5 アプリケーション管理 3

5-1 WAS V8.5 アプリケーション管理概要 3
5-1-1 インテリジェント管理によるアプリケーションのエディション管理 3
5-1-1-1 複数エディションの管理 4
5-1-1-2 複数エディションの同時稼働 5
5-1-1-3 エディションのロールアウトと妥当性検査 5
5-1-2 ビジネス・レベル・アプリケーションの管理モデル 6
5-1-2-1 アセット 6
5-1-2-2 構成単位（CU） 7
5-2 アプリケーションの導入 8
5-2-1 アプリケーション導入の流れ 8
5-2-1-1 リソースの定義 10
5-2-1-2 共用ライブラリーの定義 18
5-2-1-3 ファイルの同期 22
5-2-1-4 ファイルの配布 25
5-2-1-5 プラグインの更新 25
5-2-1-6 アプリケーションの導入手順 28
5-2-1-7 Enhanced EAR 42
5-2-2 ビジネス･レベル・アプリケーション導入の流れ 45
5-2-2-1 ビジネス･レベル・アプリケーションの作成 45
5-2-2-2 アセットのインポート 46
5-2-2-3 構成単位の登録 47
5-2-2-4 ビジネス・レベル・アプリケーションにおけるファイルの配布 50
5-3 アプリケーションの管理 52
5-3-1 Fine-Grained Update 52
5-3-1-1 Fine-Grained Update の設定手順 53
5-3-2 ロールアウト・アップデート 58
5-3-2-1 ロールアウト・アップデート処理の流れ 59
5-3-2-2 ロールアウト・アップデートの設定手順 60
5-3-2-3 ロールアウト・アップデート使用時の注意点 60
5-3-3 複数エディションの同時稼働 63
5-3-4 エディションのロールアウトと妥当性検査 66
5-3-4-1 エディションのロールアウトの計画 66
5-3-4-2 エディションのロールアウトと妥当性検査の手順 69
5-3-4-3 妥当性検査の手順 71
5-3-4-4 妥当性検査の考慮点 74
5-3-4-5 ロールアウト・アップデートとエディションのロールアウト 74
5-4 運用パターン 75
5-4-1 アプリケーション導入のケース 75
5-4-2 アプリケーション更新のケース 75
5-5 クラス・ローダー 80
5-5-1 クラス・ローダーの種類と階層構造 80
5-5-2 クラス・ローダー・ポリシー 82
5-5-2-1 アプリケーション・クラス・ローダー・ポリシー 82
5-5-2-2 WAR クラス・ローダー・ポリシー 83

5-5-3 クラス・ローダー・モード 86

5 アプリケーション管理

アプリケーションの導入、更新及び削除などアプリケーション管理業務はシステムの日常運用において欠かせない業務です。サービスへの影響を最小限に、アプリケーション管理を実現することはシステム管理者だけではなく、システムの設計者にとっても重要な課題です。この章ではまず WebSphere Application Server (以下、WAS) V8.5 で新しく採用されたアプリケーション管理モデルである、エディションの概念を説明し、アプリケーションの導入、更新及び削除の流れと具体的な手順を説明します。次にシステム構築においてよく見られる要件を取り上げて、ケース別にアプリケーション管理の考慮点について説明します。

5-1 WAS V8.5 アプリケーション管理概要

WAS V8.5 では、以前のバージョンから存在する Java EE アプリケーションの管理モデル、WAS V7.0 で導入されたビジネス・レベル・アプリケーション単位での管理モデルに加え、新たにアプリケーションのエディション管理が加わりました。これにより、同一のアプリケーションの複数のエディションをデプロイし、異なるターゲット上で複数のエディションを同時に稼働させたり、稼働させるエディションをサービス無停止で切り替えたりする事が出来るようになりました。

エディション管理を使用するためには、インテリジェント管理を使用し、オンデマンド・ルーター(ODR)(1.1.3.8 参照)を構成する必要があります。この章では必要に応じ、インテリジェント管理 / ODR を使用するケースと使用しないケースに分けて説明します。

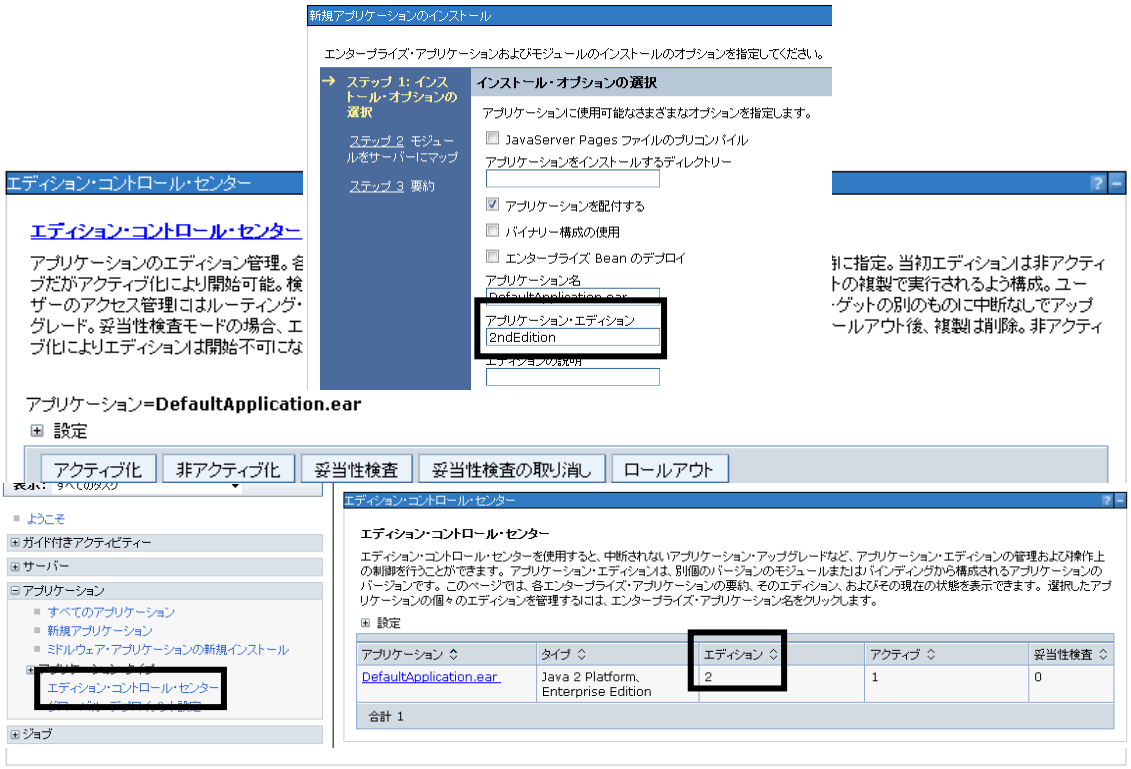
また、WAS V6.1 以前にも存在したアプリケーション管理簡略化のための機能（Enhanced EAR、Fine-Grained Update、ロールアウト・アップデート）は、WAS V8.5 でも提供されています。Enhanced EAR とは、アプリケーションが稼動するためにサーバー側で必要となる情報をアプリケーションと一緒にアーカイブしたファイルのことです。アプリケーションと一緒にアーカイブされるのは、リソース定義情報やプロパティー設定情報などです。開発ツールから Enhanced EAR をエクスポートしてサーバーにデプロイすれば、サーバー側でアプリケーションの稼動に必要なリソース定義などを行う必要がなくなります。Fine-Grained Update とは、アプリケーション（ear ファイル）全体ではなく、モジュールやファイルといった細かい単位で部分的に追加、更新、削除が行える機能です。ロールアウト・アップデート とは、クラスター上のアプリケーションを更新する際、全てのクラスター・メンバーに同時に反映させるのではなく、ノード毎に順番に自動反映させていく機能です。

5-1-1 インテリジェント管理によるアプリケーションのエディション管理

WAS V8.5 での大きな変更として、これまで WebSphere Virtual Enterprise(WVE)という別の製品で提供されていた機能がインテリジェント管理として WAS ND に統合され、WAS ND のライセンスで WVE の機能を使用することができるようになりました。
ここでは WVE / IMP の持つ機能のうち、アプリケーション管理に関わる部分について説明します。

5-1-1-1 複数エディションの管理

WAS V8.5 以降では IMP の導入により、アプリケーション(セル内で唯一の名前を持ち、配下の WAR モジュールがそれぞれセル内で唯一のコンテキスト・ルートを持つ)の複数のエディションを管理することができるようになりました。このガイドでは InfoCenter などの記述に従って「エディション」という用語を用いますが、一般にはアプリケーションの「バージョン」管理といった方がわかりやすいかもしれません。
あるアプリケーションの別エディションを導入するには、導入の際に「アプリケーション・エディション」を指定します。



DefaultApplication.ear アプリケーションのエディション数が 2 になっています。DefaultApplication.ear リンクをクリックしてみます。



図 5-1 エディション・コントロール・センターのエディションの管理画面

エディションが Base Edition となっているのは、あらかじめエディションを指定せずに導入しておいた DefaultApplication.ear アプリケーションです。エディションを指定せずにアプリケーションをインストールした場合、デフォルトで Base Edition というエディションになります。
その上にあるのが、エディションを 2ndEdition として先ほどインストールした DefaultApplication.ear アプリケーションです。
このスクリーン・ショットにはエディション列の他に「アクティブ化」「非アクティブ化」「妥当性検査」「妥当性検査の取り消し」「ロールアウト」といったボタンや、「ターゲット」「状態」の列があります。
これらがそれぞれ何であってどのように使用されるのか、以下で説明します。

5-1-1-2 複数エディションの同時稼働

同一アプリケーションの複数のエディションは、ターゲット(マップ先のアプリケーション・サーバーやクラスター)が異なっていれば、同時に稼働させることが可能です。例えば、一般ユーザーは安定版エディションへルーティングさせ、テスト・ユーザーはパイロット版エディションへルーティングさせるといったことが可能になります。
複数のエディションを同時に稼働させるためには、それらのエディションを共にアクティブ化することが必要です。ただし、それらのエディションのターゲットを分けてアクティブ化しただけでは、どのリクエストをどのエディションにルーティングするかといった問題が残ります。
リクエストの各エディションへの適切なルーティングは ODR が行います。どのリクエストをどのエディションにルーティングすればよいかは、ルーティング・ルールを設定することで ODR が判断することができるようになります。

5-1-1-3 エディションのロールアウトと妥当性検査

エディションのロールアウトにより、ユーザー(ブラウザー)に対してのサービスを無停止のまま、アプリケーションのエディションを切り替えることが可能になります。
同様の機能に WAS 8.5 以前から存在するロールアウト・アップデートがありますが、エディションのロールアウトではより洗練された仕組みにより、サービス無停止でのアプリケーション更新をより短時間に、かつ整合性を保ちつつ行うことが可能になっています。万一、ロールアウト後に新エディションの不具合が見つかった場合には、旧エディションへロールアウトすることでの迅速な戻し操作も可能になっています。
また、エディションのロールアウトを実行する前に、あらかじめ本番用クラスターと同等の環境でテストを行うための妥当性検査を使用することができます。エディション・コントロール・センターの「妥当性検査」

ボタンを押すと、本番用クラスターの構成をコピーして妥当性検査クラスターが自動的に作成され、ロールアウト先エディションのアプリケーションも自動的にデプロイされます。このことにより、本番用クラスターで現エディションを稼働させたまま、ロールアウト先エディションのテストを行うことが可能になります。テストの結果に問題がなければ、エディション・コントロール・センターの「ロールアウト」ボタンを押すことで、本番用クラスターでのエディションのロールアウトが実行され、妥当性検査クラスターは自動的に停止・削除されます。もしテストの結果、ロールアウト先エディションに問題が見つかれば、「妥当性検査の取り消し」ボタンを押すことで妥当性検査クラスターが自動的に停止・削除され、妥当性検査開始前の状態に戻すことができます。

5-1-2 ビジネス・レベル・アプリケーションの管理モデル

WAS V7.0 では新たにビジネス・レベル・アプリケーションの概念が登場しました。ビジネス・レベル・アプリケーションとは、WAS の上で稼動する単一の Java EE アプリケーション・モジュールだけではなく、そのビジネスに必要な複数のコンテンツを同時に管理するための管理モデルです。セルに予め登録した複数のコンテンツ（ビジネス・レベル・アプリケーションの管理モデルでは「アセット」と呼ばれます）の中から任意のモジュールを「構成単位」として選択し、組み合わせる事で、ビジネス・レベル・アプリケーションを構成します。

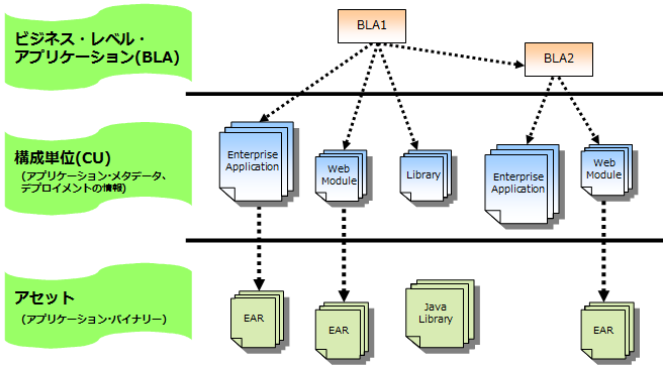


図 5-2 ビジネス・レベル・アプリケーションの管理モデル

WAS V8.5 でも全ての Java EE アプリケーションがこの管理モデルで管理されます。従来と同じ方法で Java EE アプリケーションをインストールした場合も、この管理モデルに従い、その Java EE アプリケーションを構成単位とするビジネス・レベル・アプリケーションが自動的に作成されます。

ビジネス・レベル・アプリケーションに登録された Java EE コンテンツであっても、既存の Java EE アプリケーションの管理 API や各リソースに対応する MBean（WAS 内部で実装されている、WAS のリソース管理用の JavaBean）は継続して動作しています。例えば、ビジネス・レベル・アプリケーションの Java EE 構成単位に対して始動・停止などの制御操作を行った場合は、Java EE アプリケーションの始動・停止を行う Application Manager MBean に委任されて実行される事になります。

以下にビジネス・レベル・アプリケーションの構成要素であるアセットと構成単位について記載します。

5-1-2-1 アセット

アセットは 1 つ以上のアプリケーション・バイナリー・ファイルを表します。アセットの登録時に、物理アプリケーション・ファイルを製品構成リポジトリーにインポートするか、またはアセットがある外部ロケーションを指定することができます。インポートした場合、アセットのバイナリー・ファイルはデフォルトでは以下の場所に保管されます。

<profile_root>/config/cells/*cell_name*/applications/*asset_name*
<profile_root>/config/cells/*cell_name*/assets/*asset_name*/aver/BASE/bin/
また、登録と同時に asset.xml ファイルが作成され、名前、宛先ロケーション、および他のアセットとの依存性などのアセットに関する情報がこのファイルにまとめられます。

アセットの例としては圧縮(zip)ファイル、EJB Java アーカイブ（jar）ファイル、EAR ファイル、サービス・コンポーネント・アーキテクチャー(SCA）複合 JAR ファイル、メディエーション JAR ファイル、共用ライブラリーJAR ファイル、PHP アプリケーションなどの非 Java EE コンテンツがあります。アセット名は、セル、製品管理ドメイン内で固有である必要がありますので注意して下さい。

5-1-2-2 構成単位（CU）

構成単位は、ビジネス・レベル・アプリケーションに含まれているコンテンツを表します。構成単位として登録されたアセット・コンテンツは、ビジネス・レベル・アプリケーション内の他のアセット・コンテンツとの関係性を定義したり、他のアセット・コンテンツと同時に始動したりするなどの制御が可能になります。
構成単位としては、以下の 3 つのタイプがサポートされています。

- アセット構成単位
  インポートされたアセットから作成された構成単位
- 共用ライブラリー構成単位
  クラスのライブラリーとして取り扱われる、JAR ベースのアセットから作成された構成単位
- ビジネス・レベル・アプリケーション構成単位
  既に作成済みのビジネス・レベル・アプリケーションから作成された構成単位

構成単位には、以下の情報が含まれます。
- アセット・コンテンツと特定のホスティング・ランタイムをバインドし、ランタイムがアセットをロードして実行するために必要な構成情報
- アセットが使用する外部サービス、コンポーネント、その他のリソースへの参照
- サービス定義、参照、およびその他の関連構成データ用にカスタマイズされた構成
- デプロイメント・ターゲットまたはランタイム環境のリスト、および構成単位が実行されるランタイム環境に固有の構成

例えば、エンタープライズ Bean (EJB) Java アーカイブ (JAR) アセット用の構成単位は、EJB モジュール・インスタンスです。このインスタンスには、EJB Java Naming and Directory Interface (JNDI) 名、ejb-ref 解決、および EJB JAR が実行されるアプリケーション・サーバーまたはクラスターのリストなど、必要な EJB バインディング情報が含まれています。

構成単位には下記のようなルールが適用されます。
- 構成単位は、ビジネス・レベル・アプリケーション内にのみ存在することができます。
- 複数のビジネス・レベル・アプリケーションで単一のアセット構成単位または共用ライブラリー構成単位を共有することはできません。
- 製品は、1 つのアセットに対してのみ構成単位を作成します。ただし、複数の構成単位で単一のアセットを共有することができます。これは、異なる構成で同じアプリケーションのバイナリー・ファイルを使用している場合に、特に有効です。

5-2 アプリケーションの導入

図 5-3 アプリケーション導入フローは、セル構成においてアプリケーションを稼動させるために管理者に必要な作業の流れを表しています。前提として、リモート Web サーバー構成を持つセル環境に、データベースを使用するアプリケーションを稼動させるものとします。

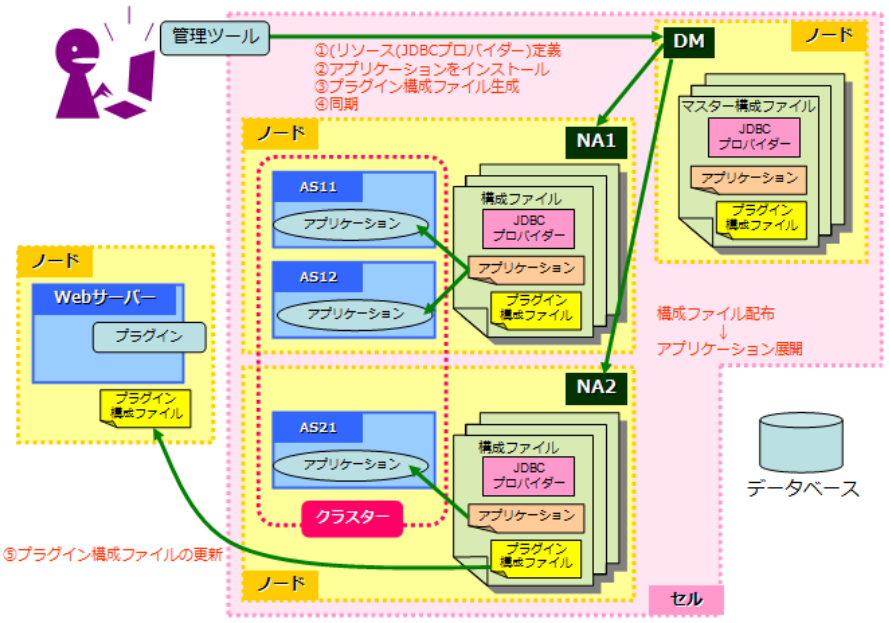


図 5-3 アプリケーション導入フロー

5-2-1 アプリケーション導入の流れ

ここからは、セル構成においてアプリケーションを稼動させるために管理者に必要な作業の流れを、実際のインストールの画面での作業手順を通して解説します。

1) まず、管理コンソールで JDBC プロバイダーなどのリソースを定義し、アプリケーションが使用するデータ・ソースや接続先データベースなどの情報を設定します。その後、アプリケーションをインストールします。この時点では、全ての構成情報はデプロイメント・マネージャー(DM)のマスター構成ファイルだけに格納されています。

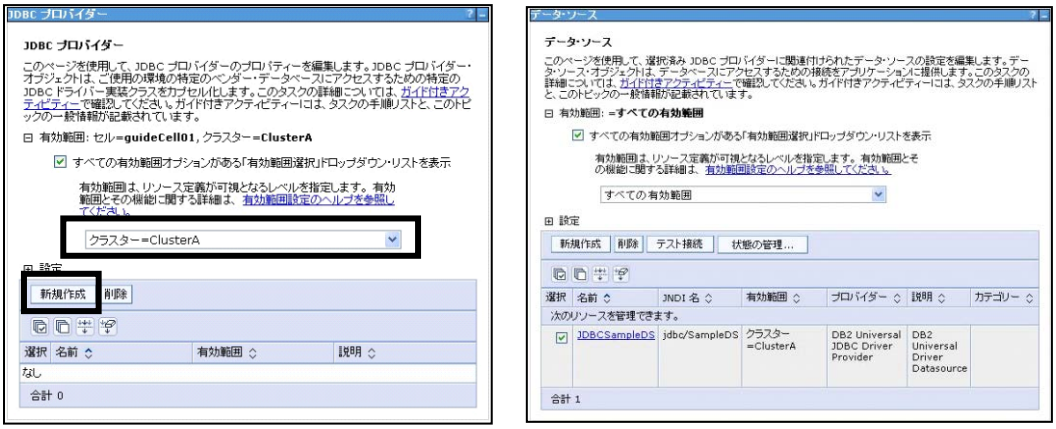


図 5-4 JDBC プロバイダー・データソース設定

2) 次に、デプロイメント・マネージャー・ノードと各管理対象ノードの同期を行います。このタイミングで、デプロイメント・マネージャーが保持している構成ファイルやアプリケーション・ファイルが各管理対象ノードに配布され、アプリケーションは配布先ノード上で展開されます。

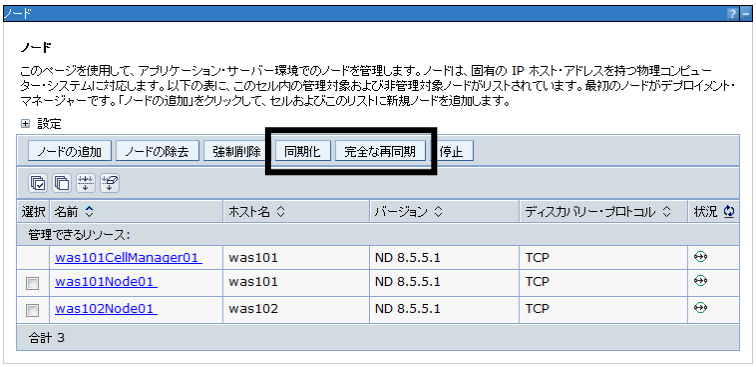


図 5-5 ノードの同期状態管理画面

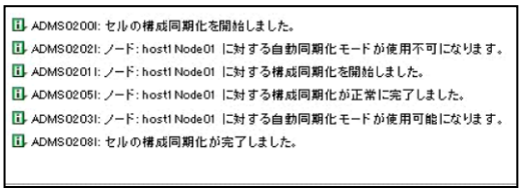


図 5-6 ノード同期開始メッセージ

3) 最後に、デプロイメント・マネージャー・ノード上あるいは各管理対象ノード上にある最新のプラグイン構成ファイルを生成、伝搬します。ODR を使用しない場合は図 5-7のように、管理コンソール上でプラグイン構成ファイルの生成・伝搬を行うことが可能です。ODR を使用する場合は、ODR または高可用性プラグイン構成生成サービス(HAPluginCfgGenerator) でプラグイン構成ファイルを生成するように構成します。また、プラグイン構成ファイルの伝搬は、シェル・スクリプト(Windows プラットフォームではバッチ・ファイル)に伝搬のための処理を記述します。詳しい手順は 2-2-1-5 節にて後述します。



図 5-8 Web サーバー管理画面

以上の作業を行うことで、Web クライアントからのリクエストを Web サーバーが受け付け、プラグインがプラグイン構成ファイルを参照して適切なアプリケーション・サーバー(AS)または ODR に割り振り、AS ではリソース定義情報をもとにデータベース接続が行われるようになります。

ODR は WebSphere プロキシー・サーバー(1.1.3.6 参照)の拡張である *ため、ODC に基づくワークロード管理を行うことができます。

これからの節では、上記流れの各ステップの詳細な構成手順について説明します。

5-2-1-1 リソースの定義

WAS 上でアプリケーションを稼動させるために必要な作業には、アプリケーションのデプロイ(インストール)とリソース定義があります。

リソース定義とは、アプリケーションがデータベースやメッセージングなどといったリソースを使用する場合に、アプリケーションとは別にサーバーに必要な情報を設定しておくことです。WAS では、JDBC プロバイダーや JMS プロバイダー、リソース・アダプター、メール・プロバイダーなどさまざまなリソース情報を定義することができます。リソース情報を定義する際、そのリソースの使用が許可されるスコープを有効範囲として設定します。有効範囲は、WAS V8.5 では V7.0 と同じくセル、クラスター、ノード、サーバー、アプリケーションという 5 階層になっています。同名のリソースを異なる有効範囲で定義した場合は、スコープの狭い方が優先されます。WAS V8.5 の管理コンソールでは、セル環境内で設定されている全ての有効範囲のリソースを表示する事が可能です。

アプリケーションをテスト環境から本番環境に、あるいは別のサーバーに移行する際は、その移行先のサーバーにて移行前と同じリソース定義が必要となります。リソース定義をアプリケーション・スコープで行っていない場合は、再度定義を行う必要があります。

---

* ODR は WebSphere プロキシー・サーバーの拡張ですが、WebSphere プロキシー・サーバーのすべての機能を持っているわけではありません。

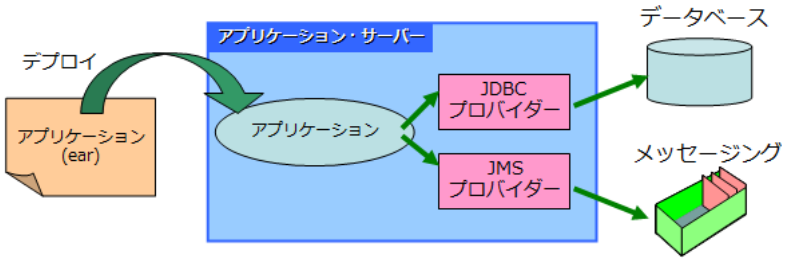


図 5-9 アプリケーションのデプロイとリソース

＜管理コンソール＞

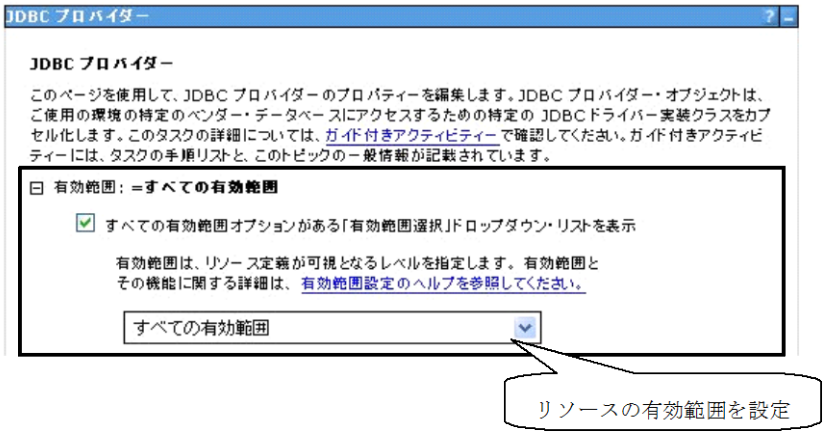


図 5-10 JDBC プロバイダー設定画面

リソース作成の例として、JDBC ドライバーの登録およびデータ・ソースの定義を紹介します。
この例では、DB2 上の DB[Sample]に接続するための設定を、クラスター[ClusterA]に行います。手順としては、DB2 の[Universal JDBC Driver]を登録し、[接続プール・データ・ソース]として[jdbc/SampleDS]を登録するという順序になります。

■ JDBCドライバーの登録

1. 有効範囲選択のプルダウン・リストから**[クラスター=ClusterA]**を選択し、**[新規作成]**をクリックします。

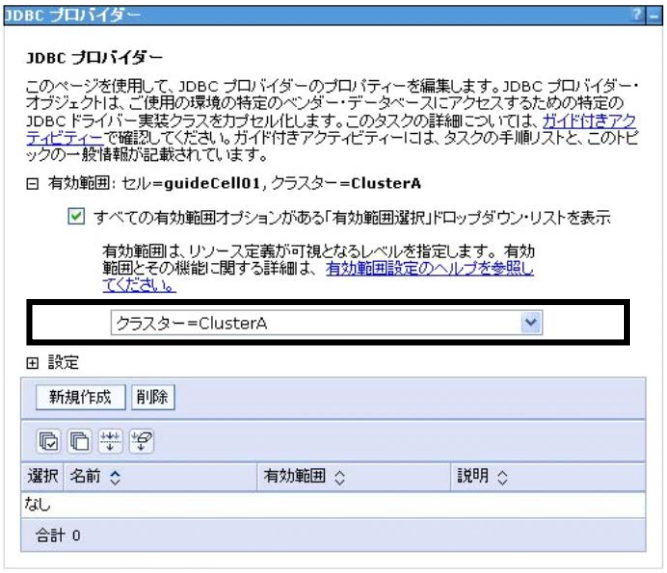


図 5-11 JDBCドライバーの登録(1)

2. [ステップ 1: 新規 JDBC プロバイダーの作成]で、**[データベース・タイプ]**として"DB2"、**[プロバイダー・タイプ]**として"DB2 Universal JDBC Driver Provider"、**[実装タイプ]**として"接続プール・データ・ソース"を選択します。

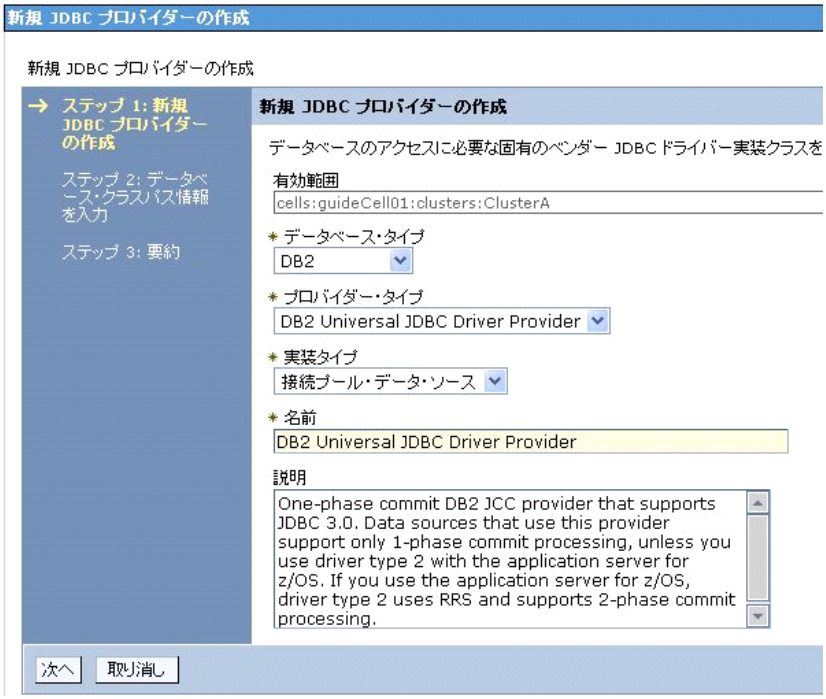


図 5-12 JDBCドライバーの登録(2)

3. [ステップ 2:データ・ソース・クラスパス情報を入力]では、JDBC ドライバーのクラスパスおよびネイティブ・ライブラリー・パスに使用される WebSphere 変数を設定し、**[次へ]**をクリックします。

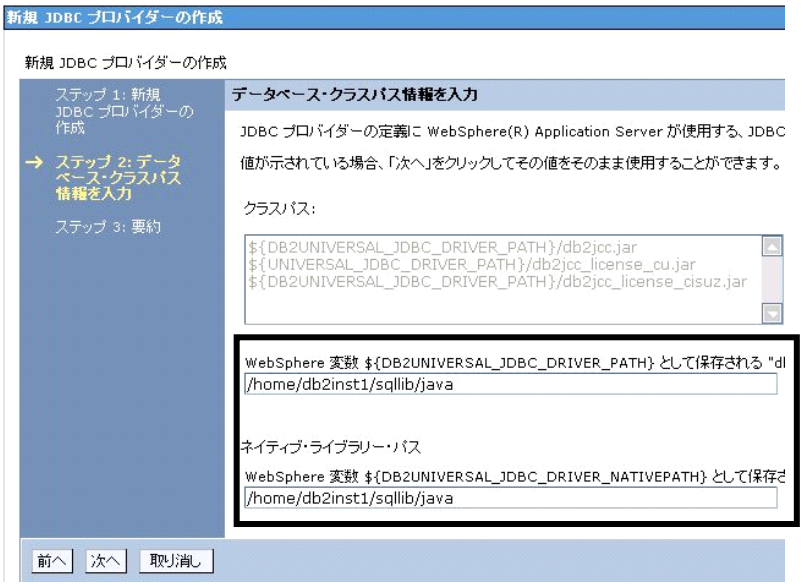


図 5-13 JDBC ドライバーの登録(3)

4. 要約の内容を確認し、**[終了]**をクリックします。
5. 構成を保存します。

■ データ・ソースの定義

1. JDBC ドライバーの登録の手順で登録した、JDBC ドライバーを選択します。

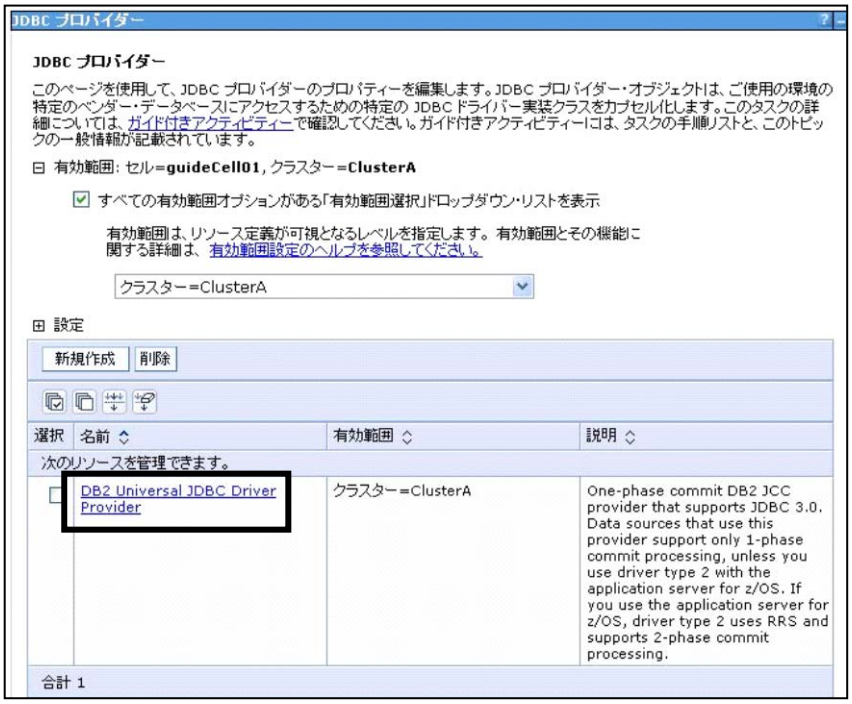


図 5-14 データ･ソースの定義(1)

2. **[追加プロパティー]**にある**[データ･ソース]**を選択します。



図 5-15 データ･ソースの定義(2)

3. **[新規作成]**をクリックします。



図 5-16 データ･ソースの定義(3)

4. [ステップ 1: 基本データ・ソース情報を入力]で、**[データ・ソース名][JNDI 名]**を設定します。**[次へ]**をクリックします。



図 5-17 データ・ソースの定義(4)

5. [ステップ 2:データ・ソースのデータベース特定プロパティーを入力]で、**[ドライバー・タイプ] [データベース名][サーバー名] [ポート番号]**と CMP 内での扱いを設定します。**[次へ]**をクリックします。

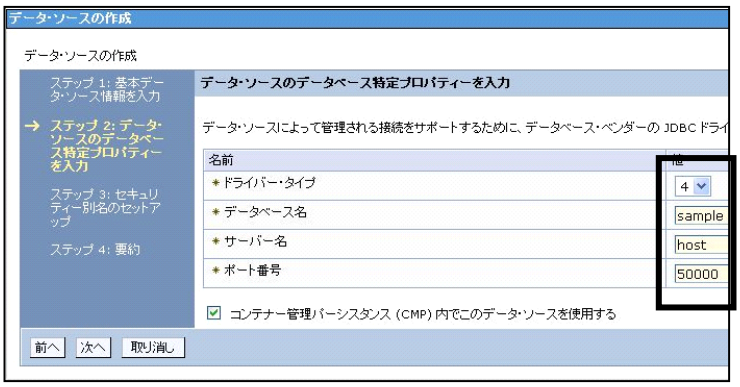


図 5-18 データ・ソースの定義(5)

6. [ステップ 3:セキュリティー別名のセットアップ]では何も入力せず、**[次へ]**をクリックします。(※セキュリティー別名は後で設定します)

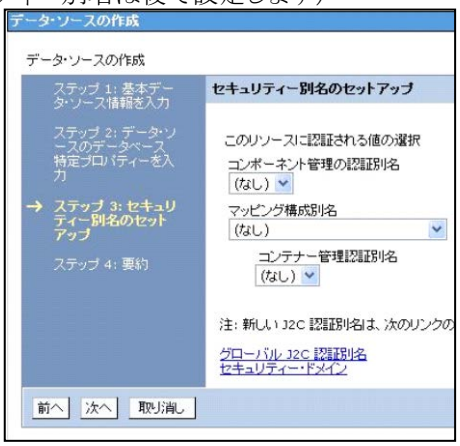


図 5-19 データ・ソースの定義(6)

7. 要約の内容を確認し、**[終了]**をクリックします。
8. 構成を保存します。
9. 作成したデータ・ソースの画面を開き、**[JAAS - J2C 認証データ]**を選択します。

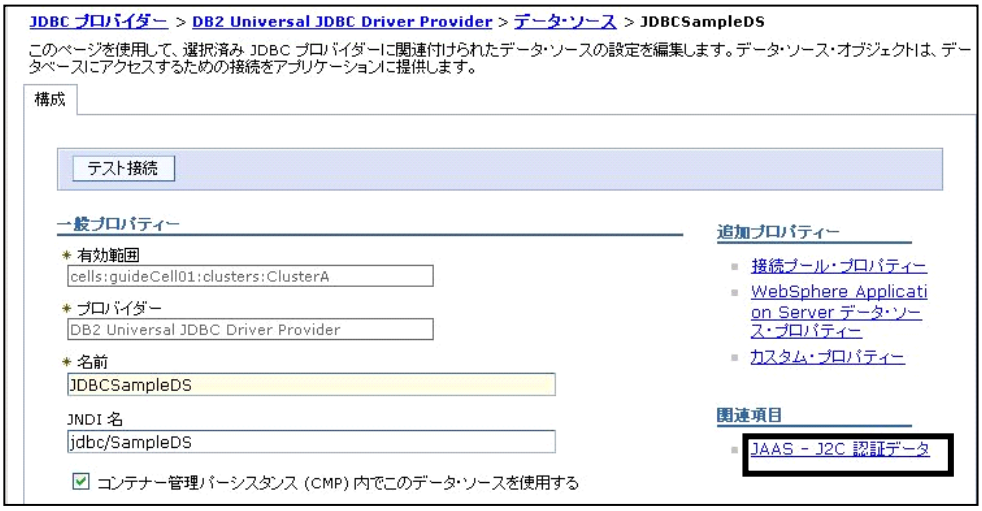


図 5-20 J2C 認証データ・エントリー作成画面(1)

10. **[新規作成]**をクリックします。

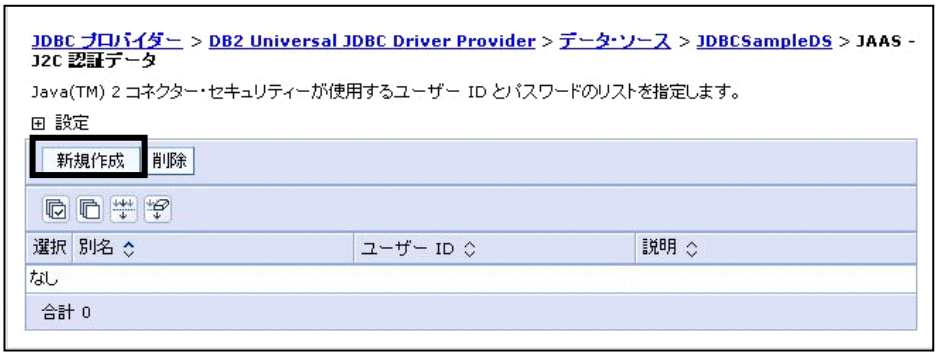


図 5-21 J2C 認証データ・エントリー作成画面(2)

11. データベースにアクセスするときに使用する、ユーザーID とパスワードを設定します。

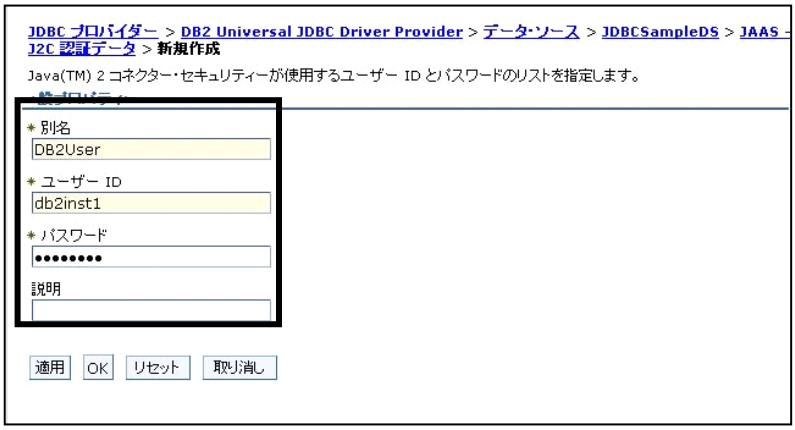


図 5-22 J2C 認証データ・エントリー作成画面(3)

12. データ・ソースの設定画面に戻ります。

13. **[セキュリティー設定]**の下の**[コンポーネント管理の認証別名]**で 11.で設定した認証別名を設定し、**[OK]**をクリックします。

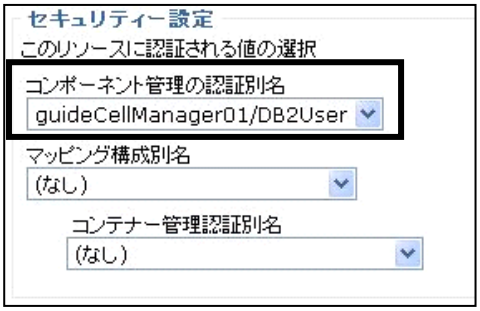


図 5-23 データ・ソースへ J2C 認証データ・エントリーを設定

14. 構成を保存します。

設定はここまでですが、設定が正しいかを確認するために、テスト接続を行います。

15. 作成したデータ・ソースを選択し、**[テスト接続]**をクリックします。

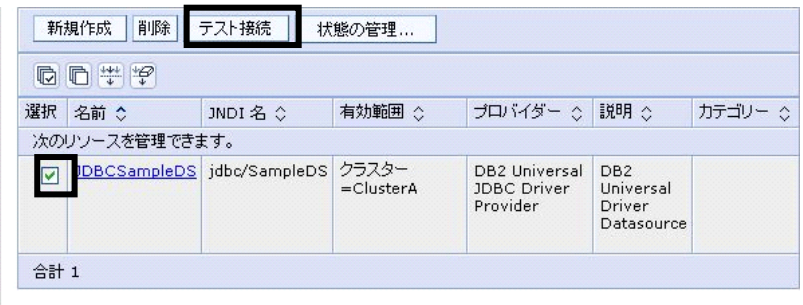


図 5-24 テスト接続

16. 下記のようなメッセージが表示されれば、正常に設定が行われています。



図 5-25 テスト接続成功時のメッセージ

5-2-1-2 共用ライブラリーの定義

アプリケーション・モジュールが複数ある場合、複数アプリケーション・モジュール間で引数や戻り値として使用されるユーザー作成のクラスや、複数アプリケーション・モジュールで共通の振る舞いを定義するためのインターフェースなど、共通するクラスを使用する場合があります。これらの依存クラスは、Jar としてまとめ、共用ライブラリーとして定義する事が出来ます。共用ライブラリーを使用すると、システム上の重複するライブラリー・ファイルの数を削減する事が出来ます。

共用ライブラリーはセル、ノード、クラスター(クラスターが作成されている場合のみ)、サーバーのレベルで作成する事が出来ます。また、作成した共用ライブラリーは「アプリケーションまたはモジュール」か、「アプリケーション・サーバー」のいずれかに関連付ける事が出来ます。更に、WAS V7.0 からは共用ライブラリーを作成する際に、共用ライブラリーで分離されたクラス・ローダーを使用するように定義する事が出来るようになりました。共用ライブラリーの範囲と、共用ライブラリーが何に関連付けられているかは、この共用ライブラリーがどのクラス・ローダーに読み込まれるか、に影響を及ぼします。クラス・ローダーについては、5-5 クラス・ローダー」を参照して下さい。

共用ライブラリーの定義方法を以下に記載します。

■ 共用ライブラリーの作成

1. 管理コンソールで、**[環境]→[共用ライブラリー]**を選択し、共用ライブラリーの有効範囲をドロップダウン・リストから選択し、**[新規作成]**をクリックします。



図 5-26 共用ライブラリーの作成(1)

2. 共用ライブラリー名、クラスパスなどを設定します。
ここで、この共用ライブラリーだけで独立したクラス・ローダーを使用したい場合には、**[この共用ライブラリーでの分離されたクラス・ローダーの使用]**にチェックを入れます。その場合には後続の共用ライブラリーを関連付ける手順で、必ずアプリケーションまたは Web モジュールに関連付けて下さい。共用ライブラリーをサーバーに関連付けると、この設定は無視され、独立した共用ライブラリーは使用されませんので注意して下さい(その場合は共用ライブラリーのファイルがアプリケーション・サーバー・クラス・ローダーに追加されます。)

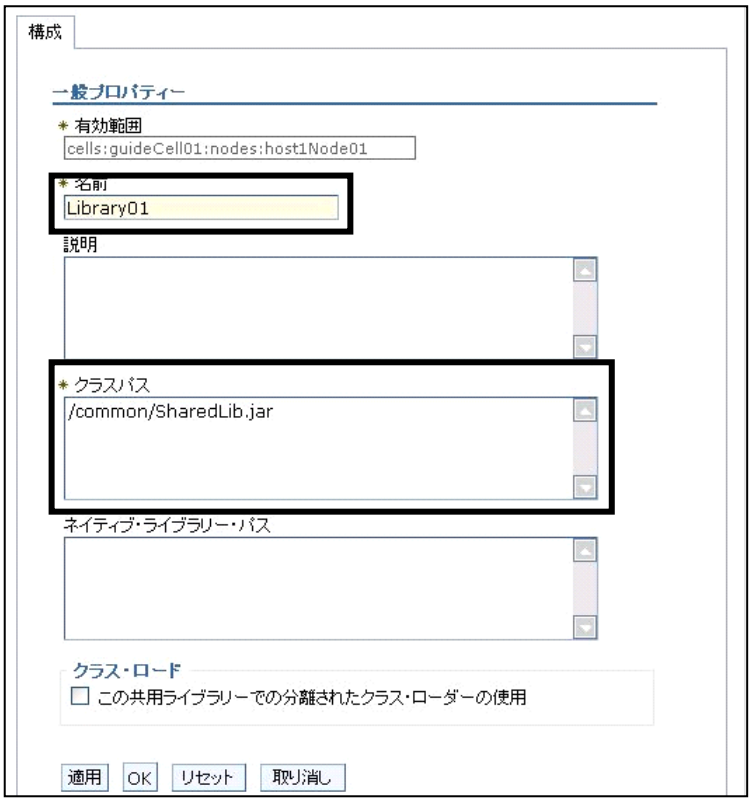


図 5-27 共用ライブラリーの作成(2)

3. 構成を保存します。

■ 共用ライブラリーの関連付け

作成した共用ライブラリーを「アプリケーションまたはモジュール」か、「アプリケーション・サーバー」のいずれかに関連付けます。

「アプリケーションまたはモジュール」と関連付けた場合は、共用ライブラリーで表されるクラスがアプリケーションのクラス・ローダーにロードされ、そのクラスがアプリケーションで使用できるようになります。

「アプリケーション・サーバー」と関連付けた場合は、共用ライブラリーで表されるクラスがサーバー全体のクラス・ローダーにロードされ、そのサーバーにデプロイされているすべてのアプリケーションでクラスが使用できるようになります。この場合、クラスのメンテナンスの観点から考慮すべき点があります。もしこの共用ライブラリーに置かれたクラスにバグが発見されて修正が必要になった場合、アプリケーション・サーバー上の全てのアプリケーションが影響を受けますので、全てのアプリケーションで修正済み依存クラスのテストを実施する必要があります。

上述のような特徴を把握した上で、どちらと関連付けるかを判断する必要があります。

◆ アプリケーションまたはモジュールと関連付ける場合

1. 管理コンソールで**[アプリケーション]→[アプリケーション・タイプ]→[WebSphere エンタープライズ・アプリケーション]**を選択し、アプリケーションを選択します。
2. **[参照]**の下から**[共用ライブラリー参照]**を選択します。

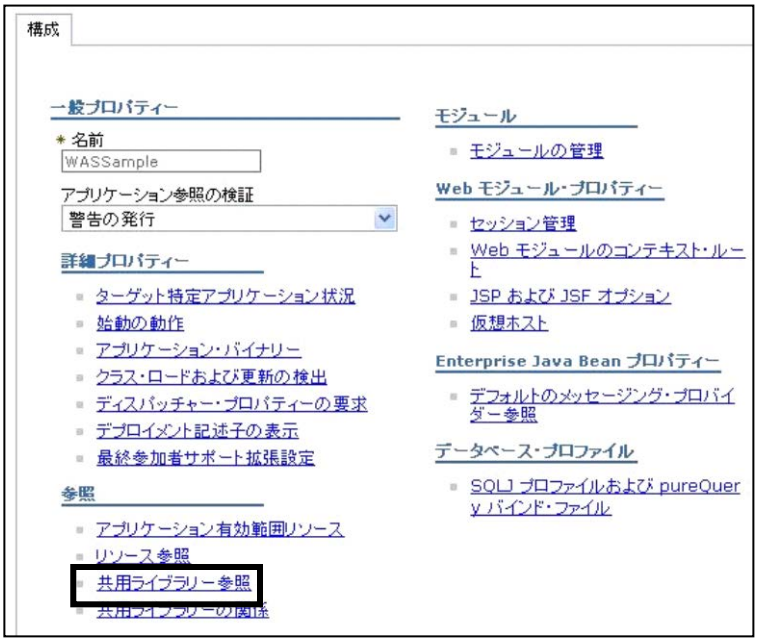


図 5-28 共用ライブラリーのアプリケーションまたはモジュールとの関連付け(1)

3. 関連付けるアプリケーションまたはモジュールにチェックを入れ、**[参照共用ライブラリー]**をクリックします。



図 5-29 共用ライブラリーのアプリケーションまたはモジュールとの関連付け(2)

4. **[使用可能]**の欄から関連付ける共用ライブラリーを選択し、右向きの矢印をクリックします。**[選択済み]**の欄に選択した共用ライブラリーが表示されているのを確認し、**[OK]**をクリックします。再び 3.の画面に戻るので、再度**[OK]**をクリックします。



図 5-30 共用ライブラリーのアプリケーションまたはモジュールとの関連付け(3)

5. 構成を保存します。

◆ アプリケーション・サーバーと関連付ける場合

1. 管理コンソールで**[サーバー]→[サーバー・タイプ] →[WebSphere Application Server]**を選択し、サーバー名を選択します。
2. **[サーバー・インフラストラクチャー]**の下で**[Java およびプロセス管理]→[クラス・ローダー]**を選択します。



図 5-31 共用ライブラリーのアプリケーション・サーバーとの関連付け(1)

3. アプリケーション・サーバー関連付け共用ライブラリー用のクラス・ローダーを選択します。まだクラス・ローダーを作成していない場合は**[新規作成]**をクリックして作成します。作成する場合、クラス・ローダーID は自動で振られますので、**[クラス・ローダ**

ー・順序]** のみを設定します。 **[クラス・ローダー・順序]**については5-5-3 クラス・ローダー・モード」も参照して下さい。



図 5-32 共用ライブラリーのアプリケーション・サーバーとの関連付け(2)

4. **[追加プロパティー]**の下の、**[共用ライブラリー参照]**をクリックします。

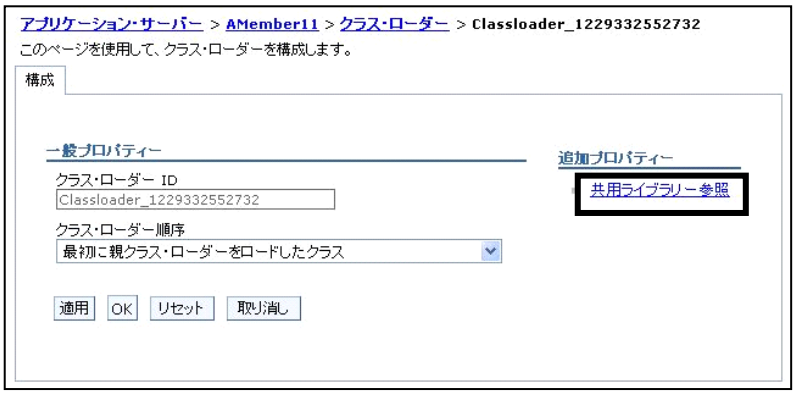


図 5-33 共用ライブラリーのアプリケーション・サーバーとの関連付け(3)

5. **[追加]**をクリックし、**[ライブラリー名]**のドロップダウン・リストからライブラリー名を選択して**[OK]**をクリックします。



図 5-34 共用ライブラリーのアプリケーション・サーバーとの関連付け(4)

6. 構成を保存します。

5-2-1-3 ファイルの同期

セル構成において、管理ツールでデプロイメント・マネージャーのマスター構成ファイルに変更を加えた場合、デプロイメント・マネージャーと各管理対象ノードのノード・エージェント(NA)が同期(ファイル同期サービス)を行うことによって、デプロイメント・マネージャーがノード・エージェントに構成情報を配布(ファイル転送サービス)し、両者が保持する構成情報の整合性を維持します。

デプロイメント・マネージャーには構成ファイルをダウンロードするためのサーブレットが導入されており、同期の際に各管理対象ノードのノード・エージェントがそのサーブレットを実行することにより、構成ファイルがそのノードに配布されることになります。

同期は、構成の保存と同時に行うか、管理コンソールから明示的に同期を指示することができます。構成の保存と同時に行う時には、「ノードと変更を同期化」のチェックを入れて保存して下さい。保存が行われると同時に、ノード間の同期も行われます。

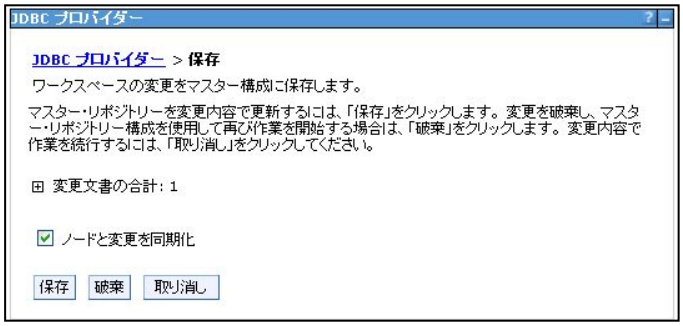


図 5-35 構成の保存画面

保存時には、同期の状態を確認することができます。出力されるメッセージの最後に、「ADMS0208I: セルの構成同期化が完了しました。」が表示されます。

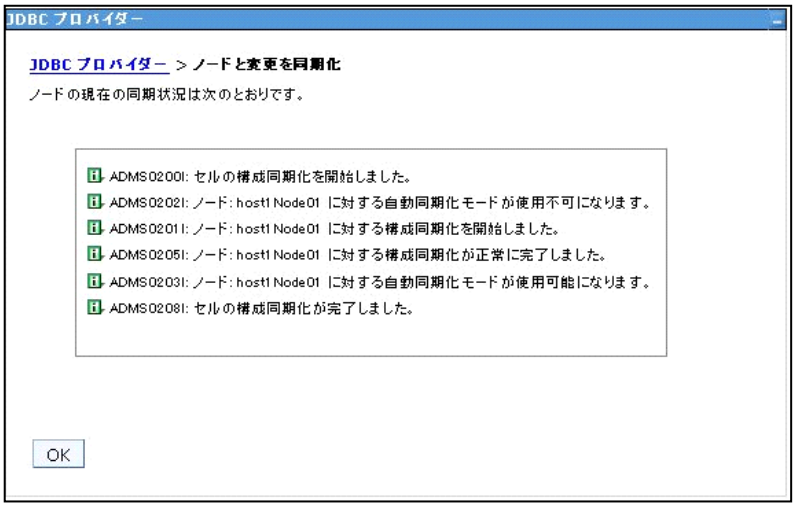


図 5-36 同期処理の状況表示

管理コンソールから明示的に同期を指示する時には、同期を取りたいノードを指定して手動による同期化あるいは完全な再同期を行います。通常の管理ツールなど JMX 経由での構成変更を配布する場合は「同期化」、手動のファイル更新や追加なども反映させたい場合は「完全な再同期」を行います。実際の処理は、デプロイメント・マネージャーからノード・エージェントに構成情報を転送するのではなく、ノード・エージェントに対してファイル同期化サービスを実行するように処理依頼をしています。

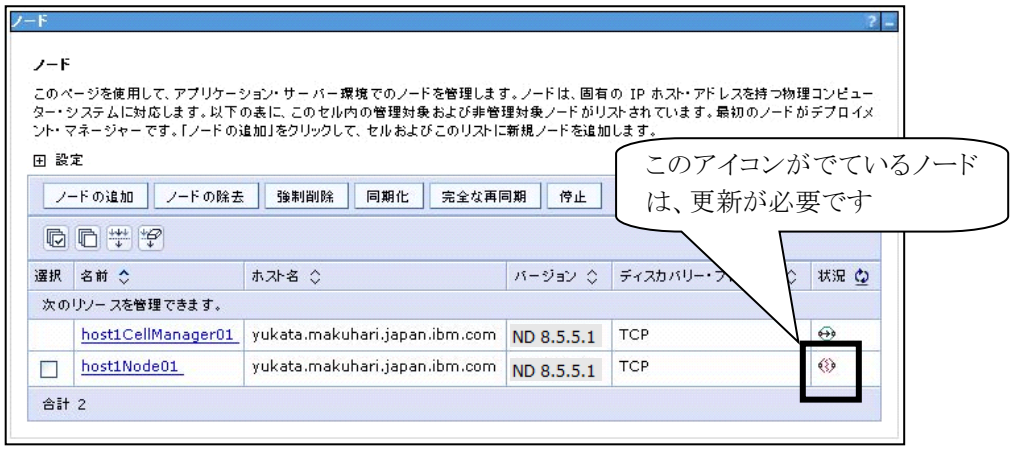


図 5-37 ノード管理画面

同期を自動で行うタイミングや時間間隔などを設定することもできます。また、同期を行いたくない特定のファイル名などを事前に指定しておくこともできます。

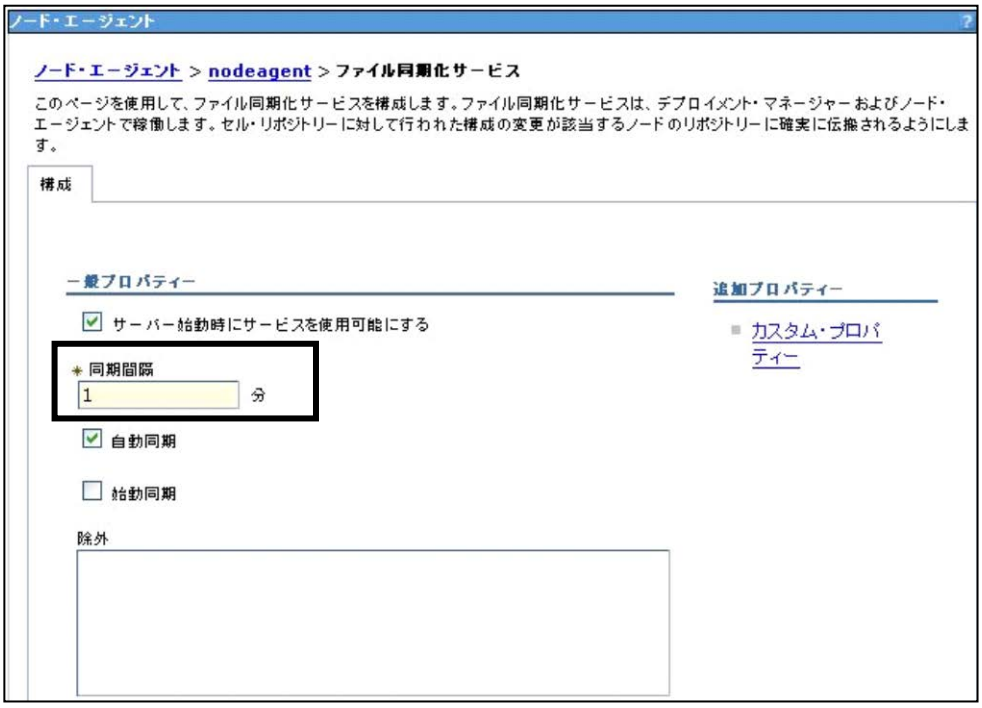


図 5-38 ノード・エージェント ファイル同期サービス設定画面

5-2-1-4 ファイルの配布

デプロイメント・マネージャー・ノードや各管理対象ノードにある、全ての構成ファイルやアプリケーション・ファイル(.ear)は config ディレクトリー配下に格納されます。各管理対象ノードにあるアプリケーション・ファイルは、デフォルトでは installedApps ディレクトリー配下に展開され、特定の AS とマッピングすることで公開され、Web クライアントからはサービスとして使用できるようになります。図 5-39 アプリケーションの配布は、セル構成におけるアプリケーションのインストールから展開までの流れを表しています。

まず、管理ツールでデプロイメント・マネージャーに対してアプリケーションをインストールすると、デプロイメント・マネージャー・ノードの config ディレクトリー配下に ear ファイル(アーカイブされた状態)が格納されます。ここには ear ファイルのデプロイメント・ディスクリプター(以下、DD)も保存されているので、管理ツールから DD を参照することができます。

次に、そのアプリケーションに「配布の使用可能」という設定がされている場合は、デプロイメント・マネージャーとノード・エージェントの同期のタイミングで、各管理対象ノードの config ディレクトリー配下に ear ファイル(アーカイブされた状態)が格納されます。そして、各管理対象ノードの installedApps ディレクトリー配下にアプリケーションが展開されます。

ビジネス・レベル・アプリケーションのファイルの配布については、「5-2-2-4 ビジネス・レベル・アプリケーションにおけるファイルの配布」を参照して下さい。

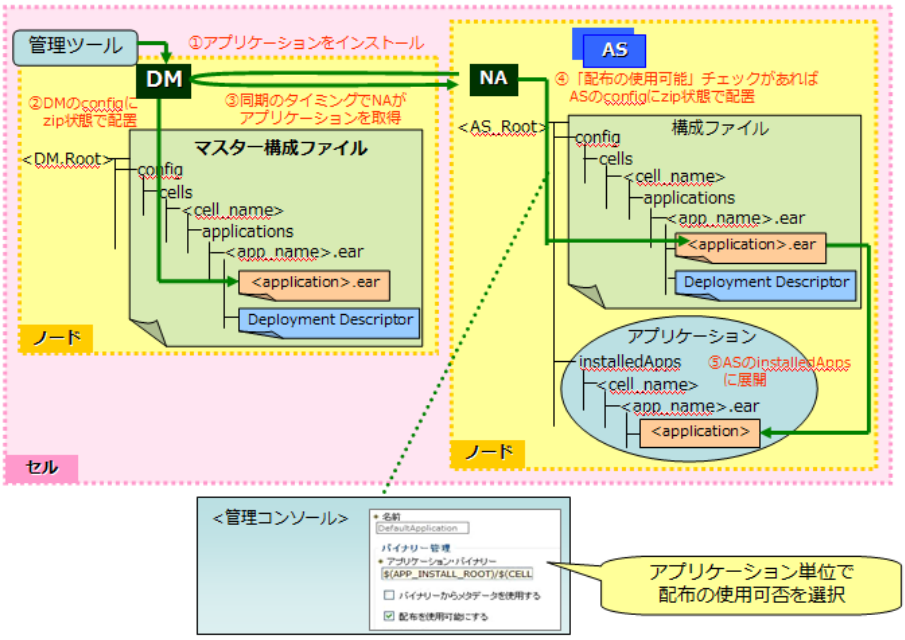
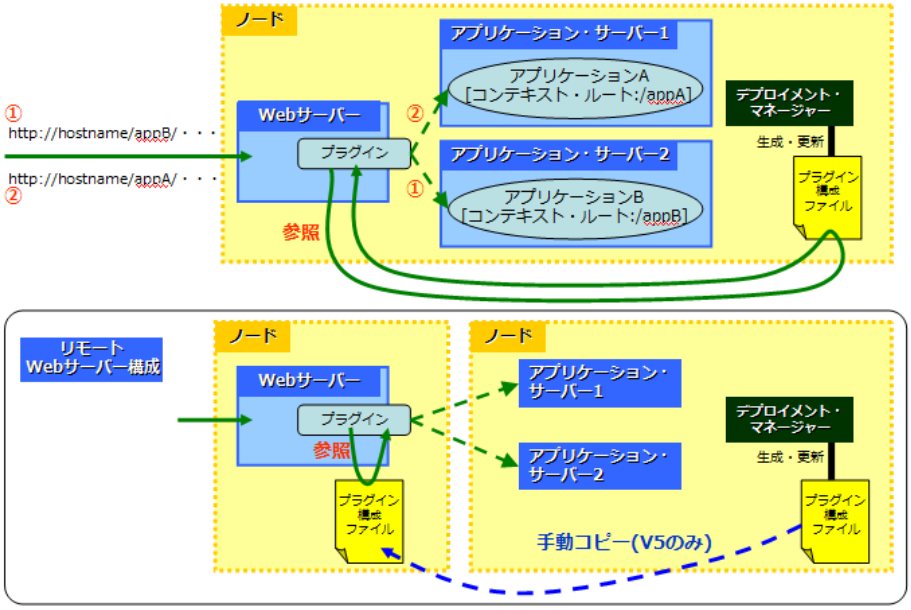


図 5-39 アプリケーションの配布

5-2-1-5 プラグインの更新

Web クライアントからのリクエストは Web サーバーが受け付け、内容に応じて適切な AS にリクエストを転送します。このとき、どの AS に転送すべきかを判断し、実際に転送を行うコンポーネントが、Web サーバー・プラグインです。Web サーバー・プラグインは WAS のコンポーネントの 1 つであり、スタンドアローン構成では AS から、セル構成で ODR を使用しない場合はデプロイメント・マネージャーから生成されるプラグイン構成ファイルを参照して、適切な転送先を決定します。ODR を構成する場合は前述のように、ODR または高可用性プラグイン構成生成サービスにプラグイン構成ファイルを生成させます。



プラグイン構成ファイルにはリクエスト URL と AS(または ODR)のマッピング情報などが記載されており、新規サーブレットの追加などでマッピング情報が変更される場合、再生成する必要があります。

図 5-40 プラグイン構成ファイルの更新

ODR を構成しない場合、WAS V8.5 では、Web サーバー管理画面からリモートにある Web サーバーに対してもプラグインの生成および伝搬ができるようになっています。また、Web アプリケーションの追加などにより、プラグイン構成ファイルの生成・伝搬が必要となった際に自動的にプラグイン構成ファイルの再生成・伝搬を行うようにすることも可能です。



図 5-41 Web サーバー プラグイン構成ファイル生成・伝搬

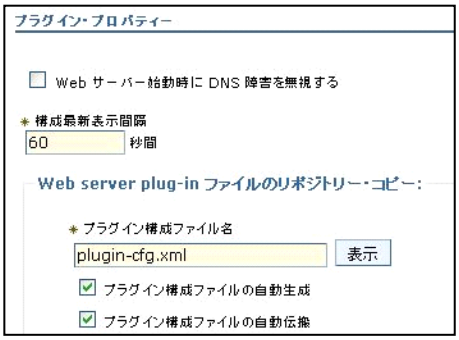


図 5-42 プラグイン構成ファイル 自動生成・伝搬設定

ODR を構成して Web サーバーから ODR に対してリクエストを転送させる場合、プラグイン構成ファイルはデフォルトで<WAS_ROOT>/etc ディレクトリーに生成されます。生成された構成ファイルをプラグイン

が読み込む所定のディレクトリー下にコピー(伝搬)するためのシェル・スクリプト(またはバッチ・ファイル)を作成しておき、そのスクリプトを ODR 構成または高可用性プラグイン構成生成サービスに指定します。こうすることで、プラグイン構成ファイルが更新された際に、その伝搬が自動的に行われるようになります。ODR を構成する場合には、図 5-42の「プラグイン構成ファイルの自動生成」「プラグイン構成ファイルの自動伝搬」のチェック・ボックスはクリアしておきます。

ODR によるプラグイン構成ファイルの生成と伝搬

プラグイン構成ファイルの生成と伝搬を ODR にさせる場合、管理コンソールの サーバー→すべてのサーバー→<ODR 名>→オンデマンド・ルーター・プロパティ→オンデマンド・ルーター設定 内の下の図の箇所で設定を行います。



「信頼されるセキュリティー・プロキシー」には、この ODR へリクエストを転送する Web サーバーのホスト名または IP アドレスを記載します。複数ある場合は、改行で区切って必要なだけ列挙します。
「プラグイン構成の生成」プルダウン・メニューでは、生成されるプラグイン構成ファイルに記載されるリクエスト転送先 ODR の存在する範囲を選択します。「すべて」は、自セルに加え、クロス・セル通信を構成された他セル内の ODR にもリクエストを転送させます。「セル」「ノード」はそれぞれ、プラグイン構成ファイルを生成させる ODR を含むセルまたはノード内のすべての ODR にリクエストを転送させます。「サーバー」はプラグイン構成ファイルを生成させる ODR のみにリクエストを転送させます。
「プラグイン構成変更スクリプト」には、生成されたプラグイン構成ファイル(デフォルトでは <WAS_ROOT>/etc/plugin-cfg.xml)を、Web サーバー・プラグインが読み込む所定の場所にコピーするためのシェル・スクリプト(別途作成し、UNIX 系 OS の場合は実行属性を付与(chmod +x)しておきます)のフルパスを記述します。この設定により、プラグイン構成ファイルが生成されるたびにここで指定したシェル・スクリプトが実行されるようになります。
プラグイン構成ファイルを ODR に生成させるよう設定した場合、当該の ODR が障害などで停止している間はプラグイン構成ファイルの生成や伝搬が行われません。可用性の観点から、2 つ以上の ODR にプラグイン構成ファイルの生成を設定するようにします。

高可用性プラグイン構成生成サービスによるプラグイン構成ファイルの生成と伝搬

高可用性プラグイン構成生成サービスは、ODR に代えてプラグイン構成ファイルを生成させることができ、デプロイメント・マネージャー、いずれかのノード・エージェントまたはいずれかの ODR のうちの 1 つのプロセスで稼働することが保証されるサービス(HAManagedItem)です。サービスが稼働しているプロセスが停止した場合、他のプロセスがサービスを引き継ぐことで高可用性を実現します。
高可用性プラグイン構成生成サービスに対する設定は GUI が用意されておらず、セルのカスタム・プロパティーとして設定を行います。設定方法の詳細は、Infocenter の以下のページを参照して下さい。

HA 環境でのプラグイン構成の生成
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wve.doc/ae/twve_haplugincfg.html

このサービスはデプロイメント・マネージャー、すべてのノード・エージェント、すべての ODR で稼働する可能性があります。そのため、これらのプロセスが稼働するすべてのノードで、前述のプラグイン構成変更スクリプトが同じパスに存在している必要があります。プラグイン構成変更スクリプトの場所は

ODCPluginCfgUpdateScript_<configName>=/path/scriptの形式でセルのカスタム・プロパティーとして設定します。詳細は上で紹介しました Infocenter のページをご覧下さい。

5-2-1-6 アプリケーションの導入手順

ここで管理コンソールからアプリケーションを導入する方法を説明します。

なお、ここではサンプル・アプリケーションとして提供されている DefaultApplication.ear のインストールを元に導入手順を記載しています。EJB3.0 や MessageDrivenBean などの導入時に表示される手順など、ここに記載のない手順については、下記の Information Center を参照して下さい。

WAS V8.5 Information Center
「コンソールを使用したエンタープライズ・アプリケーション・ファイルのインストール」
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.doc/ae/trun_app_instwiz.html

1. 管理コンソールで**[アプリケーション]→[WebSphere エンタープライズ・アプリケーション]**を選択して**[インストール]**をクリックし、インストールするエンタープライズ・アプリケーション・ファイル (.ear ファイル、別名 EAR ファイル) の絶対パス名を指定します。
   インストールしようとする EAR ファイルは、クライアント・マシン (Web ブラウザが稼働しているマシン) またはサーバー・マシン (クライアントの接続先のマシン) のいずれかに配置できます。クライアント・マシンに EAR ファイルを指定すると、その EAR ファイルは管理コンソールにより、そのコンソールが稼働しているマシンにアップロードされ、アプリケーションのインストールへと続きます。
   また、インストール用にスタンドアローンの Web アプリケーション・アーカイブ (WAR) または Java アーカイブ (JAR) ファイルを指定することもできます。**[次へ]**をクリックします。

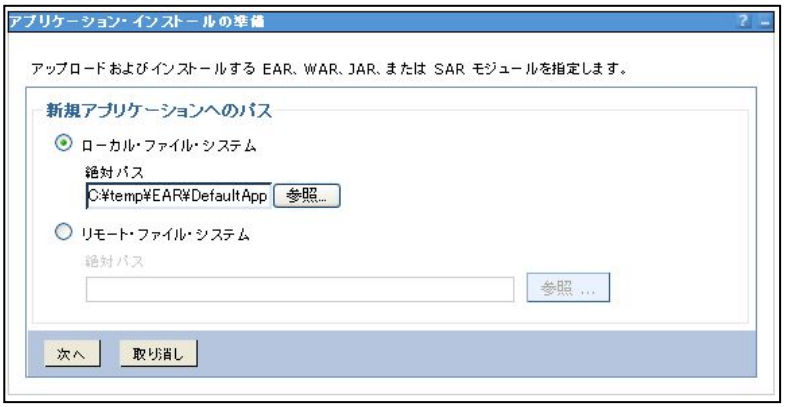


図 5-43 アプリケーション・インストール画面(1)

2. アプリケーション・インストールの方法を指定します。
   WAS V6.1 以降、不要なインストール・オプションを省略する事の出来る、ファースト・パス・インストールが追加されました。**[ファースト・パス]**を選択すると後続のインストールのステップ数が大幅に削減されます。全てのインストール・オプションおよびパラメーターを設定するには**[詳細]**を選択して下さい。当ガイドの手順では**[詳細]**を選択します。

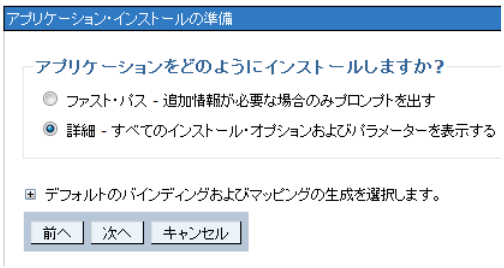


図 5-44 アプリケーション・インストール画面(2)

同画面でデフォルトのバインディングの使用またはマッピングの生成を行う事が出来ます。
デフォルト・バインディングを使用すると、アプリケーションの不完全なバインディングがすべてデフォルト値で埋められます。既存のバインディングは、変更されません。 デフォルト・バインディングを生成する際に使用されるデフォルト値はカスタマイズできます。例えば、EJB モジュールの EJB ファイルの Java Naming and Directory Interface (JNDI) 接頭部、 EJB モジュールのデフォルト・データ・ソースおよび接続ファクトリー設定、 Web モジュールの仮想ホストなどを指定できます。**[次へ]**をクリックします。



図 5-45 アプリケーション・インストール画面(2)

3. アプリケーションを WebSphere Application Server デプロイメント・ターゲットにインストールするためのオプションを指定します。

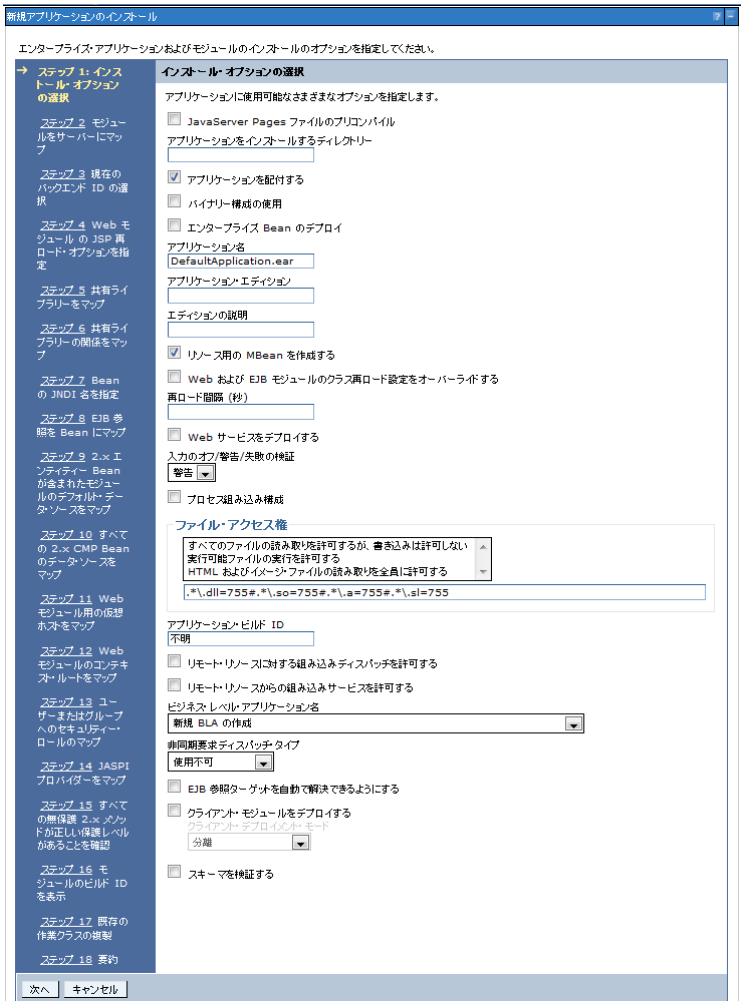


図 5-46 アプリケーション・インストール画面(3)

値を指定しない場合は、オプションのデフォルト値が使用されます。また、アプリケーションのインストール後に、これらのオプションの値の多くは管理コンソール・エンタープライズ・アプリケーション設定ページで指定することができます。指定後、**[次へ]**をクリックします。

また、左側に表示されているステップの数は、インストールするアプリケーションやインストール済みのアプリケーション・エディションによって変わります。このガイドでは製品に添付の DefaultApplication.ear の新しいエディションをインストールする場合のステップを説明します。

すべてのステップの説明は、Infocenter の以下のページを参照して下さい。

コンソールを使用したエンタープライズ・アプリケーション・ファイルのインストール

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.doc/ae/trun_app_instwiz.html

各オプションの意味は以下の通りです。

- JavaServer Pages ファイルのプリコンパイル
  インストール中に JavaServer Page (JSP) ファイルをプリコンパイルするかどうかを指定します。デフォルトでは、JSP ファイルをプリコンパイルしません。 このオプションを指定する場合はアプリケーションのデプロイメント・ターゲットが WAS V6.1 以降のサーバーである必要があります。古いバージョンのデプロイメント・ターゲットにインストールしようとすると、インストールは拒否されます。
- アプリケーションをインストールするディレクトリー

エンタープライズ・アプリケーション (EAR) ファイルのインストール先のディレクトリーを指定します。 デフォルト値は、<profile_root>/installedApps/<cell_name>です。ディレクトリーを指定しなければ、アプリケーションは<profile_root>/installedApps/<cell_name>/application_name.ear にインストールされます。

■ アプリケーションの配布

インストール中にアプリケーション・バイナリーを展開し、アンインストール中にアプリケーション・バイナリーを削除するかどうかを指定します。デフォルトでは、アプリケーションの配付は使用可能になっていますので、インストールされたアプリケーションのアプリケーション・バイナリーは、指定のディレクトリーに展開されます。また、構成に対する変更をアンインストールして保存する際には、バイナリー・ファイルも削除されます。

このオプションを使用不可にする場合、アプリケーションを実行するすべてのノードの宛先ディレクトリーで、アプリケーション・バイナリーを、EARExpander コマンドなどを使用して必ず適切に展開する必要があります。

■ バイナリー構成の使用

アプリケーション・サーバーが、アプリケーション・デプロイメント文書 deployment.xml ファイルにあるバインディング、拡張子、デプロイメント記述子を使用するか (デフォルト)、またはエンタープライズ・アーカイブ (EAR) ファイルにあるものを使用するかを指定します。デフォルトでは deployment.xml にあるバインディング、拡張子、およびデプロイメント記述子を使用しません。WAS V6.0 以降のデプロイメント・ターゲットにインストールされたアプリケーションについてのみ、この設定を選択します。この設定は、5.x デプロイメント・ターゲットにインストールされたアプリケーションについては無効です。

■ エンタープライズ Bean のデプロイ

アプリケーションのインストール時に EJBのデプロイ・ツール(EJBDeploy)を実行するかどうかを指定します。

このツールは、Enterprise Bean (EJB) ファイルの実行に必要なコードを生成します。以下の場合には、この設定を使用可能にする必要があります。

- ◆ EAR ファイルがアセンブリー・ツール (Rational Application Developer など) を使用してアセンブルされ、アセンブル中に EJBDeploy ツールが実行されていない場合
- ◆ EAR ファイルがアセンブリー・ツール (Rational Application Developer など) を使用してアセンブルされていない場合
- ◆ EAR ファイルがバージョン 5 より前の Application Assembly Tool (AAT) を使用してアセンブルされている場合

このオプションで使用の設定が可能なEJBDeployツールは、EJB3.x モジュールのインストール時には実行されません。また、この設定を使用する場合には、WAS V6.1 以降のデプロイメント・ターゲットにのみインストールして下さい。この設定を有効にし、古いバージョンのデプロイメント・ターゲットにインストールしようとすると、インストールは拒否されます。

■ アプリケーション名

アプリケーションの論理名を指定します。アプリケーション名は、セル内で固有にする必要があり、サポートされていない文字を含むことはできません。また、先頭にピリオド(.)を使用したり、先頭または末尾をスペース( )にしたりすることは出来ません。

■ アプリケーション・エディション

アプリケーションのエディション管理 (2-3-3, 2-3-4 参照) を使用する際のエディション名を指定します。指定しない場合、デフォルトで ”Base Edition” というエディション名が付与されます。

■ エディションの説明

エディションの説明 (任意) を入力します。

■ リソース用の MBean の作成

アプリケーションの開始時に、アプリケーション内にさまざまなリソース (サーブレットや JSP ファイルなど) 用の MBean を作成するかどうかを指定します。デフォルトでは、MBean を作成します。

- Web モジュールおよび EJB モジュールのクラスの再ロード設定のオーバーライド
  アプリケーションの実行時に、この製品のランタイムがアプリケーション・クラスに対する変更を検出するかどうかを指定します。この設定が有効になっている場合にアプリケーション・クラスが変更されると、アプリケーションが停止し、再始動して、更新されたクラスが再ロードされます。デフォルトでは、クラス再ロードは行いません。
- 再ロード間隔 (秒)
  更新されたファイルを探すためにアプリケーションのファイル・システムをスキャンする秒数を指定します。 デフォルトは、 EAR ファイルの IBM 拡張ファイルで指定されている、再ロード間隔属性の値です。クラスの再ロードが使用可能になっている場合にのみ、再ロード間隔属性は有効になります。再ロードを使用不可にするには、ゼロ (0) を指定します。
- Web サービスのデプロイ
  アプリケーションのインストール時に、Web サービスのデプロイ・ツール（wsdeploy）を実行するかどうかを指定します。
  このツールは、Web サービスを使用したアプリケーションの実行に必要なコードを生成します。デフォルトでは wsdeploy ツールを実行しません。 EAR ファイルに Web サービスを使用するモジュールが含まれており、以前にこれに対して wsdeploy ツールが実行されていない場合、この設定を、アセンブリー・ツールの「デプロイ」メニュー選択、またはコマンド行から使用可能にする必要があります。この設定を使用する場合には、WAS V6.1 以降のデプロイメント・ターゲットにのみインストールして下さい。この設定を有効にし、古いバージョンのデプロイメント・ターゲットにインストールしようとすると、インストールは拒否されます。
- 入力のオフ/警告/失敗の妥当性検査
  アプリケーションのインストールまたは更新中に指定されたアプリケーション参照を検査するかどうか、および妥当性検査を使用可能にする場合は、誤った参照を警告するか、またはオペレーションを失敗させるかを指定します。アプリケーションは通常、コンテナー管理のパーシスタンス (CMP) Bean のデータ・ソースを使用して、またはデプロイメント記述子で定義されたリソース参照またはリソース環境参照を使用してリソースを参照します。妥当性検査では、アプリケーションが参照するリソースがそのアプリケーションのデプロイメント・ターゲットの有効範囲で定義されるかどうかを検査します。
  リソースの妥当性検査を行わない場合は「オフ」、誤ったリソース参照について警告メッセージを出すには「警告」、誤ったリソース参照の結果、失敗するオペレーションを停止するには「失敗」を選択します。デフォルトは、「警告」です。
- プロセス組み込み構成
  組み込み構成を処理するかどうかを指定します。組み込み構成は、resource.xml および variables.xml などのファイルで構成されています。組み込み構成が選択されている、または true に設定されている場合、組み込み構成は .ear ファイルからアプリケーション有効範囲にロードされます。 .ear ファイルに組み込み構成が含まれていない場合、デフォルトは false です。 .ear ファイルに組み込み構成が含まれている場合、デフォルトは、true です。
  プロセス組み込み構成については、5-2-1-7 Enhanced EAR」も参照して下さい。
- ファイル・アクセス権
  指定のディレクトリーに展開されている、インストール済みアプリケーションのアプリケーション・バイナリーに対するアクセス権を指定します。ファイル・アクセス権を指定するには、「アプリケーションの配付」オプションが使用可能になっている必要があります。
  以下の点に注意し、設定して下さい。

- ◆ オペレーティング・システムに関係なく、ファイル・パターン内ではファイル・パス・セパレーターとして常にスラッシュ (/) を使用する必要があります。
- ◆ Windows ではファイルの読み取り権限を設定解除することはできません。
- ◆ ここで指定されるアクセス許可は、アプリケーション・レベルの許可です。ノード・レベルの構成においても、アプリケーション・バイナリーに対するアクセス権を指定することができますが、アプリケーション・レベルで指定できるアクセス許可による制限は、ノード・レベルで指定されたものと同じか、それ以上のものに限定されます。

■ アプリケーションのビルド ID

アプリケーションのビルド・バージョンを識別するストリングが指定されます。
編集は出来ません。

■ リモート・リソースに対する組み込みディスパッチを許可する

標準的な要求ディスパッチャー・メカニズムを使用して、管理対象ノード環境にある、異なる Java 仮想マシン内の Web モジュール間のリソースに対し、アプリケーションがインクルードをディスパッチできるかどうかを指定しますデフォルトは、false です。

■ リモート・リソースからの組み込みサービスを許可する

エンタープライズ・アプリケーションが、アプリケーションからのインクルード要求を処理できるかどうかを指定します。デフォルトは、false です。

■ ビジネス・レベル・アプリケーション名

この製品が、インストールしているエンタープライズ・アプリケーションを使用して新規ビジネス・レベル・アプリケーションを作成するか、またはエンタープライズ・アプリケーションを既存のビジネス・レベル・アプリケーションの構成単位にするかを指定します。デフォルトではドロップダウン・リストで[新規 BLA の作成]が選択されており、アセット名と同名のビジネス・レベル・アプリケーションが新規で作成されます。
このエンタープライズ・アプリケーションを既存のビジネス・レベル・アプリケーションに構成単位として追加するには、ドロップダウン・リストから既存のビジネス・レベル・アプリケーション名を選択して下さい。

■ 非同期要求ディスパッチ・タイプ

Web モジュールが別個のスレッド上で要求を同時にディスパッチできるかどうか、できる場合はサーバーまたはクライアントがその要求をディスパッチするかどうかを指定します。同時ディスパッチングにより、サーブレットの応答時間を改善することができます。デフォルトは、「使用不可」です。
操作が相互依存している場合は、エラーを起こす可能性があるため、非同期要求ディスパッチングを使用可能にしないで下さい。その場合には「使用不可」を選択して下さい。
サーバーが要求を同時にディスパッチできるようにするには「サーバー・サイド」を選択します。クライアントが要求を同時にディスパッチできるようにするには「クライアント・サイド」を選択します。

■ EJB 参照ターゲットを自動で解決できるようにする

不完全な EJB 参照ターゲットに対して、デフォルトの JNDI 値を割り当てるか、または不完全な EJB 参照ターゲットを自動的に解決するかを指定します。デフォルトは、false です。
このオプションを選択すると、参照が EJB 2.1 以前のモジュール、または Servlet 2.3 以前のモジュールからのものである場合に、EJB 参照ターゲットを自動的に解決するようにできます。このオプションを使用可能にすると、ランタイム・コンテナーはデフォルト値を提供するか、バインディングされていない任意の EJB 参照に対して、EJB 参照を自動的に解決します。
このオプションではデフォルトのバインディングが生成されるため、既に前の手順で「デフォルト・バインディングの生成」を選択している場合は、このオプションを選択する必要はありません。

「EJB 参照ターゲットの自動解決の許可」を選択した場合、アプリケーションにあるすべてのモジュールが、1 つのデプロイメント・ターゲットを共有する必要があります。このオプションを選択してもすべてのアプリケーション・モジュールが共通サーバーを共有していない場合、「サマリー」ページで「終了」をクリックすると、警告メッセージが表示されてアプリケーションはインストールされません。この設定を選択解除してから、「終了」をクリックしてアプリケーションをインストールする必要があります。

- ■ クライアント・モジュールをデプロイする
  クライアント・モジュールをデプロイするかどうかを指定します。
  デプロイするファイルに 1 つ以上のクライアント・モジュールがあり、そのクライアント・モジュールの環境エントリーを構成する場合に、このオプション (true に設定されている) を選択します。 さらに、このオプションを選択して、EJB 参照、リソース参照、リソース環境参照、またはメッセージ宛先参照などのリソースを構成します。 このオプションを選択すると、「クライアント・モジュールの環境エントリーのマップ」ページが表示されます。クライアント・モジュールのデプロイ先をデプロイメント・マネージャーの統合ノード (「統合済み (Federated)」) あるいはアプリケーション・サーバー (「サーバー・デプロイ済み (Server Deployed)」) にする場合、このオプションを選択して、「クライアント・デプロイメント・モード (Client deployment mode)」をデプロイメント・ターゲットの適切な値 (「統合済み (Federated)」または「サーバー・デプロイ済み (Server Deployed)」) に設定します。
  このオプションを選択した場合、バージョン 8.0 以降のデプロイメント・ターゲットに対してのみクライアント・モジュールをインストールできます。
- ■ スキーマを検証する
  公開済み Java EE デプロイメント記述子スキーマに対してデプロイメント記述子を検証するかどうかを指定します。このオプションを選択した場合、製品は各デプロイメント記述子を分析して、デプロイメント記述子の Java EE 仕様バージョンを判別し、適切なスキーマを選択してから、Java EE デプロイメント記述子スキーマに対してデプロイメント記述子を検査します。検証エラーが発生すると、エラー・メッセージが表示されます。
  Java EE デプロイメント記述子スキーマは DTD とも呼ばれます。
  このオプションを選択した場合、バージョン 8.0 以降のデプロイメント・ターゲットに対してのみアプリケーションまたはモジュールをインストールできます。

4. インストールするアプリケーションをモジュール毎に、どのサーバーもしくはクラスター上で実行するか、どの Web サーバーを経由してアクセスするかを指定することができます。ODR を構成し、リクエストを Web サーバー→ODR→AS のようにルーティングさせる場合は、モジュールを Web サーバーにマップしないようにします。

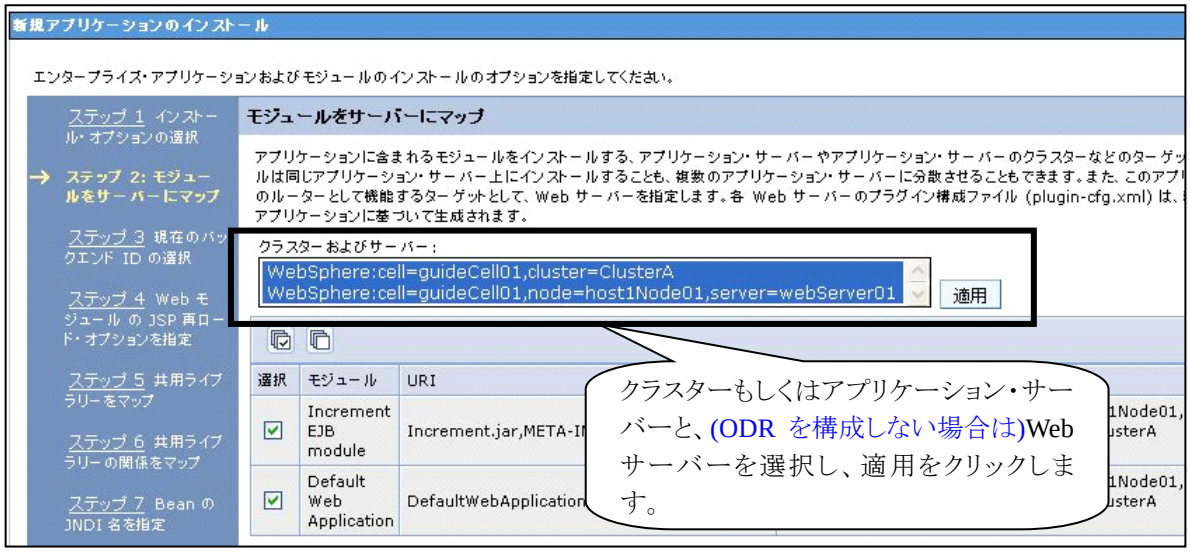


図 5-47 アプリケーション・インストール画面(4)

5. 現在のバックエンド ID を選択します。
   アプリケーション内に、複数のバックエンド・データベース用にデプロイメント・コードが生成された EJB モジュールがある場合、EJB モジュールの実行時に使用されるバックエンド・データベースを指定する事が出来ます。

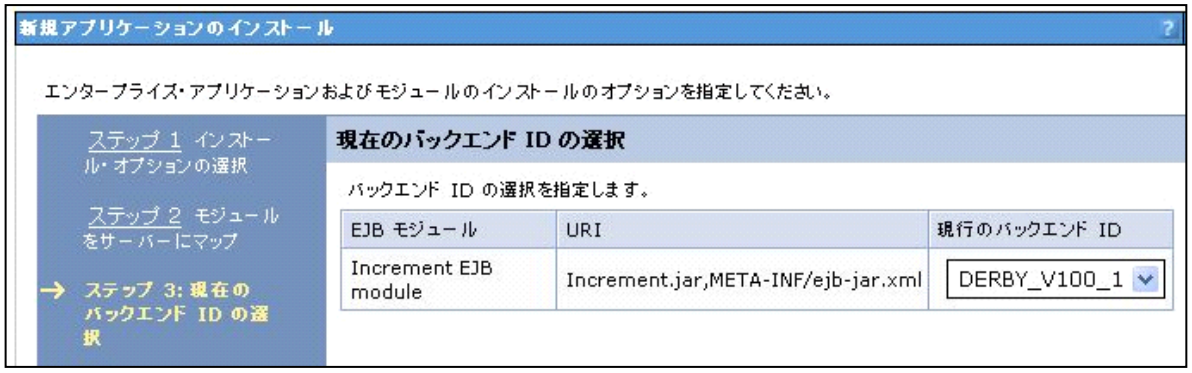


図 5-48 アプリケーション・インストール画面(5)

6. JSP 再ロード・オプションを設定します。



図 5-49 アプリケーション・インストール画面(6)

7. 定義されている共用ライブラリーとマップする場合にはチェックを入れます。

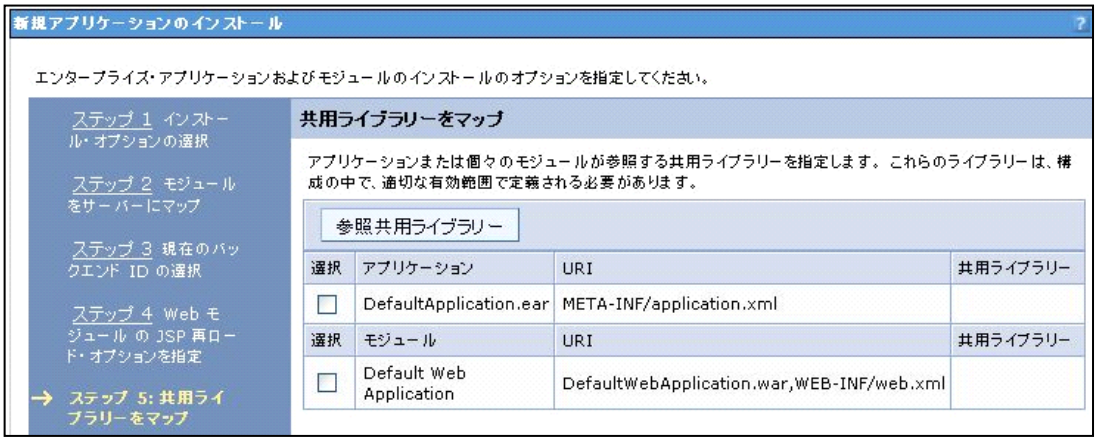


図 5-50 アプリケーション・インストール画面(7)

8. アプリケーションまたは個々のモジュールが参照する共有ライブラリーとしてアセット ID または構成単位 ID を指定します。

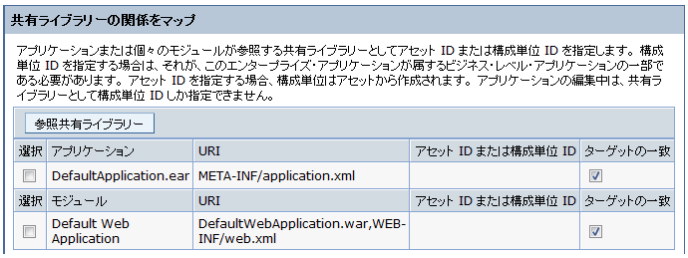


図 5-51 アプリケーション・インストール画面(6)

9. インストールするアプリケーションに EJB が含まれる場合、EJB の名称を指定することができます。通常は、変更する必要はありません。

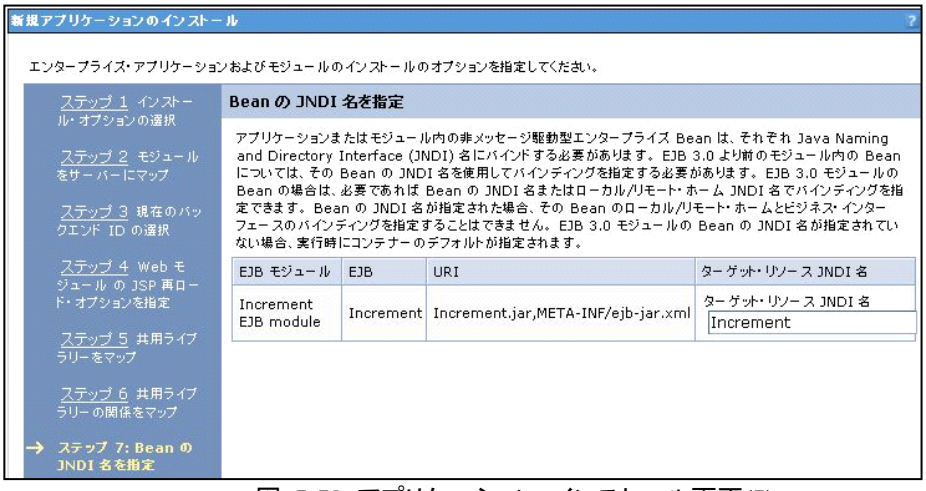


図 5-52 アプリケーション・インストール画面(5)

10. ここでは、Web モジュールなどで指定される EJB の参照名と先ほど指定した EJB の JNDI 名をマップします。

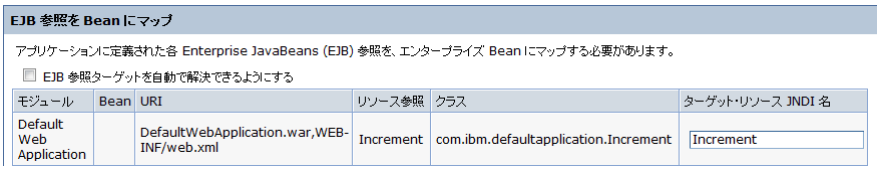


図 5-53 アプリケーション・インストール画面(7)

11. ここでは、リソース参照と定義済みリソースの JNDI 名をマップします。JNDI 名を直接編集することもできますが、**[参照]**ボタンをクリックして選択することも可能です。
また、この画面からは、複数の JNDI 名の設定、許可タイプの設定、リソース認証方式の変更、拡張プロパティーの設定を行う事が出来ます。

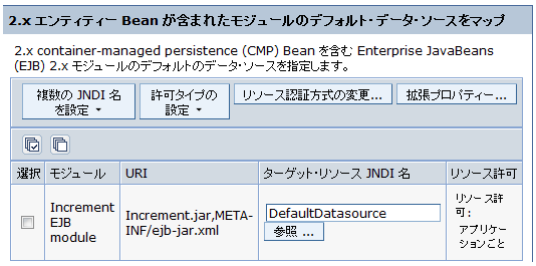


図 5-54 アプリケーション・インストール画面(8)

ここで指定できるオプションについて、手順の概要を記載します。オプションの詳細については下記の Information Center を参照して下さい。

WAS V8.5 Information Center

「すべての 2.x CMP Bean のデータ・ソースのマップ」

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.doc/ae/udat_mapdds2xcmp.html

●複数の JNDI 名を設定

1. テーブルで 1 つ以上のチェック・ボックスを選択します。
2. **[複数の JNDI 名を設定]**をクリックし、既存のリソース JNDI 名を選択します。
3. **[ターゲット・リソース JNDI 名]**に選択された JNDI 名が反映されます。

●許可タイプの設定

許可タイプの選択が「コンテナー」の場合

1. テーブルで 1 つ以上のチェック・ボックスを選択します。
2. **[アプリケーションごと]**または**[コンテナー]**を選択します。
3. **[リソース許可]**に許可タイプが反映されます。

●リソース認証メソッドの変更

1. テーブルで 1 つ以上のチェック・ボックスを選択します。
2. **[なし]**、**[デフォルト・メソッドの使用（多対 1 のマッピング）]**、**[トラステッド接続の使用（1対1のマッピング）]**、**[カスタム・ログイン構成を使用]**のいずれかを選択します。
3. **[適用]**をクリックします。リソース許可にリソース認証方式が反映されます。

●拡張データ・ソース・プロパティー

1. 拡張データ・ソース・カスタム・プロパティーの名前と値のペアを指定します。
拡張データ・ソース・プロパティーは WAS V7.0 で登場した新機能です。DB2 Universal JDBC Driver または DB2 Using IBM JCC ドライバーを使用する場合に、アプリケーションごとに異なるデータ・ソース・プロパティーを定義する事ができます。また、データ・ソースに対して既に定義されている、コア・プロパティー以外のプロパティーは設定をオーバーライドすることもできます。同じデータ・ソースに接続されているリソース間で 1 つの接続プールを使用する事も可能です。詳細については下記の Information Center を参照して下さい。
WAS V8.5 Information Center

「アプリケーション・レベルでの DB2 データ・ソース定義の拡張」
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.doc/ae/tdat_heteropool.html

12. ここでは、アプリケーションのコンテナー管理パーシスタンス (CMP) 2.x Bean を、そのアプリケーションで利用可能なデータ・ソースにマップします。テーブルの各行は、特定の EJB モジュール内の CMP Bean に対応しています。 Bean のデータ・ソース・マッピング・ターゲットの JNDI 名は、それらをアプリケーション・アセンブリー時に一緒にバインドした場合のみ行に表示されます。表示される各データ・ソースには、該当するセキュリティー構成が表示されます。

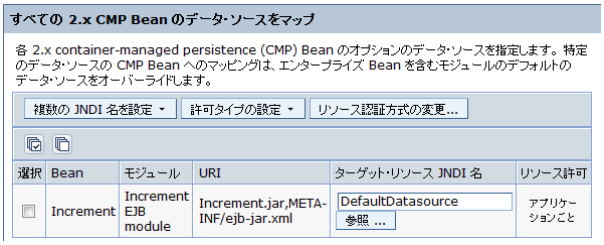


図 5-55 アプリケーション・インストール画面(9)

13. ここでは、Web モジュールで使用される仮想ホストをマップします。

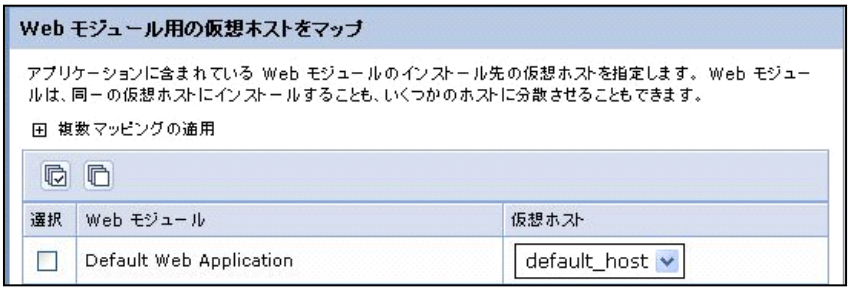


図 5-56 アプリケーション・インストール画面(10)

14. ここでは、アプリケーションのデプロイメント記述子に定義されているコンテキスト・ルートを変更する事ができます。



図 5-57 アプリケーション・インストール画面(11)

15. インストールするアプリケーションに、EJB が含まれている場合、EJB のメソッドにセキュリティー役割を追加する、もしくはセキュリティーを適用しない設定を行います。

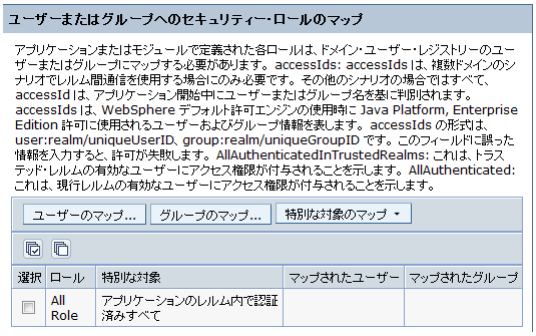


図 5-58 アプリケーション・インストール画面(12)

16. ここでは、アプリケーションあるいは Web モジュールに対して Java Authentication SPI (JASPI) 認証を有効または無効にしたり、アプリケーションまたは Web モジュールのメッセージの認証に使用される JASPI 認証プロバイダーの名前を指定したりします。

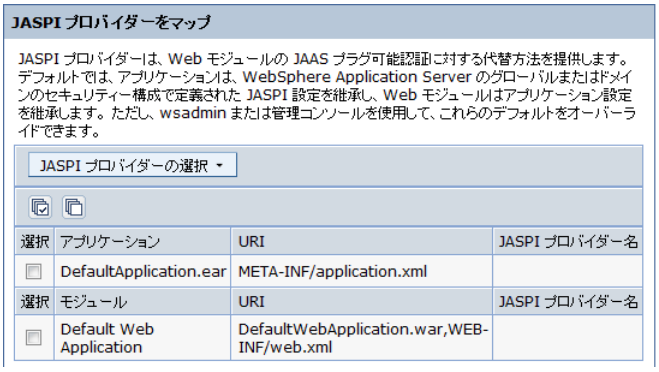


図 5-59 アプリケーション・インストール画面(13)

17. ユーザーを役割にマップする前に、このページを使用して、無保護 Enterprise JavaBeans (EJB) バージョン 2.x メソッドに、正しいレベルの保護が適用されていることを確認してください。この管理コンソール・ページで、ユーザーが特定の EJB モジュールに アクセスできるかどうかを指定することができます。



図 5-60 アプリケーション・インストール画面(14)

18. Java Platform, Enterprise Edition (Java EE) エンタープライズ・アーカイブ (EAR ファイル) のモジュールのビルド ID を表示します。モジュールのビルド ID は、モジュールまたはアプリケーションの MANIFEST.MF ファイルでビルド ID が指定されている場合に表示されます。

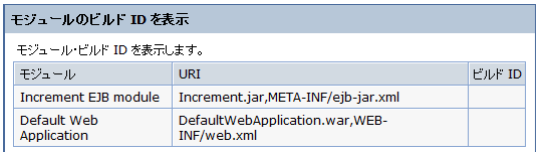


図 5-61 アプリケーション・インストール画面(15)

19. 「次のアプリケーション・エディションから既存の作業クラスの複製」リストから作業クラスを選択し、「次へ」をクリックします。 作業クラスにより、そのアプリケーション・エディションに対するデフォルトのルーティング・ルールが設定されます。 アプリケーションの作業クラスはそのアプリケーションのルーティング・ポリシーを構成します。 既存のアプリケーション・エディションを選択した場合、そのすべての作業クラスは、新規のアプリケーション・エディション名で適切に名前変更されたデフォルトを使用して複製されます。 アプリケーション・エディションが選択されない場合は、デフォルトが作成されます。

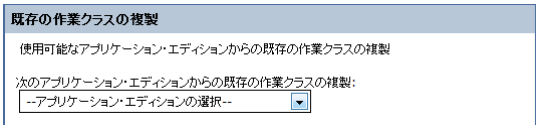


図 5-62 アプリケーション・インストール画面(16)

20. ここまで行ってきた設定の概要が表示されます。設定に問題がなければ、**[終了]**をクリックして下さい。



要約

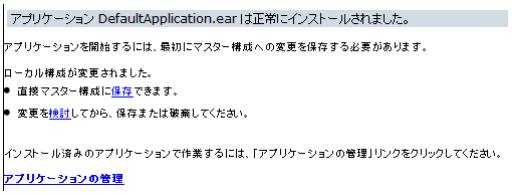
インストール・オプションの要約

オプション	値
JavaServer Pages ファイルのプリコンパイル	いいえ
アプリケーションをインストールするディレクトリー	
アプリケーションを配付する	はい
バイナリー構成の使用	いいえ
エンタープライズ Bean のデプロイ	いいえ
アプリケーション名	DefaultApplication.ear
アプリケーション・エディション	
エディションの説明	
リソース用の MBean を作成する	はい
Web および EJB モジュールのクラス再ロード設定をオーバーライドする	いいえ
再ロード間隔 (秒)	
Web サービスをデプロイする	いいえ
入力のオフ/警告/失敗の検証	警告
プロセス組み込み構成	いいえ
ファイル・アクセス権	.*\.dll=755#.*\.so=755#.*\.a=755#.*\.sl=755
アプリケーション・ビルド ID	不明
リモート・リソースに対する組み込みディスパッチを許可する	いいえ
リモート・リソースからの組み込みサービスを許可する	いいえ
ビジネス・レベル・アプリケーション名	
非同期要求ディスパッチ・タイプ	使用不可
EJB 参照ターゲットを自動で解決できるようにする	いいえ
クライアント・モジュールをデプロイする	いいえ
クライアント・デプロイメント・モード	分離
スキーマを検証する	いいえ
セル/ノード/サーバー	ここをクリックする

⚠ アプリケーション・モジュールが Web サーバーにマップされませんでした。

図 5-63 アプリケーション・インストール画面(17)

21. **[終了]**をクリックすることで、アプリケーションのインストールが行われます。この時点では、ノードに配布されていませんので、必要に応じてマスター構成に**[保存]**、変更を**[検討]**するか、**[アプリケーションの管理]**を選択して下さい。



アプリケーション DefaultApplication.ear は正常にインストールされました。

アプリケーションを開始するには、最初にマスター構成への変更を保存する必要があります。

ローカル構成が変更されました。

- 直接マスター構成に保存できます。
- 変更を検討してから、保存または破棄してください。

インストール済みのアプリケーションで作業するには、「アプリケーションの管理」リンクをクリックしてください。

アプリケーションの管理

図 5-64 アプリケーション・インストール画面(18)

5-2-1-7 Enhanced EAR

Enhanced EAR とは、Java EE アプリケーション（従来の EAR ファイル）そのものに加えて、リソース定義やプロパティーといったアプリケーションの稼動に必要となる情報を全てアーカイブしたファイルのことです。Enhanced EAR をサポートしている製品としては、Rational Application Developer V6 以降（以下 RAD）、Rational Application Developer Assembly and Deploy V7.5 以降（以下 RAD A&D）、WAS V6 以降などが挙げられます。

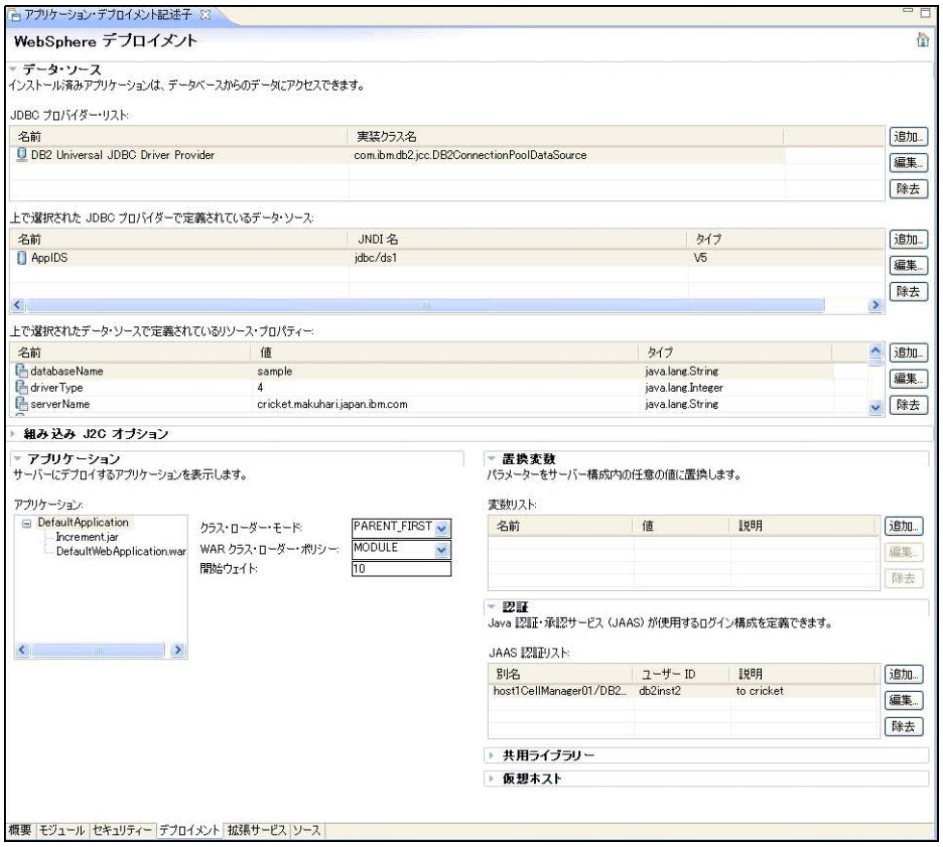


図 5-65 RAD による Enhanced EAR の設定

RAD や RAD A&D などで開発してエクスポートした Enhanced EAR を WAS にデプロイする際、Enhanced EAR にはリソース定義情報が含まれていますので、デプロイ先のサーバー側で定義する必要はありません。このように、Enhanced EAR のメリットは、さまざまなサーバーにアプリケーションを導入する場合でもサーバー毎に毎回リソース定義をする必要がない、ということです。Enhanced EAR が保持することのできる情報は以下の通りです:

表 5-1 Enhanced EAR で設定可能な項目

リソース	有効範囲
データ・ソース	アプリケーション
組み込みリソース・アダプター	アプリケーション
クラス・ローダー・ポリシー	アプリケーション
置換変数	アプリケーション
JAAS 認証	セル
共用ライブラリー	サーバー
仮想ホスト	セル

Enhanced EAR をデプロイするにあたって、リソース定義情報を含んでいるがゆえに考慮しなければならないことがあります。WAS V7.0 以降におけるリソース定義の有効範囲は「セル」、「クラスター」、「ノード」、「サーバー」、「アプリケーション」の五階層となっており、Enhanced EAR を使用した場合によく使われるデータ・ソースなどの有効範囲は「アプリケーション」となります。また、五階層の中では、アプリケーションに近い方の定義（範囲の狭い方の定義）が優先されます。例えば、サーバーと Enhanced EAR の両者で同名のリソースを定義している場合は、Enhanced EAR の中の定義情報が優先されるということになります。したがって、テスト用のリソース定義などを含んだ Enhanced EAR を本番環境に導入する際、導入先のサーバーで設定しているリソース定義を使用するためには、Enhanced EAR 内のリソース定義情報を削除するなどの対応が必要になります。

WAS V6.1 以降、管理コンソールからも Enhanced EAR 内のリソース定義情報を確認、変更する事が可能になりました。以下の手順でリソース定義情報を確認、変更します。

1. 管理コンソールで**[アプリケーション]→[アプリケーション・タイプ]→[WebSphere エンタープライズ・アプリケーション]**からアプリケーション名を選択し、**[参照]**の下にある**[アプリケーション有効範囲リソース]**をクリックします。



図 5-66 EnhancedEAR の確認・修正(1)

2. 確認、変更するリソースをクリックします。

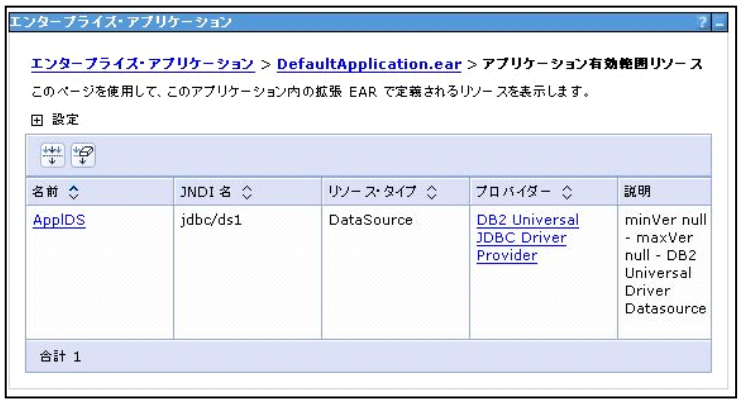


図 5-67 EnhancedEAR の確認・修正(2)

但し、この手順では新規で定義を作成する事は出来ません。EnhancedEAR の定義情報を新規で作成するには、RAD A&D や wsadmin コマンド・ツールを使用して下さい。

5-2-2 ビジネス･レベル･アプリケーション導入の流れ

「5-2-1-6 アプリケーション導入の流れ」の手順に従うと、デフォルトではアプリケーションと同名のビジネス･レベル･アプリケーションが自動的に作成されます。任意の名前のビジネス･レベル･アプリケーションを作成し、モジュールをそのビジネス･レベル･アプリケーションに登録するには、以下の手順で導入する必要があります。

1. 空のビジネス･レベル･アプリケーションを作成する
2. アプリケーション･モジュールやライブラリーなどをアセットとしてインポートする
3. 1.で作成したビジネス･レベル･アプリケーションに 2.で作成したアセットを構成単位として登録する

5-2-2-1 ビジネス･レベル･アプリケーションの作成

1. 管理コンソールで**[アプリケーション]→[アプリケーション･タイプ]→[ビジネス･レベル･アプリケーション]**を選択し、**[新規作成]**をクリックします。



図 5-68 ビジネス･レベル･アプリケーションの作成(1)

2. ビジネス･レベル･アプリケーションの名前と説明を入力し、**[適用]**をクリックします。

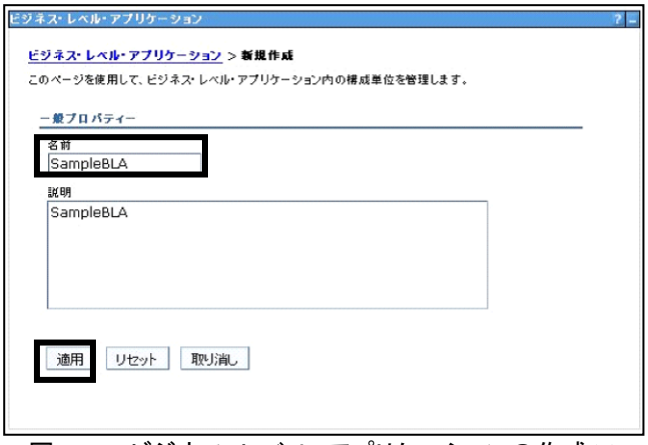


図 5-69 ビジネス･レベル･アプリケーションの作成(2)

3. 構成を保存します。

5-2-2-2 アセットのインポート

セル内で利用できるアセットを登録します。アセットとして登録できるのは、圧縮(zip)ファイル、EJB ファイル、JARファイル、EARファイル、サービス・コンポーネント・アーキテクチャー(SCA)複合JARファイル、メディエーション JAR ファイル、共用ライブラリーJAR ファイル、PHP アプリケーションなどです。

1. 管理コンソールで**[アプリケーション]→[アプリケーション・タイプ]→[アセット]**を選択し、**[インポート]**をクリックします。

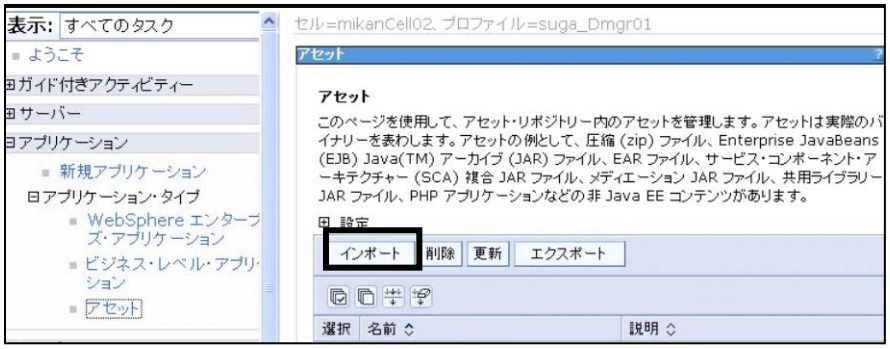


図 5-70 アセットのインポート(1)

2. インポートするアセットの絶対パス名を指定します。
[ローカル・ファイル・システム]と**[リモート・ファイル・システム]**の違いについては5-2-1-6 アプリケーション導入手順」の説明を参照して下さい。
指定後、**[次へ]**をクリックします。

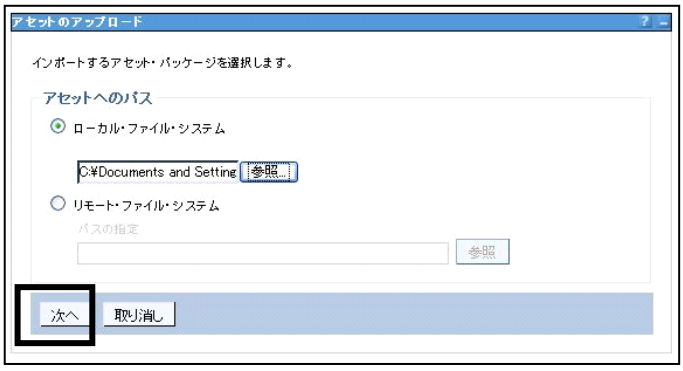


図 5-71 アセットのインポート(2)

3. アセットの設定を行います。各オプションの意味は以下の通りです。
   - アセットの説明
     任意の説明を入力します。
   - アセット・バイナリーの宛先 URL
     アセット・ファイルをインポートする先のディレクトリーを絶対パスで指定します。デフォルト値は<Profile_root>/installedAssets/<asset_name>/BASE/<asset_name>です。
   - アセット・タイプ・アスペクト
     アセット・コンテンツのタイプを示しています。この値は読み取り専用で、変更する事は出来ません。
   - ファイル・アクセス権
     [アセット・バイナリーの宛先 URL]に展開されているアセット・バイナリー・ファイルに対するアクセス権を設定します。

ここで指定されるのはアセット・レベルの許可です。ノード・レベルの構成でもアセット・バイナリーに対するアクセス権を指定する事が出来ますが、アセット・レベルのアクセス制限はノード・レベルで指定されたのと同等か、より限定された内容のものに限られるため注意して下さい。

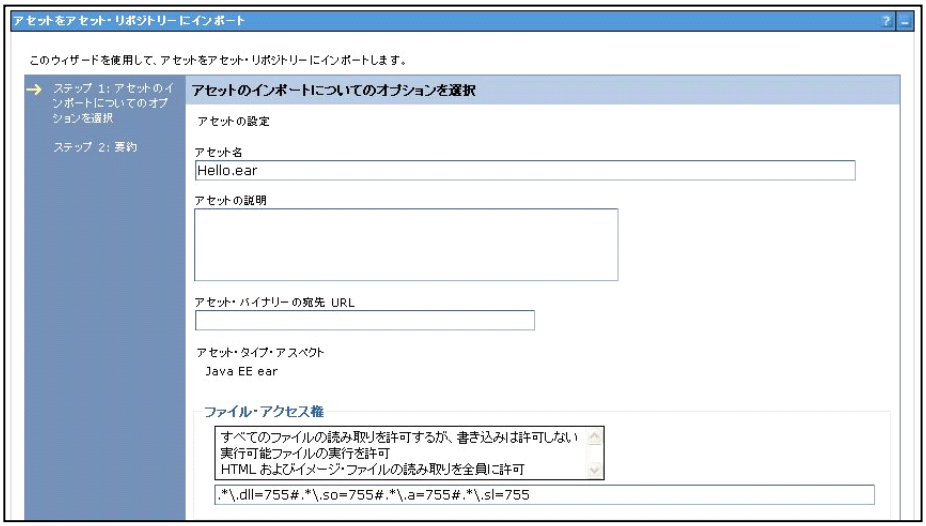


図 5-72 アセットのインポート(3)

■ アセット関係
このアセットと関係する、他のアセットを指定します。関係を追加するには**[関係の管理…]**をクリックします。

■ アセットの検証
アセットのインポートまたは更新中に、アセットが参照しているリソースがそのアセットのデプロイメント・ターゲットの有効範囲で定義されているかどうかを検証します。妥当性の検証を行うにはチェックを入れます。デフォルトでは検証を行わない設定になっています。

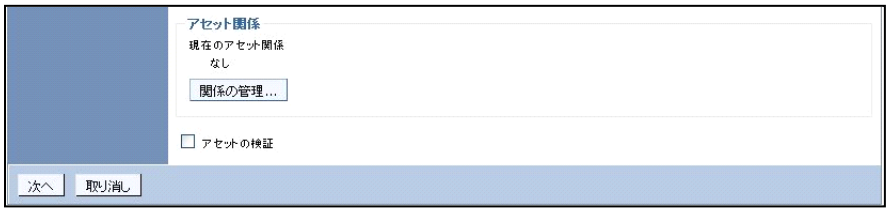


図 5-73 アセットのインポート(4)

5-2-2-3 構成単位の登録

ビジネス・レベル・アプリケーションにアセットを登録します。

1. 管理コンソールで**[アプリケーション]→[アプリケーション・タイプ]→[ビジネス・レベル・アプリケーション]**を選択し、アセットを登録するビジネス・レベル・アプリケーション名を選択します。

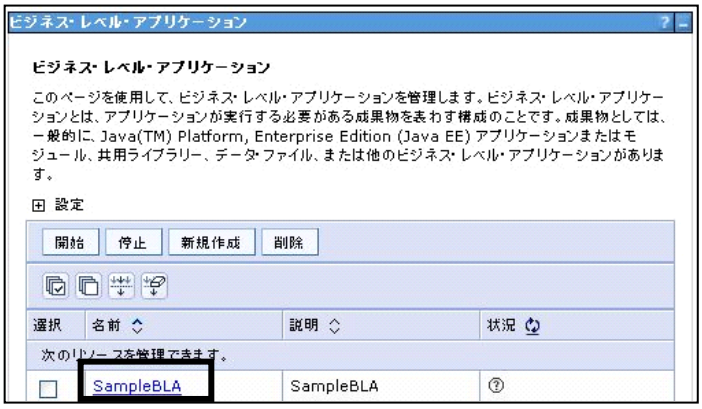


図 5-74 構成単位の登録(1)

2. **[デプロイ済みアセット]**の**[追加]**をクリックし、**[アセットの追加]**を選択します。



図 5-75 構成単位の登録(2)

3. ビジネス・レベル・アプリケーションに追加するアセットを選択し、**[続行]**をクリックします。



図 5-76 構成単位の登録(3)

以降の手順は、追加するアセットが EAR などの Java EE コンテンツであれば、5-2-1-6 アプリケーションの導入手順」と同様です。
追加するアセットが JAR ファイルなどのライブラリー・ファイルの場合は、以下の手順となります。

4. オプションの設定を行います。各オプションの意味は以下の通りです。

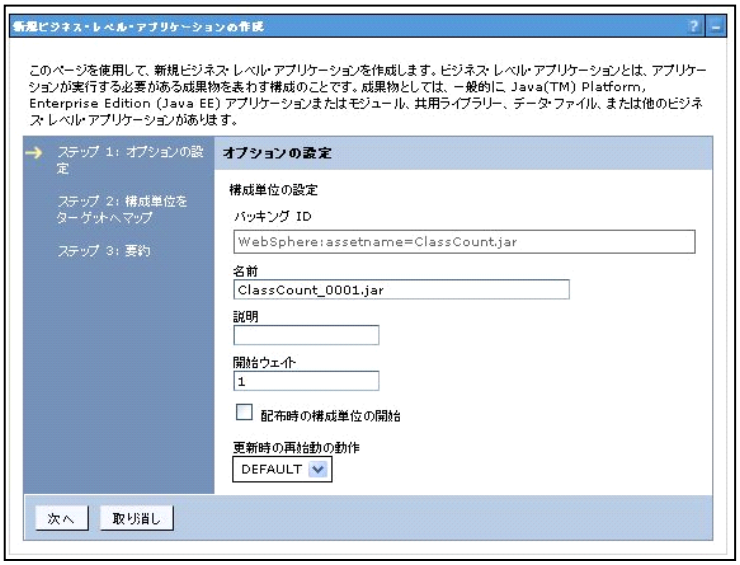


図 5-77 構成単位の登録(4)（共用ライブラリー・ファイルの場合）

● パッキング ID

アプリケーション管理ドメインに登録されている構成単位の固有 ID が設定されます。ID のフォーマットは「WebSphere:<unit_typename>=unit_name,<unit_typeversion>=version_number」です。このフィールドは通常編集する事は出来ません。

● 名前

構成単位名を入力します。ライブラリー・ファイルを登録する際、デフォルトでは、ファイル名に加えてセル環境内で連続した番号が採番されます。

● 説明

説明を入力します。

● 開始ウェイト

サーバーの始動時に構成単位がどの順序で始動するかを指定します。開始ウェイトが一番低い構成単位から開始します。

「開始ウェイト」に対して設定した値によって、ビジネス・レベル・アプリケーション内の構成単位の重要度、つまり重みが決まります。例えば、ビジネス・レベル・アプリケーション内で最も重要な構成単位の「開始ウェイト」には、1 を指定します。ビジネス・レベル・アプリケーション内で次に重要な構成単位の「開始ウェイト」には、2 を指定します。3 以降も同様です。

● 配布時の構成単位の開始

構成単位を配下のノードに配布した後、構成単位を開始するかどうかを指定します。デフォルトでは開始しない設定になっています。

● 更新時の再始動の動作

構成単位の更新後に、本製品がデプロイメント・ターゲットを再始動するかどうかを指定します。通常、構成単位は 1 つ以上のデプロイメント・ターゲットにマップされますが、この設定により、構成単位の編集後に本製品がそれらのターゲットを再始動するかどうかが決まります。デフォルトでは「DEFAULT」に設定されています。以下のオプションを設定する事が出来ます。

◆ ALL

構成単位の編集後に、構成単位の各ターゲット・ノードを再始動します。

◆ DEFAULT

構成単位の編集後に、同期プラグインにより制御されるノードを再始動します。

◆ NONE

構成単位の編集後に、ノードを再始動しません。

5. 構成単位をデプロイメント・ターゲットへマップします。**[使用可能]**の欄からマップするデプロイメント・ターゲットを選択し、右向きの矢印をクリックします。**[選択済み]**の欄に選択したターゲットが表示されているのを確認し、**[次へ]**をクリックします。

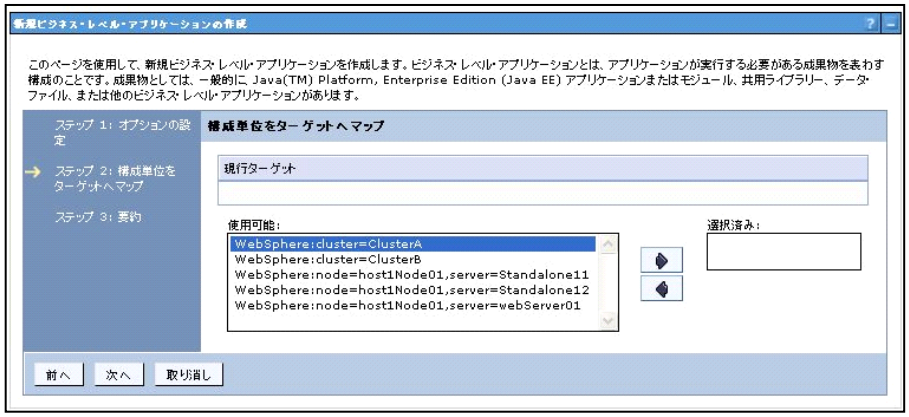


図 5-78 構成単位の登録(5)（共用ライブラリー・ファイルの場合）

6. 要約が表示されるので内容を確認し、**[終了]**をクリックします。
7. 構成を保存します。

5-2-2-4 ビジネス・レベル・アプリケーションにおけるファイルの配布

セル環境にアセットをインポートすると、デプロイメント・マネージャー・ノードのアセット・リポジトリーにアセット・ファイルが格納されます。ビジネス・レベル・アプリケーションに構成単位としてアセットが登録されると、アセット・リポジトリーから config ディレクトリー配下にアーカイブ・ファイルがコピーされ、同期のタイミングで各管理対象ノードへ転送されます。

構成単位として登録されるアセットの種類によって、ファイルの配布元ディレクトリー、配布先ディレクトリー、配布後の動きは異なります。

Java EE アプリケーションが構成単位として登録された場合、デフォルトではデプロイメント・マネージャー・ノードの config ディレクトリー配下の applications ディレクトリーの配下に、ear ファイル(アーカイブされた状態)がコピーされます。ここには ear ファイルのデプロイメント・ディスクリプター(以下、DD)も保存されているので、管理ツールから DD を参照することができます。そのアプリケーションに「配布の使用可能」という設定がされている場合は、デプロイメント・マネージャーとノード・エージェントの同期のタイミングで、各管理対象ノードの config ディレクトリー配下の applications ディレクトリー配下に ear ファイル(アーカイブされた状態)が格納されます。そして、各管理対象ノードの installedApps ディレクトリー配下にアプリケーションが展開され、特定の AS とマッピングすることで公開され、Web クライアントからはサービスとして使用できるようになります。

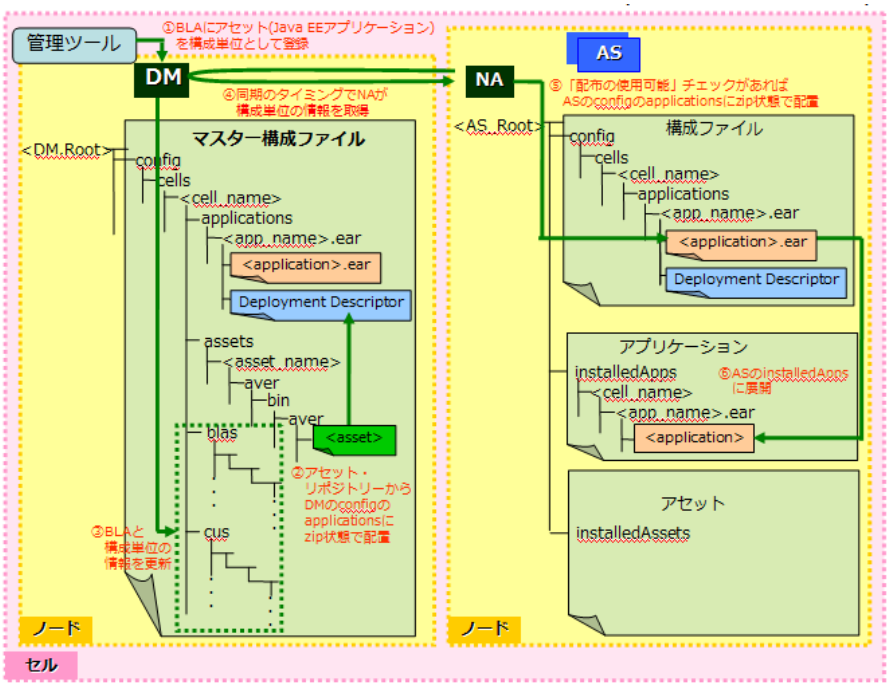


図 5-79 ビジネス・レベル・アプリケーションにおけるファイルの配布(Java EE アプリケーション)

Jar ファイルなどが構成単位として登録された場合、デフォルトではデプロイメント・マネージャー・ノードの config ディレクトリー配下の assets ディレクトリーの配下に、ファイルがコピーされます。デプロイメント・マネージャーとノード・エージェントの同期のタイミングで、各管理対象ノードの config ディレクトリー配下の assets ディレクトリー配下にファイルが格納されます。そして、各管理対象ノードの installedAssets ディレクトリー配下にファイルがコピーされ、使用可能になります。

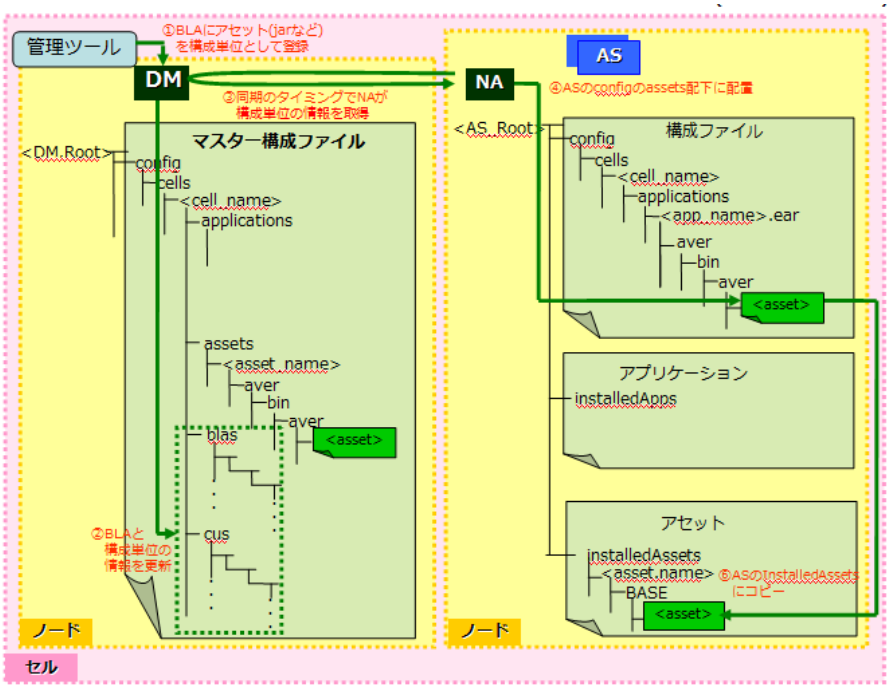


図 5-80 ビジネス・レベル・アプリケーションにおけるファイルの配布(ライブラリなど)

5-3 アプリケーションの管理

5-3-1 Fine-Grained Update

Fine-Grained Update とは、アプリケーションを部分的に追加、更新、削除できる機能です。これらの操作は、管理コンソールや wsadmin などといった WAS が提供している管理ツールを使用して実行することができます。V5.x でアプリケーションを部分的に更新するためには、アプリケーションが展開されているディレクトリーのファイルを直接置き換えるなどといった手動による方法しかありませんでした。WAS V6.0 以降のバージョンでは、管理ツールから Fine-Grained Update を実行すれば、アプリケーションの部分的な更新を展開ディレクトリーに自動で反映させることができます。

アプリケーションを更新できる単位としては、アプリケーション、モジュール、ファイル、複数ファイルという 4 種類がサポートされています。それぞれの詳細については、後述します。

この機能を使用した場合、アプリケーション・サーバーを停止することなくアプリケーションを更新することができます。更新する際、更新内容に応じて、アプリケーション全体あるいは更新の影響を受ける一部だけが自動で停止され、更新されたのち再起動されます。

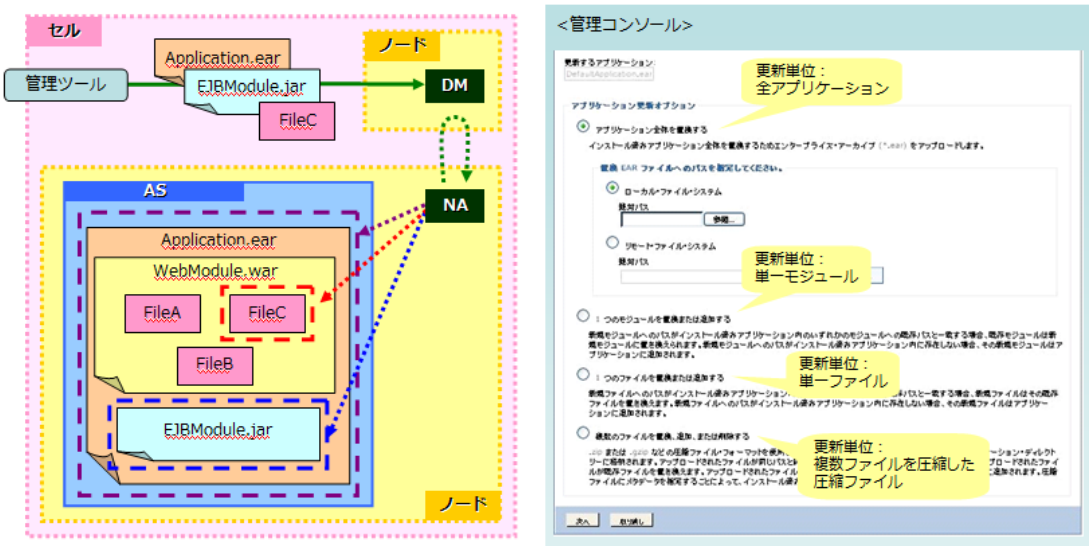


図 5-81 アプリケーションの更新単位

5-3-1-1 Fine-Grained Update の設定手順

管理コンソールを使用して Fine-Grained Update を行う手順を説明します。

1. 管理コンソールで**[アプリケーション]→[アプリケーション・タイプ]→[WebSphere エンタープライズ・アプリケーション]**を選択し、インストールされているアプリケーションの一覧から更新したいアプリケーションを選択して**[更新]**ボタンをクリックします。



図 5-82 アプリケーション更新画面(1)

ここから更新ウィザードの画面になります。更新ウィザードでは、更新オプションを選択して各設定箇所に適切な値を入力します。
更新オプションには「アプリケーション全体を置換する」、「1 つのモジュールを置換または追加する」、「1 つのファイルを置換または追加する」、「複数のファイルを置換、追加、または削除する」という 4 種類の選択肢がありますので、アプリケーションの更新単位に応じて、この中から 1 つを選択します。以降はアプリケーションのインストール・ウィザードと同様に各設定箇所に適切な値を入力しながら進めていき、最後に保存します。

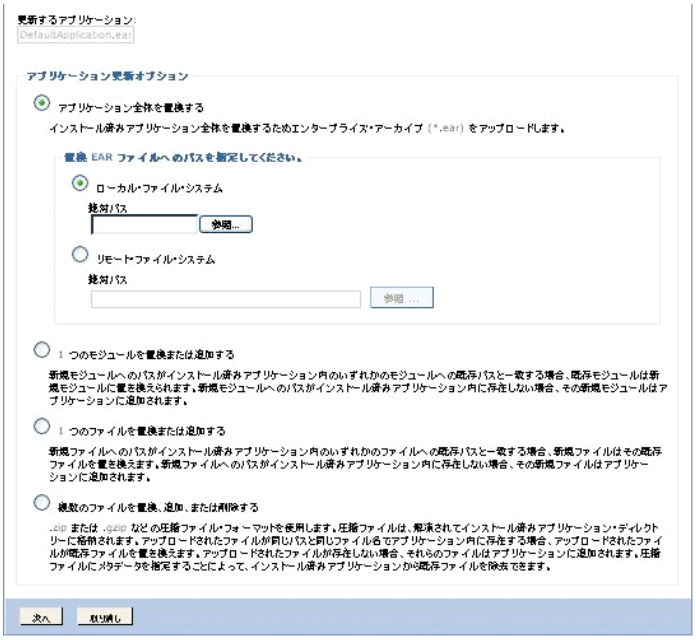


図 5-83 アプリケーション更新画面(2)

「アプリケーション全体を置換する」は ear ファイル単位での更新のことであり、「1 つのモジュールを置換または追加する」とは war ファイル、jar ファイル、rar ファイルなどの単位での更新のことです。また、「1 つのファイルを置換または追加する」とは class ファイルや jpg のような画像ファイルなどの単位での更新のことであり、これが Fine-Grained Update で更新可能な最小の単位となります。「複数のファイルを置換、追加、または削除する」とは、複数のファイルやモジュールをアーカイブした zip ファイルや gzip ファイルなどの単位での更新のことです。

更新オプションの構成方法の詳細は以下の通りです。

■ 全アプリケーションの更新をする場合

更新オプションの「アプリケーション全体を置換する」は、アプリケーション（ear ファイル）単位で更新や置換したい場合に選択します。V5.x でも全アプリケーションを更新する機能はありましたが、WAS V5.x では更新前には対象となるアプリケーションを手動で停止する必要があったのに対して、WAS V6.0 以降のバージョンでは、アプリケーションで停止および再起動が必要な部分だけが、自動で停止および再起動されるようになりました。
例えば、あるアプリケーション・サーバーにインストールされている「SampleApplication.ear」というアプリケーションをまるごと新しいバージョンの ear ファイルで置換して更新したい場合は、「アプリケーション全体を置換する」を選択します。

設定は、ファイル・システムの選択とパスの指定です。ファイル・システムは、選択するearファイルが管理コンソールを開いているブラウザと同じノード上に存在している場合には**[ローカル・ファイル・システム]**を、別のノード上に存在している場合には**[リモート・ファイル・システム]**を選択します。パスの指定は、選択する ear ファイルが存在する場所を指定します。

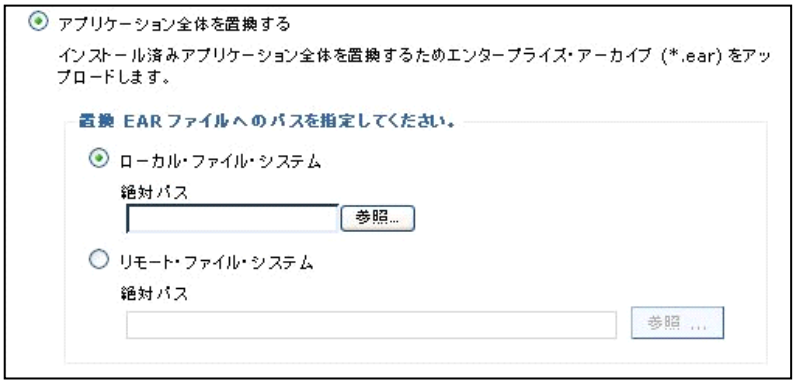


図 5-84 アプリケーション更新画面(3)

■ 1 つのモジュールを置換または追加する場合

更新オプションの「1 つのモジュールを置換または追加する」は、モジュール（war ファイル、jar ファイル、rar ファイル）単位で更新や置換したい場合に選択します。

例えば、あるアプリケーション・サーバーにインストールされている「ApplicationA.ear」というアプリケーションの「WebModuleA.war」というモジュールを「WebModuleAnew.war」というモジュールで置換して更新したい場合は、「1 つのモジュールを置換または追加する」を選択します。

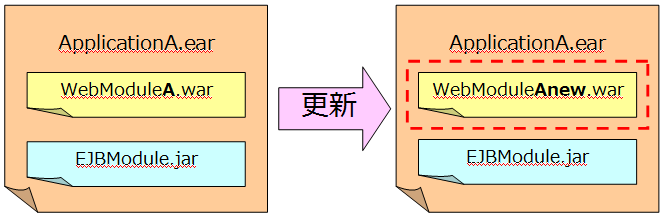


図 5-85 単一モジュールの更新

設定は、モジュールへの相対パス、ファイル・システムの選択、パスの指定、コンテキスト・ルートの指定です。モジュールへの相対パスは旧バージョンのモジュールへの相対パスを指定します。

ここに、旧バージョンのモジュールへの相対パスを指定した場合はそのモジュールの置換となり、新バージョンのモジュールへの相対パスを指定した場合はそのモジュールの追加となります。ファイル・システムは、選択するモジュールが管理コンソールを開いているブラウザと同じノード上に存在している場合には**[ローカル・ファイル・システム]**を、別のノード上に存在している場合には**[リモート・ファイル・システム]**を選択します。パスの指定は、選択するモジュールが存在する場所を指定します。コンテキスト・ルートは、スタンドアローンの Web モジュールの場合のみ使用されます。

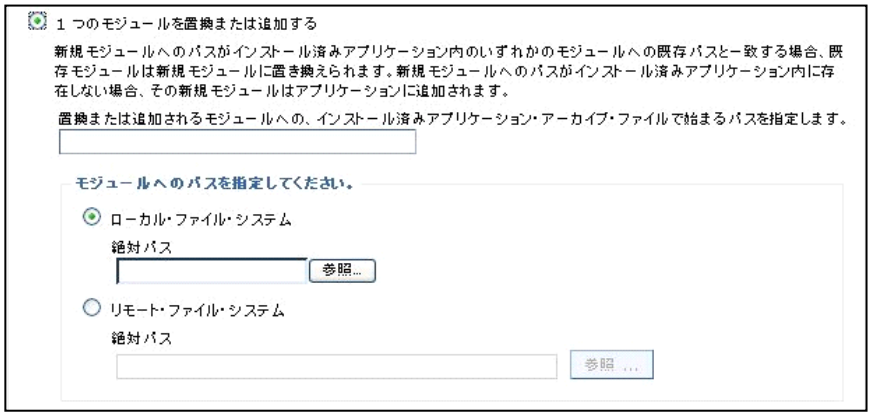


図 5-86 単一モジュールの更新画面

■ 1 つのファイルを置換または追加する場合

更新オプションの「1 つのファイルを置換または追加する」は、ファイル（class ファイル、jpg ファイル）単位で更新や置換したい場合に選択します。

例えば、あるアプリケーション・サーバーにインストールされている「ApplicationA.ear」というアプリケーション内の「WebModuleA.war」というモジュール内の「FileA.jpg」というファイルを「FileAnew.jpg」というファイルで置換して更新したい場合は、「1 つのファイルを置換または追加する」を選択します。

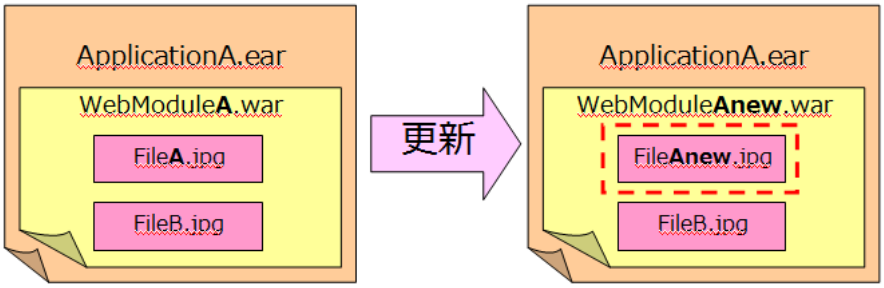


図 5-87 単一ファイルの更新

設定は、ファイルへの相対パス、ファイル・システムの選択、パスの指定です。ファイルへの相対パスは旧バージョンのファイルへの相対パスを指定します。ここに、旧バージョンのファイルへの相対パスを指定した場合はそのファイルの置換となり、新バージョンのファイルへの相対パスを指定した場合はそのファイルの追加となります。ファイル・システムは、選択するファイルが管理コンソールを開いているブラウザと同じノード上に存在している場合には**[ローカル・ファイル・システム]**を、別のノード上に存在している場合には**[リモート・ファイル・システム]**を選択します。パスの指定は、選択するファイルが存在する場所を指定します。

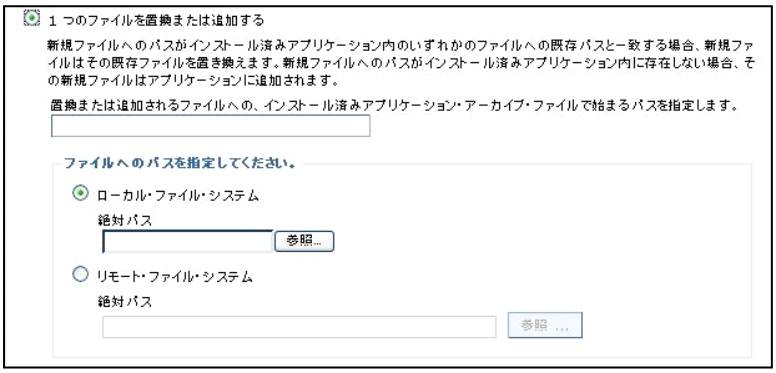


図 5-88 単一ファイル更新画面

■ 複数のファイルの更新や置換をする場合

更新オプションの「複数のファイルを置換、追加、または削除する」は、複数のファイルやモジュールをアーカイブした zip ファイルや gzip ファイルなどの単位で更新や置換したい場合に選択します。

例えば、あるアプリケーション・サーバーにインストールされている「ApplicationA.ear」というアプリケーション内の「WebModuleA.war」というモジュール内の Dir4/Dir5 ディレクトリーに「FileD.jpg」というファイルを追加し、「EJBModuleA.jar」というモジュールを「EJBModuleAnew.jar」というモジュールで置換して更新したいような場合は、「複数のファイルを置換、追加、または削除する」を選択します。アーカイブ・ファイルは、追加や更新の場合は新バージョンのファイルやモジュールをディレクトリー構造にまとめて zip や gzip でアーカイブして作成します。このとき、旧バージョンのパスと一致するファイルの場合は追加、一致しないファイルの場合は更新となります。

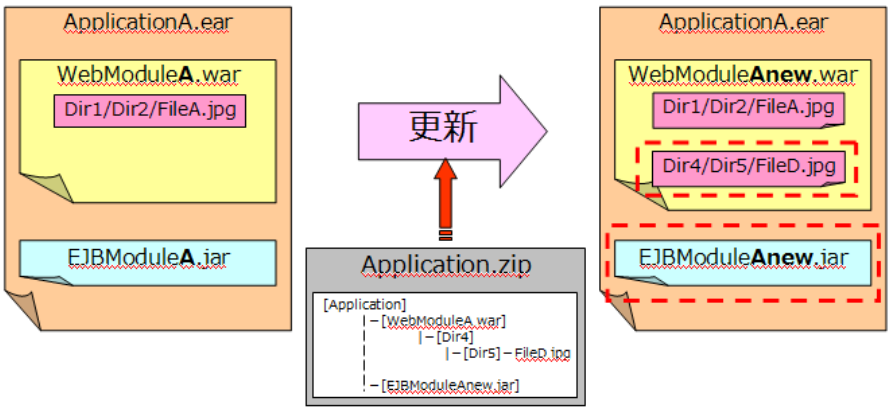


図 5-89 アプリケーションの一部を更新

設定は、ファイル・システムの選択とパスの指定です。ファイル・システムは、選択するアーカイブ・ファイルが管理コンソールを開いているブラウザと同じノード上に存在している場合には[ローカル・ファイル・システム]を、別のノード上に存在している場合には[リモート・ファイル・システム]を選択します。パスの指定は、選択するアーカイブ・ファイルが存在する場所を指定します。

■ 複数のファイルを削除する場合

更新オプションの「複数のファイルを置換、追加、または削除する」は、複数のファイルやモジュールをアーカイブした zip ファイルや gzip ファイルなどの単位で削除したい場合にも選択します。

例えば、あるアプリケーション・サーバーにインストールされている「ApplicationA.ear」というアプリケーション内の「WebModuleA.war」というモジュール内の Dir4/Dir5 ディレクトリーに「FileD.jpg」というファイルを追加し、「EJBModuleA.jar」というモジュールを「EJBModuleAnew.jar」というモジュールで

置換して更新し、更に Dir3 ディレクトリーにある「FileB.jpg」と「FileC.jpg」というファイルを削除したいような場合は、「複数のファイルを置換、追加、または削除する」を選択します。
アーカイブ・ファイルは、追加や更新の場合は新バージョンのファイルやモジュールをディレクトリー構造にまとめて zip や gzip でアーカイブして作成しますが、削除の場合は META-INF の配下の ibm-partialapp-delete.props ファイルに削除対象となるファイルのパスを指定して、一緒にアーカイブします。

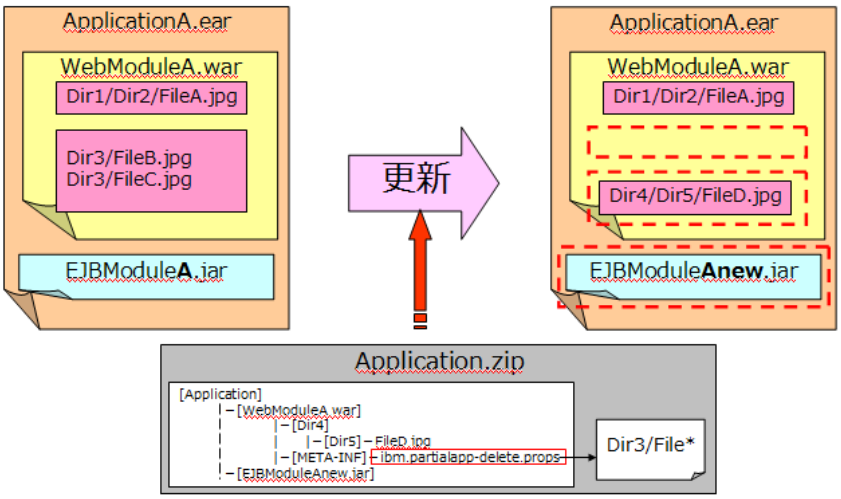


図 5-90 アプリケーションの一部を削除

設定は、ファイル・システムの選択とパスの指定です。ファイル・システムは、選択するアーカイブ・ファイルが管理コンソールを開いているブラウザと同じノード上に存在している場合には**[ローカル・ファイル・システム]**を、別のノード上に存在している場合には**[リモート・ファイル・システム]**を選択します。パスの指定は、選択するアーカイブ・ファイルが存在する場所を指定します。

5-3-2 ロールアウト・アップデート

ロールアウト・アップデート とは、クラスター上のアプリケーションを更新する際、サーバークラスターに対する更新を全てのクラスター・メンバーに同時に反映させるのではなく、ノード毎に順番に反映させていくという機能です。この機能を使用すると、クラスターを構成している複数ノードのうち常にいずれかのノード上のクラスター・メンバーは稼動しているため、サービス全体としては停止することなくアプリケーションの更新を行うことができます。

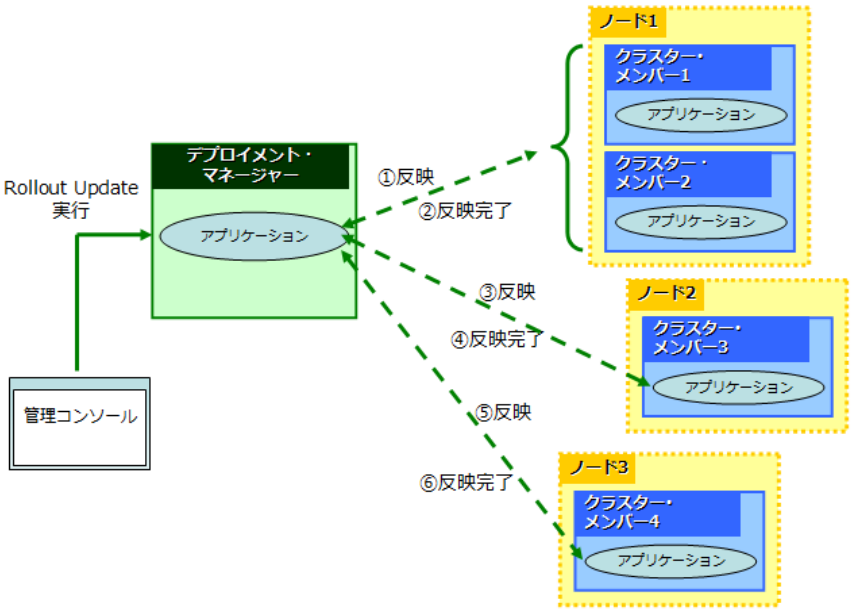


図 5-91 ロールアウト・アップデートの動作イメージ

5-3-2-1 ロールアウト・アップデート処理の流れ

各ノードへ順番に反映させる更新処理の流れについて説明します。

まず、ロールアウト・アップデート を実行すると、アプリケーションへの更新内容がデプロイメント・マネージャーの保持するマスター構成ファイルに保存されます。この時点ではまだ各ノードとの同期はとれていないので、アプリケーションの変更は反映されていません。次に、クラスター・メンバーが存在するある 1 つのノード上において、全てのクラスター・メンバーを停止します。停止が完了したら、デプロイメント・マネージャーはそのノードと同期をとり、アプリケーションの変更を反映させます。同期が完了したら、そのノード上で停止した全てのクラスター・メンバーを始動します。この時点では、このノード上のアプリケーションだけが新バージョンに更新されていることになります。ここまでの処理をクラスター・メンバーが存在する全てのノードに対して繰り返します。このようにして、最後には全てのクラスター・メンバーに対してアプリケーションの更新が完了します。

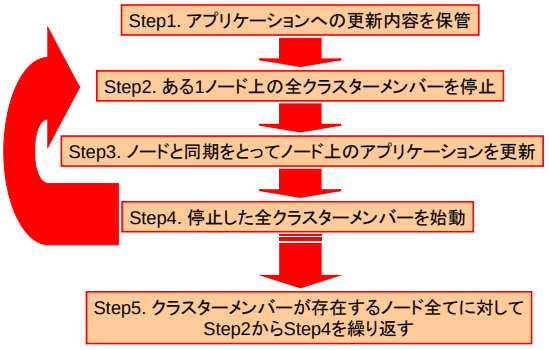


図 5-92 ロールアウト・アップデートの動作

5-3-2-2 ロールアウト・アップデートの設定手順

管理コンソールを使用して ロールアウト・アップデート を行う手順を説明します。

管理コンソールから Fine-Grained Update などでサーバークラスターに対するアプリケーションの更新を行った場合、処理が完了すると**[更新のロールアウト]**、マスター構成に**[保存]**、**[アプリケーションの管理]**の 3 つの中から次の処理内容を選択することになります。ロールアウト・アップデート を行うには、ここで**[更新のロールアウト]**をクリックするか、**[アプリケーションの管理]**をクリックしてから次の画面で**[更新のロールアウト]**ボタンをクリックします。この時点から、各ノードへの自動反映が順番に始まります。

重要:この画面でマスター構成に**[保存]**を押さないで下さい。ロールアウト・アップデートの実行時変更が自動的にマスター構成に保存されます。このリンクをクリックする必要はありません。このリンクをクリックした場合、ロールアウト・アップデートがキャンセルされ、通常のアプリケーション更新と同じ挙動になります。

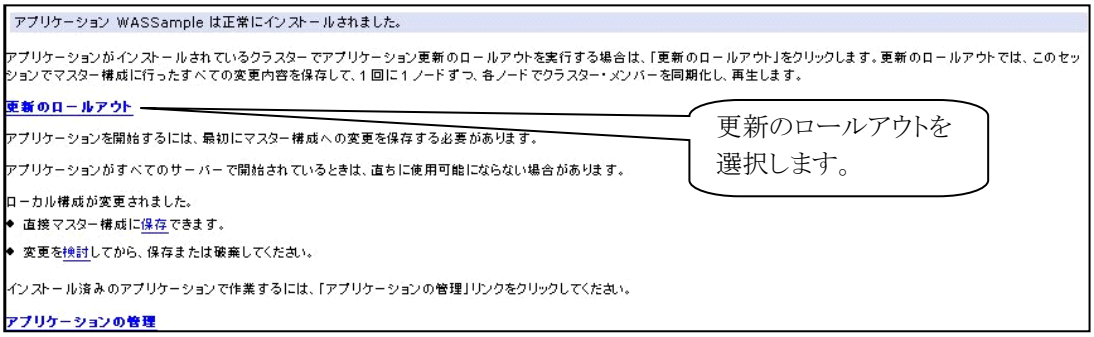


図 5-93 ロールアウト・アップデート(1)

[更新のロールアウト]を選択しますと、ロールアウト・アップデートの進捗状況が表示されます。
このときに表示されるメッセージのコードを確認することで、状況を確認できます。

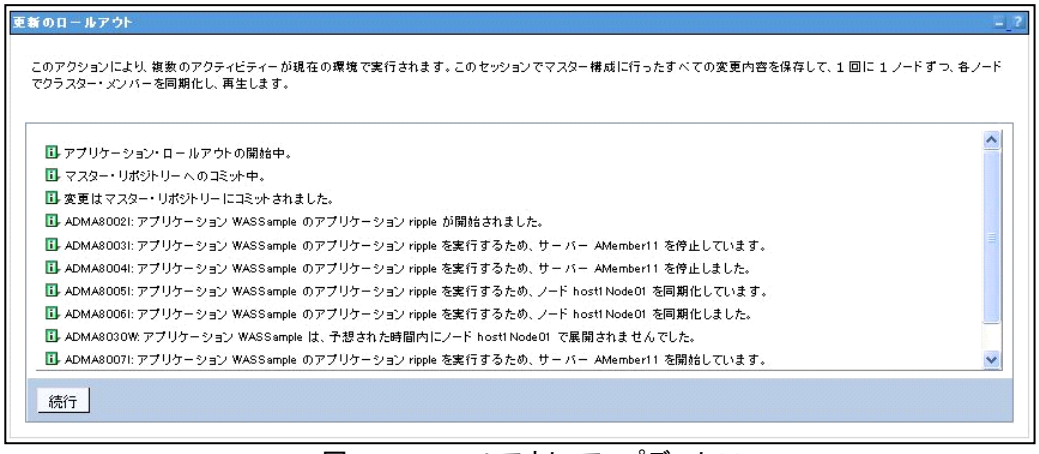


図 5-94 ロールアウト・アップデート(2)

5-3-2-3 ロールアウト・アップデート使用時の注意点

■ ノード・エージェントが必要

ロールアウト・アップデートを行うときには、クラスター・メンバーが稼働しているノードの全ノード・エージェントが起動しているようにして下さい。もし、ノード・エージェントが稼働しないノードがあると、デプロ

イメント・マネージャーはそのノードをとばして、処理を行いますので、ノード・エージェントが稼働していないノードのアプリケーションは更新されないままになってしまいます。

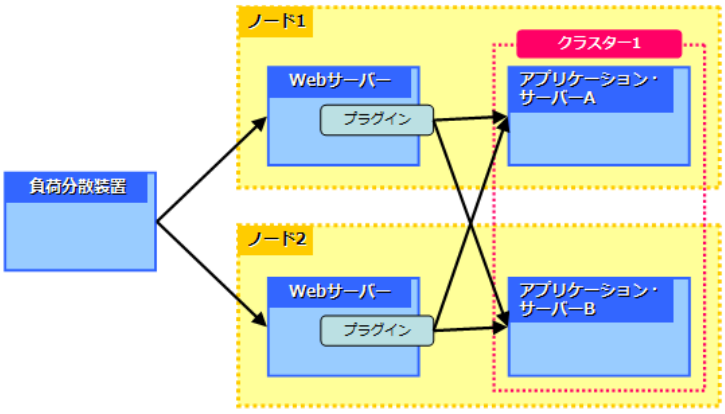


図 5-95 構成例(1)

■ 水平クラスター構成が必要

5-3-2-1のロールアウト・アップデートの処理の流れの節で説明しましたように、サーバーの起動・停止の範囲がノードです。クラスター・メンバーが同一ノード内に配置するような垂直クラスター構成ではロールアウト・アップデートを実施すると、クラスター全体が停止され、サービス不可になります。そのため図 5-95 構成例(1)のようにクラスター・メンバーが複数のノードに配置されている水平クラスター構成の場合にのみ適用できます。この際、図 5-96 構成例(2)のように、1 物理ノードに、複数のノード（複数のApplication Server/カスタム・プロファイル）を作成することで、1 物理ノードでもノード毎にロールアウト・アップデートを行うことが可能です。

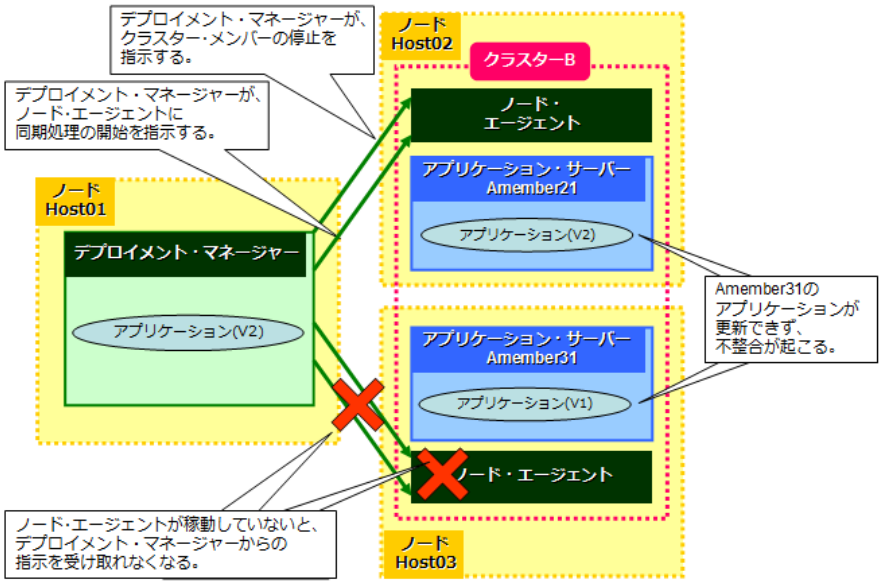


図 5-96 構成例(2)

■ プラグインのリトライ間隔との関係

図 5-97 サーバーの起動・停止タイミングとプラグインのリトライ間隔のように水平クラスター構成でロールアウト・アップデートを実施した場合、クラスター・メンバーであるサーバーA とサーバーB が順次停止、再起動されます。IHS のプラグインがクラスター・メンバーの停止を検知し、使用不可と判断します。後続のリクエストを他の稼動中のクラスター・メンバーに転送します。プラグインがあるクラスター・メンバ

ーを使用不可と判断した場合、設定されているリトライ間隔でそのクラスターメンバーサーバーに再度リクエストを投げます。デフォルトではこのリトライ間隔が 60 秒になっています。

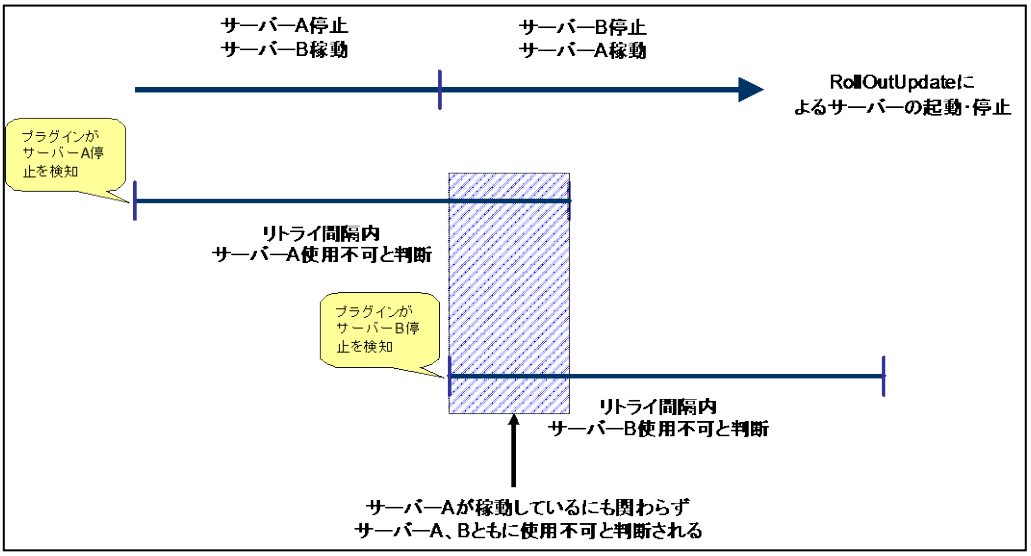


図 5-97 サーバーの起動・停止タイミングとプラグインのリトライ間隔

クラスター・メンバーの停止から再起動までの時間がこの間隔より長い場合、稼動しているクラスター・メンバーがあるにも関わらず、プラグインが全てのクラスター・メンバーが使用不可と判断し、ユーザーにエラーを返す可能性があります。クラスター・メンバーが二つだけの場合、特に注意する必要があります。リトライ間隔の設定値を調整することで、このような問題を回避できます。



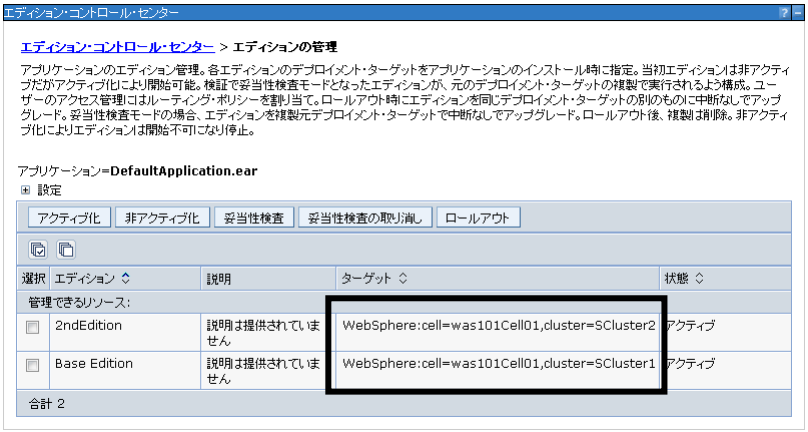
図 5-98 プラグインの再試行間隔設定

5-3-3 複数エディションの同時稼働

5-1-1-2で紹介しました、同一アプリケーションの複数エディションの同時稼働を行うための手順を説明します。

以下ではセル内に ODR が構成され使用可能であることを前提とします。

まず、前述のように、複数エディションの同時稼働では、各エディションのデプロイ先ターゲットが異なる必要があります。2 つ目(以降)のエディションをインストールする際に別のターゲットを指定するか、既に 2 つ目のエディションを同じターゲットにマップ済みの場合は別のターゲットに変更しておきます。



上の図のように、アプリケーション(ここでは DefaultApplication.ear を使用しています)各エディションが異なるターゲットにマップされるようにします。もし、いずれかのエディションが非アクティブになっている場合はそのエディションを選択して「アクティブ化」ボタンを押しておきます。

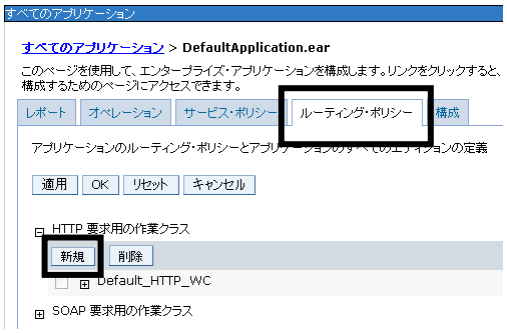
次に、どのリクエストをどのエディションにルーティングさせるかを、ルーティング・ポリシーを用いて設定します。例として、テスト・ユーザーの PC の IP アドレスが 192.168.5.1 であり、一般ユーザーの PC はそれ以外である場合を考えます。テスト・ユーザーは 2ndEdition へ、一般ユーザーは Base Edition へそれぞれルーティングすることにします。

まず、管理コンソールでアプリケーション→すべてのアプリケーションと展開し、ルーティング・ポリシーを設定するアプリケーションをクリックします。



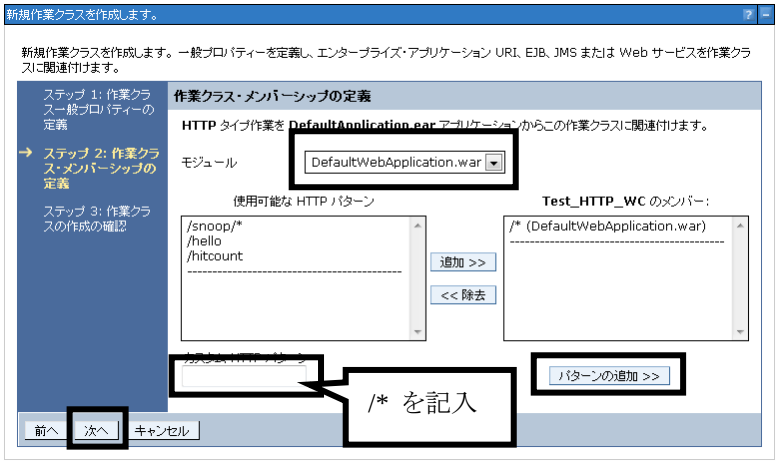
アプリケーションの管理画面で「ルーティング・ポリシー」タブをクリックします。

「HTTP 要求用の作業クラス」を展開し、「新規」ボタンを押します。



ステップ 1: 作業クラス一般プロパティーの定義で、名前を Test_HTTP_WC として「次へ」をクリックします。

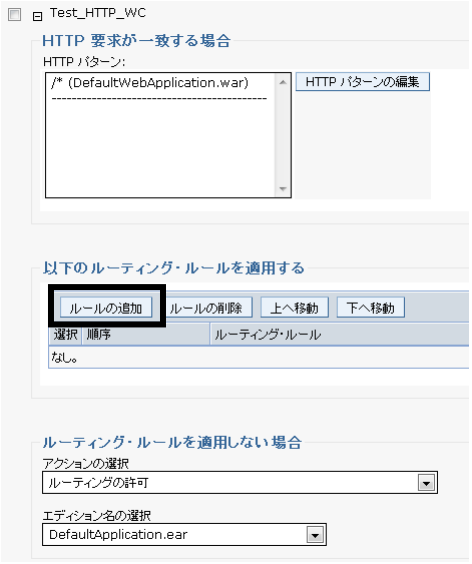
ステップ 2: 作業クラス・メンバーシップの定義で、DefaultWebApplication.war を選択し、カスタム HTTP パターンに/*を入力して「パターンの追加」ボタンを押します。



Test_HTTP_WC のメンバーに /* (DefaultWebApplication.war) が追加されたら、「次へ」ボタンを押します。

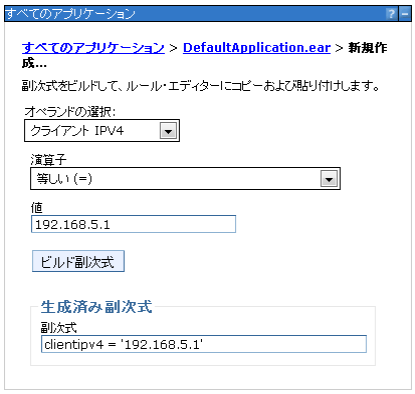
ステップ 3: 作業クラスの作成の確認では「終了」ボタンを押します。以上で作業クラス Test_HTTP_WC が作成されました。

次に、作業クラスに対してルールを設定します。Test_HTTP_WC の設定画面が開きますので、「ルールの追加」ボタンを押します。

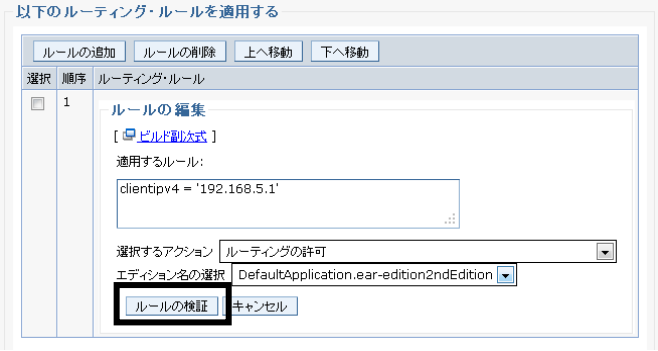


「ルールの編集」セクション内の「ビルド副次式」リンクをクリックすると、副次式ビルダーが起動します。

「オペランドの選択」プルダウン・メニューから「クライアント IPV4」、「演算子」から「等しい(=)」を選択し、「値」に 192.168.5.1 を入力して「ビルド生成式」ボタンを押します。



「生成済み副次式」の欄にルール式が生成されますので、この式をコピーして「適用するルール」欄にペーストします。「選択するアクション」に「ルーティングの許可」、「エディション名の選択」に DefaultApplication.ear-edition2ndEdition を選択して「ルールの検証」ボタンを押します。



「妥当性検査が正常に実行されました。…」のメッセージが表示されれば、ルールの設定は成功です。
その下の「ルーティング・ルールを適用しない場合」では、「アクションの選択」に「ルーティングの許可」、「エディション名の選択」に DefaultApplication.ear を選択します。これで、クライアント IP アドレスが 192.168.5.1 の場合は 2ndEdition へ、それ以外の場合は Base Edition へルーティングされることになります。画面上方の OK ボタンを押し、変更を保存します。
これでルーティング・ルールの設定は完了です。ターゲットのクラスターが起動して双方のエディションが開始されれば、以降のリクエストに対してルーティング・ルールが適用されます。

5-3-4 エディションのロールアウトと妥当性検査

5-1-1-3で紹介しました、エディションのロールアウトと妥当性検査の操作方法について説明します。
手順の説明の前に、エディションのロールアウトの計画について解説します。

5-3-4-1 エディションのロールアウトの計画

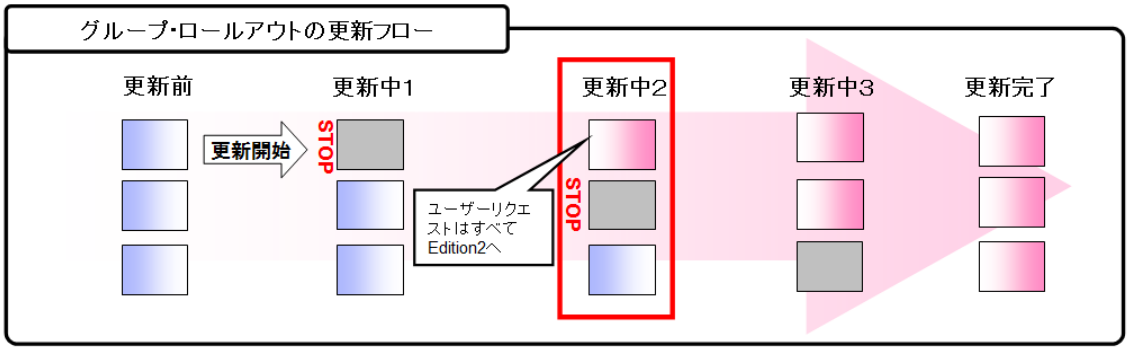
エディションのロールアウトを実行するに当たって、決めておかなくてはならないことが 3 つあります。
以下に、それらの計画・設定について説明します。

ロールアウト計画

ロールアウト計画には、「グループ化」と「アトミック」の 2 つがあります。

グループ・ロールアウト

グループ・ロールアウトでは、クラスター・メンバーを指定した「グループ・サイズ」数ずつのグループにグループ化し、グループごとにアプリケーションのロールアウトを順次実行します。グループごとのクラスター・メンバー数はできるだけ均等になるようにグループ化されます。以下にその概念を図示します。



上の図はメンバー数 3 のクラスターに対し、グループ・サイズを 1 に設定した場合のグループ・ロールアウトの更新フローを示しています。

「更新前」の状態でグループ・ロールアウトが開始されると、その後のドレーン間隔(後述)の時間経過中は、1 番目のグループ内のクラスター・メンバーに対して有効なセッション・アフィニティーを持つリクエストのみが引き続き同じクラスター・メンバーに転送され、1 番目のグループ内のクラスター・メンバーに対して有効なセッション・アフィニティーを持たないリクエストは他のグループに転送されます。

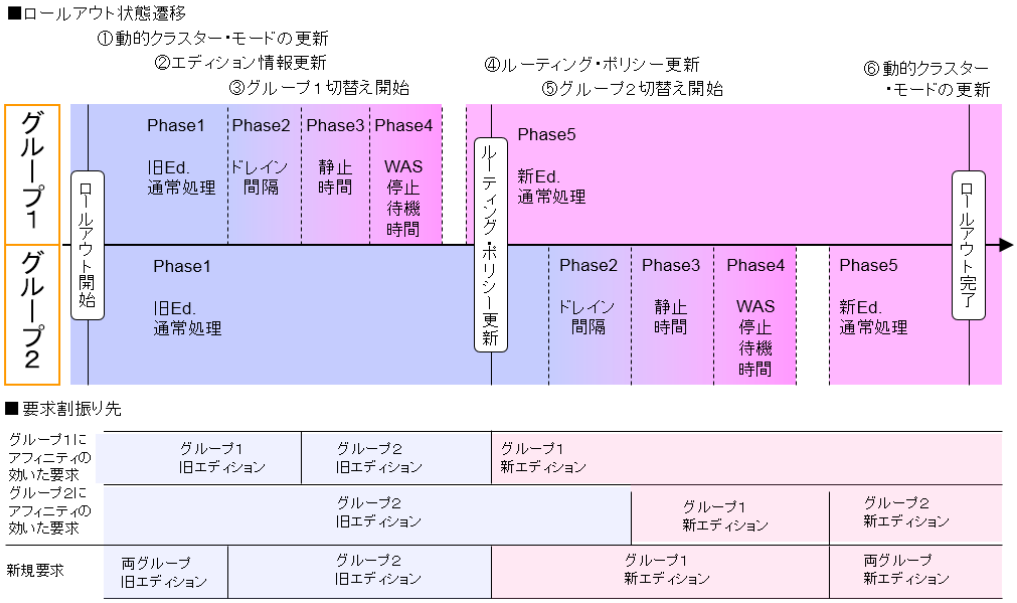
ドレーン間隔の時間が経過すると、1 番目のグループに対する ODR からのリクエスト転送が静止されて「更新中 1」の状態となります。これ以降、1 番目のグループ内のクラスター・メンバーに対して有効なセッション・アフィニティーを持つリクエストは、アフィニティーが中断されて他のグループのクラスター・メンバーで処理されることになります。従って、アプリケーションのサービスに影響を与えずにアプリケーションのロールアウトを利用するには、データベースやメモリー間複製などを使用したセッション・クラスタリング構成が必須になります。

リクエスト転送が静止された後、1 番目のグループでまだ仕掛かり中の処理があれば最大 60 秒間待ってから、(リセット計画(後述)がソフト・リセットの場合)現行エディションのアプリケーションが停止して非アクティブ化されます。続いて新エディションがアクティブ化して起動され、ODR からのリクエスト転送が再開されて「更新中 2」の状態となります。このタイミングで内部的にルーティング・ポリシーが更新され、すべてのリクエストが新エディションへルーティングされるようになります。

以後、残るグループで順次同様の処理が実行され、最終的にすべてのグループ(クラスター・メンバー)でエディションのロールアウトが完了します。

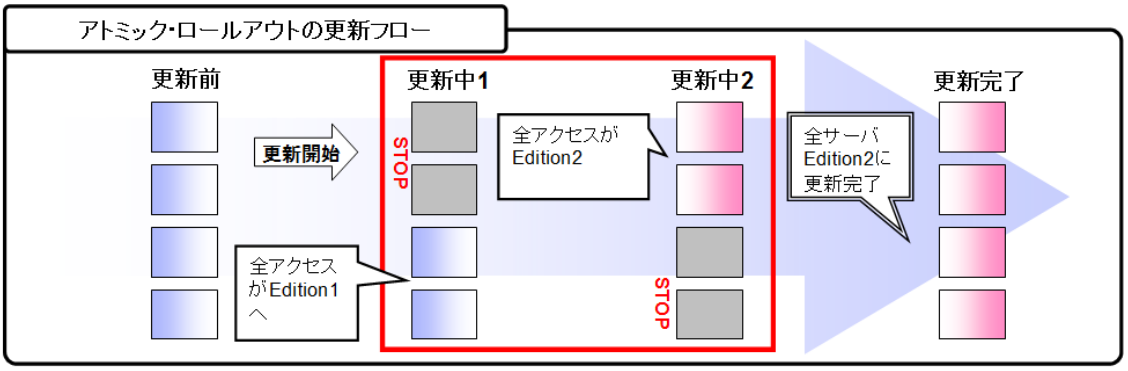
以上の動作を詳しく示したものが下の図です。

【参考】動作詳細 – グループ・ロールアウト



アトミック・ロールアウト

アトミック・ロールアウトは、グループ数が常に 2 であるグループ・ロールアウトに似ています。概念図を下に示します。



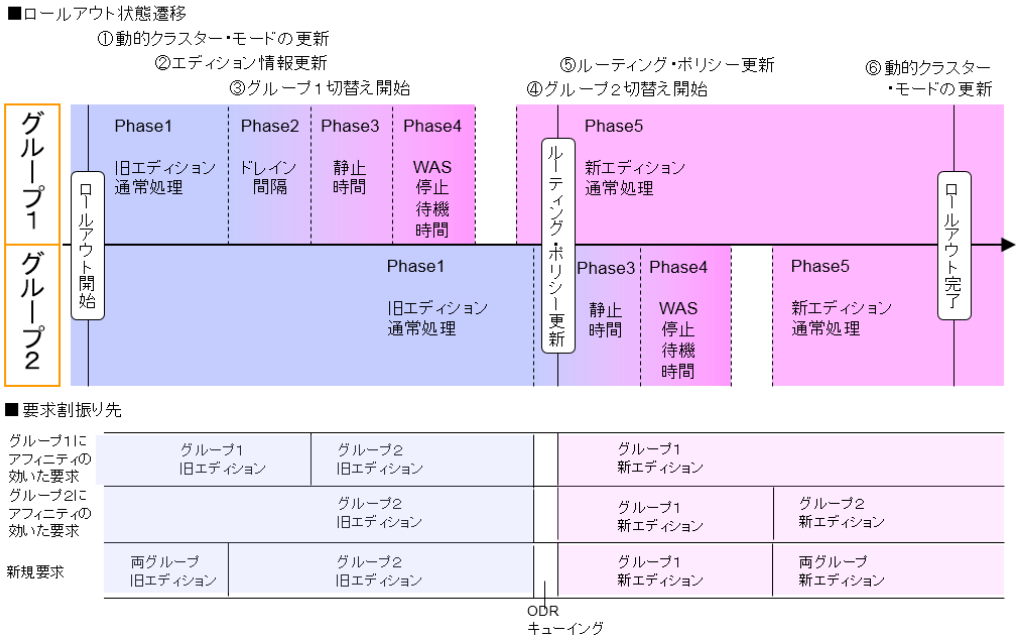
アトミック・ロールアウトを開始してから「更新中 1」の状態までの挙動は、グループ・ロールアウトと同様です。続いて、全アクセスが 2 番目のグループの現エディションに転送されている「更新中 1」の状態で 2 番目のグループのロールアウトが開始されます。その後全アクセスが 1 番目のグループの新エディションへ転送されるように、内部的にルーティング・ポリシーが更新されて「更新中 2」の状態になります。

2 番目のグループのロールアウトが開始されてから、全アクセスが 1 番目のグループの新エディションへ転送されるようになるまでの間、リクエストの転送先がない期間が発生します。この期間に ODR に到着したリクエストは、ODR 内でキューイングされることになります。

2 番目のグループのロールアウトが完了すれば、すべてのクラスター・メンバーが新エディションに切り替わります。

以上の動作を詳しく示したものが下の図です。

【参考】動作詳細 – アトミック・ロールアウト



グループ・ロールアウトではアトミック・ロールアウトと異なり、2 つ目のグループにドレーン間隔がありません。これは、前述の ODR でのキューイングが必要な時間を短縮し、キューがあふれることを防ぐためのしくみです。

リセット計画

リセット計画には、ロールアウト中にアプリケーションのエディションを切り替える際に、現エディションの停止と新エディションの開始のみを行うか、エディションの停止・開始の間にアプリケーション・サーバーの再起動も併せて行うかの 2 種類があります。前者を「ソフト・リセット」、後者が「ハード・リセット」です。
所要時間が短いため、通常はデフォルトのソフト・リセットを使用します。
ただし、ソフト・リセットはアプリケーションのエディションの停止・起動のみを行うため、JNI 経由で使用しているネイティブ・ライブラリーや、アプリケーション・サーバーに関連づけられた共用ライブラリーの更新などは反映されません。このような場合にはハード・リセットを使用する必要があります。

ドレーン間隔

ドレーン間隔は、ロールアウト中グループのクラスター・メンバーに対するアフィニティーを持ったリクエストが到着しなくなるのを待機する時間です。デフォルトは 30 秒です。
上述のようにドレーン間隔の経過中は、ロールアウト中グループへのリクエスト転送を有効なアフィニティーを持つものだけに制限し、それ以外のリクエストをロールアウト中以外のグループへ転送するようになります。

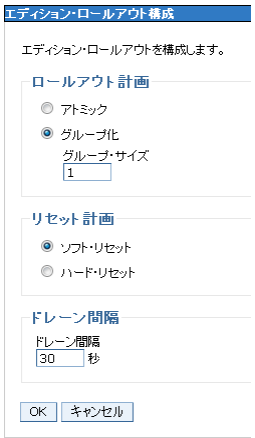
5-3-4-2 エディションのロールアウトと妥当性検査の手順

前節まで、エディションのロールアウトのために必要な設定を説明してきました。
この節ではエディションのロールアウトの手順を説明します。

エディションのロールアウトを実施するには下の図のように、現エディションと新エディションが同一のターゲットにマップされていて、現エディションがアクティブ、新エディションが非アクティブである状態が条件となります。



エディション・コントロール・センターでロールアウト先の新エディションを選択し、「ロールアウト」ボタンを押します。
右の図のように「エディション・ロールアウト構成」画面が表示されます。
ここで、前節で説明しましたロールアウト計画、リセット計画とドレーン間隔を設定し、OK ボタンを押すとロールアウトが開始されます。





ロールアウトの進行状況が逐次表示されます。「ロールアウトが正常に完了しました」のメッセージが表示されれば、新エディションへのロールアウトは完了です。
「エディションの管理」リンクをクリックしてエディション・コントロール・センターに戻ると、下の図のように新エディションがアクティブ、旧エディションが非アクティブとなっていることが確認できます。



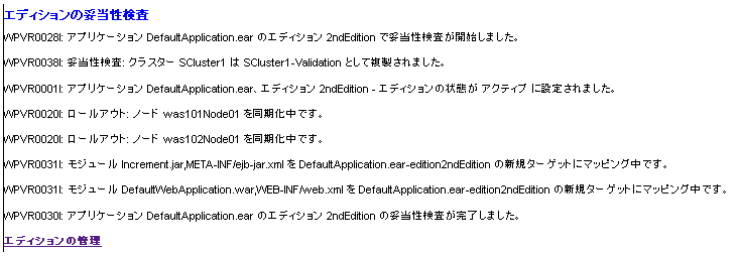
また、この状態から旧エディションを選択して同様にロールアウト操作を行うことで、新エディションのリリース後に万一不具合が見つかった場合に、旧エディションへの迅速な戻し処理が可能です。

5-3-4-3 妥当性検査の手順

妥当性検査では、新エディションの動作確認を行うための「妥当性検査クラスター」が自動的に作成され、新エディションがデプロイ、起動されます。
この状態は5-3-3節で説明しました「複数エディションの同時稼働」と同等となります。従って、妥当性検査の実行に先立って、5-3-3節と同様の手順でルーティング・ルールを作成しておきます。
以下では5-3-4-2ルーティング・ポリシーの作成と同様に、現エディションがアクティブ、新エディションが非アクティブで、ルーティング・ルールの作成が完了していることを前提とします。

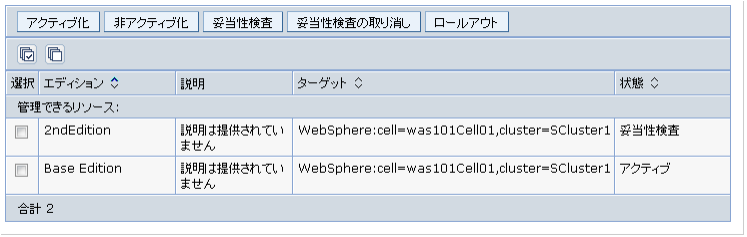


上の図のように新エディションを選択して「妥当性検査」ボタンを押します。



妥当性検査の準備状況が逐次表示されます。「妥当性検査が完了しました」のメッセージが表示されれば、妥当性検査が可能な状態になっています。

「エディションの管理」リンクをクリックしてエディション・コントロール・センターに戻ると、新エディションが妥当性検査モードになっていることが確認できます。



ここで、妥当性検査クラスターの状態を確認します。今回の例でターゲットは(動的クラスターではなく)WebSphere Application Server クラスターですので、管理コンソールで サーバー→クラスター→WebSphere Application Server クラスター の順に展開します。



もともと存在したターゲットの SCluster1(以下ではオリジナル・クラスターと呼びます)に加えて、妥当性検査クラスターの SCluster1-Validation クラスターが自動的に作成されています。WebSphere Application Server クラスターでは、作成された妥当性検査クラスターは停止されています(動的クラスターの場合は作成後に自動的に起動されます)。

SCluster1-Validationリンクをクリックして、妥当性検査クラスターの詳細を確認してみます。オリジナル・クラスターと同じ構成になっているはずです。クラスター・メンバーの構成も、オリジナル・クラスターのメンバーの構成と同じになっています。ただし、オリジナル・クラスターのメンバー数に関わらず、妥当性検査クラスターのメンバー数は 2 になります。妥当性検査クラスターの生成後にメンバーを追加することは可能です。

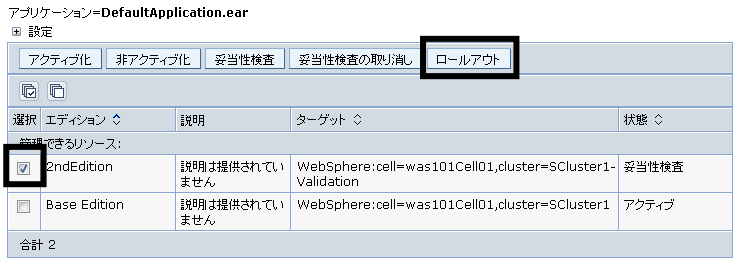
妥当性検査を始める前にルーティング・ルールが適切に設定されているかを確認し、されていなければ設定しておきます。下の図に、テスト・ユーザーが IP アドレス 192.168.5.1 の端末から DefaultApplication.ear-edition2ndEdition へアクセスして妥当性検査を行い、一般ユーザーは他の IP アドレスの端末から現行エディションの Base Edition へアクセスするルーティング・ルールの設定を示します。



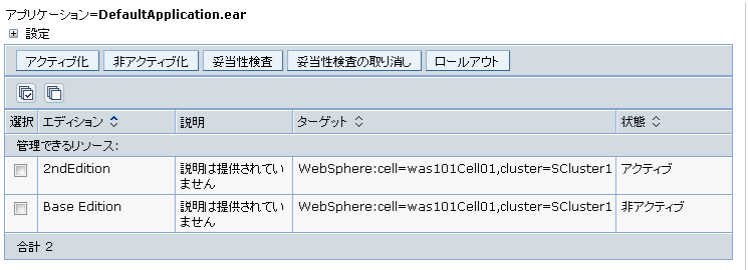
ルーティング・ルールの確認が済んだら、妥当性検査クラスターを起動して妥当性検査を実施します。妥当性検査の結果 2ndEdition に問題がなければ、オリジナル・クラスターで実際に 2ndEdition へロールアウトすることにします。問題が見つかった場合には、妥当性検査を取り消して、元の状態に戻ることができます。妥当性検査からのロールアウト、妥当性検査の取り消しのいずれを行う場合でも、あらかじめテスト・ユーザー用のルーティング・ルールを削除(または可能な場合は作業クラスごと削除)し、すべてのリクエストが適切なエディションにルーティングされるようにしておきます。

妥当性検査からのロールアウトを行う場合

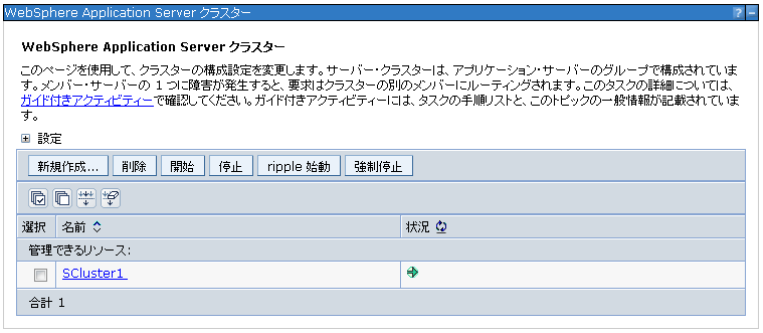
エディション・コントロール・センターで 2ndEdition を選択して、「ロールアウト」ボタンを押します。



エディション・ロールアウト構成画面に遷移しますので、ロールアウト計画とリセット計画、ドレーン間隔を設定して OK ボタンを押します。ロールアウトの進行状況が逐次表示され、ロールアウトが完了します。妥当性検査からのロールアウトが完了すると、2ndEdition がアクティブになります。



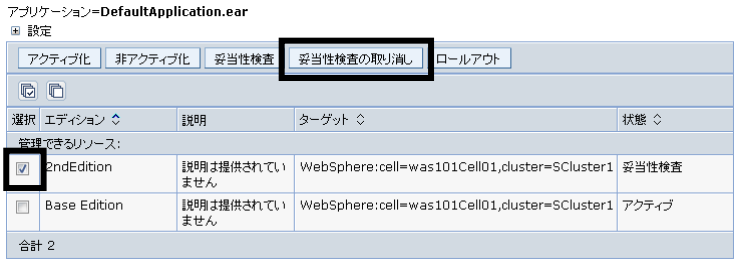
また、下の図のように、妥当性検査クラスターは自動的に削除されます。



SCluster1-Validation クラスターがなくなっていることがわかります。

妥当性検査を取り消す場合

妥当性検査を取り消す場合は、妥当性検査モードになっている 2ndEdition を選択して、「妥当性検査の取り消し」ボタンを押します。



妥当性検査の取り消しが完了すると、2ndEdition が非アクティブに戻り、妥当性検査クラスターが削除されます。

5-3-4-4 妥当性検査の考慮点

妥当性検査を使用する場合の考慮点が 2 つあります。

1) 妥当性検査クラスターの分のリソース
   妥当性検査クラスターはオリジナル・クラスターと同じノードに作成されるため、妥当性検査クラスターが必要とする CPU とメモリーを考慮してキャパシティ・プランを行う必要があります。
2) 更新系のテスト
   妥当性検査クラスターの構成はオリジナル・クラスターを引き継ぎます。
   リソース構成も引き継がれますので、データベースなど外部リソースに対して書き込みや更新がかかるテストを実施すると、本番システムのデータが変更される場合があります。
   このようなテストは妥当性検査では実施しないようにするか、妥当性検査クラスターが作成された後にデータ・ソースなどリソースの設定を変更して、テスト用データベースを使用するようにします。

5-3-4-5 ロールアウト・アップデートとエディションのロールアウト

これまで説明してきましたように、ロールアウト・アップデートとエディションのロールアウトはよく似た機能を持っています。これらの違いをまとめると、以下の表のようになります。

表 5-2 ロールアウト・アップデートとエディションのロールアウト

	ロールアウト・アップデート	エディションのロールアウト
リクエスト転送の制御	Web サーバー・プラグイン	ODR の保守モード
更新の単位	EAR、モジュール、ファイル	EAR
更新の反映方法	アプリケーション・サーバーの停止・起動	アプリケーションまたはアプリケーション・サーバーの停止・起動
リリース時間	長くなりうる	比較的短い
戻し時間	長くなりうる	比較的短い

リクエスト転送の制御は、ロールアウト・アップデートはアプリケーション・サーバーが停止したことを Web サーバー・プラグインが検知して転送を停止します。エディションのロールアウトでは、ロールアウト中のアプリケーション・サーバーに対して内部的に保守モード(第 6 章 1-8-4-4 節にて後述)を適用することで転送を停止します。
完了までの所要時間はノードやクラスターの構成に大きく依存しますが、ロールアウト・アップデートではノード単位に同期化によるアプリケーション配布が発生するのに対し、エディションのロールアウトではあらかじめ配布済みのエディションに対してアクティブ化/非アクティブ化を行うだけですので、この観点ではエディションのロールアウトの方が所要時間は短くなります。

5-4 運用パターン

今までは WAS V7.0 におけるアプリケーション管理の機能を説明しました。この章では実際の運用管理の典型パターンを想定し、これらの機能をどのように組み合わせて使用できるかについて説明します。

5-4-1 アプリケーション導入のケース

システム構築の際、開発環境と本番環境が別々になっていますが、通常同一のリソース(データ・ソース)定義を使用します。Java EE の ear を使用してアプリケーションを導入する場合、本番環境では導入の際再度リソースを定義する必要がありますが、Enhanced EAR を使用すれば必要なリソース定義情報は、Enhanced EAR の中に全て含まれていますので、再定義する作業の必要はなくなります。但しこの場合5-2-1-7 Enhanced EAR」で記述したリソースのスコープの事項に注意する必要があります。

5-4-2 アプリケーション更新のケース

アプリケーション更新の実現方法が、その更新の要件によって異なります。一般的に以下の三つにまとめられます:

■ 使用中のセッション・オブジェクトを救えるかどうか

アプリケーションの更新にはアプリケーション・サーバー再起動の必要はありませんが、更新を反映するにはアプリケーションの再ロードが必要です。再ロードの時アプリケーションは使用不可になります。更新のタイミングでユーザーのリクエストを受け付けてしまい、エラーが発生することを防ぐため、アプリケーション・サーバーを停止させる必要があります。ここで課題になるのが HTTP セッションの救済です。どこまで HTTP セッションを救う必要があるのかということをあらかじめよく考える必要があります。

もし全てのリクエストを救うことが必須の場合は、セッション DB やメモリー間複製といったセッション・パーシスタンスを構成し、書き込み頻度を「End Of Service」に設定して、すべてのリクエスト実行後にセッション情報を DB や別アプリケーション・サーバーへと保存する必要があります。

実際には完全に救済できなくてもよいが、処理中のリクエストについてはできる限り救済したい場合は、前段のコンポーネントで新規クライアントからの処理は別サーバーに割振りつつ、既存クライアントからの処理は継続して割振ります。一定時間このように既存クライアントからの処理を待った上で、どうしてもなくならないセッションについては見切りをつけるという運用が現実的であると考えられます。例えばアプリケーション特性として5分あれば必ず処理が完了する場合、10分程度待てばほとんどの処理中のクライアントからのリクエストは終了すると考えられ、それ以降も残っているセッションは、画面遷移の途中でブラウザを閉じたなどの理由で残存し、セッション・タイムアウトを待っているものである可能性があります。この運用であれば、たとえセッション・パーシスタンスを構成していない場合でも一定量のクライアントを救済することができます。

処理中の HTTP セッションを救済するには、処理を片寄せさせることで実現できます。例えば図 5-99 更新時の構成例(1)で示している構成で処理をクラスター1 のクラスター・メンバーであるサーバーB へ片寄せさせるには:

1. プラグインからサーバーA への新規リクエストの振り分けを停止するには、サーバーA のランライムの Weight を「0」に変更します。これはサーバーの再起動なしで反映できます
2. Tivoli Performance Viewer で活動中のセッション数を確認し、活動中のセッション数がゼロになった時点で、サーバーA を停止します
3. アプリケーションの更新を行い、サーバーA を再起動します
4. サーバーB に対してもステップ 1～3 を適用します

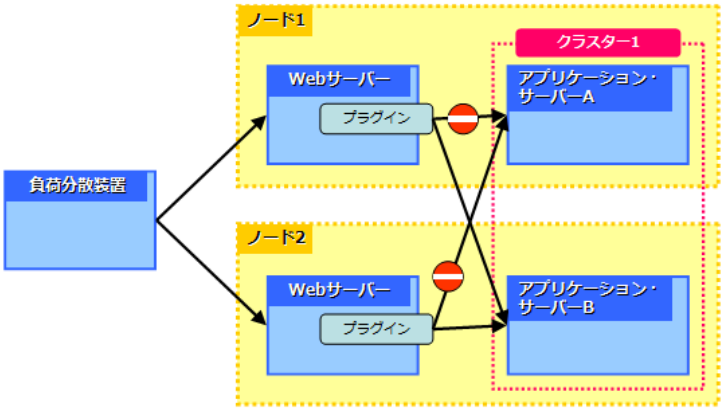


図 5-99 更新時の構成例(1)

ODR を構成して IMP の機能であるエディションのロールアウトを使用すれば、上の手順に相当する動作を自動的に実行させることが可能です。ただし、2. のように活動中のセッション数が 0 になることを確認してからアプリケーションの停止を行うことはできず、あらかじめ設定したドレーン間隔の時間だけ待って、その後に到着する有効なアフィニティーを持つリクエストに対してはセッション維持をあきらめる(見切りをつける)動作となります。

もし HTTP セッションの救済が必要ないのであれば、アプリケーション更新時、サーバーの停止と再起動だけで良いです。この場合前述のロールアウト・アップデート機能が便利です。

なお Tivoli Performance Viewer で活動中のセッション数を確認するには、管理コンソールで**[モニターおよびチューニング]→[Performance Viewer]→[現行アクティビティー]**を選択し、モニターを開始させた後、モニターするサーバー名をクリックして、**[パフォーマンス・モジュール]→[サーブレット・セッション・マネージャー]**を選択して下さい。

クラスター・メンバーのウェイトを変更するには、下記の様に設定をする必要があります。

●Web アプリケーションが稼働するクラスター・メンバーのウェイト変更

管理コンソールで**[サーバー]→[クラスター] →[WebSphere Application Server クラスター]**を選択し、クラスター名を選択します。**[追加プロパティー]**にある**[クラスター・メンバー]**をクリックします。対象となるクラスター・メンバーの**[構成済みウェイト]**欄に新しい値を入力し、**[更新]**ボタンをクリック後、構成を

保存し、Web サーバー・プラグイン構成ファイルを再生成して下さい。プラグイン・プロパティで**[構成最新表示間隔]**で設定されるプラグイン構成ファイルの再読込が行われた後、新規ウェイトが適用されます。

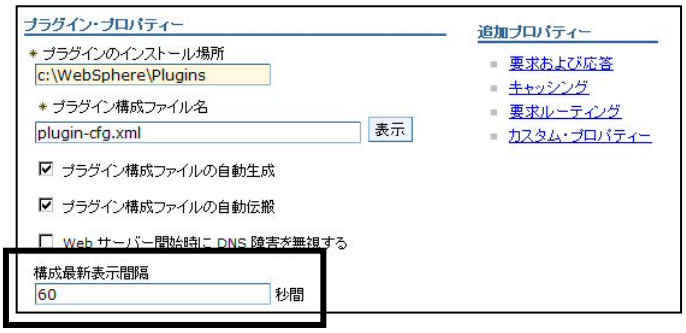


図 5-100 プラグインの「構成最新表示間隔」

●EJB アプリケーションが稼働するクラスター・メンバーのウェイト変更

管理コンソールで**[サーバー]→[クラスター] →[WebSphere Application Server クラスター]**を選択し、クラスター名を選択します。**[追加プロパティー]**にある**[クラスター・メンバー]**をクリックします。対象となるクラスター・メンバーの**[ランタイム・ウェイト]欄**に新しい値を入力し、**[更新]**ボタンをクリックします。この時点で、新規ウェイトが適用されます。

■ 新旧バージョン混在があるかどうか

Web システムによっては、24 時間×365 日で無停止運転する要件があります。このような場合アプリケーションの更新中もサービス継続する必要があるため、システムに新旧二つのバージョンのアプリケーションが混在することになります。バージョン混在時の更新方法は、アプリケーションの変更範囲によって、一般的に大きく二つに分類できます。1 つはアプリケーションロジック自体の変更だけのケースです。日常運用で見られるバグ修正などがこれに該当します。もう 1 つが外部リソースに関わる変更です。例えばデータベースに保存されているセッション・オブジェクトのデータ構造が更新されるケースです。以下それぞれのケースについて検討します。

1. アプリケーションロジックのみ変更のケース
   この場合サービスを継続するため、一般的にクラスターを構成します。クラスター・メンバー1 でのアプリケーション更新がクラスター・メンバー2 に影響しないため、ロールアウト・アップデートなどでクラスター・メンバー1、2 を順次更新します。HTTP セッションの救済が必要な場合、上記の節を参照して下さい。
2. 外部リソースに関わる変更のケース
   ここでセッション・オブジェクトのデータ構造が変更された例を取り上げて検討します。図 5-101 更新時の構成例(2)のようにクラスター・メンバーであるサーバーA とサーバーB との間データベースでセッション・パーシスタンス構成になっています。サーバーA を停止してセッション・オブジェクトの更新を行うと、サーバーB で継続している処理に影響を与えることがあります。例えばセッション・オブジェクトのクラスが変わった場合、サーバーB 上で稼動する旧いバージョンのアプリケーションに「ClassCastException」が発生する可能性があります。

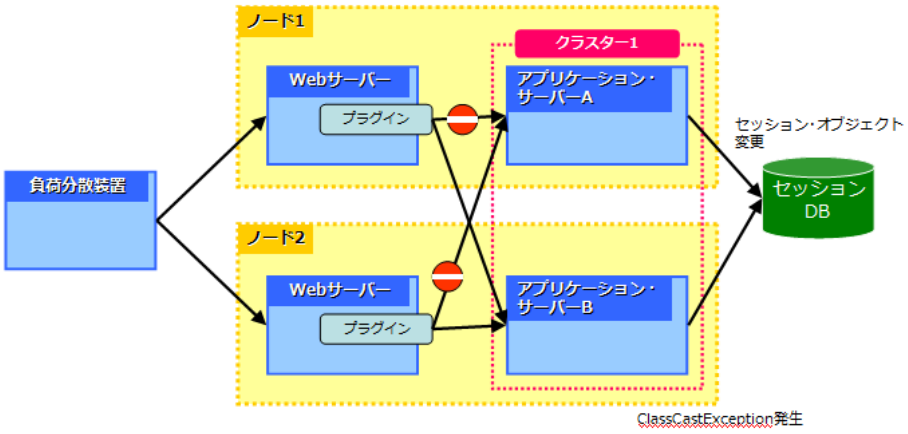


図 5-101 更新時の構成例(2)

このようなことを防ぐために、セッション・パーシスタンスの範囲を分離する必要があります。セッション・パーシスタンスの範囲がクラスター内になっていますので、複数のクラスターまたはセルに分割して、クラスターまたはセル単位でアプリケーションの更新を行えば、新旧バージョンが混在してもアプリケーションの実行に影響しません。図 5-102 更新時の構成例(3)は複数セルに分割して構成する例を示しています。Cell1 と Cell2 との間にセッション・パーシスタンスはありせんので、Cell1 でセッション・オブジェクトの構造に関わる更新が行われる時でも、Cell2 上の処理に影響を与えません。但し複数セルのような構成では、シングルセル構成と比べて構成が複雑になり、管理負荷も大きくなる可能性があるなどの考慮点があります。

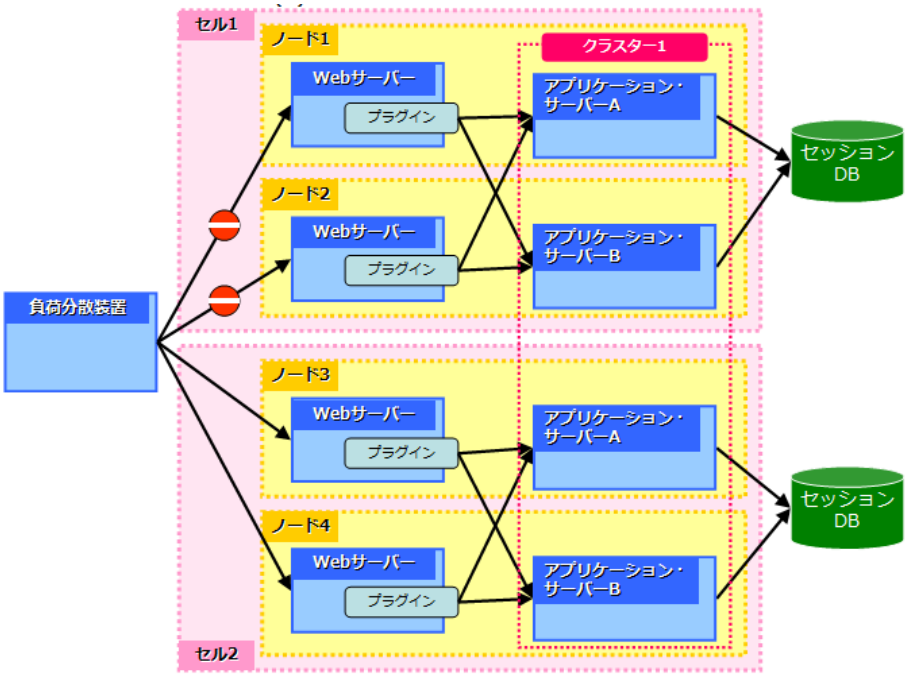


図 5-102 更新時の構成例(3)

アプリケーション更新処理のケースを表のようにまとめました。

表 5-3 アプリケーション更新処理ケース

	新旧バージョン混在なし	新旧バージョン混在あり(アプリケーションロジックのみ変更)	新旧バージョン混在あり(外部リソースも変更)
HTTP セッションを救済しない	ロールアウト・アップデート、エディションのロールアウトなど	ロールアウト・アップデートなど	N/A
HTTP セッションを救済	セッションの停止待ちが必要 またはセッション・パーシスタンスを構成	セッションの停止待ち及び処理の片寄せが必要 またはセッション・パーシスタンスを構成	セッションの停止待ち及び処理の片寄せが必要、複数セルまたはクラスターの構成が必要

アプリケーションの更新は、システム運用管理上欠かせない業務の１つです。サービス無停止運転の必要性の有無、更新に関わる範囲などにより、更新の手順だけではなく、システム全体の構成にも影響しますので、システム構築の際、アプリケーション更新の方針をあらかじめ十分に検討する必要があります。

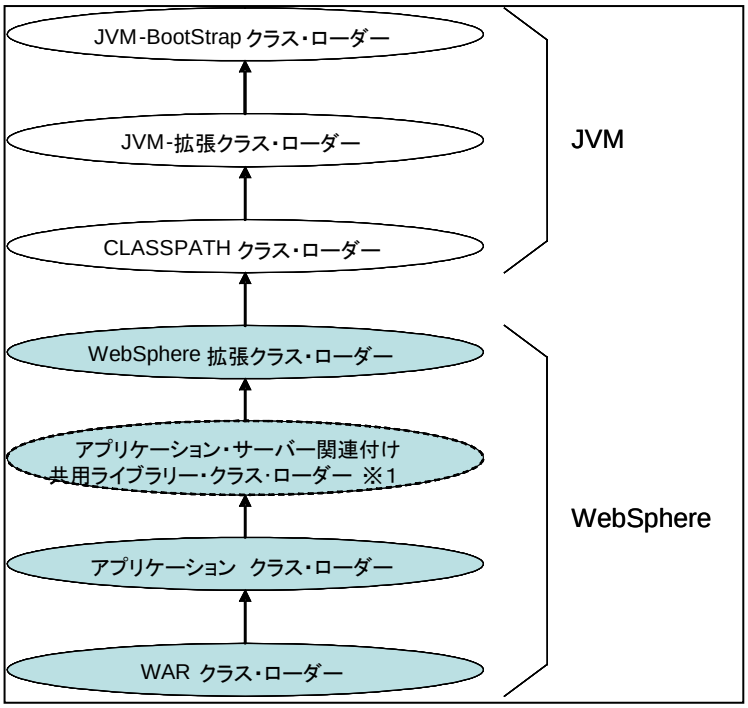
5-5 クラス・ローダー

クラス・ローダーはクラスを検索し、クラスをロードするための仕組みです。WAS では階層構造をとる複数のクラス・ローダーが用いられていますが、このクラス・ローダーの階層構造がモジュール間の可視性、クラスの可視性の問題に直結します。このため、クラス・ローダーは、アプリケーションのパッケージ化と、パッケージ化されてアプリケーション・サーバーにデプロイされたアプリケーションの実行時の振る舞いに影響を与えます。

この節では WAS V7.0 以降のクラス・ローダーの仕組みと設定方法を説明します。さらに Java EE アプリケーションのパッケージングに関する基本事項を説明します。

5-5-1 クラス・ローダーの種類と階層構造

WAS V8.5 のクラス・ローダーは以下のような階層構造をもっています。



※1　アプリケーション・サーバー関連付け共用ライブラリー・クラス・ローダーは、アプリケーション・サーバーと関連付けられた共用ライブラリーを設定したときのみ呼ばれます。

図 5-103 クラス・ローダーの階層構造

クラス・ローダーで最も重要なことは、クラス・ローダーが親子関係、つまり階層構造をとるということです。各クラス・ローダーはその上位のクラス・ローダーの子供です。子のクラス・ローダーがロードしたクラスからは、親またはその祖先のクラス・ローダーがロードしたクラスは見ることができます。一方、親のクラス・ローダーのロードしたクラスからは、子またはその子孫のクラス・ローダーがロードしたクラスを見ることはできません。

以下、各クラス・ローダーについて説明します。

JVM が提供するクラス・ローダーは以下の 3 つです。

JVM-Bootstrap クラス・ローダー

クラス・ローダー階層のトップに位置づけられるクラス・ローダーです。ネイティブ・コードで実装されており、Java 2 SDK 1.6 (または 1.7) runtime の核となるクラス（通常は jre/lib 下のクラス）をロードします。

Extensions クラス・ローダー

JVM-Bootstrap クラス・ローダーが親クラス・ローダーです。Java システム・プロパティーの java.ext.dirs に設定されたディレクトリー（通常は jre/lib/ext）下の JVM 拡張クラスをロードします。実装クラスは sun.misc.Launcher$ExtClassLoader です。

CLASSPATH クラス・ローダー

Extensions クラス・ローダーが親クラス・ローダーです。CLASSPATH 環境変数または-classpath オプションで設定されたディレクトリー下のクラス・ファイルをロードします。実装クラスは sun.misc.Launcher$AppClassLoader です。

CLASSPATH クラス・ローダーには、WebSphere Application Server 製品の Java EE API が含まれています (j2ee.jar 内)。 Java EE API がこのクラス・ローダー内にあるため、 Java EE API に依存するライブラリーをクラスパス・システム・プロパティーに追加して、サーバーのクラスパスを拡張することもできますが、これはクラスのメンテナンスの観点から問題がありますので、サーバーのクラスパスを拡張する方法としては、共用ライブラリーを追加する方が適切です。

以下が、WebSphere Application Server によって提供されるクラス・ローダーです。

WebSphere 拡張クラス・ローダー

JVM の CLASSPATH クラス・ローダーが親クラス・ローダーです。このクラス・ローダーが WebSphere ランタイムのクラスと実行時に必要な Java EE のクラスをロードします。拡張クラス・ローダーは ws.ext.dirs システム・プロパティーを使用して、クラスをロードするのに使用するパスを決定します。 ws.ext.dirs クラスパスの各ディレクトリー、およびこれらのディレクトリー内のすべての JAR ファイルまたは ZIP ファイルが、このクラス・ローダーの使用するクラスパスに追加されます。実装クラスは com.ibm.ws.bootstrap.ExtClassLoader です。

WebSphere 拡張クラス・ローダーは、サーバーにインストールされているアプリケーション・モジュールがリソース・プロバイダーに関連付けられたリソースを参照する場合、およびそのプロバイダーがリソース・ドライバーのディレクトリー名を指定した場合は、リソース・プロバイダーのクラスをサーバーにロードします。

アプリケーション・サーバー関連付け共用ライブラリー・クラス・ローダー

このクラス・ローダーは通常は呼ばれず、アプリケーション・サーバー関連付け共用ライブラリーを設定した場合にのみ呼ばれます。共用ライブラリーの設定で設定したクラスパスを用いて共用ライブラリーのクラスをロードします。

アプリケーション・クラス・ローダー

通常このクラス・ローダーの親は WebSphere 拡張クラス・ローダーです。アプリケーション・サーバー関連付け共用ライブラリーを設定した場合にのみ、アプリケーション・サーバー関連付け共用ライブラリー・クラス・ローダーが親クラス・ローダーになります。このクラス・ローダーがエンタープライズ・アプリケーションの構成要素、つまり EJB モジュール、リソース・アダプター、および依存 JAR ファイルをロードします。WAR クラス・ローダー・ポリシーを「アプリケーション」に設定した場合は、Web モジュールもこのクラス・ローダーによりロードされます。また、アプリケーション関連付け共用ライブラリーを設定した場合は、共用ライブラリーの設定で設定したクラスパスを用いて、共用ライブラリーのクラスもこのクラス・ローダーによりロードされます。

Web モジュール・クラス・ローダー

アプリケーション・クラス・ローダーがこのクラス・ローダーの親です。WAR クラス・ローダーは<WAR_Root>¥WEB-INF¥classes と<WAR_Root>¥WEB-INF¥lib ディレクトリー下のファイルをロードします。WAR クラス・ローダー・ポリシーを「アプリケーションの各 War ファイルのクラス・ローダー」に設定した場合は、Web モジュールがこのクラス・ローダーによりロードされます。WAR クラス・ローダー・ポリシーを「アプリケーションの単一クラス・ローダー」に設定した場合は、このクラス・ローダーは呼ばれません。

5-5-2 クラス・ローダー・ポリシー

クラス・ローダー・ポリシーを設定することにより、アプリケーション・モジュールの分離を制御することができます。クラス・ローダー・ポリシーの設定値により、アプリケーション・クラス・ローダーと WAR クラス・ローダーの数が決まります。あるクラスから見ることができるのは、親またはその祖先のクラス・ローダーがロードしたクラスです。子または子孫のクラス・ローダーがロードしたクラス、兄弟関係にあるクラス・ローダーがロードしたクラスは見ることができません。

5-5-2-1 アプリケーション・クラス・ローダー・ポリシー

アプリケーション・クラス・ローダー・ポリシーは、管理コンソールで**[サーバー]→[サーバー・タイプ] → [WebSphere Application Server]→[*AppServer_name*]→[クラス・ローダー・ポリシー]**で設定できます。アプリケーション・クラス・ローダー・ポリシーには「単一」と「複数」のいずれかを設定することができます。

単一（SINGLE）

アプリケーション・クラス・ローダー・ポリシーに「単一」を選択した場合、複数のアプリケーションによってクラス・ローダーが共有され、全ての EJB モジュール、依存 Jar ファイル、共用ライブラリーが同一のアプリケーション・クラス・ローダーにロードされます。各アプリケーションは互いのクラスを見ることができます。

複数（MULTIPLE）

アプリケーション・クラス・ローダー・ポリシーに「複数」を選択した場合、アプリケーション毎に独自のクラス・ローダーが用意され、それぞれのアプリケーション内の EJB モジュール、依存 Jar ファイル、共用ライブラリーをロードします。各アプリケーションは互いのクラスを見ることはできません。

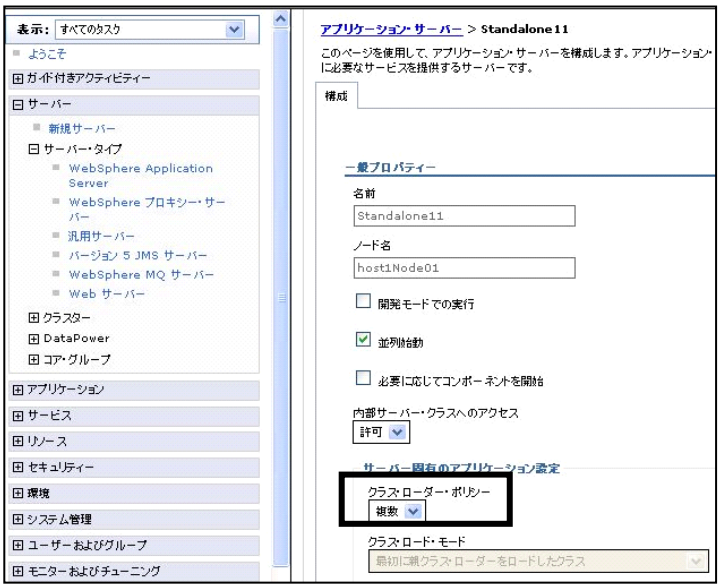


図 5-104 クラス・ローダー・ポリシーの設定画面

5-5-2-2 WAR クラス・ローダー・ポリシー

WAR クラス・ローダー・ポリシーの設定画面を開くには、管理コンソールで**[アプリケーション]→[アプリケーション・タイプ]→[WebSphere エンタープライズ・アプリケーション]→[*Application_name*]→[詳細プロパティー]→[クラス・ロードおよび更新の検出]**をクリックします。WAR クラス・ローダー・ポリシーには「アプリケーションの単一クラス・ローダー」と「アプリケーションの各 War ファイルのクラス・ローダー」のいずれかを設定できます。

アプリケーションの単一クラス・ローダー

WAR クラス・ローダー・ポリシーに「アプリケーションの単一クラス・ローダー」を設定した場合、EJB モジュール等と共に、WAR クラス・ローダーの親であるアプリケーション・クラス・ローダーにより Web モジュールもロードされます。

アプリケーションの各 War ファイルのクラス・ローダー

WAR クラス・ローダー・ポリシーに「アプリケーションの各 War ファイルのクラス・ローダー」を指定した場合、Web モジュールごとに独自の WAR クラス・ローダーが用いられます。各 Web モジュールは互いに他の Web モジュール内のクラスを見ることはできません。

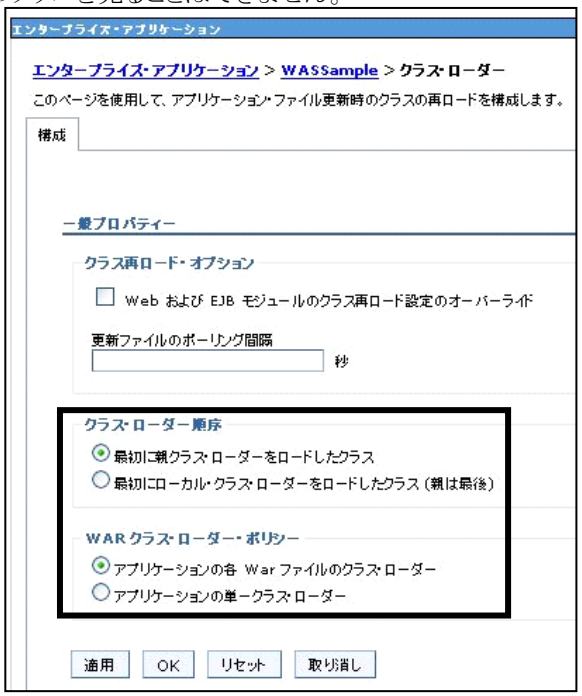


図 5-105 WAR クラス・ローダー・ポリシー設定画面

図 5-106 クラス・ローダー設定によるケース一覧は、<サンプル>のようなアプリケーションに各クラス・ローダー・ポリシーを設定した場合に、各クラス・ローダーにロードされるクラスを示しています。

この図では 2 つのアプリケーションに同一のクラス・ローダー・ポリシーを適用した場合を示していますが、WAS ではアプリケーション毎にクラス・ローダー・ポリシーを設定できるため、全てのケースを網羅しているわけではありません。また共用ライブラリー等の設定方法については、5-2-1-2 共用ライブラリーの定義」をご覧下さい。

<サンプル>

○ 共用ライブラリー（アプリケーション・サーバーと関連付け）：ServerShareLib.jar

○ アプリケーション1

- EJB モジュール：EJB1.jar
- 依存 Jar：Dependency1.jar
- 共用ライブラリー（アプリケーション関連付け）：AppShareLib1.jar
- Web モジュール：WAR1.war
  MANIFEST Class-Path: Dependency1.jar

○ アプリケーション2

- EJB モジュール：EJB2.jar
- 依存 Jar：Dependency2.jar
- 共用ライブラリー（アプリケーション関連付け）：AppShareLib2.jar
- Web モジュール：WAR2.war
  MANIFEST Class-Path: Dependency2.jar

表 5-4 クラス・ローダー設定によるケース一覧

		アプリケーション・クラス・ローダー・ポリシー	
		単一	複数
WAR クラス・ローダー・ポリシー	アプリケーションの単一クラス・ローダー（アプリケーション）	Case1	Case2
	アプリケーションの各 War ファイルのクラス・ローダー（モジュール）	Case3	Case4

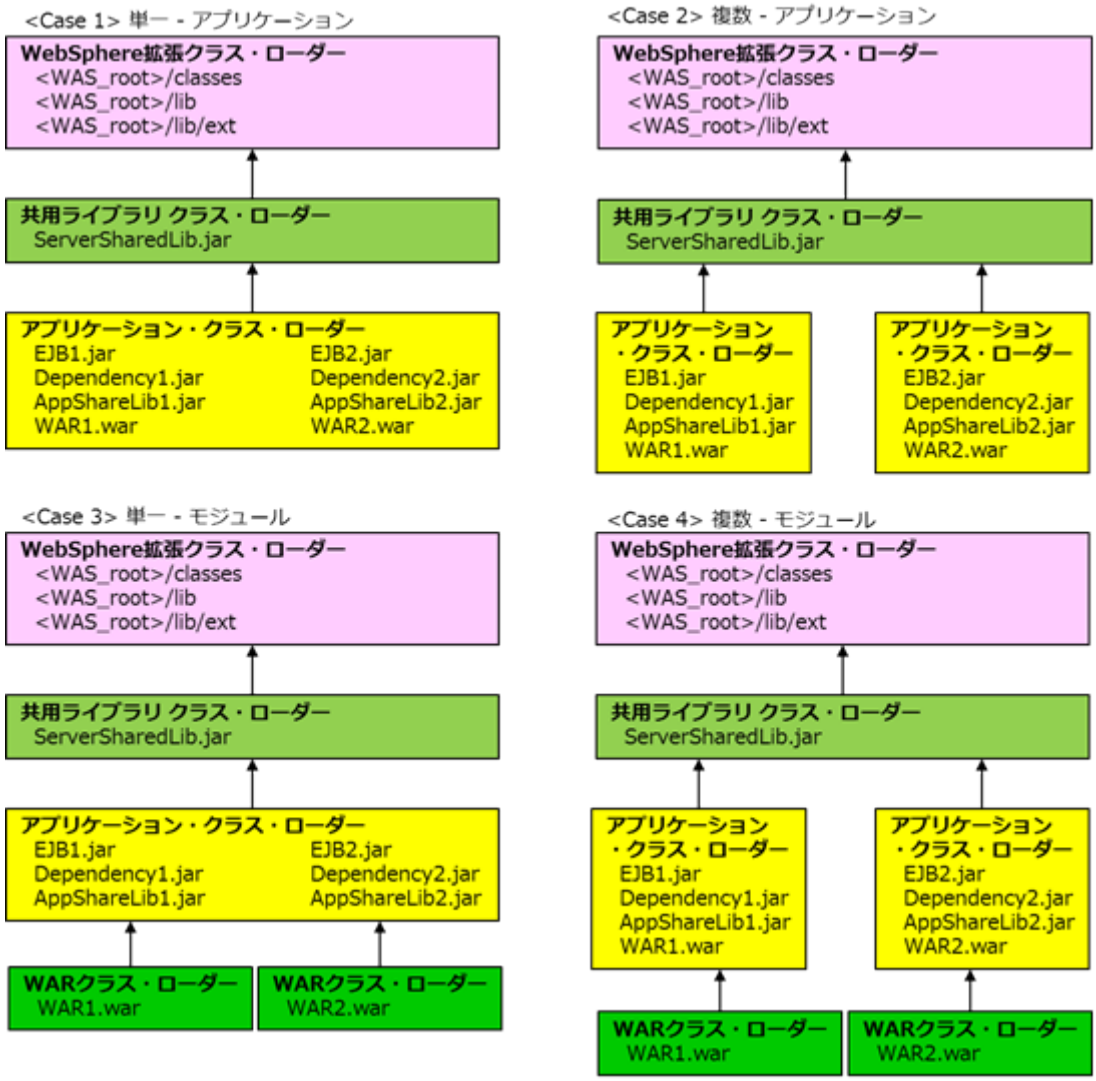


図 5-106 クラス・ローダー設定によるケース一覧

<Case 1>

アプリケーション・クラス・ローダー・ポリシーに「単一」、WAR クラス・ローダー・ポリシーに「アプリケーションの単一クラス・ローダー」を設定したケースです。

アプリケーション1・アプリケーション2内の全ての EJB モジュール、Web モジュール、依存 Jar ファイル、アプリケーション関連付け共用ライブラリー・ファイルが同一のアプリケーション・クラス・ローダーのインスタンスによりロードされます。このケースは、最もモジュールの分離レベルが低く、全てのモジュールが互いのクラスを参照することができます。

<Case 2>

アプリケーション・クラス・ローダー・ポリシーに「複数」、WAR クラス・ローダー・ポリシーに「アプリケーションの単一クラス・ローダー」を設定したケースです。

アプリケーションごとに独自のアプリケーション・クラス・ローダーのインスタンスが用意され、それぞれのアプリケーション内の EJB モジュール、Web モジュール、依存 Jar ファイル、アプリケーション関連付け

共用ライブラリー・ファイルをロードします。このケースでは、兄弟関係にあるアプリケーション・クラス・ローダーにロードされた別 EAR 内のクラスは参照することができません。

<Case 3>

アプリケーション・クラス・ローダー・ポリシーに「単一」、WAR クラス・ローダー・ポリシーに「アプリケーションの各 War ファイルのクラス・ローダー」を設定したケースです。

アプリケーション1・アプリケーション2内の全ての EJB モジュール、依存 Jar ファイル、アプリケーション関連付け共用ライブラリー・ファイルは同一のアプリケーション・クラス・ローダーのインスタンスによりロードされますが、Web モジュールはそれぞれ独自の WAR クラス・ローダーのインスタンスによりロードされます。このケースでは、例えば WAR1.war 内のサーブレットから、親のアプリケーション・クラス・ローダーにロードされたクラスは EAR に関わらず参照することができますが、同一 EAR／別 EAR に関わらず他の Web モジュール内のクラスを参照することはできません。

<Case 4>

アプリケーション・クラス・ローダー・ポリシーに「複数」、WAR クラス・ローダー・ポリシーに「アプリケーションの各 War ファイルのクラス・ローダー」を設定したケースです。

アプリケーションごとに独自のアプリケーション・クラス・ローダーのインスタンスが用意され、それぞれのアプリケーション内の EJB モジュール、依存 Jar ファイル、アプリケーション関連付け共用ライブラリー・ファイルをロードします。また Web モジュールも、Web モジュールごとに独自の WAR クラス・ローダーのインスタンスによりロードされます。このケースは、最も分離レベルが高いケースです。例えば WAR1.war 内のサーブレットから参照できるのは、自らの Web モジュール内のクラス、親のアプリケーション・クラス・ローダーにロードされたアプリケーション1内のクラスです。別 EAR であるアプリケーション2内のクラスや、アプリケーション1内であっても別の Web モジュール内のクラスを参照することはできません。

5-5-3 クラス・ローダー・モード

クラス・ローダーがクラスをロードする際には、クラス・ローダーは自らがクラスをロードしようと試みる前に、常にその親クラス・ローダーに該当クラスをロードするよう委託（Delegation）します。その親クラス・ローダーもさらにその親クラス・ローダーにクラスをロードするよう委託しますので、全ての祖先クラス・ローダーが該当クラスをロードできなかった場合にはじめて自らが該当クラスをロードすることになります。クラス・ローダーは決して子クラス・ローダーに該当クラスのロードを委託することはありません。例えば、WebSphere 拡張クラス・ローダーが Java EE モジュール内のクラスをロードしようと試みる際には、全ての上位のクラス・ローダーが該当クラスをロードできなかった場合にはじめて WebSphere 拡張クラス・ローダーにより該当クラスがロードされます。WebSphere 拡張クラス・ローダーが該当クラスをロードできなかった場合には、下位のクラス・ローダーにクラスをロードさせるのではなく、ClassNotFoundException が投げられます。

Java EE のデフォルトではこのようにクラス・ローダーは常にその親にクラスのロードを委託しますが、WAS V8.5 では、アプリケーション・サーバー関連付け共用ライブラリー・クラス・ローダー、アプリケーション・クラス・ローダー、WAR クラス・ローダーはそれぞれクラス・ローダー・モードを設定することで、親にクラスのロードを委託する前にまず自分でクラスのロードを試みるように設定することもできます。

最初に親クラス・ローダーでロードしたクラス（親が最初／PARENT_FIRST）

クラス・ローダー・モードに「最初に親クラス・ローダーでロードしたクラス」を設定した場合には、クラス・ローダーはクラスをロードする際、常に親のクラス・ローダーに該当クラスのロードをまず委託し、親のクラス・ローダーがロード不能であった場合にはじめて自らローカルのクラスパスからクラスをロードします。

最初にローカル・クラス・ローダーでロードしたクラス（親が最後／PARENT_LAST）

クラス・ローダー・モードに「最初にローカル・クラス・ローダーでロードしたクラス」を設定した場合には、クラス・ローダーはまず自らローカルのクラスパスからのクラスのロードを試み、ロードできなかった場合にのみ、親のクラス・ローダーに該当クラスのロードを委託します。

以下各クラス・ローダー・ポリシー設定時にクラス・ローダー・モードを設定可能な箇所を説明します。

アプリケーション・クラス・ローダー・ポリシーを「単一」に設定した場合

サーバー上の全てのアプリケーションに対して単一のクラス・ローダーが用いられますので、クラス・ローダー・モードはサーバー毎に設定します。管理コンソールで**[サーバー]→[サーバー・タイプ] → [WebSphere Application Server]→[*AppServer_name*]を選択し、 [クラス・ロード・モード]**欄で設定できます。

アプリケーション・クラス・ローダー・ポリシーを「複数」に設定した場合

アプリケーション毎に独自のクラス・ローダーが用いられますので、クラス・ローダー・モードはアプリケーション毎に設定します。管理コンソールで**[アプリケーション]→[アプリケーション・タイプ]→[WebSphere エンタープライズ・アプリケーション]→[*Application_name*]→[詳細プロパティー]→[クラス・ロードおよび更新の検出]**を選択し、**[クラス・ローダー順序]**欄で設定できます。

WAR クラス・ローダー・ポリシーを「アプリケーションの単一クラス・ローダー」に設定した場合

Web モジュールはアプリケーション・クラス・ローダーにロードされますので、Web モジュール毎にクラス・ローダー・モードを設定することはできません。

WAR クラス・ローダー・ポリシーを「アプリケーションの各 War ファイルのクラス・ローダー」に設定した場合

Web モジュール毎に独自のクラス・ローダーが用いられますので、クラス・ローダー・モードは Web モジュール毎に設定することができます。管理コンソールで**[アプリケーション]→[アプリケーション・タイプ]→[WebSphere エンタープライズ・アプリケーション]→[*Application_name*]→[モジュールの管理]→[*Module_name*]**で設定を行いたい WAR ファイルを選択し、**[クラス・ローダー順序]**欄で設定できます。

クラス・ローダー・モードを利用する利点としては、例えば同一アプリケーション内に同一クラス名を持つクラスがあった場合、クラス・ローダー・モードを「親が最後」に設定することにより、クラス・ローダーは親のクラス・ローダーにロードされるクラスをオーバーライドし、独自バージョンのクラスを持つことができます。

例えば、図 5-107 クラス・ローダーの参照順序例のようにロードするクラス・ローダーが異なる 3 つの場所（アプリケーション・サーバー関連付け共用ライブラリー、アプリケーション関連付け共用ライブラリー、<WAR_Root>/WEB-INF ディレクトリーの Classes ディレクトリー配下）に同一クラス名のクラス MyClass がある場合に、クラス・ローダー・モードを設定した際、EJB またはサーブレットが参照するクラスは以下のようになります。

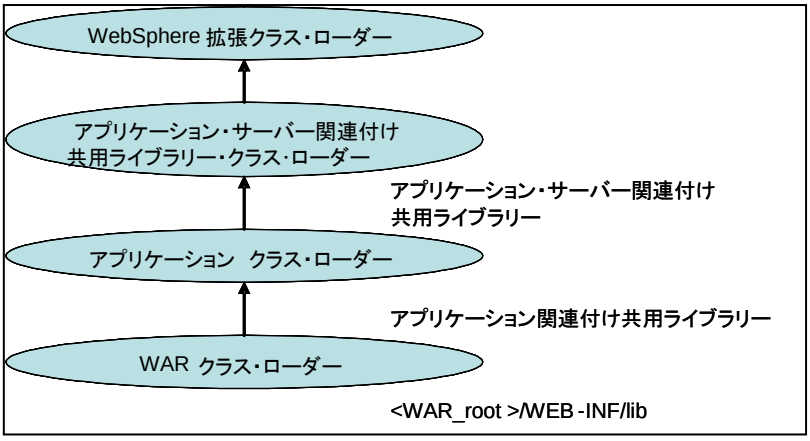


図 5-107 クラス・ローダーの参照順序例

<Case Ⅰ>アプリケーション・クラス・ローダー:親が最初 WAR クラス・ローダー:親が最初
EJB :アプリケーション・サーバー関連付け共用ライブラリー内の MyClass
サーブレット :アプリケーション・サーバー関連付け共用ライブラリー内の MyClass
<Case Ⅱ>アプリケーション・クラス・ローダー:親が最初 WAR クラス・ローダー:親が最後
EJB :アプリケーション・サーバー関連付け共用ライブラリー内の MyClass
サーブレット :<WAR_Root>¥WEB-INF¥classes 内の MyClass
<CaseⅢ>アプリケーション・クラス・ローダー:親が最後 WAR クラス・ローダー:親が最初
EJB :アプリケーション関連付け共用ライブラリー内の MyClass
サーブレット :アプリケーション関連付け共用ライブラリー内の MyClass
<CaseⅣ>アプリケーション・クラス・ローダー:親が最後 WAR クラス・ローダー:親が最後
EJB :アプリケーション関連付け共用ライブラリー内の MyClass
サーブレット :<WAR_Root>¥WEB-INF¥classes 内の MyClass

以上のように、クラス・ローダー・モードを設定することにより、親のクラス・ローダーにロードされるクラスをオーバーライドし、独自バージョンのクラスを持つことができますが、クラス・ローダー・モードに頼るのではなく、同一アプリケーション内に同一クラス名をもったクラスが存在しないよう設計することが重要です。