WebSphere Process Server for z/OS

バージョン 6.0.2

モジュールの開発およびデプロイ

お願い

本書をご使用になる前に、 47 ページの『特記事項』に記載されている情報をお読みください。

本書は、WebSphere Process Server for z/OS (製品番号 5655-N53) バージョン 6、リリース 0、モディフィケーション 2、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

IBM 発行のマニュアルに関する情報のページ

http://www.ibm.com/jp/manuals/

こちらから、日本語版および英語版のオンライン・ライブラリーをご利用いただけます。また、マニュアルに関するご意見やご感想を、上記ページよりお送りください。今後の参考にさせていただきます。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原　典： WebSphere Process Server for z/OS
Version 6.0.2
Developing and Deploying Modules

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2007.3

目次

第 1 章 モジュールの開発の概要

モジュールは、WebSphere Process Server アプリケーションのデプロイメントの基本単位です。モジュールは、アプリケーションが使用する 1 つ以上のコンポーネント・ライブラリーとステージング・モジュールで構成されています。コンポーネントは、ほかのサービス・コンポーネントを参照することができます。モジュールを開発するには、アプリケーションが必要とするコンポーネント、ステージング・モジュール、およびライブラリー (モジュールによって参照される成果物の集合) が実動サーバー上で使用可能であることを確認する必要があります。

WebSphere® Integration Developer は、WebSphere Process Server にデプロイするモジュールを開発するための主要なツールです。ほかの環境でモジュールを開発することもできますが、WebSphere Integration Developer を使用するのが最適な方法です。

WebSphere Process Server は、2 つのタイプのサービス・モジュールをサポートします。ビジネス・サービス用モジュールおよびメディエーション・モジュールです。ビジネス・サービス用モジュールは、プロセスのロジックをインプリメントします。メディエーション・モジュールは、サービス起動をターゲットが理解する形式に変換し、要求をターゲットに渡して結果をオリジネーターに戻すことによって、アプリケーション間の通信を可能にします。

以降のセクションでは、WebSphere Process Server 上でモジュールをインプリメントおよび更新する方法について説明します。

コンポーネントの概要

コンポーネントは、再使用可能なビジネス・ロジックをカプセル化するための基本要素です。サービス・コンポーネントは、インターフェース、参照、インプリメンテーションに関連付けられます。インターフェースは、サービス・コンポーネントと呼び出し側コンポーネントの間の取り決めを定義します。サービス・モジュールは、WebSphere Process Server を使用して、他のモジュールが使用できるようにサービス・コンポーネントをエクスポートしたり、サービス・コンポーネントをインポートして使用したりすることができます。サービス・コンポーネントを呼び出すために、呼び出し側のモジュールはサービス・コンポーネントとのインターフェースを参照します。呼び出し側モジュールからそれぞれのインターフェースへの参照を構成することによって、インターフェースに対する参照が解決されます。

モジュールを開発するには、以下の作業を行う必要があります。

1. モジュール内のコンポーネント用のインターフェースを定義します。
2. サービス・コンポーネントで使用されるビジネス・オブジェクトを定義、変更、または操作します。
3. インターフェースを使用して、サービス・コンポーネントを定義または変更します。

   注: サービス・コンポーネントは、インターフェースを使用して定義されます。

4. 必要に応じて、サービス・コンポーネントをエクスポートまたはインポートします。
5. コンポーネントを使用するモジュールをインストールするために使用する EAR ファイルを作成します。WebSphere Integration Developer のエクスポート EAR 機能を使用してファイルを作成するか、serviceDeploy コマンドを使用して EAR ファイルを作成し、サービス・コンポーネントを使用するサービス・モジュールをインストールします。

開発タイプ

WebSphere Process Server では、サービス指向のプログラミング・パラダイムを促進するコンポーネント・プログラミング・モデルを提供します。このモデルを使用するために、提供者はサービス・コンポーネントのインターフェースをエクスポートします。これにより、利用者はそのインターフェースをインポートして、そのサービス・コンポーネントがローカルであるかのように使用できるようになります。開発者は厳密に型指定されたインターフェースまたは動的型付きインターフェースのいずれかを使用して、サービス・コンポーネントをインプリメントしたり、呼び出したりします。インターフェースとそのメソッドについては、このインフォメーション・センターの『References』のセクションに説明があります。

サービス・モジュールをサーバーにインストールした後、管理コンソールを使用して、アプリケーションからの参照のターゲット・コンポーネントを変更することができます。新しいターゲットは、アプリケーションからの参照が要求しているものと同じビジネス・オブジェクト・タイプを受け入れ、同じ操作を実行する必要があります。

サービス・コンポーネントの開発に関する考慮事項

サービス・コンポーネントを開発する場合は、以下の点を検討してください。

- このサービス・コンポーネントがエクスポートされ、ほかのモジュールによって使用されるかどうか。

  使用される場合、そのコンポーネントに定義したインターフェースを別のモジュールが使用できることを確認してください。
- サービス・コンポーネントを実行するのに比較的長い時間がかかるかどうか。

  長時間かかる場合は、サービス・コンポーネントに非同期のインターフェースをインプリメントすることを検討してください。
- サービス・コンポーネントを分散化することが有益かどうか。

  有益である場合は、サーバーのクラスター上にデプロイされているサービス・モジュール内にサービス・コンポーネントのコピーを配置して、並列処理の利点を活かすことを検討してください。
- アプリケーションが、一相および二相コミット・リソースの混用を必要とするか。

  必要とする場合、アプリケーションの Last Participant サポートを使用可能にしてください。

注: WebSphere Integration Developer を使用してアプリケーションを作成したか、または serviceDeploy コマンドを使用してインストール可能な EAR ファイルを作成した場合、これらのツールは自動的にアプリケーションのサポートを使用可能にします。WebSphere Application Server for z/OS インフォメーション・センターで、トピック『同一トランザクション内での 1 フェーズ・コミットおよび 2 フェーズ・コミットのリソースの使用』を参照してください。

サービス・モジュールの開発

サービス・コンポーネントは、サービス・モジュール内に含まれていなければなりません。サービス・コンポーネントを含むためのサービス・モジュールを開発することが、ほかのモジュールにサービスを提供するための鍵となります。

以下のタスクでは、要件を分析した結果、ほかのモジュールで使用できるように、サービス・コンポーネントをインプリメントすると有益であると判断されていることが前提となっています。

要件を分析した結果、サービス・コンポーネントの提供と利用が効率的な情報処理手段であると判断できる場合があります。ご使用の環境にとって再使用可能なサービス・コンポーネントが有効であると判断したうえで、サービス・コンポーネントを含むためのサービス・モジュールを作成してください。

1. ほかのサービス・モジュールで使用できるコンポーネントを特定します。

   サービス・コンポーネントを特定したら、『サービス・コンポーネントの開発』に進みます。
2. ほかのサービス・モジュール内のサービス・コンポーネントを使用できる、アプリケーション内のサービス・コンポーネントを特定します。

   サービス・コンポーネントとそれぞれのターゲット・コンポーネントを特定したら、『コンポーネントの呼び出し』に進みます。
3. クライアント・コンポーネントをワイヤー経由でターゲット・コンポーネントに接続します。

サービス・コンポーネントの開発

ご使用のサーバー内の複数のアプリケーションに再使用可能なロジックを提供するための、サービス・コンポーネントを作成します。

この作業では、複数のモジュールで使用できる処理がすでに作成され、特定されていることが前提になっています。

複数のモジュールで 1 つのサービス・コンポーネントを使用することができます。サービス・コンポーネントをエクスポートすると、インターフェースを介してそのコンポーネントを参照するほかのモジュールが、そのサービス・コンポーネントを利用できるようになります。この作業では、ほかのモジュールがコンポーネントを使用できるように、そのサービス・コンポーネントを作成する方法を説明します。

注: 1 つのサービス・コンポーネントに、複数のインターフェースを設定することができます。

1. 呼び出し元とサービス・コンポーネントの間のデータの移動のためのデータ・オブジェクトを定義します。

   データ・オブジェクトおよびそのタイプは、呼び出し元とサービス・コンポーネント間のインターフェースの一部となります。
2. 呼び出し元がサービス・コンポーネントを参照するときに使用するインターフェースを定義します。

   このインターフェース定義で、サービス・コンポーネントを指定し、サービス・コンポーネント内のすべての使用可能なメソッドをリストします。
3. インプリメンテーションを定義するクラスを開発します。
   - コンポーネントが長期にわたって実行される (非同期) 場合は、ステップ 4 に進みます。
   - コンポーネントが長期にわたって実行されるものでない (同期) 場合は、ステップ 5 に進みます。
4. 非同期インプリメンテーションを開発します。

   重要: 非同期型コンポーネント・インターフェースでは、joinsTransaction プロパティーを `true` に設定できません。

   a. 同期型サービス・コンポーネントを示すインターフェースを定義します。
   b. サービス・コンポーネントのインプリメンテーションを定義します。
   c. ステップ 6 に進みます。
5. 同期インプリメンテーションを開発します。
   a. 同期型サービス・コンポーネントを示すインターフェースを定義します。
   b. サービス・コンポーネントのインプリメンテーションを定義します。
6. コンポーネントのインターフェース、およびインプリメンテーションを拡張子が .java のファイルに保管します。
7. サービス・モジュールと必要なリソースを JAR ファイルにパッケージ化します。

   ステップ 7 から 9 までの詳しい説明については、このインフォメーション・センターの『実動サーバーへのモジュールのデプロイ』のセクションを参照してください。
8. serviceDeploy コマンドを実行して、アプリケーションを格納するインストール可能な EAR ファイルを作成します。
9. サーバー・ノード上にアプリケーションをインストールします。
10. オプション: ほかのサービス・モジュール内のサービス・コンポーネントを呼び出す場合は、呼び出し元とそれに対応するサービス・コンポーネント間のワイヤーを構成します。

    このインフォメーション・センターの『管理』セクションに、ワイヤーの構成についての説明があります。

コンポーネントの開発例

この例では、1 つのメソッド CustomerInfo をインプリメントする同期型サービス・コンポーネントを示しています。最初のセクションでは、getCustomerInfo というメソッドをインプリメントするサービス・コンポーネントに対するインターフェースを定義しています。

```
public interface CustomerInfo {
 public Customer getCustomerInfo(String customerID);
}
```

以下のコード・ブロックで、サービス・コンポーネントをインプリメントします。

```
public class CustomerInfoImpl implements CustomerInfo {
 public Customer getCustomerInfo(String customerID) {
  Customer cust = new Customer();

  cust.setCustNo(customerID);
  cust.setFirstName("Victor");
  cust.setLastName("Hugo");
  cust.setSymbol("IBM");
  cust.setNumShares(100);
  cust.setPostalCode(10589);
  cust.setErrorMsg("");

  return cust;
 }
}
```

この例では、非同期型サービス・コンポーネントを作成します。コードの最初のセクションでは、getQuote というメソッドをインプリメントするサービス・コンポーネントに対するインターフェースを定義しています。

```
public interface StockQuote {

 public float getQuote(String symbol);
}
```

以下のセクションは、StockQuote に関連したクラスのインプリメンテーションです。

```
public class StockQuoteImpl implements StockQuote {

 public float getQuote(String symbol) {


     return 100.0f;
 }
}
```

以下のコード・セクションは、非同期インターフェース StockQuoteAsync をインプリメントします。

```
public interface StockQuoteAsync {

 // deferred response
 public Ticket getQuoteAsync(String symbol);
 public float getQuoteResponse(Ticket ticket, long timeout);

 // callback
 public Ticket getQuoteAsync(String symbol, StockQuoteCallback callback);
}
```

以下のセクションは、onGetQuoteResponse メソッドを定義するインターフェース StockQuoteCallback です。

```
public interface StockQuoteCallback {

 public void onGetQuoteResponse(Ticket ticket, float quote);
}
```

サービスを起動します。

コンポーネントの呼び出し

モジュールを含むコンポーネントは、WebSphere Process Server クラスターの任意のノード上でコンポーネントを使用することができます。

コンポーネントを呼び出す前に、WebSphere Process Server に、コンポーネントを含むモジュールがインストールされていることを確認してください。

コンポーネントは、コンポーネントの名前を使用し、コンポーネントに適したデータ型を渡すことによって、WebSphere Process Server クラスター内で使用可能なすべてのサービス・コンポーネントを使用することができます。この環境内でコンポーネントを呼び出すには、必要なコンポーネントを見つけてから、そのコンポーネントへの参照を作成する操作が必要です。

注: モジュール内のコンポーネントは、同一のモジュール内のコンポーネントを呼び出すことができ、これはモジュール内呼び出しと呼ばれます。提供側コンポーネント内のインターフェースをエクスポートし、呼び出し側コンポーネント内でインターフェースをインポートすることによって、外部呼び出し (モジュール内呼び出し) をインプリメントしてください。

重要: 呼び出し側モジュールが稼動するサーバーと異なるサーバー上に存在するコンポーネントを呼び出す場合は、サーバーへの追加構成を実行する必要があります。必要な構成は、コンポーネントが非同期に呼び出されるか、同期して呼び出されるかによって異なります。この場合のアプリケーション・サーバーの構成方法は、関連タスクで説明されています。

1. 呼び出し側モジュールに必要なコンポーネントを判別します。

   コンポーネント内のインターフェースの名前と、そのインターフェースに必要なデータ型を書き留めます。
2. データ・オブジェクトを定義します。

   入力または戻りは Java™ クラスでかまいませんが、サービス・データ・オブジェクトが最適です。
3. コンポーネントを探します。
   a. ServiceManager クラスを使用して、呼び出し側モジュールが使用できる参照を取得します。
   b. locateService() メソッドを使用して、コンポーネントを探します。

      インターフェースは、コンポーネントに応じて、Web サービス記述言語 (WSDL) ポート・タイプまたは Java インターフェースのいずれかを使用することができます。

4. コンポーネントを同期式、または非同期に呼び出します。

   Java インターフェースを使用してコンポーネントを呼び出すことも、 invoke() メソッドを使用してコンポーネントを動的に呼び出すこともできます。
5. 戻り値を処理します。

   コンポーネントが例外を生成することがあるので、クライアントでは例外の処理が可能である必要があります。

コンポーネントの呼び出し例

次の例では、 ServiceManager クラスを作成します。

```
ServiceManager serviceManager = new ServiceManager();
```

以下の例は、 ServiceManager クラスを使用して、コンポーネントの参照を含んでいるファイルからコンポーネントのリストを取得します。

```
InputStream myReferences = new FileInputStream("MyReferences.references");
ServiceManager serviceManager = new ServiceManager(myReferences);
```

以下のコードは、StockQuote Java インターフェースをインプリメントするコンポーネントを探します。

```
StockQuote stockQuote = (StockQuote)serviceManager.locateService("stockQuote");
```

以下のコードは、Java または WSDL ポート・タイプ・インターフェースをインプリメントするコンポーネントを探します。呼び出し側モジュールは、Service インターフェースを使用して、コンポーネントと対話します。

ヒント: コンポーネントが Java インターフェースをインプリメントする場合は、コンポーネントをインターフェースまたは invoke() メソッドのいずれかを使用して呼び出すことができます。

```
Service stockQuote = (Service)serviceManager.locateService("stockQuote");
```

次の例は、別のコンポーネントを呼び出すコード MyValue を示しています。

```
public class MyValueImpl implements MyValue {

 public float myValue throws MyValueException {

  ServiceManager serviceManager = new ServiceManager();

     // variables
         Customer customer = null;
         float quote = 0;
         float value = 0;

     // invoke
         CustomerInfo cInfo =
      (CustomerInfo)serviceManager.locateService("customerInfo");
         customer = cInfo.getCustomerInfo(customerID);

     if (customer.getErrorMsg().equals("")) {

         // invoke
       StockQuoteAsync sQuote =
      (StockQuoteAsync)serviceManager.locateService("stockQuote");
       Ticket ticket =  sQuote.getQuoteAsync(customer.getSymbol());
    // ... do something else ...
       quote =  sQuote.getQuoteResponse(ticket, Service.WAIT);
```

```
            // assign
             value = quote * customer.getNumShares();
        } else {

            // throw
            throw new MyValueException(customer.getErrorMsg());
        }
        // reply
            return value;
    }
}
```

呼び出し側モジュールの参照とコンポーネントのインターフェースの間のワイヤーを構成します。

コンポーネントの動的呼び出し

Web サービス記述言語 (WSDL) ポート・タイプ・インターフェースを指定したコンポーネントをモジュールから呼び出す場合、モジュールは invoke() メソッドを使用して、そのコンポーネントを動的に呼び出す必要があります。

この操作では、呼び出し側コンポーネントがコンポーネントを動的に呼び出すことが前提となっています。

WSDL ポート・タイプ・インターフェースの場合は、呼び出し側コンポーネントは invoke() メソッドを使用して、コンポーネントを呼び出す必要があります。呼び出し側モジュールから、この方法で Java インターフェースを指定したコンポーネントも呼び出すことができます。

1. 必要なコンポーネントを含んでいるモジュールを判別します。
2. コンポーネントが必要とする配列を判別します。

   入力配列は、次の 3 つのタイプのいずれかです。
   - 大文字の Java プリミティブ型、またはこの型の配列
   - 通常の Java クラス、またはクラスの配列
   - サービス・データ・オブジェクト (SDO)
3. コンポーネントからの応答を収容する配列を定義します。

   応答配列は、入力配列と同じタイプでかまいません。
4. invoke() メソッドを使用して、必要なコンポーネントを呼び出し、配列オブジェクトをそのコンポーネントに渡します。
5. 結果を処理します。

コンポーネントの動的呼び出しの例

以下の例では、モジュールは invoke() メソッドを使用して、大文字の Java プリミティブ・データ型を使用するコンポーネントを呼び出します。

```
Service service = (Service)serviceManager.locateService("multiParamInf");

  Reference reference = service.getReference();

  OperationType methodMultiType =
    reference.getOperationType("methodWithMultiParameter");

  Type t = methodMultiType.getInputType();
```

```
BOFactory boFactory = (BOFactory)serviceManager.locateService
  ("com/ibm/websphere/bo/BOFactory");

DataObject paramObject = boFactory.createbyType(t);

paramObject.set(0,"input1")
paramObject.set(1,"input2")
paramObject.set(2,"input3")

service.invoke("methodMultiParamater",paramObject);
```

次の例では、WSDL ポート・タイプ・インターフェースをターゲットとして持つ呼び出しメソッドを使用します。

```
Service serviceOne = (Service)serviceManager.locateService("multiParamInfWSDL");

DataObject dob = factory.create("http://MultiCallWSServerOne/bos", "SameBO");
  dob.setString("attribute1", stringArg);

DataObject wrapBo = factory.createByElement
 ("http://MultiCallWSServerOne/wsdl/ServerOneInf", "methodOne");
  wrapBo.set("input1", dob); //wrapBo encapsulates all the parameters of methodOne
  wrapBo.set("input2", "XXXX");
  wrapBo.set("input3", "yyyy");

DataObject resBo= (DataObject)serviceOne.invoke("methodOne", wrapBo);
```

モジュールとターゲットの分離の概要

モジュールを開発する際、複数のモジュールが使用できるサービスを識別します。このようにしてサービスにてこ入れすることにより、開発サイクルとコストを最小化します。多数のモジュールによって使用されるサービスがある場合は、ターゲットがアップグレードされた場合に新規サービスへの切り替えが呼び出しモジュールに対して透過的になるように、呼び出しモジュールをターゲットから分離する必要があります。このトピックでは、単純な呼び出しモデルと分離された呼び出しモデルを対比して、分離がどのように役立つかを示す例を提供します。特定の例について説明しますが、これが、ターゲットからモジュールを分離する唯一の方法というわけではありません。

単純な呼び出しモデル

モジュールを開発する際、その他のモジュールにあるサービスを使用することができます。これは、モジュールにサービスをインポートしてからそのサービスを呼び出すことによって実行します。インポートされたサービスは、WebSphere Integration Developer で、または管理コンソール内のサービスをバインディングすることによって、その他のモジュールによってエクスポートされたサービスに「関連付け」られます。『単純な呼び出しモデル』は、このモデルを示しています。

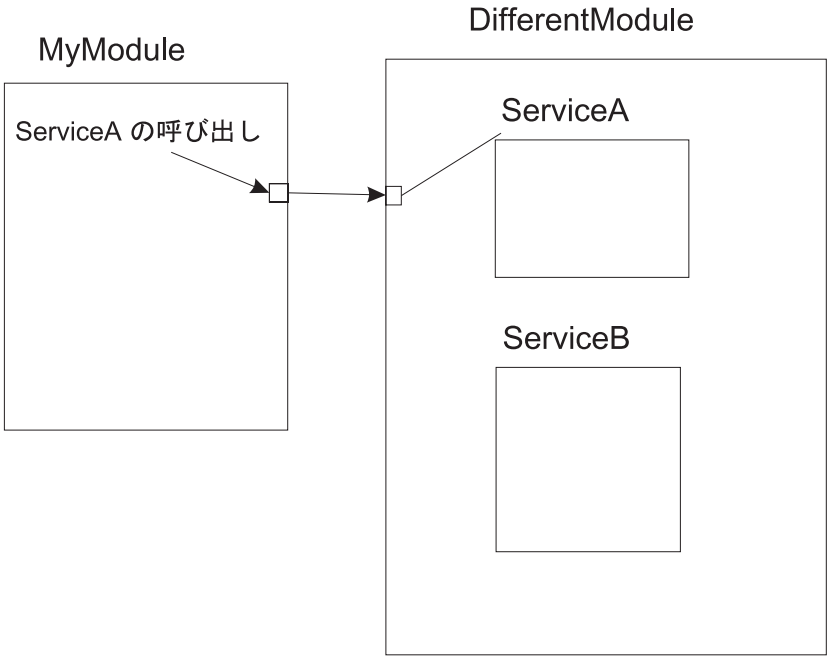

図 1. 単純な呼び出しモデル

分離された呼び出しモデル

呼び出しモジュールを停止せずに呼び出しのターゲットを変更するには、呼び出しのターゲットから呼び出しモジュールを分離します。この場合、モジュールそのものではなくダウンストリーム・ターゲットを変更しているので、ターゲットの変更中もモジュールが処理を続行できます。『アプリケーションの分離の例』は、分離によって、呼び出しモジュールの状況に影響を与えずにターゲットを変更する方法を示します。

アプリケーションの分離の例

単純な呼び出しモデルを使用した場合、同一のサービスを呼び出す複数のモジュールは、『単一のサービスを呼び出す複数のアプリケーション』のようになります。MODA、MODB、および MODC はすべて CalculateFinalCost を呼び出します。

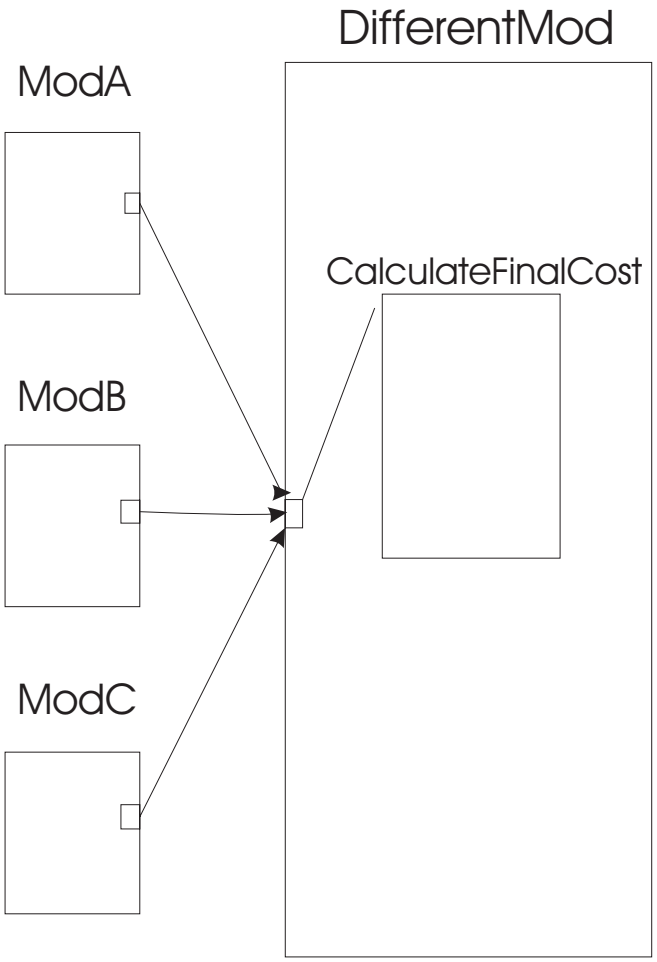

図 2. 単一のサービスを呼び出す複数のアプリケーション

CalculateFinalCost によって提供されるサービスは、そのサービスを使用するすべてのモジュールに新規コストが反映されるように、更新する必要があります。開発チームは、新規サービス UpdatedCalculateFinal を構築およびテストして、変更を取り込みます。新規サービスは実動に移す準備ができています。分離を使用しない場合は、UpdateCalculateFinal を呼び出すために、 CalculateFinalCost を呼び出すモジュールをすべて更新する必要があります。分離を使用した場合、実行する必要があるのは、バッファー・モジュールをターゲットに接続するバインディングを変更することだけです。

注: このようにサービスを変更することにより、オリジナルのサービスを、それを必要とするその他のモジュールに提供し続けることができます。

分離を使用して、アプリケーションとターゲットの間でバッファー・モジュールを作成します (『UpdateCalculateFinal を呼び出す分離された呼び出しモデル』を参照してください)。

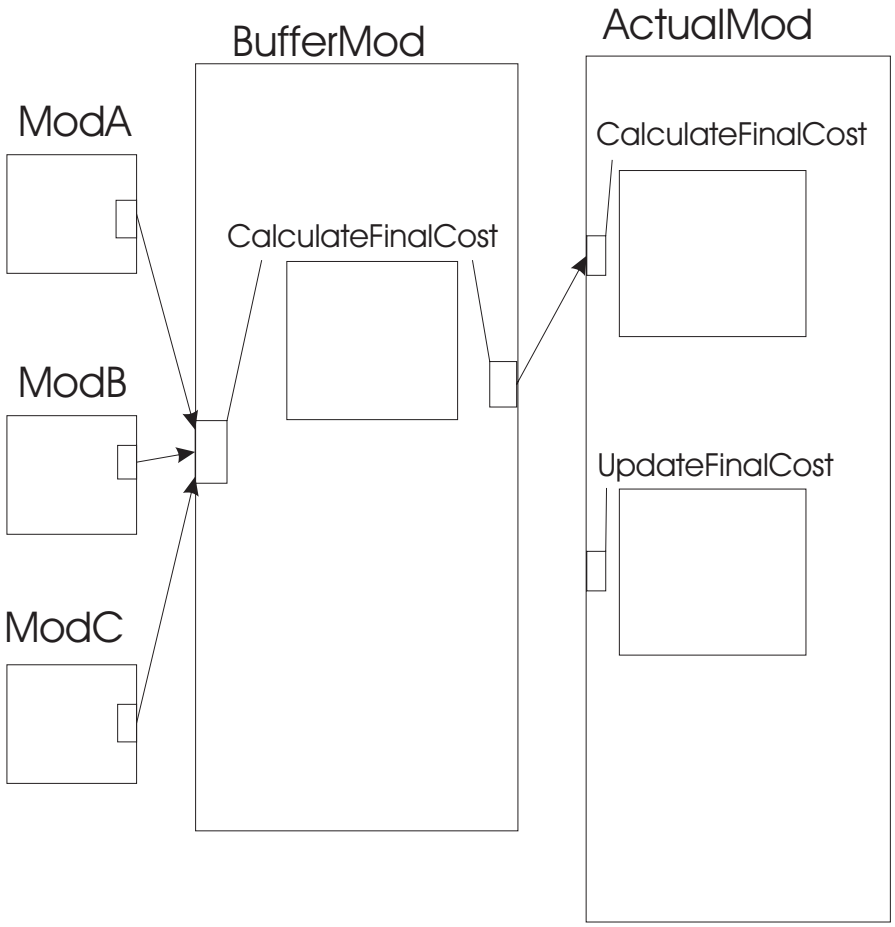

図 3. UpdateCalculateFinal を呼び出す分離された呼び出しモデル

このモデルで、呼び出しモジュールは変わりません。実行する必要があるのは、バインディングをバッファー・モジュール・インポートからターゲットへ変更することだけです (『UpdatedCalculateFinal を呼び出す分離された呼び出しモデル』を参照してください)。

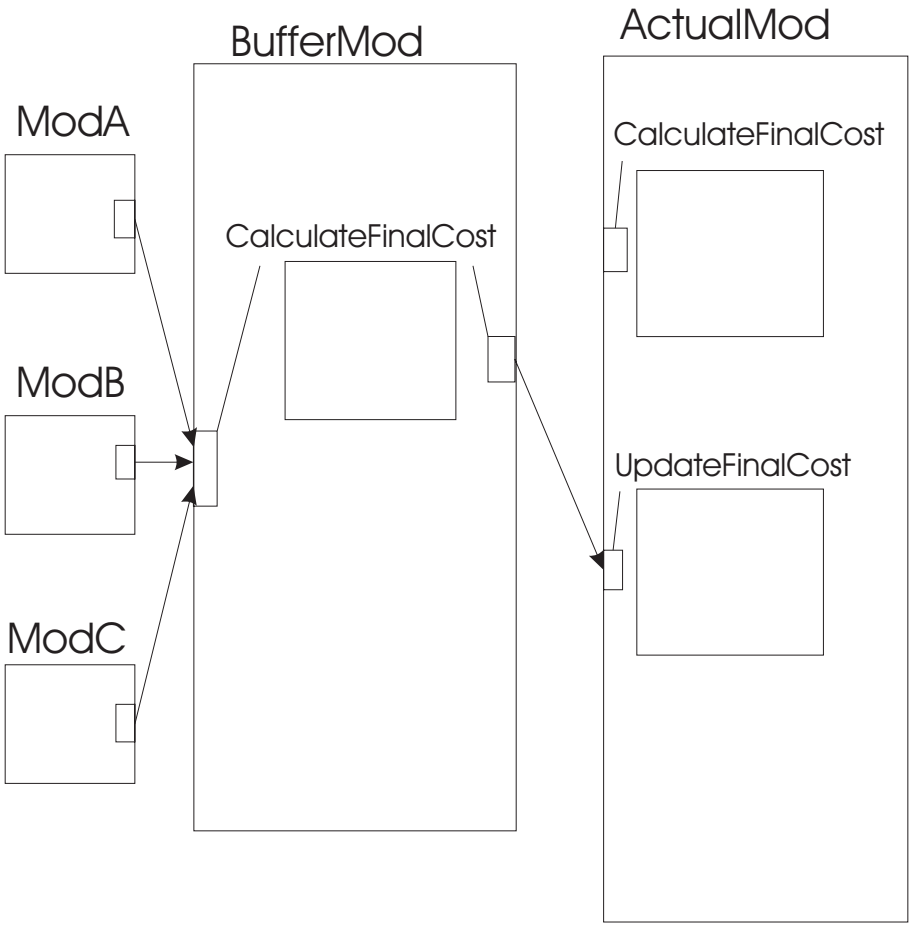

図 4. UpdatedCalculateFinal を呼び出す分離された呼び出しモデル

バッファー・モジュールがターゲットを同期で呼び出す場合、元のアプリケーションに戻される結果は、バッファー・モジュール (メディエーション・モジュール、またはビジネス・モジュール用のサービス) を再始動したときに、新規ターゲットから送信されます。バッファー・モジュールがターゲットを非同期で呼び出す場合、元のアプリケーションに戻される結果は、次の呼び出し時に新規ターゲットから送信されます。

第 2 章 モジュールの準備とインストールの概要

モジュールのインストール (デプロイとも呼ばれる) では、モジュールをテスト環境または実稼働環境のいずれかで活動化します。この概要では、テストおよび実稼働環境と、モジュールのインストールに必要な手順の一部について簡単に説明します。

注: アプリケーションを実稼働環境でインストールするプロセスは、WebSphere Application Server for z/OS インフォメーション・センターの『アプリケーションの開発とデプロイ』の項で説明されているプロセスと同様です。これらのトピックをお読みになったことがない場合、まず目を通してください。

モジュールを実稼働環境にインストールする前に、必ずテスト環境での変更内容を確認してください。モジュールをテスト環境にインストールする場合は、WebSphere Integration Developer を使用します。詳しくは、WebSphere Integration Developer インフォメーション・センターを参照してください。モジュールを実稼働環境にインストールする場合は、WebSphere Process Server を使用します。

このトピックでは、モジュールの実稼働環境へのインストールおよびその準備に必要な概念と作業について説明します。モジュールで使用されるオブジェクトを収容するファイルや、モジュールをテスト環境から実稼働環境へ移行する方法について説明しているその他のトピックがあります。モジュールを正しくインストールするためには、これらのファイルとファイルに格納されている内容を理解することが大切です。

ライブラリーと JAR ファイルの概要

モジュールでは、ライブラリー内の成果物を使用することがよくあります。成果物およびライブラリーは、モジュールをデプロイするときに指定する Java アーカイブ (JAR) ファイルに含まれています。

モジュールの開発時に、そのモジュールのさまざまな部分で使用する特定のリソースまたはコンポーネントを指定する場合があります。これらのリソースまたはコンポーネントは、モジュールの開発時に作成したオブジェクトである場合と、既にサーバー上にデプロイされているライブラリー内のオブジェクトである場合があります。ここでは、アプリケーションのインストール時に必要となるライブラリーおよびファイルについて説明します。

ライブラリーの概要

ライブラリーには、WebSphere Integration Developer 内の複数のモジュールで使用されるオブジェクトおよびリソースが格納されています。これらの成果物は、JAR ファイル、リソース・アーカイブ (RAR) ファイル、または Web サービス・アーカイブ (WAR)ファイルに格納されています。これらの成果物の例としては、次のものがあります。

- インターフェースまたは Web サービス記述子 (拡張子 .wsdl のファイル)
- ビジネス・オブジェクトの XML スキーマ定義 (拡張子 .xsd のファイル)

- ビジネス・オブジェクト・マップ (拡張子 .map のファイル)
- リレーションシップ定義とロール定義 (拡張子 .rel および .rol のファイル)

ある成果物がモジュールで必要になると、EAR クラス・パスに基づいてサーバーがこれを探し出し、メモリーにまだロードされていない場合は、ロードします。それ以降は、成果物が置き換えられないかぎりは、その成果物に対する要求では、メモリーにロードしたコピーが使用されます。図 5 に、アプリケーションとコンポーネントおよび関連するライブラリーの包含関係を示します。

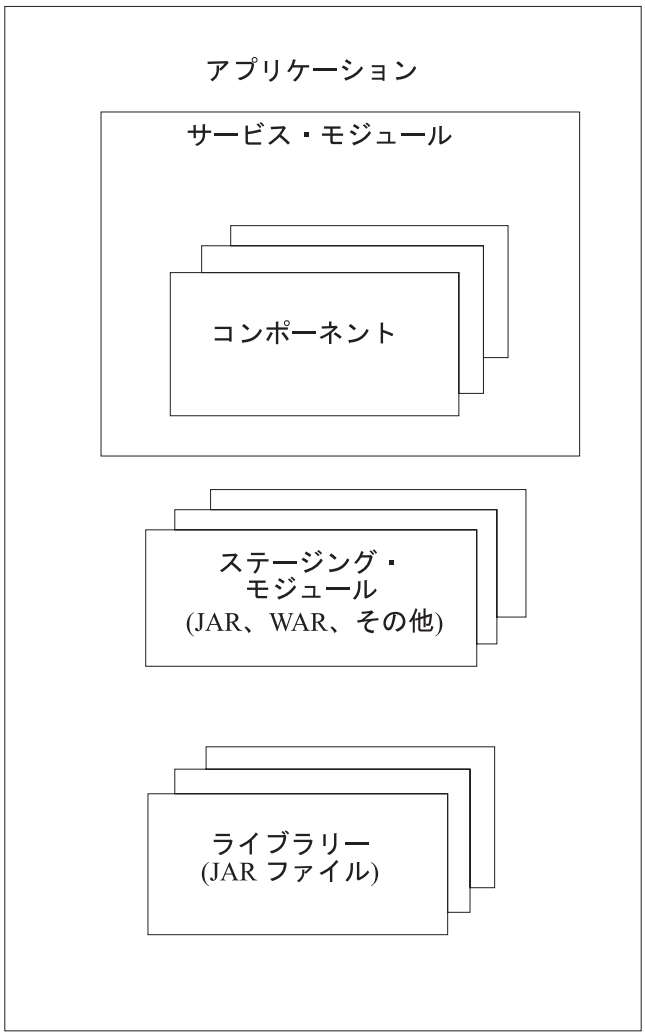

図 5. モジュール、コンポーネント、およびライブラリー間の関係

JAR、RAR、および WAR ファイルの概要

モジュールのコンポーネントを格納できるファイルは複数存在します。これらのファイルについては、Java プラットフォーム、Enterprise Edition 仕様に詳しい説明があります。 JAR ファイルの詳細については、JAR 仕様に説明があります。

WebSphere Process Server では、JAR ファイルにアプリケーション (モジュールで使用するほかのサービス・コンポーネントへの支援的な参照およびインターフェースをすべて含んだ、モジュールのアセンブル・バージョン) も格納されています。アプリケーションを完全にインストールするには、この JAR ファイル、その他のす

べてのライブラリー (JAR ファイル、Web サービス・アーカイブ (WAR) ファイル、リソース・アーカイブ (RAR) ファイル、ステージング・ライブラリー (Enterprise Java Beans - EJB) JAR ファイル、またはその他のアーカイブなど) が必要です。また、serviceDeploy コマンドを使用して、インストール可能な EAR ファイルを作成する必要があります (『 実動サーバーへのモジュールのインストール』を参照)。

ステージング・モジュールの命名規則

ライブラリー内では、ステージング・モジュールの名前についての要件があります。ステージング・モジュールの名前は、それぞれのモジュールで固有でなければなりません。アプリケーションをデプロイするために必要なその他のモジュールには、ステージング・モジュールの名前との間に競合が発生しないような名前を付けてください。*myService* という名前のモジュールの場合、ステージング・モジュール名は、次のようになります。

- *myService*App
- *myService*EJB
- *myService*EJBClient
- *myService*Web

注: serviceDeploy コマンドは、サービスに WSDL ポート・タイプ・サービスが含まれている場合にのみ、*myService* Web ステージング・モジュールを作成します。

ライブラリー使用時の考慮事項

ライブラリーを使用することにより、ビジネス・オブジェクトの整合性およびモジュール間での処理の整合性が保証されます。なぜなら、それぞれの呼び出し側モジュールは、特定のコンポーネントの専用コピーを所有するからです。不整合や障害が発生しないようにするために、呼び出し側モジュールで使用するコンポーネントおよびビジネス・オブジェクトに対する変更は、すべての呼び出し側モジュール間で整合がとれている必要があります。呼び出し側モジュールを更新するには、次の手順を実行します。

1. モジュールおよびライブラリーの最新コピーを実動サーバーにコピーします。
2. serviceDeploy コマンドを使用して、インストール可能な EAR ファイルを再作成します。
3. 呼び出し側モジュールを含む実行中のアプリケーションを停止し、再インストールします。
4. 呼び出し側モジュールを含むアプリケーションを再始動します。

EAR ファイルの概要

EAR ファイルは、サービス・アプリケーションの実動サーバーへのデプロイにおいて、重要な要素です。

エンタープライズ・アーカイブ (EAR) ファイルとは、アプリケーションがデプロイメントに必要とするライブラリー、エンタープライズ Bean、および JAR ファイルを格納した圧縮ファイルです。

アプリケーション・モジュールを WebSphere Integration Developer からエクスポートするときに、JAR ファイルを作成します。この JAR ファイルおよびその他の成果物ライブラリーまたはオブジェクトを、インストール・プロセスの入力として使用します。 serviceDeploy コマンドは、アプリケーションを構成する、コンポーネントの記述と Java コードを含む入力ファイルから EAR ファイルを作成します。

サーバーへのデプロイの準備

モジュールの開発およびテストが終了したら、モジュールをテスト・システムからエクスポートし、実稼働環境にデプロイする必要があります。アプリケーションをインストールするには、モジュールのエクスポート時に必要となるパスと、モジュールが必要とするライブラリーを認識している必要があります。

このタスクを開始する前に、テスト・サーバーでモジュールの開発およびテストを完了し、各種問題およびパフォーマンス問題を解決しておく必要があります。

このタスクでは、アプリケーションの必要な部分がすべて使用可能かどうか、および実動サーバーに移せるように正しくパッケージされているかどうか検証します。

注: WebSphere Integration Developer からエンタープライズ・アーカイブ (EAR) ファイルをエクスポートし、そのファイルを直接 WebSphere Process Server にインストールすることもできます。

重要: コンポーネント内部のサービスがデータベースを使用する場合、データベースに直接接続されたサーバーにアプリケーションをインストールします。

1. デプロイするモジュール用のコンポーネントを含むフォルダーを見つけます。

   コンポーネント・フォルダーの名前は *module-name* で、その中に *module*.module という名前のファイル (基本モジュール) があります。
2. モジュールに含まれるすべてのコンポーネントが、モジュール・フォルダーの下のコンポーネント・サブフォルダー内にあることを確認します。

   使いやすくするために、サブフォルダーには *module/component* のような名前を付けます。
3. 各コンポーネントを構成するすべてのファイルが適切なコンポーネント・サブフォルダーに格納されていて、*component-file-name*.component といった名前が付けられていることを確認します。

   コンポーネント・ファイルには、モジュール内の個々のコンポーネントの定義が記述されています。
4. 他のすべてのコンポーネントおよび成果物が、それらを必要とするコンポーネントのサブフォルダーに格納されていることを確認します。

   このステップで、コンポーネントが必要とする成果物への参照が使用可能であることを確認します。 serviceDeploy コマンドがステージング・モジュールに対して使用する名前と、これらのコンポーネントの名前が競合していてはなりません。『ステージング・モジュールの命名規則』を参照してください。
5. 参照ファイル (*module*.references) が、ステップ 1 のモジュール・フォルダー内に存在することを確認します。

参照ファイルは、モジュール内の参照およびインターフェースを定義します。

6. ワイヤー・ファイル (*module*.wires) がコンポーネント・フォルダーに存在することを確認します。

   ワイヤー・ファイルは、モジュール内の参照とインターフェース間の接続を完成させます。

7. マニフェスト・ファイル (*module*.manifest) がコンポーネント・フォルダーに存在することを確認します。

   マニフェストは、モジュールと、モジュールを構成するすべてのコンポーネントをリストします。マニフェストには、クラスパス・ステートメントも記述されています。これは、serviceDeploy コマンドが、モジュールが必要とする他のモジュールを検出できるようにするためです。

8. モジュールの圧縮ファイルまたは JAR ファイルを作成します。このファイルは serviceDeploy コマンドの入力として使用します。このコマンドは、実動サーバーにインストールするモジュールを準備してくれます。

デプロイメント前の MyValue モジュールのフォルダー構造の例

以下の例は、MyValue、CustomerInfo、および StockQuote というコンポーネントから構成される、MyValueModule モジュールのディレクトリー構造を示しています。

```
MyValueModule
   MyValueModule.manifest
   MyValueModule.references
   MyValueModule.wiring
   MyValueClient.jsp
process/myvalue
   MyValue.component
   MyValue.java
   MyValueImpl.java
service/customerinfo
   CustomerInfo.component
   CustomerInfo.java
   Customer.java
   CustomerInfoImpl.java
service/stockquote
   StockQuote.component
   StockQuote.java
   StockQuoteAsynch.java
   StockQuoteCallback.java
   StockQuoteImpl.java
```

『実動サーバーへのモジュールのインストール』の説明に従って、モジュールを実動システムにインストールします。

クラスター上のサービス・アプリケーションのインストールに関する考慮事項

クラスターにサービス・アプリケーションをインストールする場合、追加要件が発生します。クラスターにサービス・アプリケーションをインストールする際に、これらの考慮事項に注意することが重要です。

クラスターは、スケール・メリットを提供して、サーバー間の要求ワークロードのバランスを取り、アプリケーションのクライアントに対して一定レベルの可用性を

提供できるようにすることで、処理環境に多くのメリットをもたらす可能性があります。クラスター上で、サービスを含むアプリケーションをインストールする前に、以下の事項を検討してください。

- アプリケーションのユーザーが、クラスタリングにより提供される処理能力と可用性を必要としているか。

  その場合、クラスタリングが正しい解決策です。クラスタリングにより、アプリケーションの可用性と能力が向上します。
- クラスターがサービス・アプリケーション用に正しく準備されているか。

  サービスを含む最初のアプリケーションをインストールして開始する前に、クラスターを正しく構成する必要があります。クラスターを正しく構成していない場合、アプリケーションは要求を正しく処理できません。
- クラスターのバックアップはあるか。

  バックアップ・クラスターにもアプリケーションをインストールする必要があります。

第 3 章 実動サーバーへのモジュールのインストール

このトピックでは、テスト・サーバーからアプリケーションを取り出して実稼働環境にデプロイする際に行うステップについて説明します。

サービス・アプリケーションを実動サーバーにデプロイする前に、テスト・サーバー上でアプリケーションをアセンブルし、テストします。テストが終わったら、「モジュールの開発およびデプロイ」PDF の『サーバーへのデプロイの準備』の説明に従って必要なファイルをエクスポートします。次に、それらのファイルを実動システムでデプロイします。詳しくは、WebSphere Integration Developer および WebSphere Application Server for z/OS のインフォメーション・センターを参照してください。

1. モジュールおよびその他のファイルを実動サーバーにコピーします。

   アプリケーションに必要なモジュールおよびリソース (EAR、JAR、RAR、および WAR ファイル) が、実稼働環境に移動します。
2. serviceDeploy コマンドを実行して、インストール可能な EAR ファイルを作成します。

   このステップでは、アプリケーションを実稼働にインストールする準備として、サーバーにモジュールを定義します。
   a. デプロイするモジュールを含む JAR ファイルを見つけ出します。
   b. 前のステップで見つかった JAR ファイルを入力として使用して、 コマンドを実行します。
3. ステップ 2 で作成した EAR ファイルをインストールします。 アプリケーションのインストール方法は、アプリケーションをスタンドアロン・サーバーにインストールするか、セル内のサーバーにインストールするかによって異なります。

   注: 管理コンソールを使用しても、スクリプトを使用しても、アプリケーションをインストールすることができます。追加情報については、WebSphere Application Server インフォメーション・センターを参照してください。
4. 構成を保管します。 これで、モジュールがアプリケーションとしてインストールされました。
5. アプリケーションを開始します。

これで、アプリケーションがアクティブになり、モジュールを通して処理が行われるようになります。

アプリケーションをモニターし、サーバーが要求を正しく処理していることを確認します。

serviceDeploy を使用したインストール可能な EAR ファイルの作成

アプリケーションを実稼働環境にインストールするには、実働サーバーにコピーされたファイルを取得して、インストール可能な EAR ファイルを作成します。

このタスクを開始する前に、サーバーに対してデプロイしようとしているモジュールとサービスを含む JAR ファイルを用意する必要があります。詳しくは、『サーバーへのデプロイの準備』を参照してください。

serviceDeploy コマンドは、JAR ファイルや、これに依存するそのほかの EAR、JAR、RAR、WAR、および ZIP ファイルを取得して、サーバーにインストールできる EAR ファイルを作成します。

1. デプロイするモジュールを含む JAR ファイルを見つけ出します。
2. 前のステップで見つかった JAR ファイルを入力として使用して、 コマンドを実行します。

   このステップでは、EAR ファイルが作成されます。

   注: 管理コンソールで以下のステップを実行します。
3. サーバーの管理コンソールでインストールする EAR ファイルを選択します。
4. 「**保管**」をクリックして、EAR ファイルをインストールします。

ANT タスクを使用したアプリケーションのデプロイ

このトピックでは、ANT タスクを使用して、WebSphere Process Server に対するアプリケーションのデプロイメントを自動化する方法について説明します。 ANT タスクを使用すると、複数のアプリケーションのデプロイメントを定義し、サーバー上でこれらのアプリケーションのデプロイメントを無人で実行することができます。

このタスクでは、以下が前提となります。
- デプロイしようとしているアプリケーションは、すでに開発およびテスト済みである。
- アプリケーションのインストール先が、同じ 1 つまたは複数のサーバーである。
- ANT タスクについての知識がある。
- デプロイメント・プロセスを理解している。

アプリケーションの開発およびテストについての詳細は、WebSphere Integration Developer インフォメーション・センターにあります。

WebSphere Application Server for z/OS インフォメーション・センターのリファレンス部分には、アプリケーション・プログラミング・インターフェースに関するセクションがあります。ANT タスクについては、パッケージ com.ibm.websphere.ant.tasks に説明があります。このトピックに関連するタスクとして、ServiceDeploy および InstallApplicationがあります。

複数のアプリケーションを並行インストールする必要がある場合、デプロイメントの前に ANT タスクを開発します。その後 ANT タスクは、プロセスへのユーザーの介入なしにアプリケーションをサーバーにデプロイし、インストールすることができます。

1. デプロイするアプリケーションを特定します。
2. 各アプリケーションごとに JAR ファイルを作成します。
3. JAR ファイルをターゲット・サーバーにコピーします。

4. ServiceDeploy コマンドを実行する ANT タスクを作成し、サーバーごとに EAR ファイルを作成します。
5. 該当するサーバー上で、ステップ 4 で作成した各 EAR ファイルごとに、InstallApplication コマンドを実行する ANT タスクを作成します。
6. ServiceDeploy ANT タスクを実行し、アプリケーション用の EAR ファイルを作成します。
7. InstallApplication ANT タスクを実行し、ステップ 6 で作成した EAR ファイルをインストールします。

アプリケーションは、ターゲット・サーバー上に正しくデプロイされます。

アプリケーションの無人デプロイの例

以下の例は、ファイル myBuildScript.xml に入っている ANT タスクを示しています。

```
<?xml version="1.0">

<project name="OwnTaskExample" default="main" basedir=".">
 <taskdef name="servicedeploy"
   classname="com.ibm.websphere.ant.tasks.ServiceDeployTask" />
 <target name="main" depends="main2">
  <servicedeploy scaModule="c:/synctest/SyncTargetJAR"
   ignoreErrors="true"
   outputApplication="c:/synctest/SyncTargetEAREAR"
   workingDirectory="c:/synctest"
   noJ2eeDeploy="true"
   cleanStagingModules="true"/>
 </target>
</project>
```

以下のステートメントは、ANT タスクを呼び出す方法を示しています。

```
${WAS}/bin/ws_ant -f myBuildScript.xml
```

ヒント: このファイルにさらに別のプロジェクト・ステートメントを追加して、複数のアプリケーションを無人デプロイすることができます。

管理コンソールを使用して、新しくインストールしたアプリケーション・サーバーが始動され、ワークフローを正しく処理していることを確認してください。

第 4 章 ビジネス・プロセスおよびヒューマン・タスク・アプリケーションのインストール

ビジネス・プロセスまたはヒューマン・タスク、あるいはこの両方を含む Service Component Architecture (SCA) Enterprise JavaBeans™ (EJB) モジュールをデプロイメント・ターゲットに配布することができます。 サーバーまたはクラスターをデプロイメント・ターゲットとすることができます。

アプリケーションのインストール先とするアプリケーション・サーバーまたはクラスターごとに、ビジネス・プロセス・コンテナーまたはタスク・コンテナーがインストールされ、構成されていることを確認してください。

ビジネス・プロセスまたはヒューマン・タスク・アプリケーションをインストールする前に、以下の条件をすべて満たしていることを確認してください。

- アプリケーションのインストール先とするサーバーが稼動している。
- 各クラスターで、プロセスまたはタスクを使用して Enterprise JavaBeans モジュールをインストールする対象のサーバーが少なくとも 1 台実行されている。

ビジネス・プロセスおよびタスク・アプリケーションを、管理コンソールやコマンド行から、または管理スクリプトを実行してインストールすることができます。管理スクリプトを実行してビジネス・プロセス・アプリケーションまたはヒューマン・タスク・アプリケーションをインストールする場合、サーバー接続が必要です。 `-conntype NONE` オプションをインストール・オプションとして使用しないでください。

1. アプリケーションをクラスター上にインストールする場合、アプリケーションがクラスターにちなんで名前を付けられたデータ・ソースを使用していることを確認します。

   例えば、アプリケーションがデフォルト・データ・ソース `BPEDB` を使用して生成された場合、アプリケーションのデータ・ソースを BPEDB_*cluster_name* に変更します。ここで、*cluster_name* は、アプリケーションのインストール先のクラスターの名前です。
2. アプリケーションをインストールします。

すべてのビジネス・プロセス・テンプレートおよびヒューマン・タスク・テンプレートが、開始状態になります。これらのテンプレートから、プロセス・インスタンスとタスク・インスタンスを作成できます。

プロセス・インスタンスまたはタスク・インスタンスを作成するには、アプリケーションを始動する必要があります。

モデルのデプロイ

WebSphere Integration Developer またはサービス・デプロイがプロセスのデプロイメント・コードを生成するとき、プロセスまたはタスク・モデルの構成要素は、さまざまな Java 2 Enterprise Edition (J2EE) 構成要素および成果物にマップされます。すべてのデプロイメント・コードは、エンタープライズ・アプリケーション (EAR) ファイルにパッケージ化されます。デプロイするモデルの新規バージョンごとに、新しいエンタープライズ・アプリケーションにパッケージ化する必要があります。

ビジネス・プロセス・モデルまたはヒューマン・タスク・モデルの J2EE 構成要素で構成されるエンタープライズ・アプリケーションをインストールすると、モデルの構成要素が必要に応じて、*プロセス・テンプレート* または*タスク・テンプレート*として Business Process Choreographer データベースに格納されます。データベース・システムが稼働していない場合、またはデータベース・システムにアクセスできない場合、デプロイは失敗します。デフォルトでは、新しくインストールされたテンプレートは、開始済み状態となります。ただし、新しくインストールされたエンタープライズ・アプリケーションの場合は、停止状態になります。インストール済みのエンタープライズ・アプリケーションは、個々に開始したり停止したりすることができます。

新バージョンのプロセス・テンプレートまたはタスク・テンプレートの名前は同じですが、有効開始日属性が異なります。異なるエンタープライズ・アプリケーションそれぞれに、多数のバージョンのプロセス・テンプレートやタスク・テンプレートをデプロイできます。ただし、1 つのプロセスの 2 つのバージョンが同じ有効開始日を持つことはできません。1 つのプロセスの異なるバージョンをインストールする場合は、バージョンごとに異なる有効開始日を指定してください。データベースには、異なるプロセス・バージョンのすべてが格納されます。

有効開始日を指定しない場合、日付は次のように決定されます。

- ヒューマン・タスクの場合、有効開始日はアプリケーションがインストールされた日付です。
- ビジネス・プロセスの場合、有効開始日はプロセスがモデル化された日付です。

対話によるビジネス・プロセス・アプリケーションのデプロイ

アプリケーションは、wsadmin ツールと installInteractive スクリプトを使用して、実行時に対話式でインストールできます。このスクリプトにより、管理コンソールを使用してアプリケーションをインストールする場合には変更することのできない設定を変更できます。

ビジネス・プロセス・アプリケーションを対話式にインストールするには、次のステップを実行します。

1. wsadmin ツールを開始します。

   profile_root/bin ディレクトリーで、`wsadmin` と入力します。
2. アプリケーションをインストールします。

   wsadmin コマンド行プロンプトで、次のコマンドを入力します。

   ```
   $AdminApp installInteractive application.ear
   ```

ここで、*application.ear* は、ご使用のプロセス・アプリケーションを含むエンタープライズ・アーカイブ・ファイルの修飾名です。 一連のタスクにおいてプロンプトが出されるので、その時にアプリケーションの値を変更できます。

3. 構成の変更を保管します。

   wsadmin コマンド行プロンプトで、次のコマンドを入力します。

   ```
   $AdminConfig save
   ```

   変更を保管し、マスター構成リポジトリーへの更新を転送する必要があります。スクリプト・プロセスが終了し、変更を保存していなければ、変更は廃棄されます。

プロセス・アプリケーションのデータ・ソースと設定参照の設定値の構成

特定のデータベース・インフラストラクチャーで SQL ステートメントを実行するプロセス・アプリケーションは、構成する必要があります。これらの SQL ステートメントは、情報サービス・アクティビティーから発生したり、プロセスのインストールまたはインスタンスの開始時に実行するステートメントであったりします。

このアプリケーションのインストール時には、次のタイプのデータ・ソースを指定できます。

- プロセス・インストール時に SQL ステートメントを実行するデータ・ソース
- プロセス・インスタンスの開始時に SQL ステートメントを実行するデータ・ソース
- SQL 断片アクティビティーを実行するデータ・ソース

SQL 断片アクティビティーを実行するために必要なデータ・ソースは、tDataSource タイプの BPEL 変数で定義されます。SQL 断片アクティビティーで必要とされるデータベース・スキーマ名とテーブル名は、tSetReference タイプの BPEL 変数で定義されます。これらの両方の変数の初期値は、構成することができます。

wsadmin ツールを使用してデータ・ソースを指定できます。

1. wsadmin ツールを使用して、プロセス・アプリケーションを対話式にインストールします。
2. データ・ソースと設定参照を更新するタスクになるまで、タスクをステップスルーします。

   ご使用の環境でこれらの設定を構成します。次の例に、これらのタスクごとに変更可能な設定を示します。

3. 変更を保管します。

例: wsadmin ツールを使用したデータ・ソースと設定参照の更新

「**Updating data source**」タスクでは、プロセスのインストール時またはプロセスの開始時に使用される初期変数値およびステートメントのデータ・ソース値を変更できます。「**Updating set references**」タスクでは、データベース・スキーマとテーブル名に関連した設定を構成できます。

```
Task [24]: Updating data sources

//Change data source values for initial variable values at process start

Process name:  Test
// Name of the process template
Process start or installation time:  Process start
// Indicates whether the specified value is evaluated
//at process startup or process installation
Statement or variable:  Variable
// Indicates that a data source variable is to be changed
Data source name:  MyDataSource
// Name of the variable
JNDI name:[jdbc/sample]:jdbc/newName
// Sets the JNDI name to jdbc/newName
Task [25]: Updating set references

// Change set reference values that are used as initial values for BPEL variables

Process name:  Test
// Name of the process template
Variable:  SetRef
// The BPEL variable name
JNDI name:[jdbc/sample]:jdbc/newName
// Sets the JNDI name of the data source of the set reference to jdbc/newName
Schema name: [IISAMPLE]
// The name of the database schema
Schema prefix: []:
// The schema name prefix.
// This setting applies only if the schema name is generated.
Table name: [SETREFTAB]: NEWTABLE
// Sets the name of the database table to NEWTABLE
Table prefix: []:
// The table name prefix.
// This setting applies only if the prefix name is generated.
```

サーバーが稼動していないクラスターにプロセス・アプリケーションをインストール可能な場合

このトピックでは、サーバーが稼動していないクラスターにアプリケーションをインストールする必要があるような、例外的な状況について説明します。

サーバー上のビジネス・プロセス・アプリケーションのインストール時に、対応するビジネス・プロセス・コンテナーのデータ・ソースの Java Naming and Directory Interface (JNDI) 名を解決する必要があります。そのため、サーバー接続がなければ、アプリケーションをインストールできません。Network Deployment (ND) 環境では、このサーバーはデプロイメント・マネージャーです。

撤廃された制約事項

ND 環境でクラスターにビジネス・プロセス・アプリケーションをインストールする場合、以下の条件が真である場合は、クラスター内のサーバーが稼動している必要はありません。

- 必要なデータ・ソースがセル・レベルで定義されている。
- プロセス・アプリケーションがヒューマン・タスクを指定していない。

ヒューマン・タスクを持たないプロセス・アプリケーションでは、アプリケーション・サーバーのネーム・スペースでの検索操作が前に失敗した場合、データ・ソー

ス検索操作はデプロイメント・マネージャーのネーム・スペース内で実行されます。アプリケーションが正常にインストールされた場合は、アプリケーション・サーバー・ネーム・スペース内部のデータ・ソース検索操作の失敗を示す SystemOut.log ファイル内のエラー・メッセージは無視してください。

機能する場合

- デプロイメント・マネージャー・ネーム・スペース内部の検索操作は、データ・ソース JNDI 名がセル・レベルで定義されている場合のみ、正常に実行されます。
- ウィザードを使用して、スタンドアロン・サーバー上のビジネス・プロセス・コンテナーまたはヒューマン・タスク・コンテナーを構成した場合、データ・ソースはサーバー・レベルで定義されます。アプリケーション・サーバーのインストール済み環境の ProcessChoreographer/config ディレクトリーで提供されている構成スクリプト bpeconfig.jacl を使用した場合にも、同じことが当てはまります。この場合、手動でデータ・ソースをセル・レベルで定義し、ビジネス・プロセス・コンテナーのインストール時にこのデータ・ソースを使用する必要があります。
- クラスター・メンバーでウィザードを使用してビジネス・プロセス・コンテナーを構成した場合、データ・ソースは自動的にセル・レベルで定義されます。JNDI 名は、クラスター名によってスコープが決まります。アプリケーション・サーバーのインストール済み環境の ProcessChoreographer/config ディレクトリーで提供されている構成スクリプト bpeconfig.jacl を使用した場合にも、同じことが当てはまります。この場合、何も手動で変更する必要はありません。

機能しない場合

ヒューマン・タスクを持つプロセス・アプリケーションでは、追加の JNDI 名検索操作によって、スタッフ・プラグイン・プロバイダーを見つける必要があります。したがって、このようなアプリケーションのインストールを確実に正常に実行するため、稼動しているサーバーがクラスターに含まれるようにしてください。

スコープによる副次作用

名前検索の副次作用としては、アプリケーション・サーバーが稼動しておらず、データ・ソースが、サーバーまたはノード・レベルでもセル・レベルのデータ・ソースと同じ名前で定義されている場合、セル・レベルのデータ・ソースが優先されます。これは、結果的に、実行時に使用するものとは異なるデータ・ソースをデプロイメント時に使用する場合があることを意味します。

重要: 名前の競合を回避してください。手動により、セル・レベルでデータ・ソースを定義する場合、クラスター名またはサーバー名、およびノード名のスコープが追加された JNDI 名 (例えば jdbc/BPEDB_*cluster_name*) を使用します。

管理コンソールを使用した、ビジネス・プロセスおよびヒューマン・タスク・アプリケーションのアンインストール

ビジネス・プロセスまたはヒューマン・タスクが含まれるエンタープライズ・アプリケーションをアンインストールするには、以下のアクションを実行します。

1. アプリケーションのすべてのプロセスおよびタスクのテンプレートを停止します。

   このアクションにより、プロセスおよびタスクのインスタンスの作成が回避されます。

   a. 管理コンソールのナビゲーション・ペインで、「**アプリケーション**」 → 「**エンタープライズ・アプリケーション**」をクリックします。
   b. 停止するアプリケーションを選択します。
   c. 「関連項目」の下で、「**EJB モジュール**」をクリックして、Enterprise JavaBeans モジュールを選択します。 複数の EJB モジュールがある場合は、ビジネス・プロセスまたはヒューマン・タスクが含まれる Service Component Architecture (SCA) モジュールに対応する EJB モジュールを選択してください。 SCA モジュール名の後に EJB を付けると、対応する EJB モジュールを見つけることができます。例えば、SCA モジュールの名前が `TestProcess` であれば、 EJB モジュールは `TestProcessEJB.jar` になります。
   d. 「追加プロパティー」の下で、「**ビジネス・プロセス**」か「**ヒューマン・タスク**」、または両方を必要に応じてクリックします。
   e. 適切なチェック・ボックスをクリックして、プロセスおよびタスクのテンプレートをすべて選択します。
   f. 「**停止**」をクリックします。

   ビジネス・プロセス・テンプレートまたはタスク・テンプレートが含まれるすべての EJB モジュールで、このステップを繰り返します。
2. データベース、クラスターごとに少なくとも 1 つのアプリケーション・サーバー、アプリケーションがデプロイされているスタンドアロン・サーバーが動作していることを確認してください。

   Network Deployment (ND) 環境では、デプロイメント・マネージャー、すべての ND 管理対象スタンドアロン・アプリケーション・サーバー、および少なくとも 1 台のアプリケーション・サーバーが、アプリケーションがインストールされているそれぞれのクラスターごとに実行されている必要があります。
3. 実行中のプロセス・インスタンスまたはタスク・インスタンスがないこと、および終了状態で autoDelete フラグが false に設定されていないことを確認します。

   必要であれば、管理者は、Business Process Choreographer Explorer を使用して、残存するプロセス・インスタンスまたはタスク・インスタンスを削除することができます。
4. アプリケーションを停止してアンインストールするには、以下のようにします。

   a. 管理コンソールのナビゲーション・ペインで、「**アプリケーション**」 → 「**エンタープライズ・アプリケーション**」をクリックします。
   b. アンインストールするアプリケーションを選択し、「**停止**」をクリックします。

      アプリケーションでプロセス・インスタンスまたはタスク・インスタンスがまだ存在する場合は、このステップは失敗します。

c. アンインストールするアプリケーションを再度選択し、「**アンインストール**」をクリックします。
d. 「**保管**」をクリックして、変更を保管します。

アプリケーションはアンインストールされます。

管理コマンドを使用した、ビジネス・プロセスおよびヒューマン・タスク・アプリケーションのアンインストール

管理コマンドには、ビジネス・プロセスまたはヒューマン・タスクを含むアプリケーションをアンインストールするための、管理コンソールの代替方法が用意されています。

グローバル・セキュリティーが使用可能な場合は、ユーザー ID にオペレーター権限があることを確認します。

管理クライアントが接続しているサーバー・プロセスが動作していることを確認します。

- ND 環境では、サーバー・プロセスはデプロイメント・マネージャーです。
- スタンドアロン環境では、サーバー・プロセスはアプリケーション・サーバーです。

管理クライアントが自動的にサーバー・プロセスに接続できるよう、 `-conntype NONE` オプションをコマンド・オプションとして使用しないでください。

次の手順で、bpcTemplates.jacl スクリプトを使用して、ビジネス・プロセス・テンプレートまたはヒューマン・タスク・テンプレートを含むアプリケーションをアンインストールする方法を示します。また、アプリケーションをアンインストールするには、そのアプリケーションに属しているテンプレートを停止する必要があります。 bpcTemplates.jacl スクリプトを使用して、1 つの手順でテンプレートを停止およびアンインストールできます。

アプリケーションをアンインストールする前に、Business Process Choreographer Explorer を使用するなどして、アプリケーション内のテンプレートに関連付けられたプロセス・インスタンスまたはタスク・インスタンスを削除できます。また、bpcTemplates.jacl スクリプトで **-force** オプションを使用し、テンプレートに関連付けられたインスタンスの削除、テンプレートの停止、およびそれらのテンプレートのアンインストールを 1 度のステップで実行できます。

注意:
このオプションを指定すると、プロセス・インスタンスとタスク・インスタンスのすべてのデータが削除されるので、慎重に使用してください。

1. Business Process Choreographer サンプル・ディレクトリーに移動します。 次のように入力します。

   `cd install_root/ProcessChoreographer/admin`
2. テンプレートを停止して、対応するアプリケーションをアンインストールします。

```
install_root/bin/wsadmin -f bpcTemplates.jacl
                            [-user user_name]
                            [-password user password]
                            -uninstall application_name
                            [-force]
```

各部の意味は、次のとおりです。

user_name
グローバル・セキュリティーが使用可能な場合は、認証用のユーザー ID を提供します。

user_password
グローバル・セキュリティーが使用可能な場合は、認証用のユーザー・パスワードを提供します。

application_name
グローバル・セキュリティーが使用可能な場合は、認証用のユーザー・パスワードを提供します。

-force
アプリケーションがアンインストールされる前に、実行中のすべてのインスタンスを停止および削除させます。

アプリケーションはアンインストールされます。

第 5 章 アプリケーションおよび組み込み WebSphere アダプターのインストール

アプリケーションに WebSphere アダプターを組み込んで開発する場合、そのアダプターはアプリケーションと共にデプロイされます。アダプターを別途インストールする必要はありません。アプリケーションを組み込みアダプターと共にインストールする手順を以下に示します。

アプリケーションを組み込みの WebSphere アダプターを使用して開発する場合にのみ、この作業を行ってください。

1. アプリケーションとその中のリソース・アダプター・アーカイブ (RAR) モジュールをアセンブルします。『アプリケーションのアセンブル』を参照してください。
2. 『新規アプリケーションのインストール』のステップに従って、アプリケーションをインストールします。『モジュールのサーバーへのマップ』ステップで、RAR ファイルごとにターゲット・サーバーまたはクラスターを指定します。RAR モジュールに定義されているリソース・アダプターを使用するその他のモジュールはすべて、必ず同じターゲットにマップしてください。また、このアプリケーションに対する要求のルーターとして機能するターゲットとして、Web サーバーを指定します。各 Web サーバーのプラグイン構成ファイル (plugin-cfg.xml) は、Web サーバーを経由するアプリケーションに基づいて生成されます。

   注: サーバーに RAR ファイルをインストールする場合、WebSphere Process Server はコネクター・モジュールのマニフェスト (MANIFEST.MF) を探します。最初、RAR ファイルの connectorModule.jar ファイルで探し、_connectorModule.jar ファイルからマニフェストをロードします。クラスパス・エントリーが connectorModule.jar ファイルから得たマニフェストにある場合は、 RAR はそのクラスパスを使用します。インストール済みコネクター・モジュールが必要なクラスおよびリソースを見つけることを確認するには、コンソールを使用して、RAR の「クラスパス」設定を確認します。詳しくは、リソース・アダプターの設定および WebSphere リレーショナル・リソース・アダプターの設定を参照してください。
3. 変更を保管します。 「**終了**」>「**保管**」をクリックします。
4. 新規にインストールされたアプリケーションの接続ファクトリーを作成します。
   a. 管理コンソールを開きます。
   b. 新しくインストールされたアプリケーションを選択します。 「**アプリケーション**」>「**エンタープライズ・アプリケーション**」>「***application name***」をクリックします。
   c. このページの「関連項目」セクションの「**コネクター・モジュール**」をクリックします。
   d. RAR ファイルを選択します。 「***filename.rar***」をクリックします。

e. このページの「追加プロパティー」セクションの「**リソース・アダプター**」をクリックします。

f. このページの「追加プロパティー」セクションの「**J2C 接続ファクトリー**」をクリックします。

g. 既存の接続ファクトリーを更新する場合はそれをクリックし、新規に作成する場合は「**新規**」をクリックします。

注: WebSphere アダプターが EIS インポートまたは EIS エクスポートを使用して構成されている場合は、ConnectionFactory または ActivationSpec が存在し、これらを更新することができます。

Linux および UNIX: ネイティブ・パス・エレメントが含まれているアダプターをインストールする場合は、次のことを考慮してください。複数のネイティブ・パス・エレメントがあり、ネイティブ・ライブラリーの 1 つ (ネイティブ・ライブラリー A) が別のライブラリー (ネイティブ・ライブラリー B) に依存している場合、ネイティブ・ライブラリー B をシステム・ディレクトリーにコピーする必要があります。ほとんどの UNIX® システムでは制限があるため、ネイティブ・ライブラリーをロードしようとする際に現行ディレクトリーを調べません。

接続ファクトリーを作成および保管した後、適宜に、アプリケーションの各種モジュールで定義されているリソース参照を変更し、接続ファクトリーの Java Naming and Directory Interface (JNDI) 名を指定します。

注: 所定のネイティブ・ライブラリーは、 Java 仮想マシン (JVM) のインスタンスごとに 1 度だけロードできます。アプリケーションごとに独自のクラス・ローダーがあるため、組み込み RAR ファイルを持つ異なるアプリケーション同士が同じネイティブ・ライブラリーを使用することはできません。2 番目のアプリケーションは、ライブラリーをロードしようとすると例外を受け取ります。

アプリケーション・サーバーにデプロイされているアプリケーションが、ネイティブ・パス・エレメントを含む組み込み RAR ファイルを使用する場合、未解決のトランザクションがない状態で、アプリケーション・サーバーが完全にシャットダウンしたことを必ず確認する必要があります。アプリケーション・サーバーが完全にシャットダウンしない場合は、サーバーの再始動時にリカバリーを実行し、必要な RAR ファイルおよびネイティブ・ライブラリーをロードします。リカバリーが完了したら、アプリケーションに関する作業は行わないようにしてください。サーバーをシャットダウンして、再始動します。この再始動時にアプリケーション・サーバーによってこれ以上リカバリーが行われることはなく、通常のアプリケーション処理を続行できます。

WebSphere アダプター

WebSphere アダプター (または JCA アダプター、J2C アダプター) は、Java アプリケーションがエンタープライズ情報システム (EIS) への接続に使用するシステム・レベルのソフトウェア・ドライバーです。WebSphere アダプターは、JCA 仕様のバージョン 1.5 に準拠しています。

WebSphere アダプターは、アプリケーション・サーバーにプラグインし、EIS、アプリケーション・サーバー、およびエンタープライズ・アプリケーション間の接続を提供します。

アプリケーション・サーバーのベンダーは、J2EE コネクター・アーキテクチャー (JCA) をサポートするように一度そのシステムを拡張すれば、その後は複数の EIS へのシームレスな接続を保証されます。同様に、EIS ベンダーは、コネクター・アーキテクチャーをサポートするアプリケーション・サーバーにプラグインするための機能を備えた 1 つの標準の WebSphere アダプターを提供します。

WebSphere Process Server では、WebSphere リレーショナル・リソース・アダプター (RRA) のインプリメンテーションを提供します。この WebSphere アダプターは、JDBC 呼び出しによってデータ・アクセスを実現し、データベースに動的にアクセスします。接続管理は、JCA 接続管理アーキテクチャーに基づいて行われます。これは、接続プール、トランザクション、およびセキュリティー・サポートを提供します。 WebSphere Process Server バージョン 6.0 では、JCA バージョン 1.5 をサポートします。

コンテナー管理パーシスタンス (CMP) Bean のデータ・アクセスは、 WebSphere Persistence Manager から間接的に管理されます。JCA 仕様により、パーシスタンス・マネージャーは、バックエンド・ストアを特に認識せずに、WebSphere アダプターへのデータ・アクセスを代行することができます。リレーショナル・データベース・アクセスの場合、パーシスタンス・マネージャーは、リレーショナル・リソース・アダプターを使用して、データベースのデータにアクセスします。 JDBC API でサポートされるデータベース・プラットフォームに関しては、 WebSphere Process Server の前提条件を記載した Web サイトを参照してください。

IBM® は、WebSphere Process Server パッケージとは別に、多数のエンタープライズ・システム用のリソース・アダプターを用意しています。これには顧客情報管理システム (CICS®)、Host On-Demand (HOD)、情報管理システム (IMS™)、および SAP (Systems, Applications, and Products) R/3 など (ただし、これらに限定されない) が含まれます。

WebSphere Process Server では、WebSphere アダプターとのインターフェースに、EIS インポートおよび EIS エクスポートを使用します。これに代わる方法として、Rational® Application Developer などのツールを使用して、EJB セッション Bean やサービスを開発することにより、WebSphere アダプターを使用するアプリケーションを作成することもできます。セッション Bean は、javax.resource.cci インターフェースを使用して、WebSphere アダプターを介したエンタープライズ情報システムとの通信を行います。

WebSphere アダプターのデプロイメントに関する考慮事項

WebSphere アダプターのデプロイメントには、スコープに関する特定のオプションが必要です。

WebSphere アダプターは、管理コンソールを使用して、次の 2 とおりの方法でデプロイすることができます。

- スタンドアロン: アダプターがノード・レベルでインストールされ、特定のアプリケーションに関連付けられない。

注: スタンドアロンの WebSphere アダプターのデプロイメントは、WebSphere Process Server v6.0 ではサポートされません。

- 組み込み: アダプターがアプリケーションの一部であり、アプリケーションをデプロイすると、アダプターも一緒にデプロイされる。

組み込み WebSphere アダプターの場合:

- RAR ファイルを、(EIS インポートまたはエクスポートを使用する) SCA モジュール内でアプリケーション・スコープを持つように設定できます。
- RAR ファイルを、SCA 以外のモジュール内でアプリケーション・スコープを持つように設定できます。EIS インポートまたはエクスポートを含むアプリケーション自体は、別個の SCA モジュールです。

スタンドアロンの WebSphere アダプターは、インストールしないでください。

注: 管理コンソールによって、スタンドアロンの WebSphere アダプターのインストールが妨げられることはありませんが、これはインストールすべきではありません。WebSphere アダプターは、アプリケーションに組み込んでください。

組み込み WebSphere アダプターのみ、WebSphere Process Server のデプロイメントに適しています。また、組み込みの WebSphere アダプターのデプロイメントは、SCA モジュール内でアプリケーション・スコープを持つように設定された RAR ファイルでしかサポートされません。SCA モジュール以外でのデプロイメントはサポートされません。

スタンドアロンの WebSphere アダプターのインストール

スタンドアロンの WebSphere アダプターを使用する必要がある場合は、この説明に従ってインストールしてください。代わりに、関連したアプリケーションのインストールの一部として自動的にインストールされる、組み込みアダプターを使用することができます。

注: WebSphere アダプターは、アプリケーションに組み込んでください。スタンドアロンの WebSphere アダプターは、WebSphere Process Server v6.0 ではサポートされません。ここで示す説明は、参考のため記載しています。

アダプターをインストールする前に、データベースを構成する必要があります。

この操作を実行するには、管理コンソールへのアクセス権限を持ち、管理コンソールに必要なセキュリティーのロールに属していることが必要です。

1. 「RAR ファイルのインストール (Install RAR file)」ダイアログ・ウィンドウを開きます。 管理コンソールで、以下の手順を実行します。
   a. 「**リソース**」を展開します。
   b. 「**リソース・アダプター**」をクリックします。
   c. このリソース・アダプターを定義する有効範囲を選択します。この有効範囲は、ご使用の接続ファクトリーの有効範囲になります。セル、ノード、クラスター、またはサーバーを選択することができます。
   d. 「**Install RAR**」をクリックします。

ウィンドウが開きます。ここで JCA コネクターをインストールし、JCA コネクター用に WebSphere アダプターを作成することができます。「**新規**」ボタンを使用することもできますが、「**新規**」ボタンによって作成されるのは、新規リソース・アダプターだけです (JCA コネクターがシステムにインストール済みである必要があります)。

注: このダイアログを使用して RAR ファイルをインストールすると、「リソース・アダプター」ページで定義する有効範囲は、RAR ファイルのインストール場所では無効になります。 RAR ファイルは、ノード・レベルでのみインストールできます。ファイルがインストールされているノードは、「Install RAR」ページ上の有効範囲によって決定されます。 (「リソース・アダプター」ページで設定する有効範囲により、新規リソース・アダプターの有効範囲が決まります。新規リソース・アダプターは、サーバー、ノード、またはセルの各レベルでインストールすることができます。)

2. RAR ファイルをインストールします。

   ダイアログから、WebSphere アダプターを以下のようにインストールします。

   a. JCA コネクターのロケーションをブラウズします。 RAR ファイルがローカル・ワークステーション上にある場合、「**ローカル・パス**」を選択してブラウズし、ファイルを探します。RAR ファイルがネットワーク・サーバー上にある場合は、「**サーバー・パス**」を選択し、ファイルの完全修飾パスを指定します。

   b. 「**次へ**」をクリックします。

   c. 「一般プロパティー」の下で、リソース・アダプター名およびその他の必要なすべてのプロパティーを入力します。ネイティブ・パス・エレメントが含まれている J2C リソース・アダプターをインストールする場合は、次のことを考慮してください。複数のネイティブ・パス・エレメントがあり、ネイティブ・ライブラリーの 1 つ (ネイティブ・ライブラリー A) が別のライブラリー (ネイティブ・ライブラリー B) に依存している場合、ネイティブ・ライブラリー B をシステム・ディレクトリーにコピーする必要があります。ほとんどの UNIX システムでは制限があるため、ネイティブ・ライブラリーをロードしようとする際に現行ディレクトリーを調べません。

   d. 「**OK**」をクリックします。

クラスターのメンバーとしての WebSphere アダプター・アプリケーション

WebSphere アダプター・モジュール・アプリケーションは、一定の条件下で、クラスターのメンバーとして複製することができます。

WebSphere アダプター・モジュール・アプリケーションには、アダプターを介した情報の流れによって、次の 3 つのタイプがあります。

- EIS エクスポートのみを使用する WebSphere アダプター・アプリケーション: インバウンド・トラフィックのみ。
- EIS インポートのみを使用する WebSphere アダプター・アプリケーション:アウトバウンド・トラフィックのみ。

- EIS インポートとエクスポートの両方を使用する WebSphere アダプター・アプリケーション: 両方向トラフィック。

Network Deployment 環境でのアプリケーションのスケーラビリティーと可用性を実現するために、クラスターが使用されます。

インバウンド・トラフィックまたは両方向のトラフィックが発生する WebSphere アダプター・モジュール・アプリケーションは、クラスターのメンバーとして複製できません。純粋なアウトバウンド・トラフィックのみが発生するアプリケーションを、クラスターのメンバーとして複製することができます。

外部のオペレーティング・システム高可用性 (HA) 管理ソフトウェア・パッケージ (HACMP™、Veritas、Microsoft® Cluster Server など) を使用すると、インバウンドまたは双方向の (すなわち、EIS エクスポートを含む) WebSphere アダプターを使用するアプリケーションでも、Network Deployment 環境での可用性を提供することができます。

クラスター・メンバーとしての WebSphere Business Integration Adapter アプリケーション

WebSphere Business Integration Adapter モジュール・アプリケーションは、一定の条件下で、クラスターのメンバーとして複製することができます。

WebSphere Business Integration Adapter モジュール・アプリケーションには、アダプターを介した情報の流れによって、次の 3 つのタイプがあります。

- EIS エクスポートのみを使用する WebSphere Business Integration Adapter アプリケーション: インバウンド・トラフィックのみ。
- EIS インポートのみを使用する WebSphere Business Integration Adapter アプリケーション: アウトバウンド・トラフィックのみ。
- EIS インポートとエクスポートの両方を使用する WebSphere Business Integration Adapter アプリケーション: 両方向トラフィック。

Network Deployment 環境でのアプリケーションのスケーラビリティーと可用性を実現するために、クラスターが使用されます。

インバウンド・トラフィックまたは両方向のトラフィックが発生する WebSphere Business Integration Adapter モジュール・アプリケーションは、クラスターのメンバーとして複製できません。純粋なアウトバウンド・トラフィックのみが発生するアプリケーションを、クラスターのメンバーとして複製することができます。

外部のオペレーティング・システム高可用性 (HA) 管理ソフトウェア・パッケージ (HACMP、Veritas、Microsoft Cluster Server など) を使用すると、インバウンドまたは両方向の (すなわち、EIS エクスポートを含む) WebSphere Business Integration Adapter を使用するアプリケーションでも、Network Deployment 環境での可用性を提供することができます。

第 6 章 EIS アプリケーションのインストール

EIS アプリケーション・モジュールは、J2EE プラットフォームにデプロイできます。このデプロイメントでは、EAR ファイルとしてパッケージ化されたアプリケーションがサーバーにデプロイされます。J2EE のすべての成果物およびリソースが作成され、アプリケーションが構成されて、実行準備ができます。

J2EE プラットフォームへのデプロイでは、以下の J2EE 成果物およびリソースが作成されます。

表 1. バインディングから J2EE 成果物への対応

SCA モジュールでのバインディング	**生成される J2EE 成果物**	**作成される J2EE リソース**
EIS インポート	モジュールのセッション EJB で生成されるリソース参照。	ConnectionFactory
EIS エクスポート	リソース・アダプターによってサポートされるリスナー・インターフェースに応じて生成またはデプロイされるメッセージ駆動型 Bean。	ActivationSpec
JMS インポート	ランタイムにより提供されるメッセージ駆動型 Bean (MDB) がデプロイされ、モジュールのセッション EJB でリソース参照が生成される。MDB はインポートに受信宛先が指定されている場合にのみ作成される。	• ConnectionFactory • ActivationSpec • Destination
JMS エクスポート	ランタイムにより提供されるメッセージ駆動型 Bean がデプロイされ、モジュールのセッション EJB でリソース参照が生成される。	• ActivationSpec • ConnectionFactory • Destination

インポートまたはエクスポートで、ConnectionFactory のようなリソースを定義している場合、リソース参照はモジュールのステートレス・セッション EJB のデプロイメント記述子内に生成されます。また、適切なバインディングが EJB バインディング・ファイル内に生成されます。リソース参照がバインドされる名前は、モジュール名とインポート名を基にして、ターゲット属性の値 (値が存在する場合)、またはリソースに与えられたデフォルトの JNDI 検索名のいずれかになります。

デプロイ時に、実装環境により、モジュールのセッション Bean が検出され、これを使用して、リソースが検索されます。

サーバーへのアプリケーションのデプロイ中に、EIS インストール・タスクにより、バインドされたエレメント・リソースが存在するかどうかの確認が行われます。このリソースが存在しない場合、SCDL ファイルで 1 つ以上のプロパティーを指定しているときは、EIS インストール・タスクによって、このリソースが作成さ

れ、構成されます。リソースが存在しない場合は、何も処置は行われず、リソースを作成してからアプリケーションを実行するものとみなされます。

JMS インポートが受信宛先を指定してデプロイされた場合、メッセージ駆動型 Bean (MDB) がデプロイされます。これにより、送信された要求に対する応答を listen します。MDB は、JMS メッセージの JMSreplyTo ヘッダー・フィールドにある、要求で送信された Destination に関連付けられます (この Destination で listen します)。MDB は、応答メッセージが到着すると、メッセージの相関 ID を使用して、コールバック Destination に保管されているコールバック情報を取得して、コールバック・オブジェクトを呼び出します。

インストール・タスクでは、インポート・ファイル内の情報から ConnectionFactory と 3 つの宛先が作成されます。これに加えて、ActivationSpec を作成して、ランタイム MDB が受信宛先で応答を listen できるようにします。ActivationSpec のプロパティーは、Destination/ConnectionFactory プロパティーから派生しています。JMS プロバイダーが SIBus リソース・アダプターである場合、JMS の Destination に対応する SIBus の Destination が作成されます。

JMS エクスポートをデプロイする場合、メッセージ駆動型 Bean (MDB) (JMS インポートの場合にデプロイされる MDB とは異なる MDB) がデプロイされます。MDB は、受信宛先で着信要求を listen して、その要求が SCA で処理されるようにディスパッチします。インストール・タスクでは、JMS インポートの場合と同様のリソース・セット、ActivationSpec、応答の送信に使用される ConnectionFactory、および 2 つの宛先が作成されます。これらのリソースのプロパティーはすべて、エクスポート・ファイルに指定されます。JMS プロバイダーが SIBus リソース・アダプターである場合、JMS の Destination に対応する SIBus の Destination が作成されます。

J2SE プラットフォームへの EIS アプリケーション・モジュールのデプロイ

EIS モジュールを J2SE プラットフォームにデプロイすることはできますが、EIS インポートのみがサポートされます。

この作業を開始する前に、 WebSphere 統合開発環境で、JMS インポート・バインディングを使用して、EIS アプリケーション・モジュールを作成する必要があります。

メッセージ・キューを使用して EIS システムに非同期にアクセスしたい場合は、EIS アプリケーション・モジュールに JMS インポート・バインディングを提供します。

J2SE プラットフォームにデプロイするのは、バインディングのインプリメンテーションを管理対象外モードで実行できる場合のみです。 JMS バインディングでは、非同期および JNDI のサポートが必要です。このどちらも、基本の Service Component Architecture または J2SE では提供されません。J2EE Connector Architecture では、管理対象外のインバウンド通信がサポートされないため、EIS エクスポートは除去されます。

EIS インポートを使用した EIS アプリケーション・モジュールを J2SE にデプロイする場合、モジュールの依存関係だけでなく、インポートで使用される WebSphere アダプターを依存関係として、マニフェストまたは SCA がサポートするその他の形式で指定する必要があります。

J2EE プラットフォームへの EIS アプリケーション・モジュールのデプロイ

EIS モジュールを J2EE プラットフォームにデプロイすると、EAR ファイルとしてパッケージ化されたアプリケーションがサーバーにデプロイされます。J2EE のすべての成果物およびリソースが作成され、アプリケーションが構成されて、実行準備ができます。

この作業を開始する前に、 WebSphere 統合開発環境で、JMS インポート・バインディングを使用して、EIS モジュールを作成する必要があります。

J2EE プラットフォームへのデプロイでは、以下の J2EE 成果物およびリソースが作成されます。

表 2. バインディングから J2EE 成果物への対応

SCA モジュールでのバインディング	**生成される J2EE 成果物**	**作成される J2EE リソース**
EIS インポート	モジュールのセッション EJB で生成されるリソース参照。	ConnectionFactory
EIS エクスポート	リソース・アダプターによってサポートされるリスナー・インターフェースに応じて生成またはデプロイされるメッセージ駆動型 Bean。	ActivationSpec
JMS インポート	ランタイムにより提供されるメッセージ駆動型 Bean (MDB) がデプロイされ、モジュールのセッション EJB でリソース参照が生成される。MDB はインポートに受信宛先が指定されている場合にのみ作成される。	• ConnectionFactory • ActivationSpec • Destination
JMS エクスポート	ランタイムにより提供されるメッセージ駆動型 Bean がデプロイされ、モジュールのセッション EJB でリソース参照が生成される。	• ActivationSpec • ConnectionFactory • Destination

インポートまたはエクスポートで、ConnectionFactory のようなリソースを定義している場合、リソース参照はモジュールのステートレス・セッション EJB のデプロイメント記述子内に生成されます。また、適切なバインディングが EJB バインディング・ファイル内に生成されます。リソース参照がバインドされる名前は、モジュール名とインポート名を基にして、ターゲット属性の値 (値が存在する場合)、またはリソースに与えられたデフォルトの JNDI 検索名のいずれかになります。

デプロイ時に、実装環境により、モジュールのセッション Bean が検出され、これを使用して、リソースが検索されます。

サーバーへのアプリケーションのデプロイ中に、EIS インストール・タスクにより、バインドされたエレメント・リソースが存在するかどうかの確認が行われます。このリソースが存在しない場合、SCDL ファイルで 1 つ以上のプロパティーを指定しているときは、EIS インストール・タスクによって、このリソースが作成され、構成されます。リソースが存在しない場合は、何も処置は行われず、リソースを作成してからアプリケーションを実行するものとみなされます。

JMS インポートが受信宛先を指定してデプロイされた場合、メッセージ駆動型 Bean (MDB) がデプロイされます。これにより、送信された要求に対する応答を listen します。MDB は、JMS メッセージの JMSreplyTo ヘッダー・フィールドにある、要求で送信された Destination に関連付けられます (この Destination で listen します)。MDB は、応答メッセージが到着すると、メッセージの相関 ID を使用して、コールバック Destination に保管されているコールバック情報を取得して、コールバック・オブジェクトを呼び出します。

インストール・タスクでは、インポート・ファイル内の情報から ConnectionFactory と 3 つの宛先が作成されます。これに加えて、ActivationSpec を作成して、ランタイム MDB が受信宛先で応答を listen できるようにします。ActivationSpec のプロパティーは、Destination/ConnectionFactory プロパティーから派生しています。JMS プロバイダーが SIBus リソース・アダプターである場合、JMS の Destination に対応する SIBus の Destination が作成されます。

JMS エクスポートをデプロイする場合、メッセージ駆動型 Bean (MDB) (JMS インポートの場合にデプロイされる MDB とは異なる MDB) がデプロイされます。MDB は、受信宛先で着信要求を listen して、その要求が SCA で処理されるようにディスパッチします。インストール・タスクでは、JMS インポートの場合と同様のリソース・セット、ActivationSpec、応答の送信に使用される ConnectionFactory、および 2 つの宛先が作成されます。これらのリソースのプロパティーはすべて、エクスポート・ファイルに指定されます。JMS プロバイダーが SIBus リソース・アダプターである場合、JMS の Destination に対応する SIBus の Destination が作成されます。

第 7 章 失敗したデプロイメントのトラブルシューティング

このトピックでは、アプリケーションのデプロイ時の問題の原因を判別するために行うステップについて説明します。また、参考になるいくつかのソリューションも示されています。

このトピックは、以下の事項を前提としています。
- モジュールのデバッグの基本について理解している。
- モジュールのデプロイ中にロギングおよびトレースがアクティブになっている。

デプロイメントのトラブルシューティングのタスクは、エラーの通知を受け取った後に開始します。失敗したデプロイメントには、アクションをとる前に検査する必要のあるさまざまな症状があります。

1. アプリケーションのインストールが失敗したかどうか判別します。

   SystemOut.log ファイルを調べて、失敗の原因を示すメッセージを探します。アプリケーションをインストールできない理由には、以下のようなものがあります。
   - 同一の Network Deployment セル内の複数のサーバーにアプリケーションをインストールしようとしている。
   - アプリケーションの名前が、アプリケーションをインストールする Network Deployment セル上の既存のモジュールの名前と同じである。
   - EAR ファイル内部の J2EE モジュールを異なるターゲット・サーバーにデプロイしようとしている。

   重要: インストールが失敗し、アプリケーションにサービスが含まれる場合、アプリケーションの再インストールを試みる前に、失敗の前に作成された SIBus 宛先または J2C アクティベーション・スペックを除去する必要があります。これらの成果物を除去する最も簡単な方法は、失敗後に「**保管**」>「**すべて廃棄 (Discard all)**」をクリックする方法です。不注意で変更を保管した場合、SIBus 宛先および J2C アクティベーション・スペックを手動で除去する必要があります (『管理』セクションの『SIBus 宛先の削除』および『J2C アクティベーション・スペックの削除』を参照)。
2. アプリケーションが正しくインストールされている場合は、プログラムが正常に開始したかどうかを確認します。

   アプリケーションが正常に開始していない場合は、サーバーがアプリケーションのリソースを初期化しようとしたときに障害が起きています。

   a. SystemOut.log ファイルを調べて、対処法を指示するメッセージを探します。
   b. アプリケーションで必要なリソースが使用可能か、また、それらのリソースが正常に開始されたかどうかを確認します。

      開始されないリソースがあると、アプリケーションは実行されません。これは、情報が失われるのを防ぐためです。リソースが開始しない理由には次のものがあります。

- 指定されたバインディングが正しくない。
- リソースが正しく構成されていない。
- リソースがリソース・アーカイブ (RAR) ファイルに含まれていない。
- Web リソースが Web サービス・アーカイブ (WAR) ファイルに含まれていない。

c. コンポーネントが欠落していないかどうか判別します。

コンポーネント欠落の原因は、エンタープライズ・アーカイブ (EAR) ファイルが正しく作成されなかったことにあります。モジュールが必要とするすべてのコンポーネントが、Java アーカイブ (JAR) ファイルを作成したテスト・システムの正しいフォルダーに格納されているか確認します。追加情報については、**『モジュールの開発とデプロイ』>『モジュールの準備とインストールの概要』>『サーバーへのデプロイの準備』**を参照してください。

3. アプリケーションで情報が処理されているかどうかを調べます。

実行中のアプリケーションでも、情報の処理に失敗することがあります。この理由は、ステップ 2b (43 ページ) で示した理由と同様です。

a. アプリケーションが、別のアプリケーションに含まれるサービスを使用するかどうかを判別します。 その別のアプリケーションがインストール済みで、正常に開始されていることを確認します。
b. 失敗したアプリケーションが使用する別のアプリケーションに含まれる、各種デバイス用のインポート・バインディングおよびエクスポート・バインディングが正しく構成されていることを確認します。 管理コンソールを使用して、バインディングを調べ、訂正してください。

4. 問題を解決してから、アプリケーションを再始動します。

J2C アクティベーション・スペックの削除

システムは、サービスを含むアプリケーションをインストールする際に、J2C アクティベーション・スペックを作成します。アプリケーションを再インストールする前に、これらの仕様を削除しなければならない場合があります。

アプリケーションのインストールが失敗したために仕様を削除する場合は、Java Naming and Directory Interface (JNDI) 名のモジュールが、インストールに失敗したモジュールの名前と一致することを確認してください。JNDI 名の 2 番目の部分は、宛先をインプリメントしたモジュールの名前です。例えば sca/SimpleBOCrsmA/ActivationSpec では、**SimpleBOCrsmA** がモジュール名です。

サービスを含むアプリケーションのインストール後に不注意で構成を保管した場合、J2C アクティベーション・スペックを必要としなければ、アクティベーション・スペックを削除します。

1. 削除するアクティベーション・スペックを見つけます。

仕様はリソース・アダプター・パネルに含まれています。**「リソース」>「リソース・アダプター (Resource adapters)」**をクリックして、このパネルにナビゲートします。

a. **Platform Messaging Component SPI Resource Adapter** を見つけます。

このアダプターを見つけるには、スタンドアロン・サーバーの**ノード**・スコープまたは Network Deployment 環境の**サーバー**・スコープにいる必要があります。

2. Platform Messaging Component SPI Resource Adapter に関連付けられた J2C アクティベーション・スペックを表示します。

   リソース・アダプター名をクリックすると、関連する仕様が次のパネルに表示されます。

3. 削除するモジュール名に一致する **JNDI 名**の仕様をすべて削除します。
   a. 該当する仕様の横のチェック・ボックスをクリックします。
   b. 「**削除**」をクリックします。

システムが、選択された仕様を表示から除去します。

変更を保管します。

SIBus 宛先の削除

SIBus 宛先とは、アプリケーションに対してサービスを使用可能にする接続です。宛先を除去しなければならない場合もあります。

アプリケーションのインストールが失敗したために宛先を削除する場合は、宛先名のモジュールが、インストールに失敗したモジュールの名前と一致することを確認してください。宛先の 2 番目の部分は、宛先をインプリメントしたモジュールの名前です。例えば sca/SimpleBOCrsmA/component/test/sca/cros/simple/cust/Customer では、**SimpleBOCrsmA** がモジュール名です。

サービスを含むアプリケーションのインストール後に不注意で構成を保管した場合、または SIBus 宛先を必要としなくなった場合、その宛先を削除します。

注: このタスクは、SCA システム・バスからのみ宛先を削除します。また、サービスを含むアプリケーションを再インストールする前にも、アプリケーション・バスからエントリーを除去する必要があります。このインフォメーション・センターの『管理』セクションの『J2C アクティベーション・スペックの削除』を参照してください。

1. 管理コンソールにログインします。
2. SCA システム・バスの宛先を表示します。

   「**サービス統合**」>「**バス**」をクリックしてパネルにナビゲートします。

3. SCA システム・バス宛先を選択します。

   表示の中で、**SCA.SYSTEM.*cellname*.Bus** をクリックします。ここで *cellname* は、削除する宛先を含むモジュールが含まれているセルの名前です。

4. 除去するモジュールに一致するモジュール名を含む宛先を削除します。
   a. 該当する宛先の横のチェック・ボックスをクリックします。
   b. 「**削除**」をクリックします。

パネルには残った宛先のみが表示されます。

これらの宛先を作成したモジュールに関連する J2C アクティベーション・スペック
を削除してください。

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-8711
東京都港区六本木 3-2-12
IBM World Trade Asia Corporation
Intellectual Property Law & Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation577 Airport Blvd., Suite 800
Burlingame, CA 94010
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書はプランニング目的としてのみ記述されています。記述内容は製品が使用可能になる前に変更になる場合があります。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

プログラミング・インターフェース情報

プログラミング・インターフェース情報がある場合、それらはこのプログラムを使用してアプリケーション・ソフトウェアを作成する際に役立つよう提供されています。

一般使用プログラミング・インターフェースにより、お客様はこのプログラム・ツール・サービスを含むアプリケーション・ソフトウェアを書くことができます。

ただし、この情報には、診断、修正、および調整情報が含まれている場合があります。診断、修正、調整情報は、お客様のアプリケーション・ソフトウェアのデバッグ支援のために提供されています。

警告: 診断、修正、調整情報は、変更される場合がありますので、プログラミング・インターフェースとしては使用しないでください。

商標

以下は、IBM Corporation の商標です。IBM、 IBM (logo)、 AIX、 CICS、 Cloudscape、 DB2、 DB2 Connect、 DB2 Universal Database、 developerWorks、 IMS、 Informix、 iSeries、 Lotus、 Lotus Domino、 MQSeries、 MVS、 OS/390、 Passport Advantage、 pSeries、 Rational、 Redbooks、 Tivoli、 WebSphere、 z/OS、 zSeries

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標です。

Microsoft および Windows は、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

この製品には、Eclipse Project (http://www.eclipse.org/) により開発されたソフトウェアが含まれています。

IBM WebSphere Process Server for z/OS バージョン 6.0.2

Printed in Japan

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12