore.pks
ディレクトリ	applications
	cacerts
	certs
	keys
	wlai[1]

1. アダプタ、アプリケーション ビュー、および Application Integration プラグインがデプロイされているドメインに管理対象サーバを追加する場合は、wlai ディレクトリはドメイン ディレクトリにあります。

手順 3. 管理対象サーバを作成する

1. 管理サーバと Administration Console を起動します。
   a. 管理サーバを起動するには、『*WebLogic Integration の起動、停止およびカスタマイズ*』、「はじめに」の「WebLogic Integration の起動」を参照してください。
   b. コンソールを起動するには、『*WebLogic Integration の起動、停止およびカスタマイズ*』、「WebLogic Integration 管理ツールと設計ツール」の「WebLogic Server Administration Console の起動」を参照してください。
2. Administration Console のナビゲーション ツリーで [サーバ] を選択します。
3. [新しい Server のコンフィグレーション] をクリックします。
4. [名前]、[リスン アドレス] (サーバ インスタンスの IP アドレス)、および必要に応じて [外部 DNS 名] フィールドに値を入力します。

   [外部 DNS 名] に指定する値は、マルチホーム マシンのホスト名または仮想ホスト名でもかまいません。
5. [マシン] ドロップダウン リストからマシン名を選択します。
6. [作成] をクリックします。

7. [クラスタ] タブを選択します。

8. [クラスタ] ドロップダウン リストから該当するクラスタを選択します。

注意： WebLogic Server Administration Console によるサーバ、クラスタ、マシン、およびドメインの作成およびコンフィグレーションについての詳細は、次の URL にある『*Adminstration Console オンライン ヘルプ*』を参照してください。

```
http://edocs.beasys.co.jp/e-docs/wls/docs70/ConsoleHelp/index.html
```

手順 4. 新しい管理対象サーバのドメイン コンフィグレーションを更新する（オプション）

管理対象サーバを起動するには、まず、そのサーバをドメインのコンフィグレーションに追加する必要があります。管理対象サーバは、ドメインの作成時にドメイン コンフィグレーションに追加することも、ドメインの作成後に追加することも可能です。したがって、この手順は省略可能です。

この手順を実行する必要の有無、およびその方法を明らかにするには、3-24 ページの「手順 2. 新しい管理対象サーバのドメイン コンフィグレーションを更新する（オプション）」で説明したのと同じガイドラインと手順を使用してください。

手順 10. WebLogic Intergration の自動再起動のコンフィグレーション

WebLogic Integration がクラスタ環境でデプロイされているかどうかにかかわらず、システム クラッシュ、ハードウェアの再起動、サーバの不具合などが原因でシャットダウンしたサーバを自動的に再起動するように、システムをコンフィグレーションすることができます。これは、次のいずれかの方法で Node Manager をコンフィグレーションすることによって実行できます。

- Configuration Wizard によるドメイン作成時に作成したコンフィグレーション ファイルを編集します。

作成したドメインに格納されている config.xml ファイルには、変更すべき要素をすばやく特定して編集するのに役立つコメントが入っています。Node Manager をコンフィグレーションし、ドメインのコンフィグレーション ファイルを変更するには、次の手順を実行します。

a. 作成したドメインのルート ディレクトリで、config.xml ファイルをテキスト エディタで開きます。

b. NM-OPTION というラベルの付いたコメントを探し、それらのコメントに記述されている指示に従って、Node Manager および SSL をコンフィグレーションします。

c. 自己状態モニタ機能をコンフィグレーションする（すなわち、Node Manager が管理対象サーバの状態をチェックする頻度を指定する）には、4-9 ページの「手順 4. 自己状態モニタ機能をコンフィグレーションする」を参照してください。

d. Node Manager を起動するには、4-10 ページの「手順 5. Node Manager を起動する」を参照してください。

- 4-5 ページの「自動再起動のための WebLogic Integration のコンフィグレーション」で説明する手順に従って、必要なコンポーネントを WebLogic Server Administration Console を通じてコンフィグレーションします。

手順 11. 障害が発生したノードから健全なノードへ移行するための WebLogic Integration のコンフィグレーション

WebLogic Integration デプロイメントによって障害が発生したノードから健全なノードへのリソースの移行がサポートされるように、デプロイメントをコンフィグレーションするには、4-14 ページの「故障ノードから健全なノードに移行するための WebLogic Integration のコンフィグレーション」に概要が説明されている次の手順を実行します。

手順 12. WebLogic Integration のセキュリティ コンフィグレーション

作成したクラスタに対して SSL をコンフィグレーションするには、作成したドメインの config.xml ファイルにある定義済みのセクション（各 Server 要素に 1 セクションずつ）のコメントを解除します。

config.xml ファイルには、変更すべき要素をすばやく特定して編集するのに役立つコメントが入っています。SSL をコンフィグレーションし、ドメインのコンフィグレーション ファイルを変更するには、次の手順を実行します。

1. 作成した WebLogic Integration ドメインのルート ディレクトリにある config.xml ファイルを開きます。
2. SSL-OPTION というラベルの付いたコメントを探して、適切なセクションのコメントを解除して、SSL をドメインに合うようにコンフィグレーションします。

B2B Integration 機能がマルチノード クラスタにデプロイされているドメインの場合は、クラスタ内の各マシンに対して、キーストア、サーバ証明書、startWeblogic スクリプトなどもコンフィグレーションする必要があります。

実行する必要があるタスクの詳細については、以下を参照してください。

- 『*B2B Integration セキュリティの実装*』、「キーストアのコンフィグレーション」 の 「Multinode Cluster におけるキーストアの使用」
- 第 5 章「WebLogic Integration セキュリティの使い方」

警告： 作成したドメインが、WLI Domain テンプレートまたは EAI Domain テンプレートに基づいていて、ドメインに対してキーストアをコンフィグレーションする場合は、WebLogic Integration Application 要素の Deployed 属性を *false* に設定してからでないと、キーストアはコンフィグレーションできません。

次のリストは、WebLogic Integration ドメイン用の config.xml ファイルの抜粋で、Deployed 属性が false に設定されています。

コード リスト 3-6 キーストアのコンフィグレーションに先立つ Deployed 属性

の設定

```
<Application Name="WebLogic Integration" Deployed="false"
Path="C:/bea/weblogic700/integration/lib" TwoPhase="true">
```

手順 13. ドメイン内のサーバの起動

この節では、クラスタ ドメインのサーバを起動する方法について説明します。

- サーバを起動する前に
- Node Manager がコンフィグレーションされていないドメインのサーバを起動する
- Node Manager がコンフィグレーションされているドメインのサーバを起動する
- サーバをモニタおよびシャットダウンする

サーバを起動する前に

作成したドメインのサーバを起動する前に、次の手順を実行します。

1. ドメイン コンフィグレーション ファイルの WebLogic Integration Applcation 要素の Deployed 属性が `true` に設定されていることを確認します。

   3-32 ページの「手順 12. WebLogic Integration のセキュリティ コンフィグレーション」で説明したとおり、ドメインに対してキーストアをコンフィグレーションした場合は、この Deployed 属性は、*false* に設定したはずです。

   次のリストは、WebLogic Integration ドメイン用の `config.xml` ファイルの抜粋で、Deployed 属性が true に設定されています。

   ```
   <Application Name="WebLogic Integration" Deployed="true"
   Path="C:/bea/weblogic700/integration/lib" TwoPhase="true">
   ```

2. ドメイン作成時に（3-7 ページの「手順 2. WebLogic Integration ドメインの作成」参照）デフォルト以外のサーバ名を指定した場合は、ここで必ずドメイン内の次のディレクトリ名を変更してください。

DOMAIN_HOME/applications/DefaultWebApp_myserver

このパスで、*DOMAIN_HOME* は、作成したドメインのルート ディレクトリを表します。myserver という文字列を、管理サーバに対して指定した名前に置き換えます。

Node Manager がコンフィグレーションされていないドメインのサーバを起動する

Node Manager がコンフィグレーションされていないドメインのサーバを起動するには、次の手順を実行します。

1. 次の startWebLogic コマンドを実行して管理サーバを起動します。

   ```
   cd DOMAIN_HOME
   startWeblogic
   ```

   このコマンドラインで、*DOMAIN_HOME* は、作成したドメインのルート ディレクトリを表します。

2. 管理サーバが起動した後、各管理対象サーバに対して順に startManagedWebLogic コマンドを実行してドメインの管理対象サーバを起動します。言い換えれば、各管理対象サーバ インスタンスをインストールしたルート ディレクトリに移動し、次の startManagedWeblogic コマンドを実行します。

   ```
   cd DOMAIN_HOME
   startManagedWeblogic managedserver
   ```

   このコマンドラインで、*managedserver* はドメイン内の管理対象サーバの名前を表します。

   各管理対象サーバが起動すると、コマンド ウィンドウにステータス メッセージが表示されます。

Node Manager がコンフィグレーションされているドメインのサーバを起動する

Node Manager がコンフィグレーションされているドメインのサーバを起動するには、次の手順を実行します。

1. 次の `startWebLogic` コマンドを実行して管理サーバを起動します。

   ```
   cd DOMAIN_HOME
   startWeblogic
   ```

   このコマンドラインで、*`DOMAIN_HOME`* は、作成したドメインのルート ディレクトリを表します。

2. ドメインの管理対象サーバを起動します。

   a. 管理対象サーバをホストする各マシンで Node Manager を起動します（まだ起動していなかった場合）（4-10 ページの「手順 5. Node Manager を起動する」を参照）。

   b. Administration Console のナビゲーション ツリーで、各管理対象サーバの名前を順に選択します。

   c. メイン コンソール ウィンドウで、[Control] タブを選択します。

   d. [Start this Server] をクリックします。

   サーバ起動コマンドに対する、WebLogic Server Administration Console から行う他の設定の影響については、このソフトウェアおよび次の URL にある『*Adminstration Console オンライン ヘルプ*』を参照してください。

   ```
   http://edocs.beasys.co.jp/e-docs/wls/docs70/ConsoleHelp/index.html
   ```

サーバをモニタおよびシャットダウンする

起動が完了すると、WebLogic Server Administration Console を使用して、デプロイメントとステータスを確認できます。

その後、WebLogic Server Administration Console を使用して WebLogic Integration アプリケーションをシャットダウンします。コマンド ウィンドウを閉じる、〔Ctrl〕 + 〔c〕 を押す、などの操作で WebLogic Integration を閉じないこと

をお勧めします。アプリケーションを正常にシャットダウンするには、『*WebLogic Integration の起動、停止およびカスタマイズ*』、「はじめに」の「WebLogic Integration の停止」で説明するとおり、`stopWebLogic` コマンドを実行してください。

4 WebLogic Integration の高可用性

クラスタ化された WebLogic Integration アプリケーションは、スケーラビリティと高可用性を提供します。高可用性を備えたデプロイメントには、ハードウェアやネットワークに障害が発生した場合に備えた回復機能が用意されており、障害発生時にはバックアップ コンポーネントにコントロールを渡す仕組みになっています。

以下の節では、WebLogic Integration デプロイメントのクラスタ化と高可用性について説明します。

- WebLogic Integration の高可用性について
- 自動再起動のための WebLogic Integration のコンフィグレーション
- 故障ノードから健全なノードに移行するための WebLogic Integration のコンフィグレーション
- フェイルオーバと回復

WebLogic Integration の高可用性について

クラスタが高可用性を発揮するには、サービス障害から回復する能力が必要です。WebLogic Server は、複製された HTTP セッション ステート、クラスタ オブジェクト、およびクラスタ環境でのサーバに固有のサービスに対するフェイルオーバをサポートします。WebLogic Server によるそのようなフェイルオーバ シナリオの処理に関する詳細は、次の URL にある『*WebLogic Server クラスタ ユーザーズ ガイド*』の「クラスタでの通信」を参照してください。

`http://edocs.beasys.co.jp/e-docs/wls/docs70/cluster/features.html`

推奨ハードウェアおよびソフトウェア

一般的な WebLogic Integration 環境で使用できる基本コンポーネントは次のとおりです。

- 管理サーバ
- クラスタ内の管理対象サーバのセット
- HTTP ロード バランサ（ルータ）
- 共用ファイル システム－ B2B Integration 機能を使用するクラスタで高可用性を実現するためには、共用ファイル システムが必要です。Storage Area Network (SAN) またはマルチポート型のディスク システムの使用をお勧めします。
- Oracle、Microsoft SQL、または Sybase のデータベース－データベース ベンダが提供するあらゆる高可用性およびフェイルオーバ ソリューションを活用してください（4-23 ページの「データベースの回復」を参照）。
- 永続モード－永続モード（WebLogic Integration のデフォルト設定）でアプリケーションをデプロイします。WebLogic Integration システム障害が発生した後回復できるようにするには、アプリケーションは、クラスタにデプロイされているか、単独のサーバにデプロイされているかにかかわらず、永続モードで実行する必要があります。B2B Integration 機能が使用されているクラスタは、永続モードがオフになっている場合は動作しません。

  永続モードをオンにして WebLogic Integration を実行すると、オブジェクトのインメモリの動的な状態は、WebLogic Integration リポジトリの永続ストレージに保存され、必要に応じてそこから取り出すことができます。永続モードによって、異常終了またはクラッシュの際に実行時の状態を回復できることが保証されます。

クラスタ システムのネットワーク トポロジのプランニングについての論議は、この節で扱う内容の範囲を越えています。Web アプリケーションで、ロード バランサ、ファイアウォール、Web サーバについて、1 つまたは複数の WebLogic Server クラスタを組織化することにより、ロード バランシングとフェイルオーバの機能をフルに活用できる方法の詳細は、次の URL にある『*WebLogic Server クラスタ ユーザーズ ガイド*』の「クラスタ アーキテクチャ」を参照してください。

`http://edocs.beasys.co.jp/e-docs/wls/docs70/cluster/planning.html`

WebLogic Integration の回復から期待できること

高可用性を備えたデプロイメントは、システム障害の発生に備えて、回復機能を備えています。WebLogic Integration は、自動再起動または手動移行ができるようにコンフィグレーションできます。

- クラスタ環境にあるかどうかにかかわらず、管理対象サーバで自動再起動するように WebLogic Integration をコンフィグレーションすることができます。詳細については、4-5 ページの「自動再起動のための WebLogic Integration のコンフィグレーション」を参照してください。
- クラスタ環境の故障ノードから健全なノードへ手動移行ができるように WebLogic Integration をコンフィグレーションすることができます。詳細については、4-14 ページの「故障ノードから健全なノードに移行するための WebLogic Integration のコンフィグレーション」を参照してください。

注意： XOCP ビジネス プロトコルに基づく WebLogic Integration アプリケーションに対しては、高可用性はサポートされていません。そのようなアプリケーションには、回復機能がありません。

WebLogic Integration を適切にコンフィグレーションすると、そのデプロイメントに対して以下の動作を期待することができます。

- サーバに障害が発生すると、WebLogic Integration によって、アクティブなサーバへの接続が再確立され、トランザクションはそのサーバで再試行されます。
- メッセージ配信の失敗時 :
  - RosettaNet メッセージの場合は、WebLogic Integration プロトコル レイヤはメッセージを再試行せずに、HttpStatus コードをワークフロー レイヤに返します。RosettaNet ワークフローは、通常、再試行を処理する設計となっています。
  - ebXML メッセージの場合は、ebXML の配信セマンティクス（*once and only once*）を指定する際にメッセージ再試行も指定します。WebLogic Integration 再試行の指定値に基づいて、プロトコル レイヤが、配信に失敗した ebXML メッセージを再試行します。
- WebLogic Integration は、あるクラスタ内の故障ノードから健全なノードへのリソースの手動移行をサポートします。詳細については、4-5 ページの「自動再起動のための WebLogic Integration のコンフィグレーション」および

4-14 ページの「故障ノードから健全なノードに移行するための WebLogic Integration のコンフィグレーション」を参照してください。

- サーバがクラッシュする前に実行されていた WebLogic Integration リソースは、サーバが再起動またはフェイルオーバした時点で再び実行されます。
- あるクラスタの管理サーバが使用できなくなると、デプロイ要求やアンデプロイ要求は中断されますが、管理対象サーバは要求の処理を続行します。管理対象サーバは、既存のコンフィグレーションを使用して起動および再起動することができます。ただし、管理サーバが復帰するまで、クラスタのコンフィグレーションを変更することはできません（たとえば、クラスタへの新しいノードの追加などは不可）。詳細については、4-19 ページの「Administration Server に対するバックアップとフェイルオーバ」を参照してください。
- クラスタ内の管理対象サーバに障害が発生すると、要求の処理は中断されますが、そのクラスタ内の他の管理対象サーバは要求の処理を続行します。
- クラスタ内の管理対象サーバのうちいずれか 1 つが停止すると、Application Integration リソースはデプロイもアンデプロイもできません。たとえば、クラスタに 1 つでも停止したサーバがあると、Application Integration アダプタはデプロイできません。
- クラスタ環境では、フェイルオーバおよび再試行の試みにより、B2B メッセージが重複して送信される可能性があります。そのような場合は、軽微な例外として、重複メッセージ例外がログに書き込まれ、202/200 HTTP ステータスが返されます。それらが受信されると、重複したメッセージは配信されないで、ワークフロー レイヤまたはアプリケーション レイヤに配信されます。
- WebLogic Integration とデータベースが同じマシン上で実行されていて、そのマシンのプラグが抜かれた場合は、WebLogic Integration の回復を試みる前に、データベース回復手順を実行してください。データベースは別のマシン上にデプロイするのが理想的です。
- ebXML および RosettaNet ビジネス プロトコルに対しては高可用性がサポートされています（これらのビジネス プロトコルに基づくアプリケーションには回復機能がある）。

  WebLogic Integration は ebXML Message Service Specification v1.0 および RosettaNet Implementation Framework v1.1、v2.0 をサポートします。

  WebLogic Integration アプリケーションに以前のバージョンの WebLogic Integration で開発された RosettaNet ワークフローが含まれている場合は、

WebLogic Integration 7.0 でアプリケーションを実行する前に、それらのワークフローを変更する必要があります。ワークフローの移行に関する詳細については、『*WebLogic Integration 移行ガイド*』の「WebLogic Integration 2.1 から WebLogic Integration 7.0 への移行」を参照してください。

- ワークフローのインスタンスの処理中に、WebLogic Integration が処理に失敗すると、そのワークフローはロール バックされ、回復した時点で、最後に静止た点から再起動します。
- WebLogic Integration のあるインスタンスから別のインスタンスにメッセージが送信されたが、送り先のインスタンスに障害が発生した場合、サーバ コンソールに 1 つまたは複数のエラー メッセージが表示され、その後にスタック トレースが表示される場合があります。次の例で、表示されるエラー メッセージの種類を示します。
  - [Not able to send RosettaNet Message]
  - [Peer Gone Exception]
- WebLogic Integration の単独ノードへのデプロイメントおよびクラスタ デプロイメントの両方に対して、自動再起動と回復がポートされています。移行は、クラスタ デプロイメントに対してのみサポートされています。
- WebLogic Integration Business Partner (ミッドウェイトのトレーディング パートナ) の単一ノード デプロイメントに対しては、自動再起動と回復がサポートされています。ただし、クラスタ環境でミッドウェイト トレーディング パートナをデプロイすることはできません。WebLogic Integration ビジネス パートナの障害発生時に自動再起動が行われるためのデプロイメント コンフィグレーションに関する詳細は、4-5 ページの「自動再起動のための WebLogic Integration のコンフィグレーション」を参照してください。

自動再起動のための WebLogic Integration のコンフィグレーション

クラスタ環境に WebLogic Integration がデプロイされているかどうかにかかわらず、システム クラッシュ、ハードウェアの再起動、サーバの不具合などが原因でシャットダウンしたサーバを自動的に再起動するよう、システムをコンフィグレーションすることができます。

注意： この節の手順はクラスタ環境を対象としていますが、同じ手順で、非クラスタ環境、すなわち、管理サーバと管理対象サーバを 1 つずつデプロイする環境をコンフィグレーションすることも可能です。

Node Manager

この節の手順では、管理対象サーバが配置されているマシン上で Node Manager が実行されている場合に、管理対象サーバを起動するシステムをコンフィグレーションする方法について説明します。Node Manager は、WebLogic Server と同梱の Java プログラムで、管理対象サーバに対して、以下のタスクを実行します。

- ドメイン内のリモートの管理対象サーバの起動と停止
- システム クラッシュ、ハードウェアの再起度、サーバ障害などでシャットダウンした WebLogic Server インスタンスの自動再起動
- WebLogic Server インスタンスの状態の自動モニタおよび*異常*状態に達したインスタンスの再起動

Node Manager に関する詳細については、次の URL にある『*WebLogic Server ドメイン管理*』、「ノード マネージャによるサーバの可用性の管理」を参照してください。

```
http://edocs.beasys.co.jp/e-docs/wls/docs70/admin_domain/nodemgr.html
```

次の手順を実行して、自動起動するように、WebLogic Integration クラスタをコンフィグレーションします。

- 手順 1. リモート起動するように、管理対象サーバをコンフィグレーションする
- 手順 2. 管理サーバに対して SSL をコンフィグレーションする
- 手順 3. Node Manager をコンフィグレーションする
- 手順 4. 自己状態モニタ機能をコンフィグレーションする
- 手順 5. Node Manager を起動する

手順 1. リモート起動するように、管理対象サーバをコンフィグレーションする

まず、クラスタ内の各管理対象サーバを、リモート サーバから起動できるようにコンフィグレーションする必要があります。

管理対象サーバをリモート起動できるようにコンフィグレーションには、次の手順を実行します。

1. WebLogic Server Administration Console のナビゲーション ツリーで、自動起動をコンフィグレーションする管理対象サーバを選択します。
2. [コンフィグレーション] タブ、[リモート スタート] タブの順に選択します。
3. [リモート スタート] タブに表示されるフィールドに情報を入力します。必要な情報はリモート サーバ固有の情報です。このタブにあるフィールドは、次の URL にある『*Adminstration Console オンライン ヘルプ*』の「[サーバ] → [コンフィグレーション] → [リモート スタート]」で説明されています。

   ```
   http://edocs.beasys.co.jp/e-docs/wls/docs70/ConsoleHelp/domain_server_config_server-start.html
   ```

手順 2. 管理サーバに対して SSL をコンフィグレーションする

管理サーバは、SSL を使用して Node Manager と通信するため、管理サーバに対して SSL をコンフィグレーションする必要があります。次の手順を実行してください。

1. 次の行を管理サーバの `startWeblogic` コマンド ファイルに追加します。

   ```
   -Dweblogic.security.SSL.trustedCAKeyStore=WL_HOME\lib\cacerts
   ```

   このコマンドラインで *`WL_HOME`* は、WebLogic Server がインストールされたディレクトリを表します。たとえば、WebLogic Platform をデフォルト ディレクトリにインストールした場合は、*`WL_HOME`* は `C:\bea70\weblogic700\server` です。

2. democert.pem、demokey.pem、および ca.pem ファイルを BEA_HOME\weblogic700\common\templates\domains\wls.jar からドメインのルート ディレクトリに追加します。
3. WebLogic Server Administration Console のナビゲーション ツリーで、管理サーバを選択します。
4. [接続] タブ、[SSL] タブの順に選択します。
5. [SSL] タブにある各フィールドに、次の URL にある『*Adminstration Console オンライン ヘルプ*』の「[サーバ] → [接続] → [SSL]」の説明に従って情報を入力します。

   http://edocs.beasys.co.jp/e-docs/wls/docs70/ConsoleHelp/domain_server_connections_ssl.html

   democert.pem、demokey.pem および ca.pem の各ファイルは、手順 2 でドメインのルート ディレクトリにコピーしたサンプル ファイルで、初めて操作で使用できます。それらのファイルは、Administration Console で SSL のコンフィグレーションを行う際、以下のフィールドで使用できます。[サーバ証明書ファイル名]、[サーバ キーのファイル名]、[信頼性のある CA ファイル名]

手順 3. Node Manager をコンフィグレーションする

Node Manager を管理対象サーバに対してコンフィグレーションするには、WebLogic Server Administration Console を使用して、マシン作成、そのマシン上での Node Manager に対する属性の指定、そのマシンでリモート起動できるようにコンフィグレーションした管理対象サーバのデプロイなどを行う必要があります。具体的には、以下の手順を実行する必要があります。

1. Administration Console のナビゲーション ツリーで [マシン] を選択します。

   [マシン] が右ペインに表示され、ドメインで定義されているすべてのマシンがリストされています。

2. [新しい Machine のコンフィグレーション] (UNIX マシンをコンフィグレーションしている場合は [新しい UnixMachine のコンフィグレーション]) をクリックします。

ダイアログ ボックスが右ペインに表示され、新しいマシンのコンフィグレーションに関連付しているタブがリストされています。

3. [名前] 属性フィールドに新しいマシンの名前を入力し、[作成] をクリックして、指定した名前のマシン インスタンスを作成します。
4. [ノード マネージャ] タブで、Node Manager の接続属性と認証属性（Node Manager が接続をリスンするアドレスとポート）を定義します。*address:port* のデフォルト値は `localhost:5555` です。[適用] をクリックして変更を実装します。
5. [サーバ] タブで、このマシンに配置される管理対象サーバ（4-7 ページの「手順 1. リモート起動するように、管理対象サーバをコンフィグレーションする」でリモート起動できるようにコンフィグレーションした管理対象サーバ）を特定します。

   [Available] カラムからサーバ名を選択し、適切な矢印をクリックしてそのサーバを [Chosen] カラムに移動することにより、このマシンに割り当てるサーバを既存サーバから選択することもできます。
6. [適用] をクリックして変更を実装します。

   これで、新しいマシン エントリによって、このマシン上で実行されている Node Manager との接続、およびこのマシンに配置されている WebLogic Server インスタンスの特定という、2 つの目的に必要な属性が指定されました。

手順 4. 自己状態モニタ機能をコンフィグレーションする

この手順では、管理対象サーバの自動状態チェック、および Node Manager によるサーバ状態のチェック頻度をコンフィグレーションする方法について説明します。また、サーバが*異常*状態に達した場合、Node Manager が自動的にサーバを停止および起動するかどうかも指定できます。

各管理対象サーバについて、次の手順を実行します。

1. WebLogic Server Administration Console のナビゲーション ツリーで、4-7 ページの「手順 1. リモート起動するように、管理対象サーバをコンフィグレーションする」で自動起動をコンフィグレーションした管理対象サーバを選択します。
2. [コンフィグレーション] タブ、[状態モニタ] タブの順に選択します。
3. 次の情報を入力します。
   - [自動再起動] ― Node Manager による、この管理対象サーバの自動再起動を有効にするには、[自動再起動] を選択します。
   - [失敗時の自動強制停止] ― Node Manger による、故障サーバの自動停止を有効にするには、[失敗時の自動強制停止] を選択します。
   - [再起動間隔] ― Node Manager が再起動の実行に使用できる秒数がカウントされます。この属性は、Max Restarts within Interval 属性と共に、このサーバを再起動する試みを制限するために使用されます。デフォルト値は 300 秒です。
   - [期間内の最大起動回数] ― [再起動間隔] で指定された時間以内で Node Manger がこのサーバを再起動できる回数の最大値。デフォルト値は 2 回です。
   - [状態チェック間隔] ―サーバの状態チェック行う頻度（秒数）このパラメータは、サーバの自己状態モニタ機能、および Node Manager の状態照会の頻度をコントロールします。
   - [状態チェック タイムアウト] ― サーバに対する状態クエリがタイムアウトになるまでに Node Manager が待機する時間（秒数）
   - [再開始遅延] ― Node Manager がサーバの再起動までに待機すべき時間（秒数）この値は、たとえば、OS がリスン ポートをただちに再利用することを認めない場合に使用されます。

手順 5. Node Manager を起動する

Node Manager は、手動起動、OS プロンプトからの `java` コマンドの実行、自動起動、またはスクリプトの実行によって起動できます。

Node Manager 起動コマンドの構文

Node Manager を起動する java コマンドの構文は次のとおりです。

```
java [java_property=value ...] -D[nodemanager_property=value]
-D[server_property=value] weblogic.nodemanager.NodeManager
```

警告 : Node Manager は、管理対象サーバを手動で起動する場合と同じディレクトリから起動する必要があります。

上の java コマンドラインの説明

- java_property ― java 実行可能ファイルへの直接の引数、-ms または -mx. を指定します。

  注意 : メモリ不足に陥らないように、常に、Node Manger のヒープ サイズには最小限の 32MB (-Xms32m) を指定してください。

- nodemanager_property ― Node Manager プロセスの動作を定義します。表 4-1 に、有効な Node Manager プロパティを示します。

表 4-1 nodemanager_property コマンドライン引数の値

Node Manager プロパティ	説明	デフォルト値
weblogic.nodemanager.certificateFile	SSL 認証に使用される証明書ファイルへのパスを指定する。	./config/democert.pem
weblogic.nodemanager.javaHome	このマシン上の管理対象サーバを起動するために Node Manager が使用する Java のホーム ディレクトリを指定する。	なし
weblogic.nodemanager.keyFile	Administration Server との SSL 通信に使用されるプライベート キー ファイルへのパス。	./config/demokey.pem
weblogic.nodemanager.keyPassword	キー ファイルの暗号化プライベート キーにアクセスするために使用されるパスワード。	password

表 4-1 nodemanager_property コマンドライン引数の値 (続き)

Node Manager プロパティ	説明	デフォルト値
weblogic.ListenAddress	Node Manager が接続要求をリスンするアドレス。この引数により、weblogic.nodemanager.listenAddress は非推奨になる。	localhost
weblogic.ListenPort	Node Manager が接続要求数をリスンする TCP ポートの番号。この引数により、weblogic.nodemanager.listenPort は非推奨になる。	5555
weblogic.nodemanager.nativeVersionEnabled	Solaris、HP-UX 以外の UNIX システムでは、Node Manager を非ネイティブ モードで実行するために、このプロパティは false に設定する。	true
weblogic.nodemanager.reverseDnsEnabled	信頼されているホストのファイルへのエントリに DNS 名（IP アドレスではなく）を含めるかどうか指定する。	false
weblogic.nodemanager.savedLogsDirectory	Node Manager がログ ファイルを格納するディレクトリへのパスを指定する。Node Manager は、savedLogsDirectory に、NodeManagerLogs という名前のサブディレクトリを作成する。	./NodeManagerLogs
weblogic.nodemanager.sslHostNameVerificationEnabled	Node Manager がホスト名検証を実行するかどうか決定する。	false
weblogic.nodemanager.startTemplate	UNIX システムに限り、このプロパティは、管理対象サーバの起動に使用するスクリプト ファイルへのパスを指定する。	./nodemanager.sh
weblogic.nodemanager.trustedHosts	Node Manager が使用する、信頼されているホストのファイルへのパスを指定する。	./nodemanager.hosts

表 4-1 nodemanager_property コマンドライン引数の値 (続き)

Node Manager プロパティ	説明	デフォルト値
weblogic.nodemanager.weblogicHome	WebLogic Server インストールのホスト ディレクトリを指定する。このディレクトリ名は、コンフィグレーション済みのルート ディレクトリのないサーバの、-Dweblogic.RootDirectory のデフォルト値として使用される。	なし

- server_property ―新しい管理対象サーバ インスタンスの起動時にデフォルト値を指定します。表 4-2 に有効なサーバ プロパティを示します。

表 4-2 server_property コマンドライン引数の値

サーバ プロパティ	説明	デフォルト値
bea.home	現在のマシン上の管理対象サーバが使用する BEA ホーム ディレクトリを指定する。	現在のマシン上の管理対象サーバが使用する BEA ホーム ディレクトリを指定する。
java.security.policy	管理対象サーバが使用するセキュリティ ポリシー ファイルへのパスを指定する。	none
weblogic.security.SSL.trustedCAKeyStore	信頼性のある認証局の証明書が格納されている KeyStore へのパスを指定する。	java.security.keyStore

上の表にある情報および Node Manager のコンフィグレーションと実行に関する詳細は、次の URL にある、『*WebLogic Server ドメイン管理*』、「ノード マネージャによるサーバ可用性の管理」 の 「ノード マネージャの起動」 を参照してください。

http://edocs.beasys.co.jp/e-docs/wls/docs70/admin_domain/nodemgr.html

マシンの起動時に Node Manager を起動する

プロダクション環境では、Node Manager は、マシンの起動時に自動的に起動する必要があります。そのような起動方法は、UNIX システム用のスタートアップスクリプトを記述することによって、または Node Manager を Windows システム用の Windows サービスとし設定することによって、保証することができます。これらのタスクの実行方法の詳細については、次の URL にある『*WebLogic Server ドメイン管理*』、「ノード マネージャによるサーバ可用性の管理」を参照してください。

```
http://edocs.beasys.co.jp/e-docs/wls/docs70/admin_domain/nodemgr.html
```

故障ノードから健全なノードに移行するための WebLogic Integration のコンフィグレーション

管理対象サーバに障害が発生して、使用不可能とみなされた場合、サービスを障害の発生した管理対象サーバから、そのクラスタ内の健全なノードに移行することができます。システムを手動移行できるようにコンフィグレーションするには、次の手順を実行します。

- 手順 1. クラスタをコンフィグレーションする
- 手順 2. JMS サーバと JTA 回復サービスに対する移行可能ターゲットをコンフィグレーションする

クラスタ内であるノードに障害が発生した場合の移行方法の説明は、4-20 ページの「故障ノードから健全なノードへの WebLogic Integration の手動移行」を参照してください。

手順 1. クラスタをコンフィグレーションする

WebLogic Integration リソースが適切に分散されていて、クラスタ ドメインが第 3 章「クラスタ デプロイメントのコンフィグレーション」で説明したとおりにコンフィグレーションされていることを確認します。

手順 2. JMS サーバと JTA 回復サービスに対する移行可能ターゲットをコンフィグレーションする

WebLogic Integration デプロイメントの高可用性を達成するには、フェイルオーバ用の JTA サーバと JMS サーバをコンフィグレーションする必要があります。このプロセスでは、JMS サーバおよび JTS 回復サービスに対する移行可能ターゲットのコンフィグレーションも必要です。このタスクは、WebLogic Server Administration Console を使用することで、または `config.xml` ファイルを適切に編集することで実行できます。

次の手順を実行してください。

1. クラスタに対して移行可能ターゲットを作成します。
   a. Administration Console のナビゲーション ツリーで Serves ノードを選択します。
   b. コンフィグレーションするクラスタ内に配置されているサーバの名前を選択します。
   c. メイン コンソール ウィンドウで、[Control ｜ Migration Config] を選択します。移行可能ターゲットとして、制約付きサーバ候補の選択に使用できるサーバのリストが表示されます。
   d. [有効] カラムで、クラスタ内の移行可能サービスをホスティングできるすべてのサーバを選択します。矢印を使用してこれらのサーバを [選択した項目] カラムに移動します。

      注意 : 通常は、移行可能サービスの潜在的ホストとして、クラスタ内のすべての管理対象サーバが選択されます。

e. [適用] をクリックして、移行可能ターゲットに対する変更を有効にします。

指定したサーバのリストは、`MigratableTarget` 要素でコンフィグレーションされます。コード リスト 4-1 で、`MigratableTarget` 要素の `ConstrainedCandidateServers` 属性を参照してください（ドメイン コンフィグレーションには、管理対象サーバごとに `MigratableTarget` 要素が 1 つずつある）。

2. JTA フェイルオーバをコンフィグレーションします。

a. クラスタ内のすべてのサーバが、サーバのトランザクション ログ ファイルにアクセスできることを確認します。言い換えれば、JTA ログ ファイルは、共用ファイル システムに配置されている必要があります。

b. Administration Console のナビゲーション ツリーで Servers ノードを選択します。

c. コンフィグレーションするクラスタ内に配置されているサーバの名前を選択します。

d. [Control ｜ JTA Migration Config] を選択して、JTA サービスに対して移行可能ターゲットを作成します。移行可能ターゲットとして、制約付きサーバ候補の選択に使用できるサーバのリストが表示されます。

e. [有効] カラムで、クラスタ内の移行可能サービスをホスティングできるすべてのサーバを選択します。矢印を使用してこれらのサーバを [選択した項目] カラムに移動します。

f. [適用] をクリックして、新しい移行可能ターゲットに対する変更を有効にします。

注意： JTA および JMS サービスの移行は、2 ステップのプロセスです。WebLogic Integration リソースを移行するときは、まず、JTA サービスを移行してから、JMS サービスを移行することをお勧めします。詳細については、4-20 ページの「故障ノードから健全なノードへの WebLogic Integration の手動移行」 を参照してください。

移行可能ターゲットのコンフィグレーションに関する詳細については、以下を参照してください。

- 次の URL にある、『*Adminstration Console オンライン ヘルプ*』、「JTA」 の「トランザクション回復サービスの移行先になるサーバの制限」

http://edocs.beasys.co.jp/e-docs/wls/docs70/ConsoleHelp/jta.html

- 次の URL にある 『*Adminstration Console オンライン ヘルプ*』、「サーバ」の「サーバの移行タスク」

  http://edocs.beasys.co.jp/e-docs/wls/docs70/ConsoleHelp/servers.html

  注意： オンラインヘルプは、Administration Console からアクセスできます。また、次の URL にもあります。

  http://edocs.beasys.co.jp/e-docs/wls/docs70/ConsoleHelp/index.html

次のリストは、サンプルの config.xml ファイルからの抜粋で、移行可能ターゲットのコンフィグレーション方法を示しています。このリストでは、クラスタ化された WebLogic Integration 環境における JMS サーバと JTA 回復サービスの両方に対する移行ターゲットのコンフィグレーションが例示されています。このコンフィグレーション例では、クラスタに、MyServer-1 と MyServer-2 という 2 つの管理対象サーバが配置されています。

コード リスト 4-1 移行可能ターゲットのコンフィグレーション

```
<JMSServer Name="WLCJMSServer-MyServer-1"
    Store="JMSWLCStore-MyServer-2" Targets="MyServer-1 (migratable)"
   TemporaryTemplate="TemporaryTemplate">
   <JMSQueue JNDIName="com.bea.b2b.OutboundQueue-MyServer-1"
      Name="B2bOutboundQueue-MyServer-2"/>
    <JMSQueue ...
     :
</JMSServer>

<JMSServer Name="WLCJMSServer-MyServer-2"
    Store="JMSWLCStore-MyServer-2" Targets="MyServer-2 (migratable)"
   TemporaryTemplate="TemporaryTemplate">
   <JMSQueue JNDIName="com.bea.b2b.OutboundQueue-MyServer-2"
      Name="B2bOutboundQueue-MyServer-2"/>
    <JMSQueue ...
     :
</JMSServer>
...

<MigratableTarget Cluster="MyCluster"
    ConstrainedCandidateServers="MyServer-1,MyServer-2"
```

```
     Name="MyServer-1 (migratable)"
    注 =" システム生成による、サーバに対するデフォルトの移行可能ターゲット。
    手動で削除しないこと。"
    UserPreferredServer="MyServer-1"/>

<MigratableTarget Cluster="MyCluster"
     ConstrainedCandidateServers="MyServer-1,MyServer-2"
     Name="MyServer-2 (migratable)"
    注 =" システム生成による、サーバに対するデフォルトの移行可能ターゲット。
    手動で削除しないこと。"
    UserPreferredServer="MyServer-2"/>

...

<Server Cluster="MyCluster" JTARecoveryService="MyServer-1"
     ListenAddress="localhost" ListenPort="7901" Name="MyServer-1"
     ServerVersion="7.0.0.0">
     <COM Name="MyServer-1"/><ExecuteQueue Name="default" ThreadCount="15"/>
     <IIOP Name="MyServer-1"/>
    <JTAMigratableTarget Cluster="MyCluster"
     ConstrainedCandidateServers="MyServer-1,MyServer-2 Name="MyServer-1"
     UserPreferredServer="MyServer-1"/>

</Server>

<Server Cluster="MyCluster" JTARecoveryService="MyServer-2"
     ListenAddress="localhost" ListenPort="7901" Name="MyServer-2"
     ServerVersion="7.0.0.0">
     <COM Name="MyServer-2"/><ExecuteQueue Name="default" ThreadCount="15"/>
     <IIOP Name="MyServer-2"/>
    <JTAMigratableTarget Cluster="MyCluster"
     ConstrainedCandidateServers="MyServer-1,MyServer-2 Name="MyServer-2"
     UserPreferredServer="MyServer-2"/>

</Server>
```

このリストの以下の XML 要素に注意してください。

- JMS サーバー JMS サーバの Target 属性は、そのサーバに対する移行可能ターゲットの名前です。移行可能ターゲットは、デフォルトでは、各管理対象サーバに対して作成されます（詳細は、このリストの次の項目を参照）。
- MigratableTarget ー `MyCluster` の `MyServer-2` に対する移行可能ターゲットの指定です。リストで記述されているように、移行可能ターゲットは、サーバに対してシステムによって生成されるデフォルトの移行可能ターゲットで

す。クラスタ内の各管理対象サーバに対して、そのようなターゲットが 1 ずつ作成されます。

また、`MigratableTarget` 要素には、`ConstrainedCandidateServers` に対する、サーバのカンマ区切りリストも格納されています。このリストにあるサーバは、JMS サーバのバックアップとして機能する能力があるとして指定されたものです。`ConstrainedCandidateServers` のリストには、`UserPreferredServer` を含める必要があります。WebLogic Server Administration Console では、この規則を義務付けています。

- Server ― Server 要素には `JTARecoveryService.` に対する移行可能ターゲットの指定を含める必要があります。

フェイルオーバと回復

この節では、具体的なシナリオにおいて、WebLogic Integration のフェイルオーバと回復機能がどのように動作するか説明します。内容は以下のとおりです。

- Administration Server に対するバックアップとフェイルオーバ
- 故障ノードから健全なノードへの WebLogic Integration の手動移行
- データベースの回復
- JMS ストアの回復

Administration Server に対するバックアップとフェイルオーバ

管理サーバのクラッシュやその他の障害が発生した場合に迅速なフェイルオーバを実現するために、オリジナルのサーバに障害が発生した場合にただちに使用できる別のマシンに、管理サーバのインスタンスをもう 1 つ作成することもできます。

管理サーバは、コンフィグレーション ファイル（config.xml）、セキュリティ ファイル、アプリケーション ファイルを使用してドメインを運営するため、少なくとも、これらのファイルのコピーを保管しておくことことをお勧めします。そうすることで、管理サーバに障害が発生した場合も、管理対象サーバの機能を中断することなく、別のマシンで管理サーバを安全に再起動できます。

クラスタの管理サーバがクラッシュしても、管理対象サーバは要求の処理を続行します。ただし、管理サーバが回復するまでは、クラスタのコンフィグレーションを変更することはできません。また、新しいデプロイメント活動もできません。たとえば、あるクラスタの管理サーバが実行されていない場合は、新しいノードのクラスタへの追加、新しいアプリケーション ビューのデプロイメント、アプリケーション ビューに関連付けられた接続ファクトリのアンデプロイ、などを実行することはできません。

The WebLogic Integration B2B Console は、管理サーバにのみデプロイすることができます。クラスタ内の管理対象サーバにデプロイされることはありません。したがって、B2B Integration の管理およびモニタ機能は、管理サーバが停止している間は利用できません。たとえば、トレーディング パートナの追加、削除、変更は、管理サーバが回復するまで実行できません。

管理対象サーバが実行されていても管理サーバが停止している場合は、ドメインの管理は、管理対象サーバの停止や再起動を行うことなく回復できます。

管理対象サーバが実行されている場合の管理サーバの再起動に関する手順の説明は、次の URL にある『*WebLogic Server 管理者ガイド*』の「WebLogic Server の起動と停止」を参照してください。

```
http://edocs.beasys.co.jp/e-docs/wls/docs70/adminguide/startstop.
html
```

故障ノードから健全なノードへの WebLogic Integration の手動移行

この節では、コントロールされたフェイルオーバについて説明します。これは、故障ノードからクラスタ内の健全なノードにサービスを移行している間、ソースと送り先のサーバが要求を処理していない状態を指します。

WebLogic Integration を故障ノードから健全なノードに移行する準備

- アプリケーションのコンフィグレーションを、4-14 ページの「故障ノードから健全なノードに移行するための WebLogic Integration のコンフィグレーション」の説明に従って（クラスタを起動する前に）完了しておきます。
- ソース サーバが実行されていないことを確認してください。ソース サーバが停止していないが、ネットワーク上の問題により使用できない場合は、サービスは、ソース サーバから削除することなく、送り先のサーバにコピーされます。よって、同じサービスが同時に 2 つ実行されることになり、その結果、トランザクション ログや JMS メッセージの損傷を招く場合があります。

注意： JTA および JMS サービスの移行は、2 ステップのプロセスです。WebLogic Integration リソースを移行するときは、まず、JTA サービスを移行してから、JMS サービスを移行してください。

WebLogic Integration は以下のいずれかの方法で移行できます。

- weblogic.Admin コマンドライン ユーティリティを使用する方法
- WebLogic Server Administration Console を使用する方法

weblogic.Admin コマンドライン ユーティリティを使用する方法

次のコマンドライン（`MIGRATE` コマンドで `weblogic.Admin` を呼び出す）を使用して、JMS サービスまたは JTA サービスをクラスタ内のターゲット サーバに移行します。

```
java weblogic.Admin [-url http://hostname:port]
[-username username]
[-password password]
MIGRATE -jta -migratabletarget (migratabletarget_name|servername)
-destination servername [-sourcedown] [-destinationdown]
```

上のコマンドラインの説明

- `-url` ― 管理サーバの URL を、WebLogic Server がクライアントの要求をリスンする TCP ポートの番号を含めて指定します。フォーマットは *`hostname:port`* です。デフォルトは `localhost:7001` です。
- `-jta` ― JTA サービスの移行であることを指定します。`-jta` が指定されていない場合は、移行は、JMS サービスの移行であるとみなされます。

- -migratabletarget ー サービスの移行元のサーバのコンフィグレーション ファイルを指名します。各サーバについて、WebLogic Server によって自動的に移行可能ターゲットのファイルが作成されます（ファイルの名前は、JMS の場合は *servername*_migratable、JTA の場合は servername）。この移行可能ターゲット ファイルは、JMS および JTA サービスに対する優先サーバを指定するコンフィグレーション ファイルです。
- -destination ー サービスの移行先のサーバの名前です。
- -sourcedown ー ソース サーバが停止していることを明示します。

  警告： この節で前に述べたとおり、MIGRATE コマンドで weblogic.Admin を呼び出すときに、ソース サーバが停止していることが大切です。
- -destinationdown ー このオプションは、移行時に送り先サーバが停止している場合に使用します。

weblogic.Admin コマンドライン ツールの詳細については、次の URL にある『*WebLogic Server 管理者ガイド*』の「WebLogic Server コマンドライン インタフェース リファレンス」を参照してください。

http://edocs.beasys.co.jp/e-docs//wls/docs70/adminguide/cli.html

WebLogic Server Administration Console を使用する方法

weblogic.Admin コマンドライン ツールを使用する代わりに、WebLogic Server Administration Console を使用して、JTS または JMS サービスをクラスタ内のターゲット サーバに移行することができます。

1. Administration Console のナビゲーション ツリーで Servers ノードを選択します。
2. クラスタ内のサーバの名前を選択します。
3. 移行するサービスに合った [移行] タブを選択します。JTA および JMS サービスの移行は、2 ステップのプロセスです。WebLogic Integration リソースを移行するときは、まず、JTA サービスを移行してから、JMS サービスを移行してください。どの [移行] タブを選択するかは、移行するサービスの種類によって決まります。
   - JTA サービスを移行する場合は、[Control] タブを選択してから [JTA 移行] タブを選択します。

- JMS サービスを移行する場合は、[Control] タブを選択してから [移行] タブを選択します。

警告 : ソース サーバが停止していることを確認してください。WebLogic Integration に対するフェイルオーバは、ソース サーバが停止している場合にのみサポートされます。

4. [Destination Server migratable target list] からサーバを選択します。
5. [移行] をクリックします。

   手順 2 で選択したサーバで実行されていたサービスは、選択した送り先サーバに移行します。

   移行するサービスは、「手順 3.」での選択によって決まります。[JTA 移行] タブを選択した場合は、JTA サービスのみが、選択したサーバに移行します。[移行] タブを選択した場合は、JMS サービスが選択したサーバに移行します。

Administration Console を使用して、クラスタ内のターゲット サーバに JMS、JTA サービスを移行する方法の詳細については、『*Adminstration Console オンライン ヘルプ*』、「サーバ」の「サーバの移行タスク」を参照してください。

データベースの回復

WebLogic Integration は、クラシュしたデータベースの回復は試みません。データベースがクラッシュしたり停止した場合は、WebLogic Integration を再起動する必要があります。

たとえば、WebLogic Integration とデータベースが同じマシン上で実行されていて、そのマシンのプラグが抜けた場合は、WebLogic Integration の回復を試みる前に、データベース回復手順を実行してください。

JMS ストアの回復

サーバがクラッシュした後は、JMS ストアの移行は不可能です。WebLogic Integration は、JMS ストアに代えて JDBC を使用します。つまり、JDBC を使用して、他のサーバ上に配置されている JMS JDBC ストアにアクセスするのです。WebLogic Integration では、クラスタ内のすべてのノードで同一のデータベース

が使用されています。クラスタ内のノードごとに別々のデータベース インスタンスを使用する場合は、データベース ベンダが提供する高可用性やフェイルオーバ ソリューションのあらゆる利点を活用してください。たとえば、データベース クラッシュに備えて、データベースのウォーム スタンバイを利用できる場合もあります。

5 WebLogic Integration セキュリティの使い方

以下の節では、WebLogic Integration ソリューションのデプロイメントのセキュリティを設定および管理する方法について説明します。

- WebLogic Integration セキュリティの概要
- セキュリティ コンフィグレーションの考慮事項
- セキュアなデプロイメントの設定

このトピックを読み進む前に、次の URL にある『*WebLogic Security の紹介*』を参照してください。

http://edocs.beasys.co.jp/e-docs/platform/docs70/secintro/index.html

このマニュアルでは、WebLogic Platform 全体のセキュリティ機能の概要を紹介し、WebLogic Integration を他の WebLogic Platform コンポーネントと併用した場合のセキュリティ管理上の重要な注意点を示します。

WebLogic Integration セキュリティの概要

WebLogic Integration ソリューションのセキュア デプロイメントの基盤は、WebLogic Server が提供する一群のセキュリティ機能です。したがって、ご使用の環境の基礎である WebLogic Server レイヤのセキュリティをコンフィグレーションした後は、以下に示す、WebLogic Integration に固有の WebLogic Server エンティティのセキュリティをコンフィグレーションし、管理する必要があります。

- WebLogic Integration システム ユーザ、`wlisystem`
- BPM エンジンおよび WebLogis Integration Studio のユーザ、それぞれが所属するグループ

- ビジネス メッセージの送受信のため、セキュアな環境でデジタル証明書を作成しなければないため、セキュリティ管理が特に重要なトレーディング パートナ。
- アプリケーション ビュー

セキュリティ マネージャは、WebLogic Integration ドメインの作成時に一緒に作成された定義済みプリンシパルとリソースのセットに集中的に取り組む必要があります。

この概要紹介では、以下のトピックを通じて、WebLogic Integration セキュリティの十分な理解を図ります。

- セキュリティと WebLogic Integration ドメイン
- WebLogic Integration で使用される WebLogic Server のセキュリティ プリンシパルおよびリソース

注意： セキュアなデプロイメントでは、セキュリティが提供されないアプリケーションと同じ WebLogic Server インスタンスで WebLogic Integration を実行しないでください。内部 WebLogic Integration API 呼び出しは、同じインスタンスのアプリケーションから保護されません。

セキュリティと WebLogic Integration ドメイン

BEA コンフィグレーション ウィザードを使用して WebLogic Integration ドメインを作成すると、そのドメインは、デフォルトでは、互換性セキュリティを使用するようにコンフィグレーションされます。互換性セキュリティにより、ドメインでは次のことが可能になります。

- 既存のユーザおよびグループのストアを使用して WebLogic Server プリンシパルを認証する。
- アクセス制御リスト（ACL）を使用して WebLogic Integration リソースを保護する。

デフォルトでは、すべての WebLogic Integration ユーザ、グループ、および ACL は、互換性レルムと呼ばれるセキュリティ レルムに格納されます。

注意： WebLogic Integration の標準的なインストールには、WebLogic Server および WebLogic Workshop の両コンポーネントが含まれています。デフォルトでは、コンフィグレーション ウィザード、WebLogic Integration セ

キュリティをコンフィグレーションして、互換性セキュリティを使用できるようにし、ユーザやグループの格納には WebLogic Server 6.x File レルムを割り当てます。File レルムは、すべての WebLogic Integration サンプルで使用されます。WebLogic Server サンプルおよび WebLogic Workshop サンプルは、組み込み LDAP サーバをベースとしたセキュリティ コンフィグレーションに基づいていますが、今回のリリースの WebLogic Integration ではそれはサポートしていません。したがって、WebLogic Server および WebLogic Server や WebLogic WorkShop と同梱で出荷されたサンプルは、WebLogic Integration サンプルのデフォルトのコンフィグレーションとは連携動作しない場合があります。

コンフィグレーション ウィザードの詳細については、『*Configuration Wizard の使い方*』を参照してください。

http://edocs.beasys.co.jp/e-docs/platform/docs70/confgwiz/index.html

WebLogic Integration で使用される WebLogic Server のセキュリティ プリンシパルおよびリソース

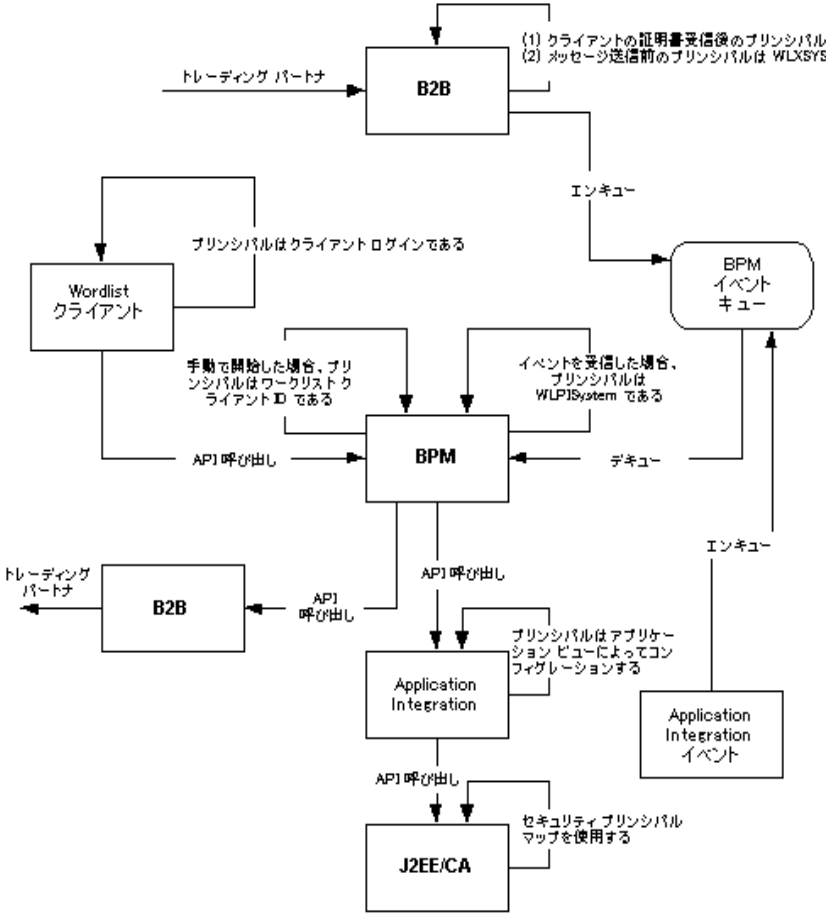
Configuration Wizard で WebLogic Integration ドメインを作成する場合、次の WebLogic Server プリンシパルおよびリソースが事前定義されています。

- `wlisystem` — WebLogic Integration システム ユーザで、トレーディング パートナが認証されて、WebLogic Integration リソースを使用する権限を与えられた後、トレーディング パートナに代わって動作します。
- `wlpiUsers` — Business Process Management ユーザ グループで、WebLogic Integration Studio や BPM エンジンなどの BPM リソースにアクセスします。
- BPM グループのセット — `wlpiUsers` グループの各メンバーは、1 つまたは複数の BPM グループに所属しています。加入するグループによって、プリンシパルによって実行できるタスクのセットが決まります。グループごとに ACL が定義されています。
- `wliPool` — WebLogic Integration リポジトリ用の JDBC 接続プール。

- `ServletFilter` ―トレーディング パートナが B2B リソースにアクセスするときに透過的に使用されるリソース。

次の図は、WebLogic Integration で使用される WebLogic Server のセキュリティ プリンシパルの概要を示しています。

図 5-1 WebLogic Integration で使用される WebLogic Server のセキュリティ プリンシパル



たとえば、B2B ベースのビジネス オペレーションの実行中、WebLogic Server プリンシパルは、以下のような機能を発揮します。

- イベント（XML、Application Integration、または B2B Integration）を介して開始された場合、ワークフローは `wlisystem` として動作します。
- 手動で開始された場合、ワークフローは Worklist クライアントに関連付けられたセキュリティ プリンシパルとして動作します。
- アプリケーション統合サービスは、アプリケーション ビューに対してコンフィグレーションされたセキュリティ プリンシパルとして動作します。
- トレーディング パートナからメッセージを受信すると、B2B エンジンは、トレーディング パートナのクライアント サイドの証明書に関連付けられている WebLogic Server プリンシパルに直ちに切り替わります。その後、`wlisystem` プリンシパルに切り替わってから、メッセージを BPM イベント キューに送信します。B2B Integration のセキュリティの詳細については、『*B2B Integration セキュリティの実装*』を参照してください。

  J2EE-CA アダプタは、リクエストを受け取ると、呼び出し元のセキュリティ プリンシパルを EIS システムで対応するプリンシパルにマップします。詳細については、次の URL にある WebLogic Server マニュアルの『*WebLogic J2EE コネクタ アーキテクチャ*』の「セキュリティ プリンシパル マップ」を参照してください。

  `http://edocs.beasys.co.jp/e-docs/wls61/jconnector/security.html`

セキュリティ コンフィグレーションの考慮事項

WebLogic Integration ドメイン用にセキュリティをコンフィグレーションする前に、以下を考慮してください。

- デジタル証明書について
- セキュア ソケット レイヤ（SSL）プロトコルの使用
- 発信プロキシ サーバまたはプロキシ プラグインの使用
- ファイアウォールまたは非 WebLogic サーバ プロキシ サーバの使用

以下の節では、これらの考慮事項について詳しく論じ、それらが WebLogic Integration のセキュリティ コンフィグレーションに及ぼす影響について説明します。

デジタル証明書について

デジタル証明書とは、インターネットなどのネットワーク上でプリンシパルとオブジェクトを一意のエントリとして識別するための電子的なドキュメントのことです。デジタル証明書によって、認証局と呼ばれる信頼性のあるサードパーティによって証明されたとおりに、ユーザまたはオブジェクトのアイデンティティが特定の公開鍵に安全にバインドされます。デジタル証明書の所有者は、公開鍵とプライベート キーの組み合わせによって一意に識別されます。

B2B 機能を使用し、企業間商取引のベースとして WebLogic Integration 環境を設定する場合は、デジタル証明書とキーの特定のセットを取得してコンフィグレーションする必要があります。以下のアイテムがセットに含まれます。

- サーバ証明書－ローカル マシン上の WebLogic Server インスタンスに対する SSL に必要です。
- ルート認証局（CA）－デジタル証明書および公開鍵の発行先のアイデンティティを保証する信頼性のある第三者組織または企業です。CA の例としては、Verisign や Baltimore があります。
- トレーディング パートナ証明書－ B2B コラボレーションに参加するローカルおよびリモートの各トレーディング パートナに必要です。これらの証明書には、クライアント証明書も含まれ、また、暗号化証明書や署名証明書が含まれる場合もあります。証明書類は、認証、認可、署名サポート、メッセージの暗号化に使用します。

デジタル証明書のフォーマット

デジタル証明のフォーマットおよびパッケージング規格が、WebLogic Server に対応していることを確認してください。デジタル証明書には、次に挙げるように、さまざまな暗号化方式があります。

- Privacy Enhanced Mail (PEM)
- Definite Encoding Rules (DER)
- Public Key Cryptography Standard 7 および同 12 (PKCS7、PKCS12)

WebLogic Server の公開鍵インフラストラクチャ（PKI）は、X.509 のバージョン 1 または 3、X.509v1、X.509v3 に適合するデジタル証明書を認識します。デジタル証明書は、Verisign、Entrust などの認証局から取得することをお勧めします。

注意： 会話に参加しているトレーディング パートナが Microsoft IIS をプロキシ サーバとして使用している場合、その会話で使用されるすべての証明書が、Verisign、Entrust などの著名な CA に信頼されている必要があります。自己署名による証明書を使用すると、IIS プロキシ サーバを通じて渡される要求が処理されません。これは、WebLogic Integration ではなく、IIS の制限です。

詳細については、『*B2B Integration セキュリティの実装*』の「セキュリティのコンフィグレーション」を参照してください。

セキュア ソケット レイヤ（SSL）プロトコルを使用する

SSL プロトコルは、次の 2 つの機能をサポートすることにより、セキュアな接続を可能にします。

- ネットワーク接続を通じてリンクされた 2 つのアプリケーションが、互いの ID を認証できるようにする。
- アプリケーション間で交換されたデータを暗号化する。

SSL 接続はデジタル証明書を交換するアプリケーション間のハンドシェークで始まり、使用する暗号化アルゴリズムの取り決め、そのセッションの残りで使用する暗号キーの生成と続きます。

B2B コラボレーションには、トレーディング パートナの認証と認可に SSL を使用することを推奨しますが、その場合は、以下のコンフィグレーションが必要です。

- WebLogic Integration ドメインのマシンごとの SSL。その方法の説明については、『*B2B Integration セキュリティの実装*』、「セキュリティのコンフィグレーション」の「SSL プロトコルと相互認証のコンフィグレーション」を参照してください。
- トレーディング パートナごとのデジタル証明書と公開鍵のセット。

- WebLogic Integration ドメイン内のマシンごとのサーバ証明書
- ルート認証局（CA）の証明書

  証明書のコンフィグレーションの詳細については、『*B2B Integration セキュリティの実装*』の「セキュリティのコンフィグレーション」を参照してください。

SSL の必須要件ではありませんが、WebLogic Integration ドメインで使用されるすべての証明書とキーを格納するキーストアの作成および活用をお勧めします。WebLogic Server には、Java Development Kit で Sun Microsystems が提供する参照キーストア実装に基づく *WebLogic Keystore プロバイダ*というユーティリティがあります。

WebLogic Keystore プロバイダは、キーストアをファイルとして実装する、標準の *JKS* キーストア タイプに基づいています。このリリースの WebLogic Server に対しては、利用できるキーストア プロバイダは JKS のみです。WebLogic Keystore プロバイダを使用してコンフィグレーションされたキーストアでは、各プライベート キーが個々のパスワードで保護されます。WebLogic Keystore プロバイダには、2 つのファイルが関連付けられています。1 つは、SSL がクライアント証明書を確認するのに使用する、CA 証明書のファイルで、もう 1 つは、ユーザのプライベート キーを格納するファイルです。WebLogic Server は、このキーストアからプライベート キーを取り出して、SSL を初期化します。

WebLogic Integration ドメイン用のキーストアの設定に関する詳細については、『*B2B Integration セキュリティの実装*』の「キーストアのコンフィグレーション」を参照してください。

発信プロキシ サーバまたはプロキシ プラグインを使用する

この節では、発信プロキシ サーバまたは WebLogic プロキシ プラグインの使用が及ぼす影響について説明します。

発信プロキシ サーバを使用する

プロキシ サーバを使用すると、トレーディング パートナはセキュリティを危険にさらすことなくイントラネットまたはインターネット経由で通信できます。高度なセキュリティで保護する必要のある環境で WebLogic Integration を使用する場合、WebLogic Integration をプロキシ サーバの後ろで使用すると効果的です。具体的には、プロキシ サーバの用途は以下のとおりです。

- WebLogic Integration のホストとなっている WebLogic Server インスタンスのローカル ネットワーク アドレスの、外部ハッカーからの隠ぺい
- 外部ネットワークへのアクセスの制限
- WebLogic Integration のホストとなっている WebLogic Server のローカル インスタンスへの外部ネットワーク アクセスのモニタ

プロキシ サーバをローカル ネットワーク上でコンフィグレーションすると、ネットワーク トラフィック（SSL プロトコルと HTTP プロトコル）は、プロキシ サーバを経由して外部ネットワークにトンネリングされます。

発信プロキシ サーバを使用する環境では、ローカル トレーディング パートナに対する転送 URI エンドポイントの指定に注意してください。HTTP プロキシを使用している場合は、Java システム プロパティの the `ssl.ProxyHost` と `ssl.ProxyPort` を指定する必要があります。詳細は、『*B2B Integration セキュリティの実装*』、「セキュリティのコンフィグレーション」の「発信 HTTP プロキシ サーバを使用するための WebLogic Integration B2B のコンフィグレーション」を参照してください。

WebLogic プロキシ プラグインと Web Server を併用する

発信プロキシ サーバを使用する代わりに、リモートのトレーディング パートナからのビジネス メッセージを処理できるようにプログラムされた、Apache サーバなどの Web サーバを使用して WebLogic Integration をコンフィグレーションすると便利な場合もあります。Web サーバは以下のサービスを提供できます。

- リモートのトレーディング パートナからのビジネス メッセージの受信。
- トレーディング パートナからのデジタル証明書の認証。

続いて、Web サーバは WebLogic プロキシ プラグインを使用します。プラグインは以下のサービスを提供するようにコンフィグレーションできます。

- Web サーバが受信したビジネス メッセージの、セキュアな内部ネットワークの内側で動作している WebLogic Integration への転送。
- Web サーバからのリモート トレーディング パートナ証明書の抽出しと、認証のための、WebLogic Server への証明書の転送 WebLogic Integration は、これを受けて、トレーディング パートナ証明書とビジネス メッセージを認証することができます。

WebLogic プロキシ プラグインをコンフィグレーションする際の考慮事項は以下のとおりです。

- WebLogic プロキシ プラグインを使用するプロキシ サーバを、要求を WebLogic Server に向けるようにコンフィグレーションする。
- プロキシ サーバと WebLogic Integration ドメイン間のネットワーク接続に使用するプロトコルを決定する。デフォルトのプロトコルは HTTP。必要な場合にのみ一方向の SSL を使用するように、プロキシ プラグインをコンフィグレーションする。
- リモートのトレーディング パートナへの転送のコンフィグレーションでは、HTTP プロトコルが WebLogic プロキシ プラグインと WebLogic Integration ドメイン間のネットワーク接続で使用されているにもかかわらず、HTTP プロトコルでリモート URI エンドポイントを指定する。
- あるトレーディング パートナから他のトレーディング パートナへビジネス メッセージを仲介する際、プロキシ サーバによっては、末端証明書のみを含めて、CA 証明書チェーンの全体は含めない場合がある。そのような状況では、トレーディング パートナが認証されない場合もある。この問題を回避するため、信頼されている CA 証明書として末端証明書の指定が推奨される（末端証明書の詳細については、『*B2B Integration セキュリティの実装*』の「WebLogic Integration B2B セキュリティ入門」を参照）。
- ローカル トレーディング パートナのサイトで、WebLogic プロキシ サーバを使ってコンフィグレーションされた Web サーバが使用されている場合は、そのトレーディング パートナの転送 URI エンドポイントを通常の方法で指定することができる。
- リモート トレーディング パートナも WebLogic Integration を使用しているが、プロキシ サーバは WebLogic プロキシ サーバではない場合、そのリモート サイトはおそらく WebLogic プロキシ プラグインを使用してコンフィグレーションされている。このような状況でリモート トレーディング パートナをコンフィグレーションする場合は、転送 URI エンドポイントとし

てトレーディング パートナのプロキシ サーバのホストとポートを指定する必要がある。WebLogic プロキシ プラグインは、そのリモート トレーディング パートナに代わって、受信したビジネス メッセージに対する必要な URL 変換を実行する。

ファイアウォールを使用する

WebLogic Integration 環境にファイアウォールが設定されている場合は、HTTP または HTTPS を介して、ローカル トレーディング パートナとの間でビジネス メッセージを自在にやり取りできるように、ファイアウォールが正しくコンフィグレーションされていることを確認してください。

セキュアなデプロイメントの設定

以下の節では、セキュア デプロイメントの設定に必要なタスクを実行する手順について説明します。

- 手順 1 : ドメインを作成する
- 手順 2 : WebLogic Server のセキュリティをコンフィグレーションする
- 手順 3 : BPM セキュリティをコンフィグレーションする
- 手順 4 : B2B Integration のセキュリティをコンフィグレーションする
- 手順 5 : Application Integration のセキュリティをコンフィグレーションする

手順 1 : ドメインを作成する

セキュリティをコンフィグレーションしようとする WebLogic Integration ドメインは、BEA コンフィグレーション ウィザードを使用して作成することをお勧めします。WebLogic Integration ドメインを作成するには、以下の手順を完了してください。

1. 次の URL にある『*Configuration Wizard の使い方*』の説明に従って、コンフィグレーション ウィザードを起動します。

   http://edocs.beasys.co.jp/e-docs/platform/docs70/confgwiz/index.html

2. WebLogic Integration ドメインのコンフィグレーションを完了します。ドメインのタイプは以下のいずれかです。
   - WebLogic Integration BPM ドメイン
   - WebLogic Integration EAI ドメイン
   - WebLogic Integration ドメイン

注意： 新しいドメインの作成には、WebLogic Server テンプレートや WebLogic Portal テンプレートではなく、必ず WebLogic Integration テンプレートを使用してください。WebLogic Integration テンプレートを指定することにより、この手順で作成するドメインが、が WebLogic Server 6.x の互換性モードのセキュリティ レルムに基づいたものとなることを保証できます。WebLogic Server 7.0 で新たに導入されたレルムは、LDAP に基づいたもので、WebLogic Integration ではサポートされていません。
WebLogic Server テンプレートを選択して新しいドメインを作成した場合は、そのドメインは、LDAP に基づく新しい WebLogic Server 7.0 セキュリティ レルムとなります。

手順 2 : WebLogic Server のセキュリティをコンフィグレーションする

WebLogic Server のセキュリティをコンフィグレーションする場合は、必ず以下を実行してください。

1. ローカルおよびリモートのトレーディング パートナのサーバ証明書の取得。SSL に対しては、トレーディング パートナ要求にかかわる WebLogic Server のインスタンスごとに証明書が必要です。
2. 次の事項の検討。
   - 証明書の通称が、対応する WebLogic Server インスタンスが実行されているマシンのホスト名と一致しているか。

この 2 つの名前が同一でない場合は、ローカルの WebLogic Server インスタンスは、ホスト名確認機能を無効にしてコンフィグレーションする必要があります。この要件は、ローカルにコンフィグレーションされたコラボレーション アグリーメントに参加するすべてのトレーディング パートナのサーバ証明書にも適用されます。ホスト名確認機能は、WebLogic Server Administration Console で、該当する Server ノードに対する [SSL] タブの [ホスト名検証を無視] 属性にチェックを付けることにより無効にできます。

- サーバ証明書とプライベート キーのフォーマットは、WebLogic Server でサポートされているリモート トレーディング パートナに対応しているか。

  サポートされている証明書フォーマットは、4-7 ページの「デジタル証明書について」にリストされています。SSL サーバが受け入れる最も一般的なサーバ証明書のフォーマットは、PEM フォーマットで暗号化される X.509 V1 または V3 です。

  プライベート キーについては、パスワード暗号化方式の PKCS8 が最も一般的なフォーマットです。WebLogic Server がプライベート キーを読み取れるように、そのパスワードを設定してください。

- WebLogic Server サーバ証明書の CA 証明書チェーンは何か。

  証明書チェーンは、信頼性のある CA のデジタル証明書の配列で、各 CA はチェーン内の先行するデジタル証明書の発行者です。

  すべての中間およびルート CA 証明書を含む 1 つのファイルを指定できます（ファイルに複数の CA 証明書が含まれていると、WebLogic Server は PEM フォーマットで暗号化されたファイルを要求する）。ルート CA キーストアを使用して信頼性のある CA 証明書を格納する場合は、必ず、チェーン全体をルート CA キーストアにインポートしてください。

3. WebLogic Keystore プロバイダのコンフィグレーション。WebLogic Server 7.0 は、キーストア機能をサポートします。キーストアの作成と WebLogic Keystore プロバイダのコンフィグレーションに関する詳細については、『*B2B Integration セキュリティの実装*』の「キーストアのコンフィグレーション」を参照してください。

   キーストアの使い方に関しては、以下の考慮事項に注意してください。

   - キーストアの使用で注意する点は、キーと証明書をエイリアスを使用してキーストアにインポートした後は、その証明書ファイルを新しい証明

書で上書きしても、新しい証明書はキーストアにインポートされないことです。

- キーストアが証明書とキーの現在のセットに対応した最新のものであること、また、WebLogic Integration リポジトリがキーストアの該当する内容を反映していることを確認してください。

手順 3 : BPM セキュリティをコンフィグレーションする

WebLogic Integration の Business Process Management (BPM) 機能用のセキュリティ モデルには、以下のエンティティで構成されています。

- ユーザとグループ－ユーザは、プログラミングや営業などの一定のタスクを遂行する特定の個人です。グループは、同じタスクを遂行する 1 人以上のユーザまたは 1 つ以上のグループの集合です。たとえば、グループ A はプログラマの集合を表し、グループ B は営業部員の集合を表します。

  1 つのセキュリティ レルム内では、管理者（`wlisystem` プリンシパルで表される）が、ワークフローやその他のリソースの使用を希望するユーザおよびグループに与えるアクセスのレベルを指定できます。ユーザおよびグループは、1 つの WebLogic Server セキュリティ レルム内で維持されます。WebLogic Integration Studio で、グループの定義やグループへのユーザの追加を行うことができます。追加された各ユーザは、自動的に、また同時に、WebLogic Server ユーザのリストに追加されます。

- 組織－さまざまな事業組織、所在地、その他、特定のビジネスまたは企業に関係する識別アイテムを表すエンティティです。

  組織は、セキュリティ レルムの外にある BPM 固有のエンティティです。

- ロール－あるグループに属する個人が共通して担う一定の範囲の責任分担、能力、またはパーミッション レベルです。ロールは、1 つのオーガニゼーションに属する場合もありますが、同じ名前を複数のオーガニゼーションで使用することもできます。

  ロールは、WebLogic Server グループにマップされます。

BPM セキュリティのコンフィグレーションは、基本的には、ユーザ、グループ、オーガニゼーション、およびパーミッション レベルを定義するというタスクです。オーガニゼーションとロールを定義できるので、BPM リソースにアクセス

するユーザおよびグループのオーガニゼーション化において、非常に大きな柔軟性を発揮することができます。Studio では、ユーザ、グループ、ロール、オーガニゼーションを作成および変更するのに使用できる各種ツールを使用できます。また、Studio では、ユーザ、グループ、およびロールのパーミッションをきめ細かく管理する方法が用意されています。

BPM セキュリティの詳細については、以下のトピックを参照してください。

- 『*WebLogic Integration Studio ユーザーズ ガイド*』、「データの管理」の「セキュリティ レルム」
- 『*WebLogic Integration の起動、停止およびカスタマイズ*』の「WebLogic Integration のカスタマイズ」の「BPM セキュリティ モデルについて」。特に「カスタム セキュリティ レルムのコンフィグレーション」の項を参照
- 『*BPM クライアント アプリケーション プログラミング*』の「セキュリティ レルムのコンフィグレーション」

手順 4 : B2B Integration セキュリティをコンフィグレーションする

ファイアウォール越しに行われるトレーディング パートナ間のメッセージ交換に関与する WebLogic Integration ソリューションには、トレーディング パートナの認証および認可や否認防止性など、特別なセキュリティ要件があります。

B2B セキュリティをコンフィグレーションするには、以下のタスクを実行します。

- B2B コラボレーションを実施するのに必要な証明書とキーの取得。WebLogic Server の各インスタンスに対して、ルート CA 証明書、トレーディング パートナ証明書、先に説明したキー、およびサーバの証明書とキーが必要です。
- WebLogic Integration 環境で使用するキーストアの作成および WebLogic Keystore プロバイダへのキーストアの登録。
- ローカル トレーディング パートナのコンフィグレーション。
- リモート トレーディング パートナのコンフィグレーション。
- 否認防止サービスの実装（必要な場合）。

- 使用するビジネス プロトコルに対するセキュリティ要件の実装。

以下の節では、各タスクに関する推奨事項および考慮事項を示します。

証明書を取得する

WebLogic Integration のセキュリティのコンフィグレーションに着手する前に、特に B2B 交換を実施するプランがある場合は、以下の証明書とキーを必ず取得してください。

- B2B 交換で使用される、WebLogic Server の各インスタンスに対するサーバ証明書とキー。この要件は、リモート トレーディング パートナにも適用されます。コンフィグレーションの対象となる各トレーディング パートナについて、そのサーバ証明書が必要です。これらの証明書は、SSL が必要とします。
- この環境で使用される各証明書に対するルート CA 証明書。

  ルート CA ディレクトリには、トレーディング パートナのクライアント、暗号化方式、および署名の各証明書に対するルート CA 証明書*のみ*が格納されている必要があります。ルート CA ディレクトリには、サーバ証明書に対する CA 証明書が格納されていては*いけません*（サーバ証明書は、そのドメインの `config.xml` ファイルでコンフィグレーションされる）。
- 各トレーディング パートナのクライアント証明書（ローカル、リモートの両方）。ローカル トレーディング パートナについては、クライアント証明書のプライベート キーの場所も取得する必要があります。
- すべてのトレーディング パートナについての暗号化方式と署名に対する証明書。ローカル トレーディング パートナについては、各証明書のプライベート キーの場所も取得する必要があります。

キーストアを作成する

B2B コラボレーション用に WebLogic Integration ドメインの設定時には、以下のキーストアを作成するため、WebLogic Keystore プロバイダをコンフィグレーションする必要があります。

- プライベート キーストア

ローカル トレーディング パートナのプライベート キーと B2B コラボレーションに通常必要とされる証明書（クライアント、サーバ、署名、暗号化方式の証明書）を格納します。WebLogic Server は、このキーストアからプライベート キーを取り出して、SSL を初期化します。

- ルート CA キーストア

  すべての信頼性のある認証局（CA）の証明書を格納します。WebLogic Keystore プロバイダは、WebLogic Server が使用する、信頼性のある CA のキーストアを作成します。これは、デフォルトでは、SSL がクライアント証明書を確認するために使用する、信頼性のある CA を見つけるのに使用します。

JavaSoft JDK `keytool` ユーティリティまたは WebLogic Server `ImportPrivateKey` ユーティリティを使用して、各キーストアを作成し、プライベート キーを追加することができます。上のキーストアがいずれも作成されていない場合は、プライベート キーを追加するために、どちらかのユーティリティが初めて使用される際に作成されます。

キーストアを作成して、キーの初期セットを入力し、「手順 2: WebLogic Server のセキュリティをコンフィグレーションする」の説明に従って、WebLogic Keystore プロバイダに登録します。

ローカル トレーディング パートナをコンフィグレーションする

ローカル トレーディング パートナは、HTTP または HTTPS を使用してリモート トレーディング パートナにメッセージを送信します。B2B コラボレーションで SSL を使用している（推奨）場合は、各トレーディング パートナについて、クライアント証明書とキーをコンフィグレーションする必要があります。

ローカル トレーディング パートナのクライアント証明書とキーについては、以下に注意してください。

- プレーン キーとパスワードで保護されたキーの両方がサポートされている。PEM、DER、PKCS8 の各フォーマットがサポートされている。
- PEM または DER 方式で暗号化された X509 V1 または V3 証明書がサポートされている。

- プライベート キーのパスワードは、大文字小文字の区別が意味を持ち、Java システム プロパティの startWeblogic スクリプトのコマンドラインで指定される。

ローカル トレーディング パートナの署名と暗号方式の証明書とキーについては、以下に注意してください。

- パスワードで保護された、PKCS8 を使用するプライ ベート キーのみがサポートされている。
- DER 暗号化方式の証明書を推奨。
- ルート CA 証明書を CA 証明書ディレクトリが指示する場所にコピーする必要がある。
- プライベート キーのパスワードは、大文字小文字の区別が意味を持ち、Java システム プロパティの startWeblogic スクリプトのコマンドラインで指定される。

リモート トレーディング パートナをコンフィグレーションする

ローカル トレーディング パートナの場合と同様に SSL を使用している場合は、各リモート トレーディング パートナのクライアント証明書とキーをコンフィグレーションする必要があります。

リモート トレーディング パートナのクライアント証明書とキーについては、以下に注意してください。

- リモート トレーディング パートナは、クライアント証明書を使用して接続先ドメインの WebLogic Server との双方向 SSL 接続を確立する。この証明書が、WebLogic Integration リポジトリでそのトレーディング パートナに対して指定さている証明書と一致することを確認してください。
- リモート トレーディング パートナも WebLogic Server を使用している場合は、この情報はそのトレーディング パートナのクライアント証明書としてリポジトリに格納されている。
- リモート トレーディング パートナが WebLogic Integration に基づいていない場合は、有効なクライアント証明書をそのトレーディング パートナから取得すること（WebLogic Integration - Business Connect を使用している場合は、そのソフトウェアの管理ツールを使用して、企業の証明書をファイルにエク

スポートし、そのファイルの WebLogic Integration 内の場所を指定することができる）。

リモート トレーディング パートナの署名と暗号方式の証明書とキーについては、以下に注意してください。

- 必ず、WebLogic Integration リポジトリで署名および暗号化方式の証明書とキーをコンフィグレーションする前に、それらの証明書を取得すること。これらの証明書のすべてが同一である必要はありませんが、1 つの証明書を暗号化方式と署名の両方に使用することが可能です。
- CA 証明書も取得して、ルート CA キーストア（または、キーストアを使用していない場合は該当するディレクトリ）に追加すること。自己署名証明書が使用されている場合は、それも追加してください。

リモートトレーディング パートナのサーバ証明書については、以下に注意してください。

- リモート トレーディング パートナのサイトの HTTP（S）サーバ証明書を取得する必要がある。これが、SSL 接続先のサーバです。
- リモート トレーディング パートナのサイトも WebLogic Integration に基づいており、リモート サイトの Web サーバが WebLogic Server である場合は、トレーディング パートナのサーバ証明書として、その Web サーバの証明書を使用すること。

ビジネス プロトコルに対するセキュリティ要件を実装する

以上の他に、コラボレーション アグリーメントをコンフィグレーションするためのビジネス プロトコルに固有のセキュリティ要件が存在する可能性があるため注意してください。

次の表には、さまざまなビジネス プロトコルに関する追加の情報源がリストされています。

表 5-1 B2B で使用されるビジネスプロトコルに関する追加情報

トピック	参照すべき節のタイトル	出典マニュアル
RosettaNet セキュリティ	「はじめに」の「RosettaNet セキュリティのコンフィグレーション」	*Implementing RosettaNet for B2B Integration*

表 5-1 B2B で使用されるビジネスプロトコルに関する追加情報

トピック	参照すべき節のタイトル	出典マニュアル
cXML セキュリティ	「cXML の管理」の「セキュリティ」	*Implementing cXML for B2B Integration*
ebXML セキュリティ	「ebXML の管理」の「セキュリティ」	*B2B Integration ebXML の実装*

手順 5 : Application Integration のセキュリティをコンフィグレーションする

WebLogic Integration は、Application Integration の機能を使用して作成および管理される統合ソリューションの部分を対象とした以下のセキュリティ メカニズムを提供します。

- エンタープライズ情報システム（EIS）に接続するには、アプリケーションがログイン名やパスワードなどの特定の資格を提供しなければならない場合がある。詳細については、『*Application Integration ユーザーズ ガイド*』、「カスタム コードの作成によるアプリケーション ビューの使用方法」の「シナリオ 1: 特定の資格に基づいて接続する」を参照。
- アプリケーション ビューをデプロイする場合、セキュリティ設定をコンフィグレーションして、WebLogic Server のユーザまたはグループに対してアプリケーション ビューの読み書きアクセス権を付与または無効にすることができる。詳細については、『*Application Integration ユーザーズ ガイド*』、「アプリケーション ビューの定義」、「アプリケーション ビューの定義手順」の「アプリケーション ビューのデプロイ」を参照。

6　パフォーマンスのチューニング

以下の節では、WebLogic Integration デプロイメントのパフォーマンスをチューニングする方法について説明します。

- WebLogic Integration のパフォーマンスのチューニング
- 実行時パフォーマンスのモニタおよびチューニング
- ハードウェア、オペレーティング システム、およびネットワークのリソースのチューニング
- データベースのチューニング

WebLogic Integration のパフォーマンスのチューニング

以下の節では、WebLogic Integration のパフォーマンスのチューニング方法について説明します。

- 一次チューニング リソース
- WebLogic Server のパフォーマンスをチューニングする
- Java 仮想マシン（JVM）をモニタおよびチューニングする

一次チューニング リソース

この節では、サーバが実行する作業を管理するためのチューニング可能な WebLogic Integration の一次リソースについて説明します。

- BPM の場合、イベント駆動型ワークフローをチューニングするための一次リソースは、イベント リスナ メッセージ駆動型 Bean です。

- Application Integration の場合、チューニング方法は処理のタイプによって異なります。
  - 同期サービス呼び出しの場合、一次リソースはアプリケーション ビュー Bean です。
  - 非同期サービス呼び出しの場合、一次リソースは、非同期要求プロセッサのスレッド プール サイズです。
  - 通常、イベント アダプタはチューニング不要です。

  また、J2EE-CA リソースのプール サイズをアダプタごとに設定する必要があります。アダプタのチューニング方法については、アダプタのマニュアルを参照してください。
- B2B Integration の場合、チューニングできる一次リソースはありません。

その他の WebLogic Integration リソースはすべて、一次リソースをサポートするためにのみ変更します。

WebLogic Server のパフォーマンスをチューニングする

以下の節では、WebLogic Integration デプロイメント用の WebLogic Server リソースのコンフィグレーション方法について説明します。

- EJB のプール サイズおよびキャッシュ サイズをコンフィグレーションする
- JDBC 接続プールのサイズをコンフィグレーションする
- 実行スレッド プールをコンフィグレーションする
- J2EE コネクタ アーキテクチャ アダプタ用のリソース接続プールをコンフィグレーションする
- B2B の大容量メッセージ サポートをコンフィグレーションする

WebLogic Server のパフォーマンスのチューニングの概要については、次の URL にある『*BEA WebLogic Server パフォーマンス チューニング ガイド*』を参照してください。

`http://edocs.beasys.co.jp/e-docs/wls/docs70/perform/index.html`

EJB のプール サイズおよびキャッシュ サイズをコンフィグレーションする

EJB のプール サイズおよびキャッシュ サイズをコンフィグレーション、つまりデフォルト設定で開始して必要に応じて変更することで、WebLogic Integration のパフォーマンスをチューニングできます。一般に、パフォーマンスという点では、プールまたはキャッシュのサイズは小さすぎるよりも大きすぎる方が問題は少なくなります。これらの設定の詳細については、次の URL にある『*WebLogic エンタープライズ JavaBeans プログラマーズ ガイド*』の「WebLogic Server への EJB のデプロイ」を参照してください。

`http://edocs.beasys.co.jp/e-docs/wls/docs70/ejb/deploy.html`

BPM イベント リスナ メッセージ駆動型 Bean のプール サイズをコンフィグレーションする

`wlpi-mdb-ejb.jar` ファイルには、イベント キューのイベントを取り出すイベント リスナ メッセージ駆動型 Bean のプールが格納されています。プール サイズの設定は、受け取ったイベントに基づいて、WebLogic Integration システムで実行されるワークフロー数を制御します。イベントおよび検証イベントキュー用の message-driven bean 同様、時限イベント リスナも、`wlpi-mdb-ejb.jar` ファイルで指定されたクラスタにデプロイされます。これらの message-driven bean は、`com.bea.wli.bpm.TimerQueue` から作業を取り出します。

EventListener と TimeListener のデフォルトのプール サイズにアクセスするには、Administration Console で WLI-BPM イベント プロセッサ EJB (`wlpi-mdb-ejb.jar`) に対する [EJB 記述子の編集] を選択します。たとえば、イベント リスナ message-driven bean のデフォルト設定は 10 (順序付けされているリスナと順序付けされていないリスナが 5 つずつ) です。

システムに対してカスタム JMS キューをコンフィグレーションする場合は、MDB Generator ユーティリティを使用してプールサイズと関連付けられたキューを設定します。『*WebLogic Integration の起動、停止およびカスタマイズ*』、「WebLogic Integration のカスタマイズ」の「カスタム Java Message Service キューのコンフィグレーション」を参照してください。

最初は 20 個の Bean を設定し、モニタを行って Bean の追加が必要かどうか判断することをお勧めします。詳細については、6-22 ページの「メッセージ駆動型 Bean 数の確認する」を参照してください。

その他の EJB のプール サイズおよびキャッシュ サイズをコンフィグレーションする

システムのチューニングを行う場合、以下のキャッシュおよびプール サイズについて考慮することをお勧めします。これらのパラメータは、WebLogic Integration クラスタ内のノードごとにチューニングすることができます。

BPM ワークフロー プロセッサ Bean のキャッシュサイズ

BPM ワークフロー プロセッサ Bean については、1-12 ページの「ワークフロー プロセッサ Bean」 参照。

このキャッシュ サイズは、BPM イベント リスナ メッセージ駆動型 Bean プールのサイズと同じかそれ以上とする必要があります。また、サブワークフローや Worklist クライアントからの予想される負荷にも対応できる値を設定する必要があります。デフォルト設定は 100 です。`WorkflowProcessor` EJB とその `Max Beans In Cache` の設定にアクセスするには、WebLogic Server Administration Console で [Edit EJB Descriptor for the WLI-BPM Server EJB (`wlpi-ejb.jar`)] を選択します。

BPM テンプレート エンティティ Bean のキャッシュ サイズ

BPM テンプレート エンティティ Bean については、1-13 ページの「テンプレート Bean」 参照。

このキャッシュ サイズには、WebLogic Integration システムで同時に処理するユニークなテンプレートの数と同じかそれ以上の値を設定する必要があります。デフォルト設定は 100 です。`TemplateDefinitionRO` EJB とその [`Max Beans In Cache`] の設定にアクセスするには、WebLogic Server Administration Console で [Edit EJB Descriptor for the WLI-BPM Server EJB (`wlpi-ejb.jar`)] を選択します。

BPM テンプレート エンティティ Bean のキャッシュ サイズ

BPM テンプレート エンティティ Bean については、1-14 ページの「インスタンス Bean」 参照。

このキャッシュ サイズには、ワークフロー インスタンス プロセッサ数と同じかそれ以上の値を設定する必要があります。デフォルト設定は 100 です。`WorkflowInstance` EJB とその [`Max Beans In Cache`] の設定にアクセスするには、WebLogic Server Administration Console で [Edit EJB Descriptor for the WLI-BPM Server EJB (`wlpi-ejb.jar`)] を選択します。

アプリケーション ビューの ステートレス セッション Bean のプールサイズ

アプリケーション ビューの ステートレス セッション Bean については、1-16 ページの 「Application Integration リソース」 を参照。

Max Beans In Free Pool のデフォルト設定は 200 で、ほとんどのデプロイメントではこの値で十分です。

com.bea.wlai.client.ApplicationView EJB とそのプールの設定値にアクセスするには、WebLogic Server Administration Console で、[Edit EJB Descriptor for the WLI-AI Server EJB (wlai-server-ejb.jar)] を選択します。

アプリケーション ビューの ステートフル セッション Bean のキャッシュ サイズ

Max Beans In Cache のデフォルト設定は 100 です。

com.bea.wlai.client.StatefulApplicationView EJB とその [Stateful Session Cache] の設定にアクセスするには、WebLogic Server Administration Console で [Edit EJB Descriptor for the WLI-AI Server EJB (wlai-server-ejb.jar)] を選択します。

非同期サービス プロセッサ message-driven bean のプール サイズ

非同期サービス プロセッサ message-driven bean については、1-16 ページの 「Application Integration リソース」 を参照。

Max Beans In Free Pool のデフォルト設定は 1000 です。WLI-AI Async Processor message-driven bean とそのプールの設定にアクセスするには、Administration Console で、[Edit EJB Descriptor for the WLI-AI Async Processor EJB (wlai-asyncprocessor-ejb.jar)] を選択します。

イベント プロセッサ message-driven bean のプール サイズ

イベント プロセッサ message-driven bean については、1-16 ページの 「Application Integration リソース」 を参照。

Max Beans In Free Pool のデフォルト設定は 1000 です。WLI-AI イベント プロセッサ message-driven bean とそのプールの設定にアクセスするには、Administration Console で、[Edit EJB Descriptor for the WLI-AI Event Processor (wlai-eventprocessor-ejb.jar)] を選択します。

JDBC 接続プールのサイズをコンフィグレーションする

JDBC 接続プールのサイズをコンフィグレーションすることで、WebLogic Integration のパフォーマンスをチューニングできます。概要については、1-8 ページの「JDBC 接続プール」を参照してください。

WebLogic Integration クラスタ内の各ノードで必要な JDBC 接続プールのサイズを決めるには、次の表のガイドラインに基づいて、サーバごとに必要な接続数を計算します。

表 6-1 JDBC 接続プールの接続数の計算

対象リソース	必要な JDBC 接続数の計算方法
BPM イベント リスナ メッセージ駆動型 Bean (順序付けされていない Bean + すべての順序付けされている Bean)。	イベント リスナ メッセージ駆動型 Bean のプール サイズに 2 を掛ける。たとえばイベント リスナ メッセージ駆動型 Bean のプール サイズが 10 の場合、JDBC 接続プールに 20 接続を追加する。 イベント リスナは常に、少なくとも 1 つ、場合によっては 2 つの JDBC 接続を使用する。最大数を使用する場合を想定して因子 2 を掛けているので、必要であれば、計算値よりも小さい値を使用できる。 **注意 :** Worklist クライアントからワークフロー プロセッサを実行する場合、さらに接続数を追加する必要がある。
B2B Integration	JDBC 接続プールに 10 接続を追加する。
Application Integration	アプリケーション ビュー Bean (デフォルトは 5) ごとに 1 接続を追加し、非同期要求プロセッサ リスナ（デフォルトは 2）ごとに 1 接続を追加する。
Application Integration アダプタ	アダプタ（イベント アダプタとサービス アダプタ）で必要な数の接続を追加する。たとえば DBMS アダプタの場合は、J2EE-CA リソース コネクタ プール内のリソースごとに 1 接続を追加する。

各リソースに必要な接続数を計算したら、すべてのリソースに必要な合計を計算し、その値に基づいてクラスタ内の各ノードの JDBC 接続プールをコンフィグレーションします。

パフォーマンスを最適化するには、初期容量と最大容量に同じ値を設定してください。

JDBC 接続のモニタ方法については、「JDBC 接続数を確認する」 を参照してください。

JDBC 接続プールの詳細については、以下の節を参照してください。

- 次の URL にある 『*BEA WebLogic Server パフォーマンス チューニング ガイド*』 の 「JDBC 接続プール サイズのチューニング」

  http://edocs.beasys.co.jp/e-docs/wls/docs70/perform/WLSTuning.html

- 次の URL の 『*WebLogic Server 管理者ガイド*』 の 「JDBC 接続の管理」

  http://edocs.beasys.co.jp/e-docs/wls/docs70/adminguide/jdbc.html

実行スレッド プールをコンフィグレーションする

1-9 ページの 「実行スレッド プール」 で説明されているように、実行スレッド プールをコンフィグレーションすることで、WebLogic Integration のパフォーマンスをチューニングできます。WebLogic Integration クラスタ内の各ノードについて、次の表のガイドラインに基づいて実行スレッド数を計算します。

表 6-2 実行スレッド数の計算

対象リソース	必要な実行スレッド数の計算方法
BPMBPM	BPM のオーバーヘッド用に 1 スレッドを追加する。
BPM イベント リスナ メッセージ駆動型 Bean	イベント リスナ メッセージ駆動型 Bean ごとに 1 スレッドを追加する。
同時 Worklist クライアント要求	予想される同時 Worklist クライアント要求ごとに 1 スレッドを追加する。
B2B IntegrationB2B Integration	RosettaNet または ebXML message-driven bean ごとに 2 スレッドを追加する。
Application Integration	Application Integration のオーバーヘッド用に 1 スレッドを追加する。

表 6-2 実行スレッド数の計算 (続き)

対象リソース	必要な実行スレッド数の計算方法
Application Integration アダプタ	アダプタごとに 3 スレッドを追加する。
アプリケーション	アプリケーション用として必要な数の実行スレッドを追加する。

各リソースに必要なスレッド数を計算したら、すべてのリソースに必要な合計を計算し、その値に基づいてクラスタ内の各サーバのスレッド プール サイズをコンフィグレーションします。

WebLogic Server Administration Console を使用してスレッド プール サイズをコンフィグレーションする方法については、次の URL にある『*Adminstration Console オンライン ヘルプ*』、「サーバ」の「サーバ コンフィグレーション タスク」を参照してください。

```
http://edocs.beasys.co.jp/e-docs/wls/docs70/ConsoleHelp/servers.html
```

スレッドのモニタ方法については、6-14 ページの「スレッド数を確認する」を参照してください。

J2EE コネクタ アーキテクチャ アダプタ用のリソース接続プールをコンフィグレーションする

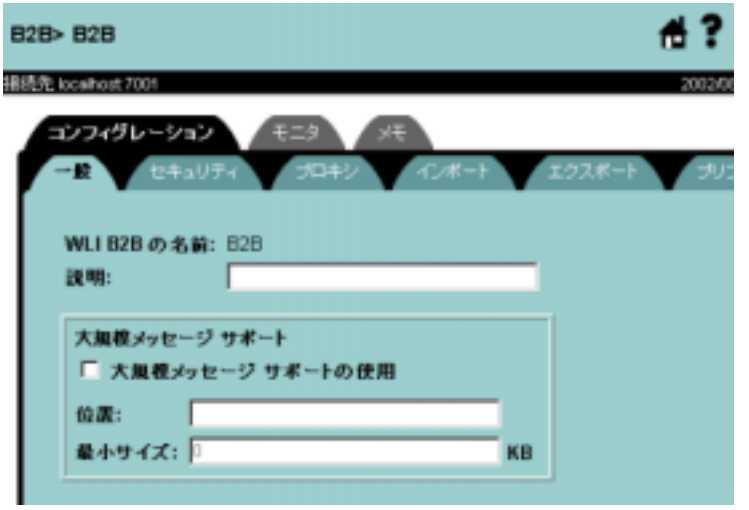
J2EE コネクタ アーキテクチャ（J2EE-CA）アダプタのリソース接続プールをコンフィグレーションすることで、WebLogic Integration のパフォーマンスをチューニングできます。1-9 ページの「J2EE コネクタ アーキテクチャ」を参照してください。特定のアダプタのリソース接続プールをチューニングする方法については、アダプタのマニュアルを参照してください。

B2B の大容量メッセージ サポートをコンフィグレーションする

B2B の会話時に交換されるメッセージが大きくメモリに収まらない場合、WebLogic Integration B2B Console で大容量メッセージ サポート機能を有効にしてから、サーバを再起動します（20MB を超えるメッセージは、大容量メッセージとみなされる）。図 6-1 は、大容量メッセージ サポート機能を有効にする際に使用するコンソール パネルの領域です。

注意： 大容量メッセージに対する EJB トランザクションのコンフィグレーションに関する詳細については、6-9 ページの「EJB トランザク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。

図 6-1 B2B Console の大容量メッセージ サポート領域



EJB トランザクションをコンフィグレーションする

メッセージ処理中にトランザクション タイムアウトを示す例外が返される場合は、次の BPM リソースにあるトランザクション タイムアウト パラメータをチューニングすることをお勧めします。

- WLI-BPM Server (`wlpi-ejb.jar`)

- WLI-BPM イベント プロセッサ EJB (`wlpi-mdb-ejb.jar`)

注意： トランザクション タイムアウトは、容量の小さいメッセージの場合より大容量メッセージの処理中の方が発生しやすくなります。

トランザクションタイムアウトパラメータをチューニングするには、`wlpi-ejb.jar` ファイルと `wlpi-mdb-ejb.jar` ファイルの `trans-timeout-seconds` 属性を、90 秒から 1090 秒に変更することをお勧めします。JAR ファイルにアクセスする手順は次のとおりです。

1. Administration Console のナビゲーション ツリーで [デプロイメント | EJB] を選択します。
2. [WLI-BPM Server EJB] または [WLI-BPM Event Processor EJB] を選択します。
3. [Edit EJB Descriptor] をクリックして、EJB 記述子の編集ができる新しいウィンドウを表示します。

Java 仮想マシン（JVM）をモニタおよびチューニングする

Java 仮想マシン（JVM）は、WebLogic Integration Java コードの実行場所です。WebLogic Integration デプロイメントのパフォーマンスを最適化するには、JVM のコンフィグレーションをチューニングします。たとえば、JVM ヒープ サイズによって、VM がガベージ コレクションを行う頻度と時間が決定されます。WebLogic Integration の場合、最小推奨ヒープ サイズは 512KB です。JVM のコンフィグレーションの詳細については、次の URL にある『*BEA WebLogic Server パフォーマンス チューニング ガイド*』の「Java 仮想マシン（JVM）のチューニング」を参照してください。

`http://edocs.beasys.co.jp/e-docs/wls/docs70/perform/JVMTuning.html`

Sun HotSpot JVM ヒープの構成とガベージ コレクションの詳細については、次の URL を参照してください。

`http://java.sun.com/docs/hotspot/gc/index.html`

Sun Hotspot JVM のコマンドライン オプションの完全なリストについては、次の URL を参照してください。

```
http://java.sun.com/docs/hotspot/VMOptions.html
```

JVM オプションの多くは、`setenv.cmd` または `setenv.sh`、および `startWeblogic.cmd` または `startweblogic.sh` で設定します。ローエンド システムに対応するため、一部のデフォルト値は低めに設定されています。大規模システムの場合は、JVM をチューニングすることでパフォーマンスを向上させることができます。次の節では、一般的に使用されるオプションについて説明します。

JVM を選択する

WebLogic Server に付属する JDK のバージョンは、何種類かの JVM 実装をサポートしています。Solaris では、サーバ JVM を使用することをお勧めします。サーバ JVM の使用を指定するには、`-server` 引数を使用します。この引数は、Java 実行プログラム名の直後に指定する必要があります。

Windows システムでは、Hotspot JVM を使用します（WebLogic Integration スクリプトでは、デフォルトでは、`-hotspot` オプションが使用される）。Windows システムでの Hotspot JVM の使用に問題が発生する場合は、スクリプトに次のオプションを追加することをお勧めします。

```
-xxMaxPermSize=131072K
```

Classic JVM では、JIT コンパイラが使用できない、推奨していません。また、Classic JVM を使用すると、Hotspot JVM や サーバ JVM を使用した場合に比べ、サーバの実行速度が低下します

実行時に使用されるコンパイル アルゴリズムを除くと Hotspot JVM とサーバ JVM は、まったく同様です（Hotspot JVM は、クライアント JVM と呼ばれることもある）。

JVM ヒープ サイズをチューニングする

JVM ヒープの最小（初期）サイズと最大サイズは、同じにします。大規模な WebLogic Integration サーバの場合、次のオプション設定のように両方のサイズを 512MB にすることをお勧めします。

```
-Xms512m -Xmx512m
```

Solaris システムでは、非常に大きなヒープ向けの追加オプションが用意されています。仮想メモリを使用せずに、物理メモリをヒープに対して直接使用できるオプションは、特に重要です。この機能は、「Intimate Shared Memory（親和型共有メモリ）」と呼ばれます。詳細については、以下の URL を参照してください。

```
http://java.sun.com/docs/hotspot/ism.html
```

Hotspot JVM でのガーベジコレクション制御

Hotspot JVM のヒープ領域は、*育成*ヒープと終身ヒープという 2 つの領域に分割されます。

すべての新しいオブジェクトは、育成ヒープで作成されます。ガベージ コレクションの実行後に残ったオブジェクトのみが、育成ヒープから終身ヒープに移されます。終身ヒープのガベージ コレクションは、終身ヒープほど頻繁には実行されず、そのコレクション処理は、育成ヒープに対するコレクション処理よりもコストが高くなります。

一般的に、育成ヒープのコンフィグレーション時には、一時オブジェクトを格納するのに十分なサイズを設定してください。一般的なアプリケーション サーバ、特に WebLogic Integration の場合、アプリケーションの実際の状態はデータベースに保存されます。要求の処理中に割り当てられたメモリの大部分は、要求が終了すると解放されます。したがって、1 つの要求の中で使用されるオブジェクトが終身ヒープに移されることを防ぐには、育成ヒープのコンフィグレーション時に十分な容量を割り当てることが重要になります。このようにコンフィグレーションすることで、育成ヒープでのガベージ コレクションに比べ、終身ヒープに対するガベージ コレクションの頻度が大幅に減ります（このため、この方法はガベージ コレクションの頻度低下と呼ばれることがある）。

育成ヒープのガベージ コレクションは世代別に実行されます。一般的なガベージコレクションでは、すべての新しいオブジェクトは、育成ヒープ領域から割り当てられます。育成ヒープ領域のすべてのオブジェクトは、オブジェクトの*若い世代*を構成します。育成ヒープ領域がフルになると、ガベージ コレクタは、部分的なガベージ コレクションを実行します。また、アクセスできなくなった、言い換えれば*死滅した*オブジェクトが占有していた育成領域のメモリを回復します。育成領域にある、まだ*生存している*オブジェクトは、古いオブジェクトを格納するメモリ領域に移動します。世代別ガベージ コレクションの場合、ガベージ コレクタは死滅したオブジェクトのすべてのメモリを検索する必要がないため、フル ガベージ コレクションと比べて、大幅に実行速度が向上します。

ガベージ コレクションおよび JVM パフォーマンスについての詳細は、次の URL にある、『*A Test of JavaTM Virtual Machine Performance*』を参照してください。

```
http://developer.java.sun.com/developer/technicalArticles/Programming/JVMPerf/
```

グローバル ヒープが 512MB の場合、育成ヒープの妥当なサイズを 128MB にすることをお勧めします。次のコマンドラインの指定では、初期育成ヒープ サイズは 128MB に、サイズの最大値は 128MB に、それぞれ設定されています。

```
-XX:NewSize=128m -XX:MaxNewSize=128m
```

育成領域は、Eden 領域 1 つと同サイズの下位領域 2 つから構成されています。ガベージ コレクションの実行中に、最後まで残るオブジェクトは下位領域に移動します。*サバイバル*領域は、2 つの下位領域を合わせたサイズです。

サバイバル領域のサイズをチューニングするには、*SurvivorRatio* パラメータを使用します。サバイバル領域の比率の初期推奨値は 2 です。アプリケーションをモニタして、この値を変更する必要があるかどうか判断します。サバイバル領域の比率を 2 と指定するには、次のオプション設定を使用します。

```
-XX:SurvivorRatio=2
```

このパラメータにより、Eden 領域と*各*下位領域の比率は 2:1 となります。言い換えれば、Eden 領域のサバイバル領域に対する比率は 1:1 で、各下位領域は、育成ヒープのサイズの 1/4 です（同じサイズの下位領域が 2 つあるので 1/2 にはならない）。サバイバル領域が小さすぎた場合、コレクションのコピー時に、直接古い世代用の領域にオーバーフローしてしまいます。サバイバル領域が過剰に大きい場合は、無用な空き領域が発生します。

JVM のヒープ使用率をモニタする

次のフラグを指定して、冗長ガベージ コレクションを有効にすると、ヒープの使用率とガベージ コレクションを最も効率的にモニタできます。

```
-verbosegc
```

結果は、標準形式で出力されます。Hotspot JVM の場合は、2 種類の行が表示され、Eden (GC) ヒープおよび 終身ヒープ（フル GC）内のコレクション数が示されます。

また、Weblogic Server Administration Console を使用して、実行時のヒープ使用率をモニタすることもできます。これは、ヒープの要件を定義したり、メモリのリークを識別する際に便利です。

実行時パフォーマンスのモニタおよびチューニング

以下の節では、WebLogic Integration のデプロイメントで実行時のパフォーマンスをモニタする方法について説明します。

- WebLogic Server のパフォーマンスをモニタおよびチューニングする
- BPM のパフォーマンスをモニタおよびチューニングする
- B2B Integration のパフォーマンスをモニタおよびチューニングする
- Application Integration のパフォーマンスをモニタおよびチューニングする
- アプリケーションのプロファイリング

WebLogic Server のパフォーマンスをモニタおよびチューニングする

WebLogic Server Administration Console を使用すると、サーバ、JDBC 接続プール、JCA、HTTP、JTA サブシステム、JNDI、EJB などのリソースを含む WebLogic Server ドメインの状態とパフォーマンスをモニタできます。詳細については、次の URL にある『*BEA WebLogic Server ドメイン管理*』の「WebLogic Server ドメインのモニタ」を参照してください。

```
http://edocs.beasys.co.jp/e-docs/wls/docs70/admin_domain/monitoring.html
```

スレッド数を確認する

システムに十分なスレッドがコンフィグレーションされているかを判断する手順は次のとおりです。

1. WebLogic Server Administration Console のナビゲーション ツリーで、サーバ名を選択します。
2. [モニタ] タブ、[一般] タブの順に選択します。

3. [すべてのアクティブなキューのモニタ] をクリックします。

   次の図は、WebLogic Server Administration Console に表示されるアクティブキューに関する情報を示したものです。

図 6-2 アクティブ実行キュー テーブル

ThreadPoolSize パラメータが、スレッド数を制御します。ThreadPoolSize パラメータは、サーバごとに個別に設定されます。

1. WebLogic Server Administration Console のナビゲーション ツリーで、サーバ名を選択します。
2. [モニタ] タブ、[パフォーマンス] タブの順に選択します。

   [スループット]、[キューの長さ]、[メモリ使用状況] の 3 つのグラフが表示されます。[アイドル スレッド数] フィールドは、グラフの上に表示されます。次の図は、WebLogic Server Administration Console に表示されるパフォーマンスに関する情報を示したものです。

図 6-3 サーバのパフォーマンス情報

3. [アイドル スレッド数] フィールドにゼロが表示されることがある場合は、スレッドを追加する必要があります。

スレッドを追加するには、次のように、デフォルト キューを選択して、スレッド数を指定します。

1. WebLogic Server Administration Console のナビゲーション ツリーでサーバを選択し、次に、[モニタ | 一般] タブを選択します。
2. [すべてのアクティブなキューのモニタ] をクリックします。
3. [Execute Queue のコンフィグレーション] を選択します。

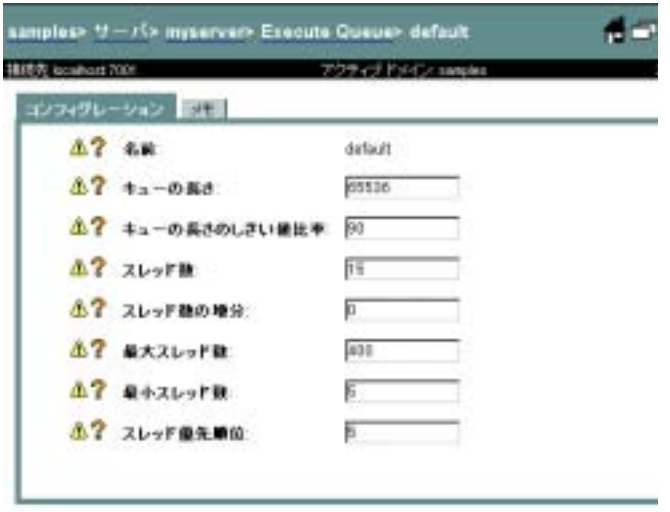
   次の図のように、実行キュー情報が WebLogic Server Administration Console に表示されます。

図 6-4 実行キュー テーブル

スレッドは、次のようにして指定することができます。

1. WebLogic Server Administration Console のナビゲーション ツリーで、サーバ名を右クリックしてドロップダウン メニューを表示します。
2. ドロップダウン メニューから [実行キューを見る] を選択して、次の図に示すとおりのウィンドウを表示します。スレッド数を指定します。

図 6-5 デフォルトの実行キーのコンフィグレーション



Solaris システムでは、スレッド数の変更によってパフォーマンスが向上するかどうかを調べるには、負荷レベルを変えずにスレッド数の設定変更の前後に `mpstat` コマンドを実行します。コンテキストが切り替わる回数が低下すれば、パフォーマンスが向上したことになります。

トランザクションの実行回数を確認する

各種トランザクションの実行回数を表示するには、Weblogic Server Administration Console で該当するサーバ名を選択します。右フレームで、[モニタ] タブ、[JTA] タブの順に選択します。[すべてのインスタンスのモニタ] を選択します。

BPM フレームに関連付けられている一部のトランザクションは、変更できません。アプリケーションに関連付けられているトランザクションは変更可能です。トランザクション タイプを変更したり、トランザクションを組み合わせるには、次の手順を実行します。

1. Administration Console のナビゲーション ツリーでサーバを選択します。
2. [モニタ] タブを選択してから、[JTA] タブを選択して、次の図に示すように、トランザクションをモニタする際に使用するウィンドウを表示します。

図 6-6 [モニタ] タブ

JDBC 接続数を確認する

JDBC 接続はデータベースへの接続で、データベースにアクセスするたびに新しい接続を確立する必要がなくなるため、スレッドのパフォーマンス問題を回避できます。JDBC 接続用に複数のプールを設定できます。各プールに十分な接続数を設定し、スレッド接続の待機時間が長くならないようにしてください。アクティブ JDBC 接続プールをモニタする手順は次のとおりです。

1. WebLogic Server Administration Console のナビゲーション ツリーで [サービス] を選択します。
2. [JDBC ｜接続プール] を選択して、すべての JDBC 接続メイン ウィンドウに表示します。
3. プールの名前をクリックしてから [Monitor All Active Pools] をクリックして、次の図のように、アクティブ JDBC 接続を表示します。

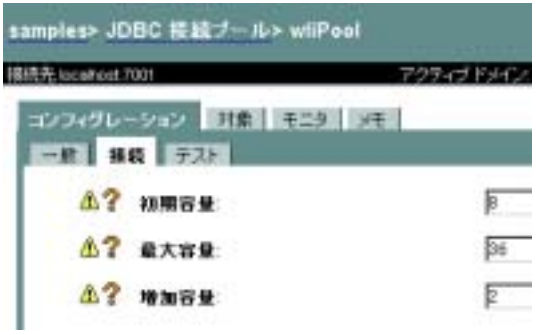
図 6-7 JDBC 接続プール

4. [接続] の欄を、表示される接続数が、このプールに対してコンフィグレーションされている合計接続数に近い値であるかをチェックします。[最大接続数] の値が、このプールに対してコンフィグレーションされている合計接続数と等しいかをチェックします。この 2 つのどちらか一方でも当てはまる場合、同様の状況下または負荷が少し増加した場合に、接続数を増やします。

接続プールのコンフィグレーションを変更する手順は次のとおりです。

1. Administration Console のナビゲーション ツリーで [サービス] を選択します。
2. [JDBC ｜接続プール｜ wliPool] を選択します。
3. メイン ウィンドウで、[接続] タブを選択します。

図 6-8 接続プールのコンフィグレーション



4. [初期容量] フィールドと [最大容量] フィールドの値を同じ数に設定します。

BPM のパフォーマンスをモニタおよびチューニングする

WebLogic Integration Studio を使用すると、ワークフローおよびワークフロー変数のステータスを含むワークフローのパフォーマンスのさまざまな側面をリアルタイムでモニタできます。Studio では、ワークフロー インスタンスを削除したり、負荷およびパフォーマンスの統計に関するレポートを表示したりすることができます。詳細は、『*WebLogic Integration Studio ユーザーズ ガイド*』の「ワークフローのモニタリング」を参照してください。

BPM のパフォーマンスの主な測定項目は以下のとおりです。

- *インスタンス数*—指定した期間内に開始されたワークフローの数です。インスタンス数には、同時実行しているクライアントによって開始された処理（ワークフローのインスタンス化、タスクの実行、およびサーバへのイベント送信）数が含まれています。
- *終了数*—指定した期間内に終了した（完了ノードに達した）ワークフローの数です。終了数には、サーバサイドから実行された処理（ワークフローのインスタンス化、タスクの実行、クライアントからのイベント受信、ビジネス オペレーションの実行、およびタスクの終了マーキング）数が含まれています。
- *メッセージ配信の保証*—必要に応じて、主な BPM 機能を使用して、メッセージが失われることなく、ターゲットのワークフローに配信されることを保証します。

これらのパフォーマンス測定項目の統計値を取得する 1 つの方法は、SQL 文を使用してデータベース インスタンス テーブルから統計値を抽出することです。たとえば、次の SQL コードでは、インスタンス数に関する統計値が計算されます。

コード リスト 6-1 ワークフロー インスタンスの統計値を調べるための SQL コード

```
select 'INSTANTIATIONS', count(*),
avg((completed-started)*86400),
max((completed-started)*86400),
86400*(max(started)-min(started)) total_duration,
from instance
```

次の SQL コードでは、終了数に関する統計値が計算されます。

コード リスト 6-2 ワークフロー終了数の統計値を調べるための SQL コード

```
select 'COMPLETIONS',  count(*),
avg((completed-started)*86400),
max((completed-started)*86400),
86400*(max(completed)-min(started)) total_duration
from instance where completed is not null
```

メッセージ駆動型 Bean 数の確認する

メッセージ駆動型 Bean に関する情報を表示する手順は、次のとおりです。

1. WebLogic Server Administration Console のナビゲーション ツリーで、サーバ名を選択します。
2. [モニタ] タブ、[JMS] タブの順に選択します。
3. [すべての JMS サーバのモニタ] をクリックします。
4. [アクティブな JMS の送り先] をクリックします。次の図に示すように、アクティブな JMS 送り先が表示されます。

図 6-9 イベント キューのモニタ

5. WLI_BPM_Event のキューの長さ（待機中のメッセージ）を確認します。この値が大きい場合には、パフォーマンスの最適化を妨げるキューが発生しています。この場合、message-driven bean を追加するとパフォーマンスが向上します。

message-driven bean の数を変更する手順は次のとおりです。

1. Administration Console のナビゲーション ツリーで [デプロイメント | EJB] を選択します。
2. [WLI-BPM Event Processor] を選択します。
3. [EJB 記述子の編集] をクリックして、EJB 記述子の編集ができる新しいウィンドウを表示します。
4. 左のナビゲーション ウィンドウで、[weblogic-ejb-jar | weblogic-enterprise-bean | EventListener | message-drive-destination | プール] を選択して、次の図のようなコンフィグレーション ウィンドウを表示します。

図 6-10 MDB のコンフィグレーション

5. [max-beans-in-free-pool] のパラメータ値を編集します。
6. 変更内容を有効にするには、WebLogic Server を再起動する必要があります。

Bean タイプ数を確認する

WebLogic Server Administration Console を使用して、Bean のタイプと数量を表示できます。

1. Administration Console のナビゲーション ツリーで [デプロイメント | EJB] を選択します。
2. 特定の EJB の名前を選択します。たとえば、WLI-BPM Server を選択します。
3. メイン ウィンドウでは、[モニタ] タブと表示する Bean のタイプを選択します。たとえば、ステートフル セッション Bean に関する情報を表示するには、[すべてのステートフル EJB ランタイムのモニタ] をクリックします。
4. 表示する情報を変更するには、[このビューをカスタマイズ ...] をクリックします。列を追加または削除したり、ソート順序を変更したりできます。

   以下の列は、特に重要です。

   - Number of beans in use
   - Number of beans in cache

   以下の JAR ファイルは、特に重要です。

   - `wlpi-ejb.jar`
   - `wlpi-mdb-ejb.jar`
   - `wlai-server-ejb.jar`
   - `wlai-eventprocessor-ejb.jar`
   - `wlai-asyncprocessor-ejb.jar`
   - アプリケーション固有の EJB の JAR ファイル

これらの JAR ファイルの詳細については、6-3 ページの「EJB のプール サイズおよびキャッシュ サイズをコンフィグレーションする」 を参照してください。次の各図（図 6-11、図 6-12、図 6-13）は、ステートフル Bean、エンティティ Bean、およびメッセージ駆動型 Bean が表示されるウィンドウの領域を示したものです。

次の図は、WLI-BPM Server EJB (`wlpi-ejb.jar`) に対するステートフル EJBRuntimesb を示しています。

図 6-11　ステートフル Bean 情報

次の図は、WLI-BPM Server EJB (`wlpi-ejb.jar`) に対するエンティティ EJBRuntimesb を示しています。

図 6-12　エンティティ Bean 情報

次の図は、WLI-BPM イベント プロセッサ EJB (`wlpi-mdb-ejb.jar`) に対するメッセージ駆動型 EJBRuntimes を示しています。

図 6-13 MDB 情報

*キャッシュ フル*に関するシステム メッセージが表示される場合、EJB 記述子を編集して、Cache パラメータで該当する EJB の [Max Beans] の値を増やします。

キャッシュがフルになるまでパッシベーションが行われないエンティティ Bean が多数ある場合は、そのエンティティ Bean の Idle Timeout Seconds パラメータ値を小さくできます。

1. Administration Console のナビゲーション ツリーで [デプロイメント | EJB] を選択します。
2. 特定の EJB の名前を選択します。たとえば、WLI-BPM Server を選択します。
3. [EJB 記述子の編集] をクリックして、EJB 記述子の編集ができる新しいウィンドウを表示します。
4. 左のナビゲーション ウィンドウで、[weblogic-ejb-jar | weblogic-enterprise-bean | WorkflowProcessor | stateful-session-descriptor | stateful-session-cache] を選択して、次の図のようなコンフィグレーション ウィンドウを表示します。
5. Idle Timeout Seconds パラメータを編集します。

図 6-14 アイドル タイムアウトのコンフィグレーション

メッセージ送信を保証する

ビジネス プロセスの設計要件に応じて、ワークフローへのメッセージの送信を保証する 2 つの機能を活用すると便利な場合があります。この節で要約する機能は、具体的には、JMS クライアントからワークフローに送信されるメッセージに適用されます。これには、ワークフロー間のメッセージ送信が含まれます。トレーディング パートナ間で送信されるビジネス メッセージは含まれません（トレーディング パートナのビジネス メッセージはデフォルトでアドレス指定メッセージ配信を使用する）。

以下の機能によってメッセージ配信が保証されます。

- アドレス指定メッセージ配信。メッセージ送信先のワークフロー インスタンスの ID を指定できます。アドレス指定メッセージングを使用すると、インスタンス化されたワークフローの受信側イベント ノードがフローでアクティブ化されていない場合でも、現在のワークフローとの会話を開始した（Start Workflow アクションでワークフローを呼び出すことにより、または以前に送信された XML メッセージにより、ワークフロー内の開始ノードまたは イベント ノードをトリガすることにより）特定のワークフロー インスタンスへの応答メッセージの配信を保証できます。
- メッセージがトランザクションが正しく終了してコミットが発行された場合に限りメッセージが送信されるようにするための、Post XML イベント アクションでトランザクション モードの設定。この設定は、ワークフローが静止状態に達するまではメッセージが送信されないことを保証することにより、

メッセージ配信を保証するのに必要なリソースが割り当てられてコミットされることを保証します。

これら 2 つの機能を併用することで、ワークフローへのメッセージの配信が保証されます。これらの機能の使い方については、以下を参照してください。

- 『*BPM クライアント アプリケーション プログラミング*』、「JMS 接続の確立」の「メッセージ配信の保証」
- 『*WebLogic Integration Studio ユーザーズ ガイド*』、「アクションの定義」にある「JMS トピックや JMS キューへの XML メッセージの通知」

アドレス指定メッセージ配信の使い方の例については、『*BPM ユーザーズ ガイド*』、「サンプルについて」の「ビジネス プロセスおよびワークフローのモデル化」を参照してください。

トレーディング パートナ間のビジネス メッセージの配信に関する詳細については、『*B2B Integration ワークフローの作成*』を参照してください。

B2B Integration のパフォーマンスをモニタおよびチューニングする

B2B Integration の機能のパフォーマンスをモニタするには、以下のヒントを検討します。

- WebLogic Integration B2B Console を使用すると、トレーディング パートナのセッション、配信チャネル、会話、およびコラボレーション アグリーメントを含む B2B Integration の機能のさまざまな側面をモニタおよび制御できます。
- 実行時のパフォーマンスをモニタするには、システムで受け取った HTTP 要求のトラッキング用のファイル、`access.log` を調べます。このファイルを調べると、システム管理者がネットワーク /TCP インタフェースの状態を検証できます。タイムスタンプから、要求着信の速度がわかります。
- XOCP メッセージ フローのボトルネックを検出するには、コンシューマ トレーディング パートナ サイドの `QualityOfService` クラスの `getHopTimestamps()` メソッドを使用します。このメソッドは、メッセージのすべてのホップのタイムスタンプを返します。データを正しく解釈できるように、すべてのマシンの時計を同期させます。

B2B Integration のパフォーマンスの主な測定項目は以下のとおりです。

- *スループット*—指定の期間内にハブで処理（送信と受信）されたメッセージ数です。
- *所要時間*—あるスポークからハブを経由して別のスポークに達するまでに必要な時間です。

詳細は、『*B2B Integration 管理ガイド*』の「B2B Integration のモニタ」を参照してください。

B2B アクティビティをモニタする

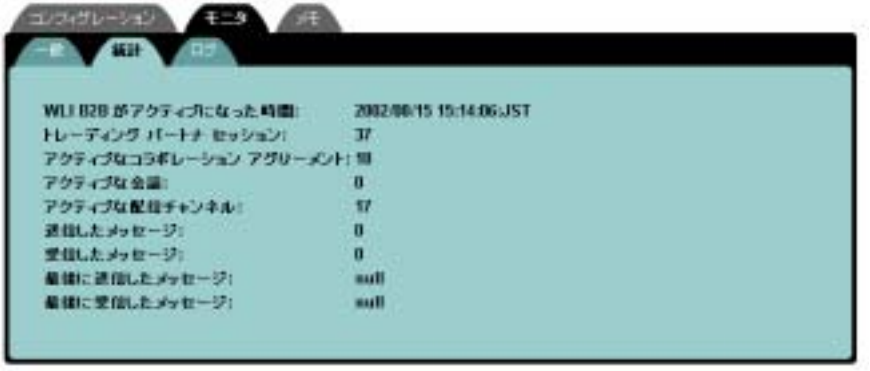
WebLogic Integration B2B Console を使用して B2B アクティビティのレベルを判断します。

1. B2B Console のナビゲーション ツリーで、親ノードの B2B を選択します。
2. メイン ウィンドウで、[モニタ | ログ] タブの順に選択して、次の図のように、`wli.log` ファイルを表示します。

図 6-15　B2B ログのモニタ

3. メイン ウィンドウで、[モニタ | 統計] タブの順に選択して、次の図のように、B2B の統計データを表示します。

図 6-16 B2B 統計のモニタ



Application Integration のパフォーマンスをモニタおよびチューニングする

この節では、Application Integration のモニタとチューニングに関する情報を提供します。内容は以下のとおりです。

- Application Integration のパフォーマンスをモニタおよびチューニングする
- Application Integration 用の EJB プールをモニタおよびチューニングする

Application Integration のパフォーマンスをモニタおよびチューニングする

アプリケーション ビューに対して十分な接続数が設定されているかどうかを確認するには、以下の手順で Weblogic Server Administration Console を使用します。

1. Administration Console のナビゲーション ツリーで [デプロイメント | コネクタ] を選択します。
2. 該当するアプリケーション ビューに対してデプロイされている接続ファクトリを選択します。接続ファクトリ名は、次の形式で表示されます。

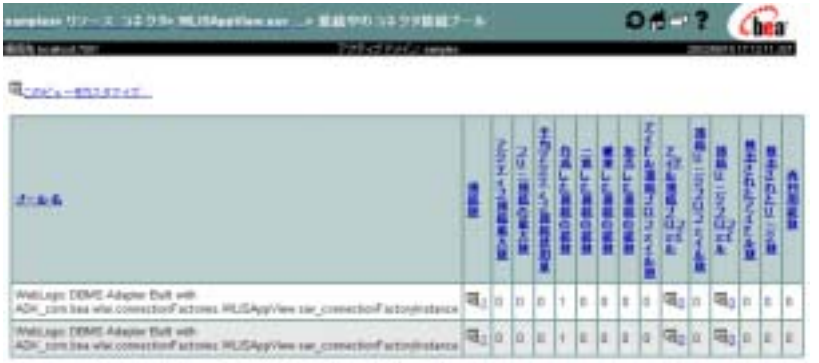
*ApplicationViewName*_connectionFactory.

3. [モニタ] タブを選択し、次に [すべての接続中のコネクタ接続プールのモニタ ...] をクリックします。

   次の図に示すように、アプリケーション ビューで定義されている EIS への接続が表示されます。

   JDBC 接続を使用すると、データベースにアクセスするたびに新しい接続を確立する必要がなくなるため、スレッドのパフォーマンス問題を回避できます。各プールに対して十分な接続数を設定し、スレッド接続の待機時間が長くならないようにしてください。

図 6-17 アプリケーション ビューの接続のモニタ



4. 接続数を確認してください。
   - 表示される接続数が、このプールに対してコンフィグレーションされている合計接続数に近い値になっていますか。
   - [最大接続数] の値が、このプールに対してコンフィグレーションされている合計接続数と等しいですか。

   この 2 つのどちらか一方でも当てはまる場合、同様の状況下または負荷が少し増加した場合に、接続数を増やします。

アプリケーション ビューに対する最大接続数を表示または変更する手順は次のとおりです。

1. Application View Console を起動します。

2. アプリケーション ビューを選択してから、[Deploy] タブを選択します。次の図は、プールのサイズに最大値をモニタするのに使用される [Application View Console] タブの使い方を示しています。

図 6-18 最大プール サイズのモニタ

[Maximum Pool Size] の値は、接続の最大数を表しています。

最大プールサイズの変更手順は次のとおりです。

1. アプリケーション ビューが現在デプロイされている場合は、[Undeploy] をクリックします。
2. アプリケーションビューがアンデプロイされているときは、[Edit] をクリックします。

   イベントやサービスを編集できるウィンドウが表示されます。
3. [Continue] をクリックして、次の図に示すウィンドウを表示します。このウィンドウで、最大プール サイズを編集できます。

図 6-19 最大プール サイズのモニタ

4. [Maximum Pool Size] の値（接続の最大数）を編集します。
5. [Deploy] をクリックして、新しい最大プール サイズを設定したアプリケーション ビューを再デプロイします。

Application Integration 用の EJB プールをモニタおよびチューニングする

Application Integration のパフォーマンスのチューニングを行う場合は、以下の EJB プールのチューニングについて考慮してください。

- 非同期サービス プロセッサ message-driven bean プール（`wlai-asyncprocessor-ejb.jar`）
- イベント プロセッサ message-driven bean プール（`wlai-eventoprocessor-ejb.jar`）
- アプリケーション ビューのステートフル およびステートレス の セッション EJB プール (`wlai-server-ejb.jar`

EJP のプールのモニタリングおよびチューニングに関する情報については、6-23 ページの「Bean タイプ数を確認する」および 6-22 ページの「メッセージ駆動型 Bean 数の確認する」参照してください。

アプリケーションのプロファイリング

Java プロファイラ ツール（Jprobe や OptimizeIt など）を使用すると、実行時にアプリケーションのプロファイリングできます。これらのツールでは、システムのパフォーマンスのボトルネックやスレッドの競合を識別できます。起動時のパフォーマンスではなく、必ず実行時のパフォーマンスをプロファイリングしてください。

ハードウェア、オペレーティング システム、およびネットワークのリソースのチューニング

以下の節では、ハードウェア、オペレーティング システム、およびネットワークをチューニングする際に考慮すべき要因について説明します。

- ハードウェアをチューニングする
- オペレーティング システムをチューニングする
- ネットワークのパフォーマンスをチューニングする

詳細については、次の URL にある『*BEA WebLogic Server パフォーマンス チューニング ガイド*』の「ハードウェア、オペレーティング システム、およびネットワーク パフォーマンスのチューニング」を参照してください。

`http://edocs.beasys.co.jp/e-docs/wls/docs70/perform/HWTuning.html`

パフォーマンスのボトルネック

デプロイメントにおける WebLogic Integration のパフォーマンスを最適化するには、以下のハードウェア リソースが互いに会話する方法を理解しておく必要があります。パフォーマンスのボトルネックは、これらのハードウェア リソースが適切にチューニングされていないことが原因で発生します。

表 6-3 パフォーマンスのボトルネック

ハードウェア リソース	ボトルネック
CPU	スループットが不十分なため、ページングやスワッピングが頻発する。
メモリ	システム メモリが不十分なため、ページングやスワッピングが頻発する。
ネットワーク リソース	大量のネットワーク トラフィックを処理するだけの帯域幅が不十分である。ネットワークの衝突が頻発する。
ディスク I/O およびコントローラ	大量の I/O 要求を処理するだけの容量が不十分である。

ハードウェアをチューニングする

デプロイメントにおける WebLogic Integration のパフォーマンスを最適化するには、以下のハードウェアの要因を検討します。

- 負荷の平均時とピーク時に許容範囲のパフォーマンス レベルで WebLogic Integration を実行するために必要なマシンの台数（とマシンごとの CPU 数）。
- ストレージの種類、コンフィグレーション、および許容できるサイズ。RDBMS のパフォーマンスを高めるには、より高速なディスクを使用します。
- 平均時とピーク時の負荷を許容範囲のパフォーマンス レベルで処理するために必要なメイン メモリの容量。

オペレーティング システムをチューニングする

デプロイメントにおける WebLogic Integration のパフォーマンスを最適化するには、以下のオペレーティング システムの要因を検討します。

- コンフィグレーション可能なファイル記述子の制限
- ユーザ プロセス用のメモリ割り当て

- コンフィグレーション可能な TCP チューニング パラメータ
- コンフィグレーション可能なスレッド処理モデルの設定
- vmstat、mpstat、netstat、iostat などのモニタ ツールの使用

Windows NT/2000 でコンフィグレーション可能な TCP のチューニング パラメータ

Windows NT または Windows 2000 を搭載したサーバの場合、`TcpTimedWaitDelay` パラメータをデフォルト値の 240 秒ではなく、60 秒に設定することをお勧めします。このパラメータは Windows レジストリにあり、`regedit` utility (`regedit.exe`) を使用して変更、編集することができます。このエントリは次の場所にあります。

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
```

デフォルトでは、このエントリは存在しません。

`TcpTimedWaitDelay` では、TCP が終了した接続を解放し、そのリソースを再利用できるまでに必要な経過時間を定義します。接続の終了から解放までの期間は、`TIME_WAIT` ステート、または `2MSL` ステートと呼ばれます。この間は、新しい接続を確立するよりも非常に低いコストで、クライアントやサーバへの接続を再開できます。

RFC 793 では、ネットワーク上にあるセグメントの最大有効期間の少なくとも 2 倍以上の間（`2MSL`）、TCP は終了した接続を維持する必要があります。接続が解放されると、そのソケットの組み合わせと TCP 制御ブロック（TCB）を使用して、別の接続をサポートできます。デフォルトで、MSL は 120 秒に定義されているので、この値は MSL の 2 倍、つまり 4 分になります。ただし、レジストリ エントリを使用して、この間隔をカスタマイズできます。

このエントリの値を小さくすると、終了した接続がより早く解放されるため、新しい接続に対して提供されるリソースが増えます。ただし、この値を小さくしすぎると、接続が完了する前に接続リソースが解放されることがあります。この場合、接続を再確立するためには、追加リソースが必要になります。

注意： 通常、このエントリ値を超えるまで、終了した接続が解放されることはありません。ただし、TCP 制御ブロック（TCB）の範囲外で接続が行われている場合には、この期間を過ぎる前に TCP は接続を解放できます。システムで作成される TCB 番号は、`MaxFreeTcbs` のパラメータ値で示されます。

Windows NT/2000 システムをモニタする

パフォーマンス モニタ（perfmon.exe）を使用すると、すべてのシステム リソースおよびタスク マネージャ（CPU、メモリ、スレッドなど）をモニタできます。

Solaris のスワップ領域をコンフィグレーションする

ヒープやスレッド限度が小さすぎるあんど、スワップ場所が不十分な場合、メモリ不足エラーを発生することがあります。

Solaris ネットワークをチューニングする

Solaris システムにおけるネットワークのチューニング方法については、次の URL にある WebLogic Server プラットフォームに関するページを参照してください。

http://e-docs.bea.com/wls/platforms/sun/index.html

Solaris システムをモニタする

Solaris システムをモニタする際の推奨コマンドを次の表で示します。

モニタ対	使用コマンド
メモリ使用率	vmstat
CPU 利用率	mpstat 5.（このコマンドを使用すると、CPU 使用率の他に、プロセッサごとのコンテキストの切り替わり回数も表示される）。CPU の合計使用率を表示するには、sar コマンドを使用する。
Disk I/O	iostat
Network I/O	netstat -sP tcp. このコマンドは、各種 TCP パラメータをモニタする際に使用する。

ネットワークのパフォーマンスをチューニングする

デプロイメントにおける WebLogic Integration のパフォーマンスを最適化するには、ネットワークで高いパフォーマンスを実現するための以下の要件を検討します。

- WebLogic Integration およびアーキテクチャ内の他の層（クライアント接続やデータベース接続など）への接続用の十分なネットワーク帯域幅
- LAN/WAN の十分なスループット
- ネットワーク パフォーマンスを最適化するためのコンフィグレーション可能なオペレーティング システムの設定
- ピーク時の負荷を処理できるだけの十分な容量

データベースのチューニング

デプロイメントにおける WebLogic Integration のパフォーマンスを最適化するには、基盤となるリソースを最大限に活かす必要があります。WebLogic Integration は、実行時の処理を行うためやアプリケーション データの永続性を保証するためにデータベース リソースに深く依存しています。以下の節では、WebLogic Integration デプロイメント内のデータベースのチューニング方法について説明します。

- 一般的なデータベース チューニングの注意点
- Oracle データベースをチューニングする
- Microsoft SQL Server データベースをチューニングする
- Sybase データベースをチューニングする

これらの節には、WebLogic Integration のパフォーマンスを最適化する作業を行う場合に検討すべき問題からなるチェックリストを用意してあります。特定のデータベース製品についての詳しい手順については、各製品の適切なマニュアルを参照してください。

一般的なデータベース チューニングの注意点

以下の節では、デプロイメントのさまざまなパラメータおよび機能の設定を調整することでデータベースのパフォーマンスを最適化する方法について説明します。

- オープンしているカーソル
- ディスク I/O の最適化
- データベースのサイズ変更とテーブル スペースの編成
- チェックポイント
- データベースの互換性
- データベースのモニタ

オープンしているカーソル

一般的に、複数のカーソルを使用することで、並行して実行する処理を増やす（1 つのオープン カーソルが更新を実行する間に、別のオープン カーソルが挿入を実行するなど）ことができますが、データベース サーバが対応できる最大カーソル数には限度があります。この最大プールは、データベース サーバのすべてのセッションおよび接続で共有されます。1 つの接続で使用するカーソルが多すぎた場合、他の接続でカーソルが不足するため、データベースのパフォーマンスとシステムのスケーラビリティが低下します。データベース サーバが対応できる最大オープン カーソル数および同時接続ユーザ数の平均値から、システムに適した値を見積もることができます。その他に、カーソルのオープン時間を最小限に抑えるという方法もあります。

ディスク I/O の最適化

ディスク I/O の最適化は、主要なデータベース チューニング パラメータの中でスループットとスケーラビリティに直接関連するものです。最速のディスクでも、アクセス速度はメモリ アクセスよりはるかに低速です。できる限り、ディスク アクセス数を最適化します。一般に、I/O のブロック / バッファ サイズを大きくするとディスク アクセス数が減るので、プロダクション環境で大きな負荷がかかった場合の実質的なスループットが向上する可能性があります。

推奨設定については、後述するデータベースのチューニングに関する節で該当するデータベースの説明を参照してください。

データベースのサイズ変更とテーブル スペースの編成

データベースの負荷を複数のディスク間で分散すると、ディスクの過負荷を防いだり、過負荷の発生回数を抑えたりできます。データベースのパフォーマンスを最適化する手順は次のとおりです。

- 頻繁にアクセスが行われるテーブルと索引は、異なるディスク上に置きます。このメカニズムは、データベースによって異なります。データベースの保存構造については、ローカル データベースの管理ガイドを参照してください。

  たとえば、各ワークフロー インスタンスとその子は、`WORKFLOWINSTANCE` テーブル内の行を作成します。これらのテーブルは、挿入および更新処理用に最適化する必要があります。このテーブルでの削除処理は、WebLogic Integration Studio を通じてバッチ処理されます。ワークフロー インスタンスを削除するためのバッチ削除処理については、削除処理に対応できるように、十分なサイズを持つロールバック セグメントをコンフィグレーションします。

- やり直しログ、アーカイブ ログ、およびデータベース テーブルをそれぞれ別のディスクに置きます。

- 一部のデータベースでは、Raw ディスク I/0 と標準ファイルシステム I/0 のどちらかを選択できます。通常、Raw ディスク I/O を選択すると、書き込みパフォーマンスが向上します。一方、ファイルシステム I/O を選択すると、OS レベルのキャッシングが実行されるため、読み込みパフォーマンスが向上します。このため、OLTP アプリケーションでは、Raw ディスク I/0 を使用することをお勧めします。ファイルシステム I/O は、意思決定支援アプリケーションで使用してください。Raw I/O を使用する場合は、データベース バッファのキャッシュ サイズを増やして、OS レベルのキャッシング不足を補う必要があります。

チェックポイント

チェックポイントは、不要なキャッシュ データを定期的にクリーン アップするための機能です。チェックポイントの実行時には、I/O アクティビティおよび使用されるシステム リソースが増加します。チェックポイントを頻繁に実行する

と、ディスク上のデータの整合性は増しますが、データベースのパフォーマンスは低下します。多くのデータベース システムにはチェックポイントの機能がありますが、すべてのデータベース システムがユーザレベルの制御を提供している訳ではありません。たとえば、Oracle の場合、システム管理者はチェックポイントの頻度を設定できますが、ユーザは SQLServer 7.X のチェックポイントは制御できません。推奨設定については、使用しているデータベースの製品マニュアルを参照してください。

データベースの互換性

推奨されているバージョンのクライアントおよびサーバのみを使用します。サポートされているデータベースのリストについては、使用している WebLogic Integration のリリースの『*WebLogic Integration リリース ノート*』にあるソフトウェア要件を参照してください。

データベースのモニタ

データベース使用の以下の面をモニタします。

- *ディスクの空き容量*—データベース用の領域が不足しないようにシステムをモニタします。主要なテーブル（`WORKFLOWINSTANCE` など）は、十分な領域が割り当てられていることをモニタする必要があります。領域のデフラグメンテーションのために定期的なテーブルの再編成をスケジューリングし、ディスクの空き領域を確保します。
- *パフォーマンス*—データベースに付属しているプロファイリングまたはモニタ ツールを使用して、ボトルネックを識別したり、パフォーマンスをチューニングするための推奨設定を取得したりすることができます。

Oracle データベースをチューニングする

この節では、Oracle 8.1.7 のパフォーマンスのチューニング方法について説明します。

V$ テーブル

Oracle 8.1.7 には V$ テーブルと呼ばれる一連の動的パフォーマンス ビューが搭載されているため、ユーザは SQL クエリを使用してシステム統計をモニタできます。このとき、ユーザは SYS または SYSTEM ユーザとしてデータベースにログインするか、これらの動的ビューにアクセスするための管理者権限が必要です。これらの動的ビューについては以下の節で説明します。動的ビューの詳細については、Oracle 管理者ガイドおよびチューニングガイドを参照してください。

初期化パラメータ

初期化パラメータ ファイル（init.ora）には、Oracle サーバの初期化パラメータとその値が記述されています。

Windows NT/2000 の場合、このファイルのパスは次のとおりです。

d:\oracle\admin*sid*\pfile\init.ora

このパス名において、*d:\oracle* はインストール先のディレクトリで、*sid* is データベースのインスタンス ID です（例：d:\Oracle\admin\hsundb\pfile\init.ora）。

このファイルは、PROCESSES = 100 のように、属性と値の組み合わせで編成されます。

ファイルを変更する前に、必ずバックアップを作成してください。変更内容を反映するには、サーバを再起動する必要があります。

サーバの再起動後、このファイルが変更されていることを確認してください。確認を行うには、SQL 文または SQL*Plus コマンドを使用します。パラメータとその値は、動的パフォーマンス ビュー V$PARAMETER に保存されます。

PROCESSES パラメータに対して行われた変更の有効性は、次のクエリを使用して検証します。属性名は、小文字で指定してください。

```
SELECT name, value FROM v$parameter WHERE name = ‘processes’
```

あるいは、SQL*Plus シェルで SHOW PARAMETERS *parameter_name* コマンドを使用します。たとえば、次のコマンド、

```
SHOW PARAMETERS “parameter”
```

は、次のクエリと同様の機能があります。

```
SELECT name, value FROM v$parameter WHERE name LIKE ‘%parameter%’;
```

パラメータを完全に理解したうえで、パラメータ値を変更してください。パラメータの詳細については、Oracle のマニュアルを参照してください。

共有プールのサイズ

共有プールは、Oracle サーバのシステムグローバル領域（SGA）の重要な部分です。SGA は共有メモリ構造のグループで、Oracle データベース インスタンスに関するデータと制御情報が格納されます。複数のユーザが同時に同一インスタンスに接続すると、そのインスタンスの SGA 内のデータがユーザ間で共有されます。

SGA の共有プール部分では、2 つの主要領域であるライブラリ キャッシュとディクショナリ キャッシュにデータが格納されます。ライブラリ キャッシュには、SQL 関連の情報および制御構造（例 : 解析済みの SQL 文、ロック）が格納されます。ディクショナリ キャッシュには、SQL 処理に必要なメタデータが格納されます。

ほとんどのアプリケーションにおいて、共有プールのサイズは Oracle システムのパフォーマンスに重要な影響を及ぼします。共有プールが小さすぎる場合、サーバは限られた使用可能領域の管理のためにリソースを確保する必要があります。Oracle ではさまざまなキャッシュの並行管理に制限があるため、CPU リソースが消費されて、リソースの競合が発生します。使用するトリガや保存手順が多いほど、大きな共有プールが必要です。

HARED_POOL_SIZE 初期化パラメータでは、共有プールのサイズをバイト単位で指定します。プロダクション システムの場合、この値を 9MB 以上にすることをお勧めします。75MB の共有プールを必要とするシステムも珍しくありません。共有プール内の空き容量をモニタするには、次のクエリを使用します。

```
SELECT * FROM v$sgastat
WHERE name = 'free memory' AND pool = 'shared pool';
```

共有プール内に常に空き容量がある場合、プール サイズを増やしてもほとんどメリットはありません。また、共有プールが一杯だからといって、必ず問題が発生するとは限りません。一旦共有プール内に格納されたデータは補助記憶装置に移すことができます。アプリケーションとデプロイメントの要件は異なる場合があるため、特定のデプロイメントおよびアプリケーションにもとづいて、この値を設定する必要があります。

最大オープン カーソル

Oracle サーバ内のすべてのリソースが 1 つの接続によって占有されるのを防ぐため、システム管理者は OPEN_CURSORS 初期化パラメータを使用して、接続ごとに最大オープン カーソル数を制限できます。このパラメータのデフォルト値は、WebLogic Server や WebLogic Integration などのシステムでは小さすぎます。175 から 255 の範囲内で値を指定することをお勧めします。カーソル情報は、次のクエリを使用してモニタできます。

```
SELECT name, value FROM v$sysstat
WHERE name LIKE 'opened cursor%';
```

最大プロセス数

多くのオペレーティング システムでは、Oracle サーバに接続するたびに、接続に対するシャドウ プロセスが発生します Oracle サーバが対応できる最大プロセス数は、同時接続ユーザ数および Oracle サーバによって使用されるバックグラウンド プロセス数によって決まります。多くの並行処理をサポートする必要のあるシステムの場合、デフォルト値では小さすぎます。200 から 255 の範囲内の値を指定することをお勧めします。固有の情報については、Oracle 管理者ガイドを参照してください。このパラメータの現行設定は、次のクエリを使用して確認できます。

```
SELECT name, value FROM v$parameter WHERE name = 'processes';
```

データベースのブロック サイズ

ブロックは、Oracle システムにデータを保存する際の基本単位で、I/O の最小単位です。1 つのデータ ブロックは、ディスク上にある物理データベース領域の特定バイト数に対応しています。ブロックに関するこの概念は、Oracle RDBMS 固有のものです。基盤となるオペレーティング システムのブロック サイズと混同しないでください。このブロック サイズは物理的な記憶域に影響を与えるため、この値を設定できるのはデータベースの作成時に限られます。データベースの作成後は変更できません。

WebLogic Integration リポジトリの特徴とアクセス パターンから想定すると、WebLogic Integration で使用するデータベースを作成する際、ブロック サイズを 8K にすることをお勧めします。このパラメータの現行設定は、次のクエリを使用して確認できます。

```
SELECT name, value FROM v$parameter WHERE name = 'db_block_size';
```

一般的なブロック サイズの利点と欠点を以下の表に示します。

ブロック サイズ	利点	欠点
2K-4K (小容量)	同一ブロック上で複数のトランザクションが動作する場合、ブロックの競合が減少する。データベースの列が小さい場合やランダム アクセスの回数が多い場合に適する。	I/O のオーバヘッドが比較的大きい。 列のサイズによっては、各ブロックに保存できる列の数が少なくなる。
8K (中容量)	列のサイズが中くらいの場合、1 回の I/O で複数の列をバッファ キャッシュに取り込むことができる。 ブロック サイズが小さい場合には、1 つの列しか取り込むことができない	ブロック サイズが大きく、小さい列にランダム アクセスを行う場合、Oracle バッファ キャッシュ内の領域が無駄になる。たとえば、ブロック サイズが 8KB で列のサイズが 50 バイトの場合、ランダム アクセスを行うと、バッファ キャッシュ内の 7,950 バイトが使用されない。
16K-32K (大容量)	オーバヘッドが比較的少ないため、多くのデータ保存領域を確保できる。順次アクセスを行う場合や列のサイズが大きい場合に適する。	索引のリーフ ブロックで競合が多く発生するため、大容量ブロックは OLTP タイプの環境で使用される索引ブロックには適さない。

システム管理者向けのオプションをチューニングする

この節で説明するチューニング手順は、該当するシステムに精通したシステム管理者またはユーザが実行する必要があります。

警告 : この節で説明するチューニング オプションすべてを使用してもパフォーマンスが向上するとは限りません。経験的アプローチでパラメータ値を指定しなければならない場合があります

SNP プロセス

デフォルトで、Oracle サーバにはスケジュール済みのタスクを実行するためのバックグラウンド処理がいくつか作成されています。これらのタスクは、Job Queue 機能または Advanced Replication 機能を使用した場合にのみスケジュールできます（詳細は、Oracle のマニュアルを参照）。これらの Oracle 機能を使用し

ない場合は、バックグラウンド処理はリソースを無駄に消費します。これらのプロセスを、実際に必要になるまで無効にするには、init.ora ファイルを変更します。

init.ora ファイルで以下のセクションをコメント行にするのが、最も安全な方法です。

```
# The following parameters are needed for the Advanced Replication
#Option

#job_queue_processes = 4
#job_queue_interval = 10
```

ソート領域のサイズ

ソート領域を拡張すると、クエリの実行時にメモリ内でソートを実行できるため、大容量データのソート処理のパフォーマンスが向上します。どの時点においても、各接続で使用されるソート領域は 1 つしかないため、ソート領域の拡張は重要です。init.ora パラメータのデフォルト値は、通常 6 ～ 8 個のデータ ブロックのサイズになります。OLCP 処理の場合は、この値で十分ですが、意思決定支援処理、大容量データの一括処理、および大容量の索引関連処理（索引の再作成など）の場合は、この値を増やす必要があります。これらのタイプの処理を実行する際、次の init.ora パラメータをチューニングしてください（現行では、8K のデータ ブロックが設定されている）。

```
sort_area_size = 65536
sort_area_retained_size = 65536
```

テーブルの物理記憶域パラメータ

挿入、更新、削除処理によって、データベース テーブルのサイズは拡張および縮小されます。テーブルが拡張されると追加 I/O が発生するため、データベースの処理速度は低下します。このため、予想されるアクセス数および使用パターンに応じて、各テーブルの物理記憶域のパラメータを設定する必要があります。したがって、パラメータは、主に、テーブルを使用するアプリケーション次第で決まることになります。通常、Oracle システムで使用されているデフォルト値のままでも特に支障はありませんが、パラメータをチューニングすることで、パフォーマンスを飛躍的に向上させることができるインスタンスが数多くあります。この作業は、Oracle RDBMS に精通した DBA のエキスパートが実行してください。以下の節では、スキーマ オブジェクトに共通する保存域パラメータの中でも特に CREATE TABLE コマンドにとって重要なパラメータについて説明します。これらのパラメータの推奨値については、このマニュアルでは扱いません（詳細については、Oracle マニュアルまたは DBA を参照）。ここでは、問題点を事前にチェックする際に便利なパラメータとクエリをいくつか紹介します。

- INITRANS and MAXTRANS

  トランザクションによってブロックに変更が加えられると、ブロックのヘッダにフラグが付けられます。トランザクションのコミット時に、このフラグは削除されます。フラグによってブロック内の領域が使用されるため、トランザクションのフラグが増えるとデータの保存領域が減ります。フラグがない場合、トランザクションはブロックへの変更が許可されず、処理を待機する必要があります。Oracle では、ユーザがテーブルごとにブロック当たりのフラグ数を制御できます（更に細かい制御を行うことができるテーブルもありますが、そのような制御については、このマニュアルでは扱わない）。NITRANS パラメータを使用すると、各ブロックに割り当てる初期フラグ数を指定できます（最小値は 1）。MAXTRANS で指定した数まで、フラグを追加できます。空きフラグがない場合、トランザクションはブロックされます。トランザクションがブロックされると、デッドロックが発生する可能性が高くなります（つまり、トランザクションは処理を完了できず、ロックされたリソースが解放されるまで待機する）。MAXTRANS のデフォルト値は 255 ですが、次のクエリを使用して、OLTP に含まれるテーブルに適した値がパラメータに指定されているかを確認してください。

  ```
  SELECT owner, table_name, ini_trans, max_trans, FROM all_tables;
  ```

  各ワークフローはアプリケーションのライフサイクルの中で一連のトランザクションを WORKFLOWINSTANCE に対して実行するので、これらの設定はアプリケーションが同時に複数のワークフローに関与する場合に重要となります。

- MINEXTENTS and MAXEXTENTS

  これらのパラメータによって、テーブルの拡張および縮小が制御されます。拡張テーブルは、任意の数のデータ ブロックから構成されます（「データベースのブロック サイズ」 を参照）。これらのパラメータによって、テーブルの作成時に割り当てられる拡張テーブルの数（テーブルのサイズは、MINEXTENTS で指定されている値より小さくはできない）、および 1 つのテーブルに割り当てられる拡張ブロックの最大数が制御されます。通常、テーブルを作成する際に、次の設定オプションを使用します。

  ```
  CREATE TABLE foo (col1 number, col2 date)
  STORAGE (MINEXTENTS 1 MAXEXTENTS UNLIMITED);
  ```

  これらのパラメータ値を確認するには、次のクエリを使用します。

  ```
  SELECT owner, table_name, min_extents, max_extents
  FROM all_tables;
  ```

MAXEXTENTS で UNLIMITED オプションを指定すると、クエリによって返される値が非常に大きな整数（2147483645 など）となります。

Redo ログをスワップする

回復処理をサポートするため、Oracle RDBMS に対して実行されるすべての処理は、Redo ログに記録されます（特定の処理に対するロギングが明示的に無効になっている場合を除く）。時間の経過とともに、ログ内の情報量が増え、最終的には処理のパフォーマンスに影響を与え始めます。回復処理はバックアップデータを使用して実行できるため、データベースのバックアップを実行した直後は、Redo ログ内の情報は必要なくなります。このため、バックアップ後に不要になった情報をクリーン アップして 新しい Redo ログを開始し、システム パフォーマンスを回復することをお勧めします。この処理を実行するには、次の SQL コマンドを使用します。

```
ALTER SYSTEM SWITCH LOGFILE
```

Redo ロギング、Redo ログとログ グループの管理、および RDBMS のバックアップの最適な実行方法の詳細については、Oracle のマニュアルを参照してください。

テーブルの再編成

SQL 処理（OLTP および Bulk）によってテーブルが拡張または縮小される際、テーブルの保存領域が断片化する場合があります。これによってパフォーマンスが低下するため、領域のギャップを回収してテーブル データをひとまとめにするユーザ介入処理が必要になります。通常、この処理はテーブルの再編成と呼ばれます。Oracle 8.1.7 には、この処理をサポートする機能が組み込まれていないため、ユーザはこの処理を手動で行う必要があります。推奨手順に従って、データベースのバックアップ直後に、この処理を実行してください。テーブル foo の再編成手順は、次のとおりです。

1. 次の SQL 文を使用して、テーブルをコピーします。

   ```
   CREATE TABLE foo_bkup AS SELECT * FROM FOO;
   ```

   これは新しいテーブルで、回収する領域がないため、データのコピー処理によって、データの再編成が実行されます。

2. 次の SQL 文を使用して、旧テーブルを削除します。

   ```
   DELETE TABLE foo;
   ```

3. 新しいテーブル名を、次の SQL 文を使用して旧テーブル名に変更します。

```
RENAME foo_bkup TO foo
```

プロセスの各手順には、DDL 文（CREATE TABLE、DROP TABLE など）が含まれます。DDL statements are not transactional in Oracle. 厳密には、DDL 文は完全に独立したトランザクションの中で実行されます。このため、テーブルの再編成中は ROLLBACK コマンドは実行されません。

Microsoft SQL Server データベースをチューニングする

次の表では、Microsoft SQL Server データベースに固有のパフォーマンス チューニング パラメータについて説明します。パラメータの詳細については、Microsoft SQL Server のマニュアルを参照してください。

表 6-4 Microsoft SQL Server データベースのチューニング パラメータ

パラメータ	推奨設定
Tempdb	tempdb を高速の I/O デバイスに格納する。
回復間隔	perfmon によって I/O が増加する場合は、回復間隔を長くする。
I/O ブロック サイズ	2Kb より大きな I/O ブロック サイズを使用する。

Sybase データベースをチューニングする

次の表では、Sybase データベースに固有のパフォーマンス チューニング パラメータについて説明します。パラメータの詳細については、Sybase のマニュアルを参照してください。

表 6-5 Sybase データベースのパフォーマンス チューニング パラメータ

パラメータ	推奨設定
回復間隔	回復間隔を短くすると、チェックポイント処理が多くなるので、I/O 処理が増える。

表 6-5 Sybase データベースのパフォーマンス チューニング パラメータ (続き)

パラメータ	推奨設定
I/O ブロック サイズ	2KB より大きな I/O ブロック サイズを使用する。
最大オンライン エンジン	対称マルチプロセッサ（SMP）環境のエンジン数を制御する。Sybase では、CPU 数から 1 を引いた数に設定することを推奨している。

A WebLogic Integration クライアント アプリケーションのデプロイメント

WebLogic Integration には、クライアント アプリケーションの開発に役立つアプリケーション プログラミング インタフェース（API）が備わっています。WebLogic Integration API が使用されているクライアント アプリケーションでは、WebLogic Integration クラスのインポート、エンタープライズ JavaBean (EJB) の検索などができます。

この節では、WebLogic Integration クライアントのデプロイメントを正常に行うために役立つ情報を提供します。

JAR ファイル

クライアントの CLASSPATH で、次の JAR ファイルを指定します。

- `%BEA_HOME%\integration\lib\wliclient.jar` ― EJB リモート インタフェース、ホーム クラス、API を公開した EJB に対する署名クラスを格納します。
- `%BEA_HOME%\integration\lib\wlicommon.jar` ―サーバおよびクライアントに共通のクラスを格納します。

ここで、`BEA_HOME` は、WebLogic Platform がインストールされれているディレクトリを表す環境変数です。

要件および推奨事項

クライアント アプリケーションをデプロイする場合は、前の節の説明に従って JAR ファイルを指定する他、以下の要件および推奨事項も考慮してください。

- クライアントが簡単な Java アプリケーションである場合は、システムの CLASSPATH で `wliclient.jar` and `wlicommon.jar` を指定できます。
- WebLogic Integration と同じ Java 仮想マシン（JVM）を実行するエンタープライズ アプリケーションをデプロイする場合は、manifest クラスパスに JAR ファイルを指定する必要があります。

  J2EE 仕様に、クラスの補助 JAR ファイルを必要とすることを明示するコンポーネントに対して、`Class-Path` エントリが用意されています。作成したアプリケーションに対して、JAR または WAR ファイルを作成する場合、必要な JAR ファイルを参照する `Class-Path` 要素とともにマニフェストファイルも組み込みます。この manifect ファイルの使い方については、次の URL にある『*WebLogic Server 開発者ガイド*』の「WebLogic Server J2EE アプリケーション クラスローディング」を参照してください。

  `http://edocs.beasys.co.jp/e-docs/wls/docs70/programming/classloading.html`
- これは必須要件ではありませんが、WebLogic Integration と WebLogic Integration クライアントアプリケーションが別々の JVM で実行されるデプロイメント シナリオでは、`wliclient.jar` ファイルおよび `wlicommon.jar` ファイルを、manifest `Class-Path` で指定することをお勧めします。これらの JAR ファイルを manifest Class-Path でリストすることにより、両方のアプリケーションが完全に独立して実行できることを保証できます。
- JAR ファイルは、サーバとクライアントで同一バージョンを維持する必要があります。

BPM クライアント アプリケーションの開発方法については、『*BPM クライアント アプリケーション プログラミング*』を参照してください。

B　リソース アダプタのデプロイ

この節では、ソース アダプタをクラスタ内のサーバを起動した後でデプロイする方法について説明します。クラスタ デプロイメントの設定および起動に関する情報、WebLogic Integration ドメインでデフォルトでデプロイされるアダプタについては、第 3 章「クラスタ デプロイメントのコンフィグレーション」を参照してください。

クラスタでサーバを起動後、以下の方法のいずれかを使用してリソース アダプタをデプロイすることができます。

- weblogic.Deployer コマンドライン ユーティリティの使用法
- WebLogic Server Administration Console の使用法

weblogic.Deployer コマンドライン ユーティリティの使用法

weblogic.Deployer ユーティリティは、Java ベースのデプロイメント ツールで、WebLogic Server デプロイメント API に対するコマンドライン インタフェースを提供します。詳細は、『*WebLogic Server 開発者ガイド*』、「WebLogic Server デプロイメント」の「デプロイメント ツールおよび手順」を参照してください。

`http://edocs.beasys.co.jp/e-docs/wls/docs70/programming/deploying.html`

サンプル DBMS アダプタのデプロイ

次の例では、WebLogic Integration ソフトウェアに付属のサンプルの DBMS アダプタを MyCluster というクラスタにデプロイする方法を例示します。このクラスタには、MyServer1 と MyServer2 という 2 つの管理対象サーバが含まれています。次の表では、クラスタ コンフィグレーションについて説明します。

サーバ名	サーバの種類	リスン アドレス : ポート
MyAdmin	管理サーバ	127.0.0.5:7005
MyServer1	Managed Server	127.0.0.1:7001
MyServer2	管理対象サーバ	127.0.0.1:7002

次のコマンドを使って、この例のクラスタに DBMS アダプタをデプロイします。

注意 : このコードは、1 つのコマンドを表します。この例では複数の行に分けて読みやすくしてあります。ただし、実際のコマンドラインでは 1 行で入力されます。

コード リスト B-1 DBMS アダプタのデプロイメントに使用する、weblogic.Deployer コマンドライン

```
java -classpath WL_HOME\lib\weblogic.jar weblogic.Deployer
-adminurl t3://127.0.0.5:7005 -user username -password password
-upload -stage
-source WLI_HOME\adapters\dbms\lib\BEA_WLS_DBMS_ADK.ear
-name BEA_WLS_DBMS_ADK
-targets BEA_WLS_DBMS_ADK.rar@MyCluster,
BEA_WLS_DBMS_ADK_Web.war@MyCluster,
BEA_WLS_DBMS_ADK_EventRouter.war@MyServer1
-activate
```

上記のコマンド ラインの説明 :

- `-adminurl` ―クラスタ内の管理サーバの URL を指定します。
- `-user` ―管理サーバが認証に使用するユーザ名を指定します。

- `-password` ー管理サーバが認証に使用するユーザ名を指定します。
- `-upload` ー EAR ファイルを管理サーバにアップロードします。管理サーバで `weblogic.Deployer` ユーティリティを実行する場合は、このオプションは省略可能です。ただし、`weblogic.Deployer` ユーティリティを管理サーバで実行しない場合は、このオプションは必須です。
- `-stage` ー WebLogic Server デプロイメント機能に対して、すべての管理対象サーバに対する EAR ファイルのステージングをサーバの起動前に実行するよう指示します。
- `-source` ーリソース アダプタ用の ERA ファイルの場所を指定します。(*`WLI_HOME`* は、WebLogic Integration をインストールしたディレクトリを表します（例：`C:\bea\weblogic700\integration`）。
- `-name` ーリソースアダプタ用のエンタープライズ アプリケーションの名前を指定します。アダプタの論理名と同じ名前です。これは、リソース アダプタの一意の識別子です。
- `-targets` ー先にアダプタに対して定義しておいた ERA ファイルに格納されているサブコンポーネントを指定する。

  これは、サブコンポーネントのカンマ区切りリストです（リストにあるアイテム間にスペースがないことに注意）。2-32 ページの「アダプタのデプロイ」で説明したように、サンプル アダプタ用イベント ルータ WAR ファイルは、クラスタ内の 1 つのノードにデプロイする必要があります。このサンプル コマンドは、RAR と設計時 Web アプリケーションがクラスタにデプロイされること、また、EventRouter Web アプリケーションは所定の管理対象サーバにデプロイメントされることを指定します。
- `-activate` ーこのドメインでアプリケーションを起動します。

WebLogic Server Administration Console の使用法

1. 管理サーバと Administration Console を起動します。

   a. 管理サーバを起動するには、『*WebLogic Integration の起動、停止およびカスタマイズ*』、「はじめに」の「WebLogic Integration の起動」を参照してください。

   b. コンソールを起動するには、『*WebLogic Integration の起動、停止およびカスタマイズ*』、「WebLogic Integration 管理ツールと設計ツール」の「WebLogic Server Administration Console の起動」を参照してください。

2. Administration Console のナビゲーション ツリーで、アダプタをデプロイするドメインの [アプリケーション] ノードを選択します。

   [*Domain_Name* | デプロイメント | アプリケーション]

3. [新しい Application のコンフィグレーション] をクリックします。

   メイン コンソール ウィンドウに WebLogic Server ウィザードが表示されます。ウィザードに従って、アダプタのデプロイメント プロセスを進めます。

4. WebLogic Server と併用する EAR、WAR、JAR、または RAR ファイルを見つけます。たとえば、WebLogic Integration ソフトウェアに付属の、サンプルの DBMS アダプタをデプロイする場合は、次のディレクトリにある `BEA_WLS_DBMS_ADK.ear` ファイルを選択します。

   ```
   WLI_HOME\adapters\dbms\lib\BEA_WLS_DBMS_ADK.ear
   ```

   ここで、*`WLI_HOME`* は、WebLogic Integration をインストールしたディレクトリ、たとえば、`C:\bea\weblogic700\integration` を表します。

   注意： アプリケーションまたはコンポーネントのディレクトリを展開した形でコンフィグレーションすると、WebLogic Server は、指定されたディレクトリ内およびそのディレクトリ下にあるすべてのコンポーネントをデプロイします。

5. ウィザードのプロンプトに従って、コンフィグレーションおよびデプロイメントを完了します。たとえば、ターゲットとステージング モードを指定する必要があります。詳細は、B-1 ページの「weblogic.Deployer コマンドライン ユーティリティの使用法」の `-targets` と `-stage` オプションを参照してください。

WebLogic Server Administration Console を使用してアプリケーションをデプロイする方法の詳細については、次の URL にある『*Adminstration Console オンライン ヘルプ*』、「アプリケーション」の「コンフィグレーションおよびデプロイメント タスク」を参照してください。

`http://edocs.beasys.co.jp/e-docs/wls/docs70/ConsoleHelp/applications.html`

索引

L

M

N

O

P

R

S

T

V

W

X

あ

い

え

お

さ

し

す

せ

そ

る

ろ

わ