Novell exteNd Composer™ Enterprise Server

for IBM WebSphere Application Server

4.2

www.novell.com

ユーザガイド

Novell®

保証と著作権

米国 Novell, Inc.
1800 South Novell Place
Provo, UT 85606

www.novell.com

Novell exteNd Composer Enterprise Server ガイド
2003 年 1 月
000-000000-000

オンラインマニュアル： この製品およびその他の Novell 製品のオンラインマニュアルや更新情報については、www.novell.com/documentation を参照してください。

目次

このガイドについて

目的

このガイドでは、Composer アプリケーションを IBM WebSphere Application Server 上に配備するために exteNd Composer Enterprise Server を使用する方法について説明します。

対象読者

このガイドは、アプリケーションサーバの管理者、および Composer サービスの配備に携わるユーザを対象としています。

前提条件

このガイドでは、exteNd Composer の設計時環境および Composer のアプリケーション構築例についての予備知識が必要です。また、Java Archive 形式 (WAR、EAR、JAR)、および一般的な Web サービスの配備の概念について理解している必要があります。

構成

このガイドは、次のように編成されています。

章	説明
第 1 章、「exteNd Composer Enterprise Server へようこそ」	exteNd スイート製品の定義および概要について説明します。
第 2 章、「サーバ概要」	exteNd Composer Enterprise Server の仕様と運用ランタイム環境について簡単に説明します。
第 3 章、「配備の計画」	Composer サービスを配備する前に考慮すべき、環境およびリソースに関連する重要な要因の概要を説明します。

章	説明
第 4 章、「プロジェクトの配備」	使用可能なサービストリガのオプション、および exteNd Composer の Deployment Wizard の使用方法を説明します。
第 5 章、「配備フレームワークの使用」	非標準配備のために、アプリケーションサーバのフレームワーククラスをカスタマイズまたは拡張する方法を説明します。カスタムサービストリガを使用する必要がある場合は、この章をお読みください。
第 6 章、「トランザクションの管理」	アプリケーションのトランザクションを制御するオプションについて説明します。
付録 A、「配備オブジェクトの内容」	アプリケーションサーバにインストールされるファイルの内容について説明します。
付録 B、「Deployment Framework API マニュアル」	exteNd Composer Enterprise Server の Java フレームワークファイルについて説明します。
付録 C、「用語集」	このガイドで使用されている用語を定義します。

表記規則

このガイドで使用する様式および表記規則は、次のとおりです。

手順での太字の **Serif** フォントは、次のアクション項目を示します。

- メニューの選択
- フォームの選択
- ダイアログボックス項目

太字の **Sans-Serif** フォントは、次の項目を示します。

- Uniform Resource Identifier
- ファイル名

「斜体」のフォントは、次の項目を示します。

- 入力する変数情報
- 新出の技術用語
- 他の Novell 出版物のタイトル

「モノスペース」のフォントは、次の項目を示します。

- メソッド名

- コードの例
- システム入力
- オペレーティングシステムオブジェクト

追加のドキュメント

Novell exteNd Director に関する完全なドキュメンテーションについては、次の Novell マニュアルの Web サイトを参照してください。

http://www.novell.com/documentation-index/index.jsp

1 exteNd Composer Enterprise Server へようこそ

Novell exteNd Composer は、eXtensible Markup Language (XML) を主要な情報交換媒体として使用することによって、強力な eCommerce アプリケーションを作動、統合、および配置するために必要な時間を大幅に短縮する、B2B 統合サーバ製品のスイートです。exteNd Composer スイートは、3 つの製品で構成されます。

- **exteNd Composer** - B2B 統合アプリケーションを作成するための視覚的な設計環境
- **exteNd Composer Enterprise Server** - exteNd Composer Composer で作成されたアプリケーションを実行するランタイム環境
- **exteNd Composer Connectors** - exteNd Composer Composer およびサーバの機能を拡張して、データベース、ホストアプリケーション、および Java コンポーネントなどの企業情報ソースの XML 対応を可能にする製品ファミリです。

このガイドの焦点は、exteNd Composer Enterprise Server です。exteNd Composer Composer の詳細については、『exteNd Composer ユーザガイド』、および eCommerce アプリケーションに組み込む必要がある特定の exteNd Composer Connectors のユーザガイドを参照してください。

XML とは

Standard Generalized Markup Language (SGML)のサブセットである(XML) eXtensible Markup Language は、1998 年始めの World Wide Web Consortium で採用されたドキュメント標準です。XML は、プラットホームに依存しない構造化データ交換を容易にするように設計された、汎用メタ言語です。XML を正しく使用すると、ソフトウェアアプリケーションは、ビジネス情報がどのように作成されたかがわからなくても、その情報を受け取ったり処理したりすることができます。さらに、XML の作成者は、Web の主要情報交換プロトコルである Hypertext Transfer Protocol (HTTP) 上を移動できるデータ言語の作成を意識していました。

exteNd Composer Enterprise Server とは

exteNd Composer Enterprise Server(または、略してサーバ) は、exteNd (Composer) で開発されたアプリケーションのランタイム環境です。企業のアプリケーションサーバのコンテキストで実行される Java アプリケーションです。サーバにより、Composer から配備された XML メタデータを解釈したり処理したりする「ランタイム実行エンジン」、およびアプリケーションサーバによって提供されるサービス (例 : スレッド管理、接続プール、負荷分散、フェイルオーバとセキュリティ) と統合できる「アプリケーションサーバに合わせたフレームワーク」の両方を装備できます。フレームワークは、配備された exteNd Composer サービスについて、特にローカルマシンの環境、ネットワーク、またはインターネットで他のプログラムとどのように統合するか、サーバ側カスタマイズも提供します。

このガイドでは、exteNd Composerアプリケーションを WebSphere Application Server に配備するように選択したという前提で解説していきます。

コンポーネントおよびサービスについて

exteNd Composer は、「コンポーネント」と「サービス」という 2 つの主な処理構成要素を含むアクションモデルアーキテクチャに基づきます。コンポーネントは、ユーザの初期設定および具体的な統合のニーズによって多かれ少なかれ細分化された、(アクションリストとして実装された) 作業の実行可能な単位です。たとえば、JDBC コンポーネントは通常、着信した XML 要求ドキュメントを検証して、ドキュメントの重要な情報を SQL 問い合わせにマップし、SQL 結果セットを XML 応答ドキュメントにマップします。

「*exteNd Composer* サービス」は通常、複数のコンポーネントを構成したり、イベント処理、ルーティング、ロギングと例外処理などの重要なフロー管理作業を実行したりします。一般的なサービスには、単数または複数の XML ドキュメントの受信、洗練されたドキュメントマップおよび変換の実行、JDBC(または他の) データソースからの追加情報の収集、トランザクションの実行、エラー条件の処理、状況依存型の電子メール通知の送信、または元のリクエスタへ単数または複数の XML 応答ドキュメントの返信、あるいはそのすべてを実行するコンポーネントが含まれます。サービスは、exteNd Composer の「配備の単位」であり、アプリケーションサーバ上のサーバレットまたは EJB からインスタンス化されたオブジェクトです。

Web サービスと JMS サービスという 2 つの主なタイプのサービスがサポートされています (JMS は Java Messaging Service の略で、メッセージ指向ミドルウェアに対する Sun 定義のインタフェースです)。Web サービスのカテゴリには、HTTP に着信するデータによって呼び出されるよう設計されたすべてのサービスが含まれます。JMS サービスのカテゴリには、メッセージキューまたはメッセージトピックのメッセージの着信によって呼び出されるよう設計されたサービスが含まれます。

注記： JMS サービスは、Novell exteNd Composer JMS Connector を購入してインストールした場合にのみ該当します。

サーバがツールの exteNd Composer スイートとどのように適合するは、次の図のとおりです。

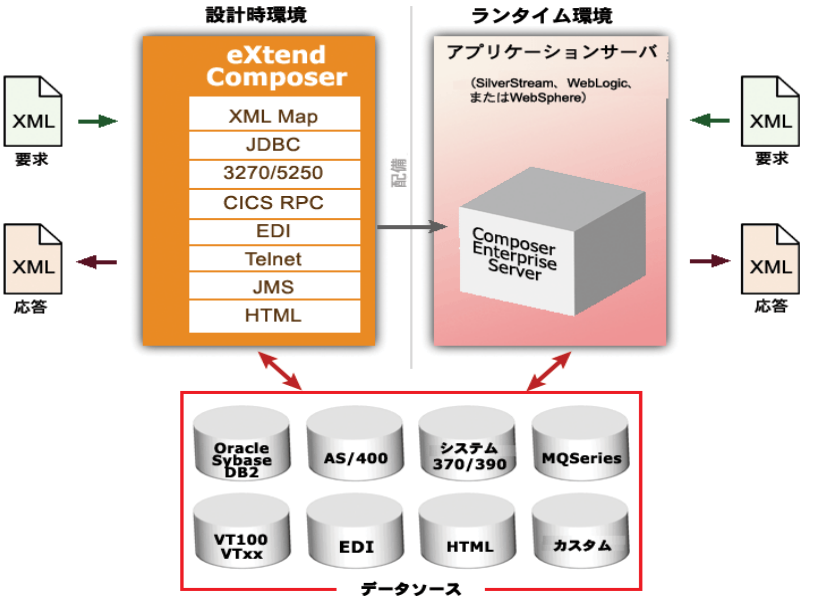


図 1-1

図のように、Composer を使用して、必要な統合作業を実行するコンポーネントやサービスを作成します。次に、コンポーネントおよびサービスが exteNd Composer Enterprise Server によって実行される商用アプリケーションサーバ環境に、コンポーネントおよびサービスを配備します。exteNd Composer の Connectors は、企業ビジネスシステムの XML 対応、およびアプリケーションとの相互作用のランタイム管理を提供します。

全体として、exteNd Composer 製品スイートでは次のことができます。

- 異種ドキュメントマップ

  XML に対応した任意のアプリケーションから XML 形式のデータを受信します。続いて、受信したデータを異なる XML ドキュメントタイプにマップまたは変換し、生成された XML 形式のデータを、XML に対応したその他のアプリケーションに送信します。

- 統合制御処理

イベントと、構成要素、条件、エラー、ログ、カスタム ECMAScript 関数などの繰り返しとグループ化を含む、統合アプリケーションに関連付けられている処理動作をすべて管理します。

- 端末データインタフェースを使用したホストアプリケーションの XML 対応

  XML ドキュメントからデータを読み込み、そのデータを端末トランザクションに直接マップ、変換、または転送するか、あるいは端末トランザクションの「結果」からデータを直接読み込んで、そのデータを XML ドキュメントにマップします。一般的な端末トランザクションタイプの例は、3270、および 5250 です。端末データインタフェースの XML 対応は、exteNd Composer の 3270Connector、および 5250Connectoror を通じて行われます。

- トランザクションベースおよびメッセージベースのプログラミングインタフェースを使用した、ホストアプリケーションの XML 対応

  XML ドキュメントからデータを読み込み、そのデータを COBOL/CICS Procedure Division に直接マップ、変換、または転送するか (例 : COMMAREA やメッセージキューを使用)、あるいは COBOL Procedure Division からデータを読み込んで、そのデータを XML ドキュメントにマップします。トランザクションベースおよびメッセージベースのプログラミングインタフェースの XML 対応は、exteNd Composer の CICS RPC and JMS Connectorors から行います。

- ウェブサイトのコンテンツ取得 (「スクリーンスクレーピング」) を使用したホストアプリケーションの XML 対応

  リモートウェブページからデータを読み込みます。続いて、HTML DOM 要素を XML DOM 要素にマップまたは変換し、生成された XML 形式のデータを、XML に対応したその他のアプリケーションに送信します。この機能は exteNd Composer の HTML Connectoror から利用できます。

- JDBC インタフェースを使用したデータベースの XML 対応

  XML ドキュメントからデータを読み込み、そのデータを SQL トランザクションに直接マップ、変換、または転送するか (JDBC を使用)、あるいは SQL トランザクションの「結果」からデータを直接読み込んで、そのデータを XML ドキュメントにマップします。JDBC データソースの XML 対応は、exteNd Composer の JDBC Connectoror を通じて提供されます。

- Java の XML 対応

  Java オブジェクトを開発して、統合アプリケーションに直接組み込みます。XML データは、これらのオブジェクトに渡して Java で処理し、exteNd Composer に返信して、さらに操作やマップを行うことができます。また、Java オブジェクトからは、exteNd Composer Enterprise Server の「フレームワーク API」にアクセスして (後の章で説明します)、カスタムドキュメント管理やイベント処理などの高度な操作を実行できます。

2 サーバ概要

サーバ仕様

exteNd Server は、ランタイムエンジンおよび拡張可能なアプリケーションフレームワークから構成される、100%Java および XML アプリケーションです。ランタイムエンジンには次の機能があります。

- XML の解析
- XSL の処理
- XML アプリケーションオブジェクトのメタデータの解釈と処理
- インストール可能なファクトリによる Connect オブジェクトのインスタンスの作成および実行

アプリケーションフレームワークの一部は、環境に依存しないベースクラス、およびexteNd Serverが実行される各アプリケーションサーバに適したクラスで構成されます。アプリケーションサーバに固有のクラスには、次のクラスが含まれます。

- ログ
- 接続プール
- トランザクション制御
- API 転送
- マルチパートのリクエスト処理のような固有のサービストリガ

サーバのフレームワークを拡張して、XML ドキュメントの処理前後、強化されたセキュリティ、および SOAP(Simple Object Access Protocol) 処理などの追加サービスを組み込むこともできます。

運用ランタイム環境

メタデータアーキテクチャ

各アプリケーションオブジェクト (例 : コンポーネント、サービス、接続、コードテーブル) は、XML ドキュメント (メタデータ) として保存されます。アプリケーションのメタデータはサーバのランタイムエンジンによって実行され、定義された操作を実行します。アプリケーションオブジェクトを XML メタデータとして表すことによって、exteNd は、多様な標準ツールで表示、管理できるコンポーネントおよびサービスを生成します。

配備オプション

exteNd では、プロジェクトをパッケージしてアプリケーションサーバに配備するためのウィザードが Composer のメインメニューに用意されています。Deployment Wizard には次の機能があります。

- プロジェクトのメタデータを JAR ファイルにパッケージ化する
- サービストリガ (EJB およびサーブレット) の Java コードを生成する
- EJB ラッパおよび配備記述子ファイルを生成する (必要な場合)
- 生成されたすべてのコードとメタデータ JAR をウィザードから直接アプリケーションサーバへ自動的にロードする

最初にアプリケーションを配備するときに選択した内容は XML ファイル (**xc_deployment_info.xml**) に保存され、次回 Deployment Wizard を起動するときに復元されます。それ以降の配備セッションでは、選択内容を変更または維持できます。配備オプションの変更が必要でない場合は、Deployment Wizard の最初の画面からプロジェクトを完全に再配備できます。

オプションとして手動で配備するには、生成した配備 JAR ファイルをステージングディレクトリに格納するように Composer を設定します。一般的に、この方法はアプリケーションの配備を担当するサーバの管理者が大規模なアプリケーションサーバをインストールする場合に使用されます。Composer ですべての必要な配備ファイルが生成されると、管理者は IT 組織によって定義された標準を使用してアプリケーションサーバにインストールできます。

Composer Enterprise Server の管理制御

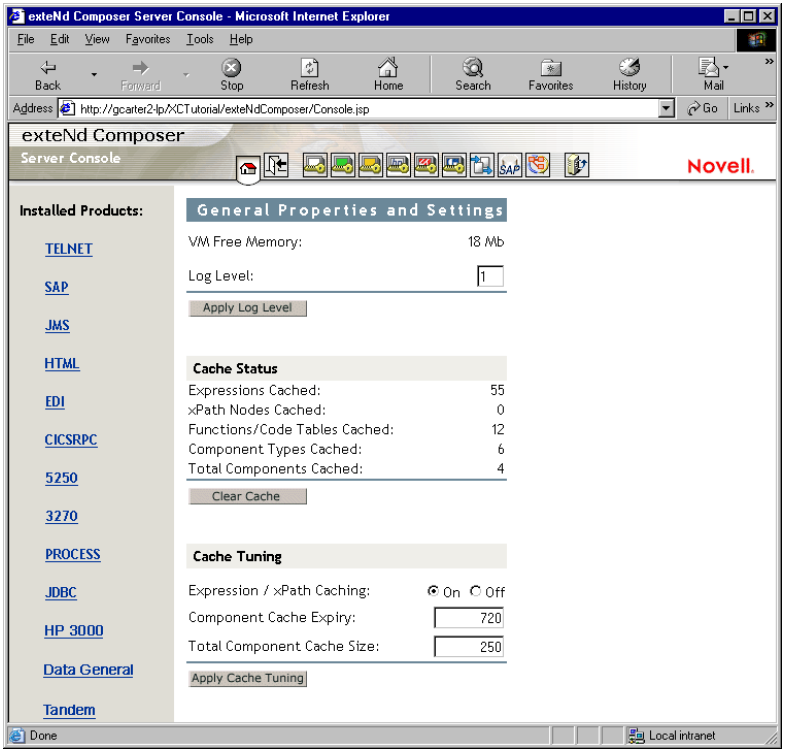
HTML ブラウザコンソールを使用して、Composer Enterprise Server のランタイム操作のあらゆる側面を管理できます。

➢ サーバ管理コンソールにアクセスする

1 Composer Enterprise Server がインストール済みおよび動作中で、アプリケーションサーバが実行中であることを確認してください。

2 Web ブラウザを起動します。

3 サーバを使用するホストアドレスおよびポートまで移動し、配備プロジェクトを探します (URL は **http://localhost:7001/exteNdComposer/Console** のように表示されます)。



4 Composer　プロジェクトのログメッセージのしきい値を変更したい場合は、［**Apply Log Level**］ボタンをクリックします。

5 キャッシュからすべてのオブジェクトをただちにパージしたい場合は、［**Clear Cache**］ボタンをクリックします。

6 必要に応じて(次の説明を参照してください)新しいキャッシュを入力してから［**Apply Cashe Settings**］ボタンをクリックすると、新しい設定が有効になります。

注記：［**Clear Cache**］ボタンと［**Apply**］ボタンはリアルタイムで作動します。変更内容を有効にするために Composer Enterprise Server を再起動する必要はありません。変更内容はただちに有効になります。

キャッシュの調整

Composer Enterprise Server の管理コンソールの［Advanced］タブにはリソースのキャッシュについての情報があり、さまざまなキャッシュパラメータを設定できます。キャッシュの調整についてユーザがコントロール可能なパラメータを次に説明します。

「コンポーネント」という単語は［Advanced］コンソールの数か所で使用されていますが、キャッシュされたオブジェクトには、Composer コンポーネントだけでなく他の xObject も含まれます。つまり、コンポーネントのみがキャッシュされるのではなく、サービス、リソース (コードテーブル、接続オブジェクト、カスタムスクリプトオブジェクトなど)、XML テンプレートなど、他の xObject タイプもキャッシュされます。

Least-Recently-Used (LRU) キャッシュアルゴリズム

Composer Enterprise Server は、*Least-Recently-Used* (LRU) アルゴリズムにより自動的にキャッシュ管理を処理します。xObjects をインスタンス化すると、キャッシュされたオブジェクトがあらかじめ定義された数に達したり超えたりしないかぎり、xObject はメモリに維持されます (またはキャッシュされます)。新しく到着するオブジェクトのためにスペースを作る必要がある場合は、もっとも古いオブジェクトはパージされます。「あらかじめ定義された数」は自分で設定できます。大きな数に設定すると、使用可能な仮想マシンのメモリの空き容量が消費されますが、Composer Enterprise Server のメモリに多数のオブジェクトを維持できます。小さい数に設定すると、メモリには少数のオブジェクトのみが維持される代わりに RAM の空き容量が増えます。

注記：キャッシュの容量が大きい方がパフォーマンスに優れているわけではありません。たとえば、ルーチンの JVM ガベージコレクション (メモリの圧縮とパージ) には、キャッシュが大きい場合はより時間がかかり、キャッシュの LRU 分析 (および削除) にも時間がかかります。異なるキャッシュ設定を試してみて、運用環境の「スイートスポット」を探す必要があります。

ユーザが調整可能なパラメータは次のとおりです。

- ［**Expression/XPath caching**］- このオプションを有効(ラジオボタンをオン)にすると、Composer Enterprise Server により、ランタイム時にできる限り ECMScript 式および Xpath オブジェクトが再使用されます。

- ［**Component Cache Expiry**］- この設定によって、アクティブでない (しかしまだキャッシュされている)xObject の有効期限 (分単位) を入力できます。デフォルトは 720 分 (12 時間) で、アクティブでないコンポーネントは 12 時間を超えるとメモリに残りません。(つまり、オブジェクトが 12 時間メモリにあって一度も使用されなかった場合は、おそらくそれ以上メモリにある必要はないということです。)
- ［**Total Component Cache Size**］- これは、ランタイム時にキャッシュに保存されるコンポーネントオブジェクトの最大数です。

キャッシュの有効期限および合計コンポーネントのキャッシュサイズは、独自のスレッドで動作するデーモン処理 (キャッシュプルーナ) により強制されます。プルーナは 10 秒ごとにキャッシュを検査し、オブジェクトが「時間切れ」でないか (非アクティブの有効期限に達していないか) を調べます。期限切れの場合、キャッシュがいっぱいかどうかに関係なく、オブジェクトは即座にキャッシュから削除されます。プルーナは、前回の検査以降にキャッシュが［Total Component Cache Size］制限を超えていないかも調べます。制限を超えている場合は、適切な数の古いオブジェクトは (時間切れかどうかに関係なく) パージされ、キャッシュサイズが制限内に保持されます。

パフォーマンスの調整

パフォーマンステストは、複雑な課題です。1 つのアドバイスではすべての状況に適用できません。テストを通じて、経験からキャッシュ設定を定義する必要があります。

Composer Enterprise Server コンソールによりキャッシュ設定を変更した場合、サーバを再起動する必要はありません。ただし、余裕を持ったキャッシュ設定はメモリの空き容量に大きな影響を与えるので、Composer Enterprise Server に割り当てられた仮想マシンのメモリ量を増やす必要がある場合もあります。その場合は再起動が必要です。Composer Enterprise Serve に割り当てられたメモリの量を変更するには、サーバをシャットダウンしてから、Composer サーバの **\lib** ディレクトリにある *xconfig.xml* と名前の付いたファイルを開きます (*xconfig.xml* ファイルが **\bin** ディレクトリにある、設計時の Composer 環境とは異なります)。*xconfig.xml* で、`VM_PARAMS` と名前の付いた要素を探します。`VM_PARAMS` 要素の設定を変更して、サーバに割り当てる RAM の最初の量および最大量を反映します (これらは標準的な JVM コマンドラインオプションです。JVM 起動パラメータと設定方法の詳細については、Sun のドキュメントを参照してください)。

注記： Composer Server の実行中は、*xconfig.xml* を編集しないでください。シャットダウン時に、Composer によって変更内容が上書きされます。*xconfig.xml* を編集する前に、Composer またはサーバ、あるいはその両方を終了してください。

余裕を持ったキャッシュ設定と RAM 設定によってパフォーマンスが良くなるとは限らないことに注意してください。パフォーマンスを決定する要因は多数あり、複雑に関係しています。慎重なテストを行って、運用環境に適した設定を決定する必要があります。

ライセンスのアップデート

Composer Enterprise Serve に関連付けられているライセンス文字列をアップデートする必要が生じた場合は、UpdateLicense.bat ファイル (exteNdComposer\bin ディレクトリにあります) を使用して実行できます。コマンドラインから、次のコマンドを実行します。

```
updateLicense product newLicense [Composer/Server]
```

ここで、*product* には (ライセンスをアップデートする) 特定の製品の名前を指定し、*newLicense* にはライセンス文字列を指定します。最後の引数 (*Composer* または *Server* の一方) は、目的の製品の設計時バージョンまたはランタイムバージョンのどちらをアップデートするかを指定します。

インストールされている製品のリストは、次のコマンドを実行して参照できます。

```
updateLicense -L
```

3 配備の計画

厳密に言えば、exteNd Composer でアプリケーションの設計を開始する前に、最終的なアプリケーションをどのように配備するかを理解する必要はありませんが、アプリケーション設計のできるだけ早い段階で配備を考慮しておくことをお勧めします (特に、トランザクション管理に関係する大規模で複雑なアプリケーションやサービスの場合にお勧めします)。Deployment Wizard を起動して、配備可能なオブジェクトを作成する前に、配備アプリケーションをどのように構成するかを決める必要があります。たとえば、次のような項目を考えてください。

- サービスをどのようにインスタンス化するか(たとえば、パラメータのある着信 URI に応答するか、または別のプログラムで行うかなど)
- 接続の要件 (接続プールが必要かどうか)
- トランザクション制御が必要かどうか (トランザクションモデルは、「コンテナ」管理または「Bean」管理のどちらにするか)
- 使用するサービストリガオブジェクトのモデル (EJB、サーブレット、または SOAP) また、サービスを複数のマシンに提供するか。
- サービストリガで、セキュリティ制限または特定の役割アクセス、あるいはその両方が必要かどうか
- ビジネスパートナーが、XML リソース (DTD、XSL、XSD、および WSDL スタイルシート) にアクセスできる必要があるかどうか
- 特別な Java クラス (カスタムまたは既存) に対するアクセスが必要かどうか

サービスのインスタンス化

Composer サービスは、完全なビジネスプロセスを提供し、そのビジネスプロセスを実行するためにインスタンス化が必要なスタンドアローンエンティティとして配備することができます。また、より大きなアプリケーションに統合することもできます。配備されるサービスは、XML のソースがわからなくても XML ドキュメントに実行するため、exteNd では、配備時に簡単にこの決定を下すことができます (詳細については、第 1 章 ,「プロジェクトの配備」を参照してください)。

Web サービスと JMS サービス

Composer は、Web サービスおよび JMS サービスという 2 つのタイプのサービスを作成できます (JMS は Java Messaging Service の略で、Java アプリケーションとメッセージ指向ミドルウェアを統合する Sun の API です)。Web サービスは、World Wide Web (または HTTP) を介して送信される XML データに対応して実行されます。JMS サービスは、メッセージキューに送信されるメッセージに対応して実行されます。配備に関する考慮事項は、2 つのサービスタイプで異なります。

Web サービスは、着信 HTTP データに対応するサーブレットである「サービストリガ」オブジェクトにより呼び出されるまで、停止しています。また、Web サービスは、サーバ上で実行する別の Java プロセスにより直接インスタンス化することもできます。

これに対して、JMS サービスは、配備されたときから、管理者がオフにするまでアクティブです。配備時に、JMS サービスはメッセージ指向ミドルウェア (MOM) 環境のメッセージキュー(またはメッセージトピック) で *MessageListener* オブジェクトを登録します。JMS サービスがキューを待機すると、受信メッセージはサービスの `onMessage()` ハンドラをトリガし、こうしてサービスを呼び出します。Web サービスは、XML 入力を処理するように設計されていますが、JMS サービスは、「メッセージ」に応答します。このメッセージには、様々なペイロード (XML、COBOL コピーブック、および属性バイトストリームなど) が含まれます。

注記： JMS サービスタイプの説明は、exteNd JMS Connect を購入してインストールした場合にのみ適用されます。この接続がない場合は、JMS サービスを作成できません。

JMS サービスの詳細については、『exteNd JMS Connect ユーザガイド』を参照してください。

スタンドアロン配備

スタンドアロン配備では、サービスは exteNd Server のコンテキスト内でインスタンス化され、すべての必要な手順を実行します (入力パラメータを XML 文字列に変換する、必要な HTTP 要求を実行する、システムイベントをログする、など)。Deployment Wizard を使用すると、プロジェクトメタデータのパッケージ化、および XML 入力の受け取り方法に基づいたサービストリガの作成を実行できます。

XML 入力は、いくつかの方法で受け取ることができます。たとえば、パラメータを持つ Web ページ要求または HTML フォームからデータを取得したり、取引パートナーの URI から情報を受け取ることもできます。Deployment Wizard では、このようなさまざまな種類の XML 入力の対応するために必要なサービストリガが、自動的に作成されます。配備オプションの詳細については、18 ページ「配備オプション」を参照してください。

注記 : exteNd JMS Connect がインストールされている場合、Deployment Wizard には、JMS サービス配備に関するパネルが追加されます。この点以外は、Deployment Wizard は、両方のサービスタイプで同じです。

統合アプリケーション配備

作成するサービスによっては、他のアプリケーションに統合する必要があります。このタイプの統合では、前に説明したとおり、サービストリガを使用したり、外部アプリケーションのJavaオブジェクトからサービスを直接インスタンス化することができます。

Java オブジェクトからサービスをインスタンス化するため、eXtend では、exteNd Server フレームワークの JavaDoc API および参照可能ソースコードが提供されています (詳細については、付録 B,「Deployment Framework API マニュアル」を参照してください)。

注記 : 外部アプリケーションを統合する必要のあるサービスを配備する場合、正確な JAR ファイルのパッケージ化および CLASSPATH 仕様を考慮する必要があります。**xcs-all-ws.jar** ファイルには、exteNd Server の実行に必要なクラスが含まれます。また、このファイルでは、**xconfig.xml** およびプロジェクトメタデータの両方が CLASSPATH になければなりません。

接続プール

通常、外部リソースとの通信を管理する場合、最もリソースを消費する操作の 1 つに、「接続管理」があります。各要求に対して、各トランザクションで接続を開いたり、閉じたりできるようにすると、アプリケーションサーバのオーバーヘッドが急増します。このオーバーヘッドを最小化するため、exteNd Server では、アプリケーションサーバの「接続プール」機能を利用できます。

データベース接続プール

アプリケーションサーバを通して提供されたサービスの 1 つに、データベース「接続プール」があります。WebSphere Application Server では、これらのプールは、**jdbc/datasource_name** の形式を受け入れる JNDI 名によって識別されます (ここで、**datasource_name**は、WebSphereデータソースの名前で置き換える必要があります)。

WebSphere Application Server の接続プール機能を利用するには、ターゲットデータベースの接続リソースに、設計時に指定したプール名が必要です。exteNd Composer で、JDBC 接続リソースを開いてから、[Properties] パネルの [Connection Info] タブに移動します (詳細については、『Novell exteNd Composer ユーザガイド』のリソースの作成に関する章で「接続について」を参照してください)。

Connects 用リソースプール

exteNd Server では、非データベースリソースへの接続に対してアプリケーションサーバの接続を補う接続のプール機能が提供されています。exteNd Server の接続プールは、個別の exteNd コンソールを使用して設定および管理できます。

非データベースで、Connect 固有の接続プールを利用するには、配備された各接続タイプ (3270、5250、Telnet、CICS RPC、JMS など) の Connect ガイドを参照してください。

セキュリティ役割

役割を使用すると、配備した exteNd Composer サービストリガに対するアクセス権限を規定できます。役割は、配備時に指定できます。役割を指定すると、管理者は、Application Server のコンソールで抽象的な役割に実際のセキュリティ定義を提供できます。セキュリティ役割を使用すると、特定の URL パートナーの HTTP アクションを制限することができます。

トランザクション制御

トランザクションが Composer サービスの一部である場合、設計時に配備によって起こりうる影響をすべて考慮することが特に重要です。たとえば、サービスのトランザクション動作は比較的単純で、2 段階のコミットが必要ない場合、直接的なサーブレットの配備で十分に適切となります。さまざまなバックエンドシステムに及ぶような、さらに複雑な動作が実装される場合、複雑なトランザクションでは、EJB 配備が必要になる場合があります。その場合、トランザクション制御が Bean 管理またはコンテナ管理のいずれであるか決定する必要もあります (詳細については、第 1 章 ,「トランザクションの管理」を参照してください)。

XML リソースの公開

B2B プロセスを確立する場合、ビジネスパートナーが必要とする特定のファイルを公開しなければならないことがあります。このようなファイルの例としては、請求書を提出するための XSL スタイルシート、およびサイトから送信したドキュメントを検証するための DTD/ スキーマファイルがあります。

WebSphere は、サーブレットおよび Web パスの両方を、Web アプリケーションに関連付けます。Web パスは、クライアントが使用できるファイルのリソースを含むディレクトリを参照します。公開されたリソースをアプリケーションの Web パスに配置すると、サービスコンポーネントのすべてのユーザが使用可能になることを確認できます。

XML Interchange アクションからのファイルアクセス

XML Interchange アクション (『*Composer* ユーザガイド』で高度なアクションに関する章を参照してください) を使用する Composer サービスは、ランタイム時に読み込み権または書き込み権、あるいはその両方を必要とします。WebSphere/NT 環境では、WebSphere のプロセスは、ネットワークのリソースに対して権限を持つユーザ ID の下で実行される必要があるということです。それ以外では、「ファイルが見つかりません」という例外が生じる場合があります。

EJB アクセス

➢ WebSphere アプリケーションサーバで配備された EJB にアクセスする

1 必要な JAR ファイルのクラスパスエントリを追加することによって、**xconfig.xml** 設定ファイルの SYSTEM/RUNTIME セクションをアップデートします。これには、次のものが含まれます。

exteNd クラスに基づく JAR ファイル

- **remote-ejb.jar** (EJB が配備されたときに生成されたリモートのスタブを含む JAR ファイル)
- xcs-all-ws35.jar

WebSphere のインストールにある JAR ファイル

- ejs.jar
- ujc.jar

たとえば、アップデートされた JAR エントリは次のようになります。

```
<SYSTEM>
   <RUNTIME>
   . . . .
   <JAR>D:\WebSphere\AppServer\lib\ejs.jar;D:\WebSphere\AppServ
er\lib\ujc.jar</JAR>
   </RUNTIME>
</SYSTEM>
```

2 **exteNd\Composer\jre\lib\orb.properties** ファイルを編集します。「#」記号を使用して、Novell's JBroker 製品を参照する 5 行をコメントアウトします。WebSphere 固有のプロパティに関連する 2 行から、コメントを削除します。

次に示す行をコメントアウトします。

```
org.omg.CORBA.ORBClass=com.sssw.jbroker.ORB
org.omg.CORBA.ORBSingletonClass=com.sssw.jbroker.orb.SingletonO
RB

javax.rmi.CORBA.StubClass=com.sssw.jbroker.rmi.StubDelegate
javax.rmi.CORBA.UtilClass=com.sssw.jbroker.rmi.UtilDelegate
```

```
javax.rmi.CORBA.PortableRemoteObjectClass=
     com.sssw.jbroker.rmi.PortableRemoteObjectDelegate
```

次に示す行のコメントを解除します。

```
org.omg.CORBA.ORBClass=com.ibm.rmi.iiop.ORB
org.omg.CORBA.ORBSingletonClass=com.ibm.rmi.corba.ORBSingleton
```

この時点で、exteNd を実行したり、(たとえば)ECMAScript を使用して Function アクションを通じて、配備された EJB にアクセスしたりすることができます。

追加の Java クラス

新しい Java クラスをサービスに統合することが便利または必要なことがあります。アプリケーションに別の Java クラスが必要な場合、exteNd Composer および exteNd Server で使用 (参照) できるようにする必要があります (exteNd Composer に適用可能な説明については、『*eXtend Composer* ユーザガイド』の付録 A を参照してください)。主な要件は、JAR ファイルまたはクラス、あるいはその両方が、exteNd によってリソースの検索に使用される CLASSPATH 環境変数にバインドされることです。

WebSphere Administrative Console を使用して Java クラスを追加する

3 WebSphere AdminServer を起動します。

4 WebSphere Administrative Console を起動します。

5 ［Topology］タブをクリックします。

6 Composer アプリケーションが置かれているノードを拡張します。

7 Composer アプリケーションが置かれているアプリケーションサーバをクリックします。

8 アプリケーションサーバのプロパティ内で、[General Properties]タブをクリックします。

9 ［General］プロパティシートで「Command-Line Arguments」フィールドをクリックして、新しいディレクトリまたは JAR ファイルを含むように CLASSPATH の設定を更新します。

4 プロジェクトの配備

配備プロセス

Composer プロジェクトを WebSphere 環境に配備するには、次の手順が必要です。

- プロジェクトのサービスに対するランタイムコンテキストを決める。例 : サービスをスタンドアロンプロセスとして実行するか、または (代わりに) 別のアプリケーションの一部として実行するかどうか。
- exteNd Composer の Deployment Wizard を使用して配備オブジェクトを作成する。
- 配備オブジェクトを WebSphere Application Server にインストールする。

はじめに

自動配備を利用する場合は、プロセスを配備する前にターゲット管理サーバが実行されており、Composer Deployment Manager サーブレットがインストールされていることを確認する必要があります (事前に該当するノードに exteNd をインストールしている場合、このサーブレットがすでに存在しています)。

ランタイム環境の決定

Composer プロジェクトを配備する前に、サービスをスタンドアロンで実行するか、アプリケーションサーバの別のアプリケーションに統合するかを決める必要があります。スタンドアロンサービスを配備する場合、「サービストリガ」オブジェクトを作成する必要があります。このオブジェクトは、exteNd により自動的に作成されます。また、自分で作成することもできます。他のアプリケーションにサービスを統合するには、外部アプリケーションから Composer サービスをインスタンス化および実行する Composer 配備フレームワークオブジェクトを使用する必要があります。

配備オブジェクトの作成

Composer プロジェクト配備には、次のオブジェクトタイプが含まれます。

表 4-1

オブジェクト	用途	注記 :
プロジェクト JAR ファイル	プロジェクトのサービス、コンポーネント、およびリソースを含みます。	XML リソース (DTD、スキーマ、XSL) は、別々に公開される必要があります。公開するの形式の例については、次の XML リソース JAR を参照してください。
PROJECT.xml	ユーザ定義のプロジェクト変数に関する情報を含みます。	
xc_deployment_info.xml と呼ばれる配備プロファイルファイル	Deployment Wizard が最後に実行されてからの配備プロファイルを含みます。	自動的に作成されます(前のプロジェクト JAR に存在)。配備が次に実行されるときに、配備情報を exteNd により回復できます。

exteNd Composer の Deployment Wizard では、これらのオブジェクトを作成し、Deployment Manager を介して WebSphere Application Server に自動的にインストールできます。希望する場合は、Deployment Wizard をいつでも再度実行して、一部または全部のオプションを変更したり、変更したプロジェクトコンポーネントを再度配備したりできます

これらのオブジェクトの内容の詳細については、付録 A,「配備オブジェクトのコンテンツ」を参照してください。

配備オブジェクトのインストール

WebSphere ノードに配備オブジェクトをインストールするには、2 つのオプションがあります。最初のオプションは、Deployment Wizard が生成する JAR ファイルを、WebSphere ノードで実行中の Deployment Manager に、自動的に送信させる方法です。2 つめのオプションは、オブジェクトを手動でインストールする方法です。コマンドラインモードで Deployment Manager を実行し、および必要なパラメータを指定します。

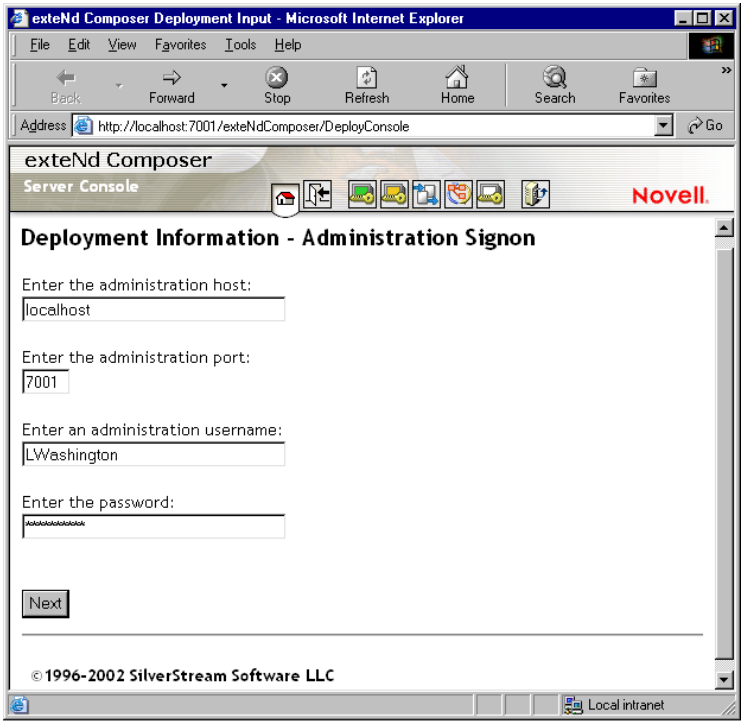
ブラウザベースの配備

(前に説明したように) 配備オプションの 1 つとして、Composer 配備コンソールを使用して、ブラウザセッション内で配備プロセスを初期化、管理できるようにします。この方法の利点は、設計時と配備時の操作を明確に区分できる点です。(Composer の設計側エディタで作業している) アプリケーション開発者は、配備可能なプロジェクト JAR を作成して、ネットワーク上のステージングディレクトリにポストできます。その後しばらくして、サーバ管理者または配備専門家が、ここで説明するコンソールを使用した方法によって JAR を取得、配備できます。

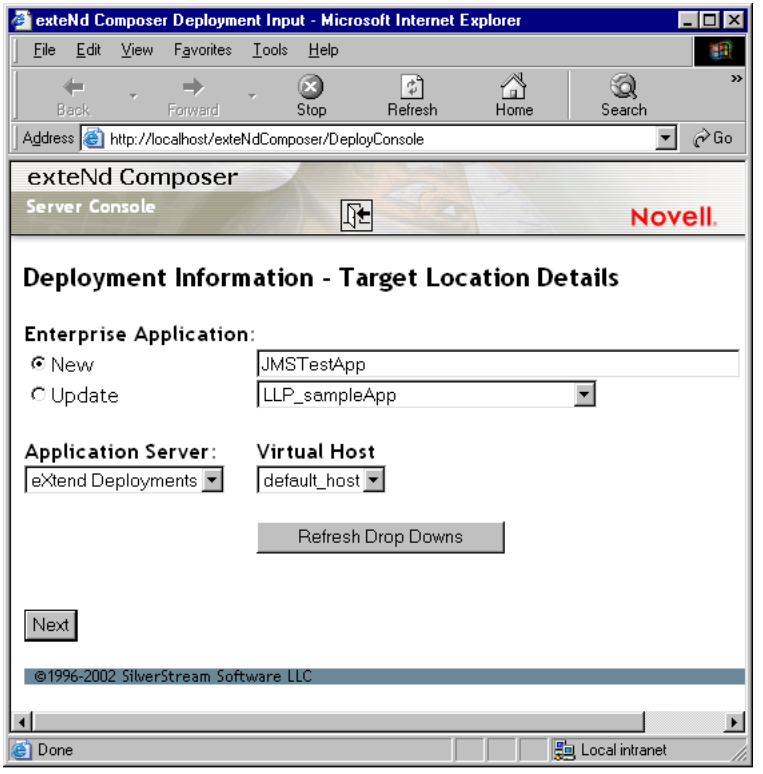
➢ 配備コンソールを使用する

1. Composer Enterprise Server がインストール済みおよび動作中で、アプリケーションサーバが実行中であることを確認してください。
2. Web ブラウザを起動して、メイン Composer サーバコンソールに移動します (「サーバ概要」の説明を参照してください)。
3. ツールバーの［**Deploy from Browser**］ボタンをクリックします。

4. 表示される画面 (次の図を参照) で、必要な情報を入力します。

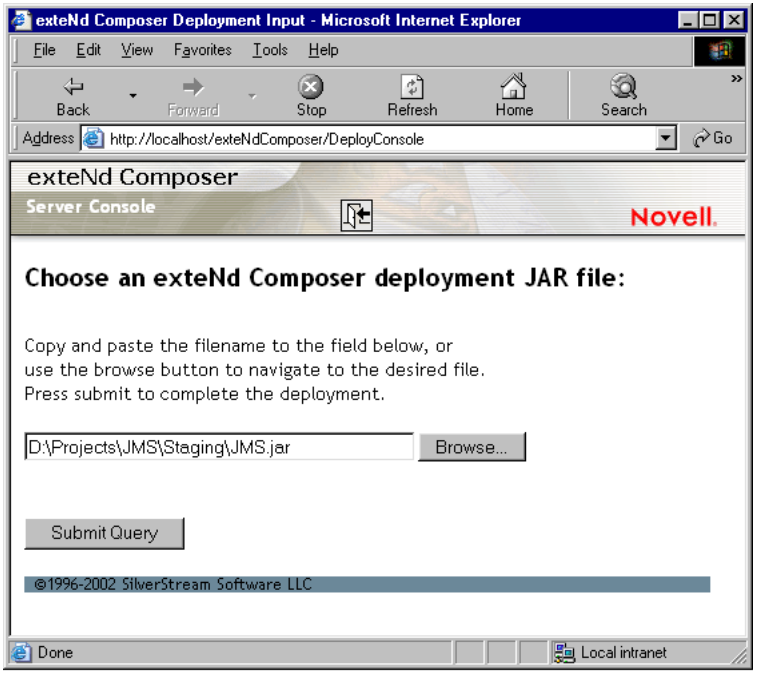


5 ［**Next**］ボタンをクリックします。新しい画面が表示されます。



6 アプリケーション名、サーバ名、および仮想ホスト名を入力します (配備を行うのはこれが初めてでない場合は、[Update] ラジオボタンをオンにします)。

7 [**Next**] ボタンをクリックします。新しい画面が表示されます。



8 配備対応 JAR ファイルへのパスが分かる場合は、示されたテキストフィールドにパスを入力します。それ以外の場合は、［**Browse . . .**］ボタンをクリックして、JAR の現在の場所 (ステージングディレクトリ) に移動します。

9 ［**Submit Query**］ボタンをクリックして、配備を完了します。ファイルがサーバにコピーされ、レポートがブラウザに生成されます。

この時点で、適切なサーバファイルはすべて最新となり、サーバを再起動すると Composer サービスが「ライブ」になります。

Deployment Manager について

exteNd Server インストールの **xcs-all-ws.jar** ファイルとして含まれるサーブレットである Deployment Manager では、配備処理の最後に配備オブジェクトが自動的に配備されます。手動で配備する場合は、*xcs-deploy* ユーティリティを使用してプロジェクトを配備できます。図 4-1 を参照してください。

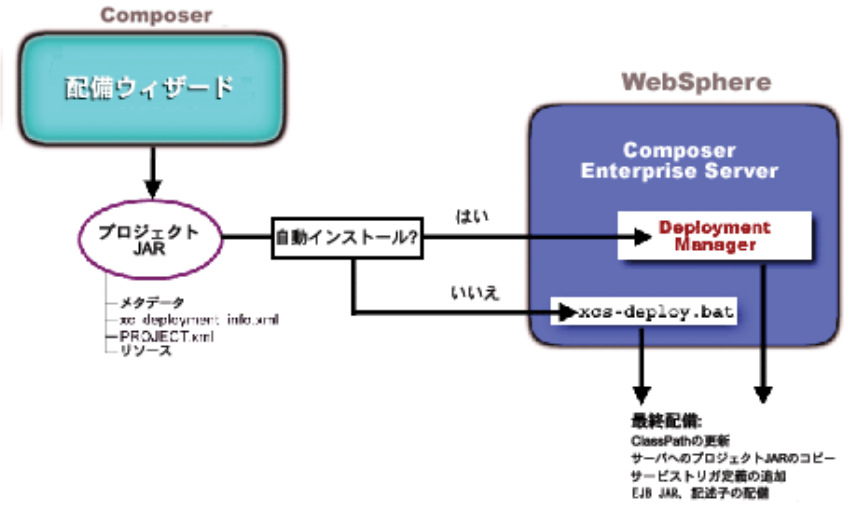


図 4-1

Deployment Manager の URI は、次のとおりです。

http://<hostname>/<Virtual_host_path>/exteNdComposer/DeployManager

ここで、

- **hostname** は、WebSphere ノードのホスト名です。
- **Virtual_host_path**は、exteNd Webアプリケーションがインストール時にインストールされた仮想ホストを表すパスです。

たとえば、次の WebSphere 環境にすでに exteNd をインストールしているとします。

ノード : `accounting`

仮想ホスト : `default_host` (path = '/')

その結果、Deployment Manager の URI は、次のようになります。

http://accounting/exteNdComposer/DeployManager

注記 : Deployment Wizard の最後のパネルで [Launch browser to complete deployment] というラベルが付いたチェックボックスをオンにしている場合、Deployment Manager によってあらゆる自動インストール処理の詳細が考慮されます。配備オブジェクトを手動でインストールしたい場合は、*xcs-deploy* ユーティリティを使用します (50 ページ「手動による配備オブジェクトのインストール」を参照)。

Web サービスのサービストリガについて

サービスは、アプリケーションサーバ環境における実行の基本単位です。サービスを含めすべての exteNd オブジェクトはメタデータの指示として作成され、配備 JAR の中にある XML ファイルに保存されます。配備するとき、サービスは適当なサーバトリガメカニズムに接続する必要があります。Deployment Wizard で、Web サービスの 3 つのサービストリガである Java サーブレット、Enterprise Java Beans、および SOAP を生成できます。さらに、exteNd Server のフレームワークを使用すると、サービストリガをカスタマイズしたり、手動で作成したりできます。

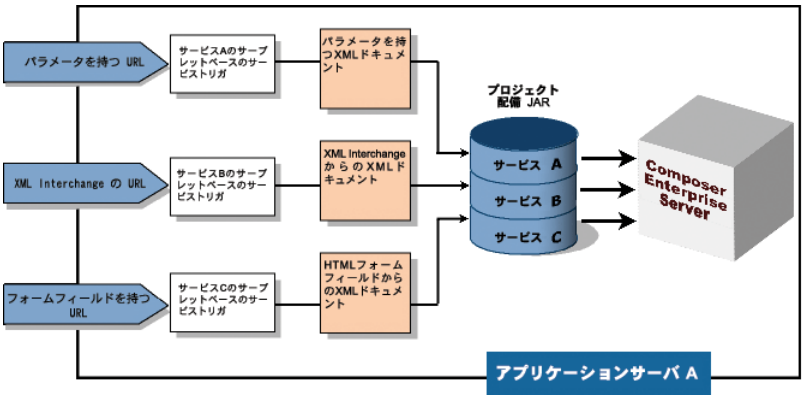
トリガに関する次の説明は、Web サービスの場合です。JMS サービスは Web サービスとは異なる方法で実行されます。ただし、単一の配備可能なプロジェクトに Web サービスまたは JMS サービス、あるいはその両方が存在する場合があるため、いずれのサービスでも同じ Deployment Wizard が使用されます。

サーブレットベースのサービストリガ

Java サーブレットはサービストリガとして使用されます。サービストリガにより URI と特定の Composer サービスが関連付けられ、入力データが HTTP 要求から XML ドキュメントに変換されます。サービスは XML ドキュメントを入力データとして受け入れることができ、最終的にサービスが実行されます。また、EJB、着信 SOAP 要求、または JMS メッセージから、サービスが直接的または間接的にトリガされるようにサーブレットを生成することもできます。EJB からサービスをトリガするには、EJB ベースのトリガをまず生成して (37 ページ「EJB ベースのサービストリガ」を参照)、それによって使用するサーブレットを生成する必要があります。

サーブレットベースのサービストリガは、Composer Deployment Wizard により自動的に作成されます。必要な場合は、これらのサーブレットベースのサービストリガを変更したり、Composer により作成されたサービストリガとは別のものを定義したりできます。

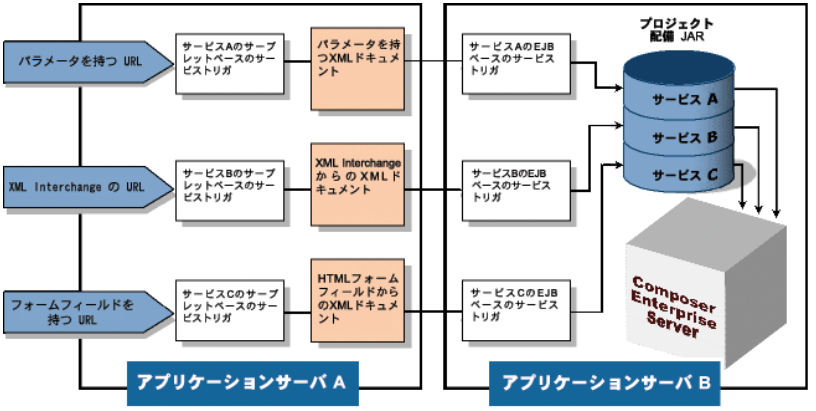
次の図では、Deployment Wizard により生成されたサーブレットタイプに関する一般的なシナリオが説明されています。



EJB ベースのサービストリガ

サービストリガの 2 つ目のタイプは、EJB(Enterprise Java Bean) です。EJB は、配備の柔軟性および宣言型トランザクション制御に優れているなど、さまざまな理由でサービストリガとして使用されます。Composer のウィザードによって生成された EJB サービストリガは、あらゆる Java アプリケーションで使用して Composer サービスを直接実行できます。

EJB の実行は、サーブレットの実行ほど直接的ではありません。そのため、exteNd では EJB ベースのサービスをトリガするためにその前のセクションで参照されるタイプの Java サーブレットを生成できます。サーブレット -EJB ベースのサービストリガでは、1 つではなく、2 つのオブジェクトの間でこれらのサービスの呼び出しタスクを分割しています。サーブレットが URI を関連付け、入力 HTTP データを XML に変換し、EJB を実行します。同じマシンに exteNd プロジェクト JAR として置く必要のある EJB は、サービスを実行します。次の図を参照してください。



アプリケーションベースのサービストリガ

Composer で生成されたサービストリガオブジェクトを使用する以外の方法として、Composer の外でサービストリガオブジェクトを作成するか、Composer サービスの使用が必要な別のアプリケーションにサービストリガの機能性を統合する方法があります。このようなアプリケーションベースのいずれのトリガオプションでも、Composerオブジェクトにインタフェースを記述するComposer Deployment *Framework* が必要です。

SOAP ベースのサービストリガ

特殊なサービストリガは、Composer の SOAP ベーストリガです (この独自のパネルが Deployment Wizard にあります。詳細は以下を参照してください)。このタイプのトリガを選択すると、HTTP POST を介して着信 SOAP 要求を受け取り、サービスに渡す前に SOAP 本文をアンラップするサーブレットが Composer により生成されます。

JMS サービストリガ

配備するとき JMS サービスにより、メッセージキューまたはメッセージトピックを持つ `onMessage()` ハンドラを登録する *MessageListener* オブジェクトのインスタンスが生成されます。JMS サービスのリスナオブジェクトは exteNd Server が実行されるたびに (つまりアプリケーションサーバが起動されるたびに) ロードされます。 サービスそのものはメッセージがキューに着信するまで、つまり `onMessage()` ハンドラが起動するまで実行されません。したがって、メッセージの着信は、サービスが応答するイベントとして機能します。

本来 JMS サービスの寿命は無限で、サーバがアクティブな限りサービスもアクティブであるため、他の Composer サービスとは別に管理する必要があります。ブラウザベースの管理コンソールは、この目的のために提供されます。『*exteNd JMS Connect ユーザガイド*』を参照してください。

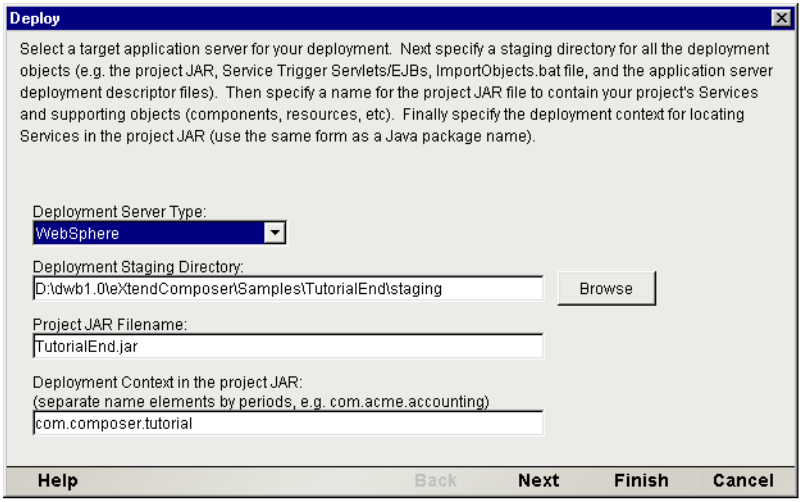
注記： この説明は、exteNd JMS Connect がインストールされている場合にのみ適用されます。詳細については、『*exteNd JMS Connect ユーザガイド*』を参照してください。

Composer Deployment Wizard の使用

Composer Deployment Wizard では、配備のさまざまな特性に関する一連の質問が表示されます。

Composer Deployment Wizard を使用して、プロジェクトのためのプロジェクト JAR、サービストリガ、およびその他の配備オブジェクトを生成するには、Composer メニューバーから［**File**］、［**Deploy**］の順に選択します。

一般情報パネル



このパネルを使用して、配備の一般情報を指定します (以下で詳しく説明します)。すでにプロジェクトが配備されている場合、最新の配備で使用された値が表示されます。

配備サーバタイプ

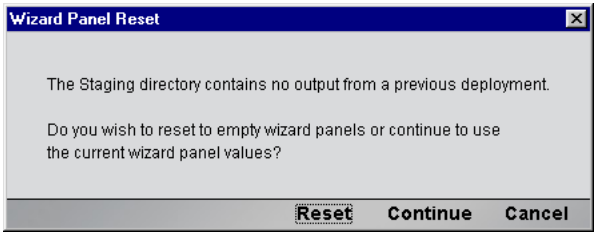
この配備のアプリケーションサーバタイプを［Novell］、［IBM WebSphere］、または［BEA WebLogic］から選択します。これにより、選択されたサーバ環境に固有な配備ファイルが Composer により作成されます。

配備ステージングディレクトリ

Composer により作成されるすべての配備オブジェクトを配置するディレクトリを指定します。［**Browse**］ボタンを選択して既存のディレクトリに移動するか、新しいディレクトリを作成します。

注記：各プロジェクトには、専用のディレクトリが必要です。2 つのプロジェクトを同じステージングディレクトリに配備すると、配備オブジェクトが上書きされます。

プロジェクトを再配備するが、ステージングに選択したディレクトリが空 (新しい) の場合、次のようなプロンプトが表示されます。



このプロンプトでは、ウィザードのすべてのフィールドを再設定するか、前の配備の設定を使用して新しい配備を続行できます。

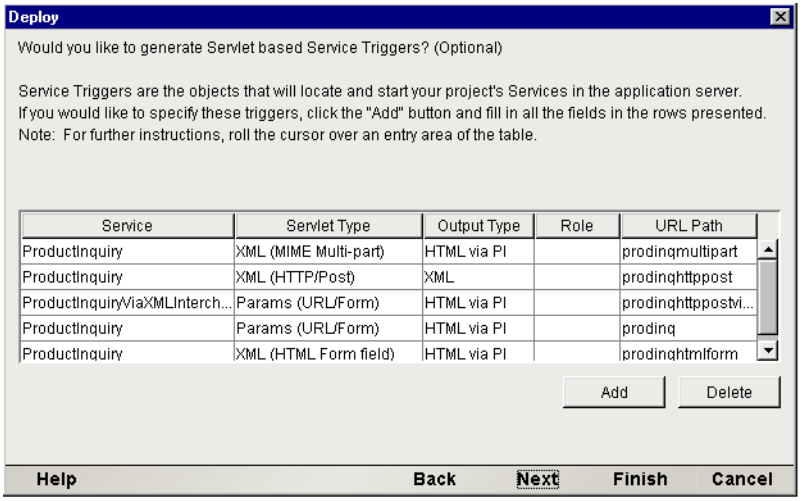
プロジェクト JAR ファイル名

このプロジェクトのすべての xObject を含む JAR ファイルの名前を入力します。このファイルは、ステージングディレクトリに作成されます。

プロジェクト JAR の配備コンテキスト

このコンテキストの名前は、自由に付けることができます。文字列の部分をピリオドで区切って、文字列を入力します。配備コンテキストは、(プロジェクトが異なる)2 つの Composer サービスの名前が同じ場合、ネームスペース競合を避けるためこれらのサービスを区別するときに使用されます。

サーブレットベースのサービストリガパネル



このパネルを使用して、サーブレットベースのサービストリガで配備するWebサービスを選択します。同じステージングディレクトリを使用して、プロジェクトがすでに配備されている場合、最新の配備で使用された値が表示されます。配備される各サーブレットベースのサービストリガに対して、［**Add**］ボタンを押して、次のパラメータを指定します。

Service

プロジェクトのサービスは、各行の最初のフィールドに表示されます。フィールドをクリックすると表示されるドロップダウンリストから、配備するサービスを選択します (プロジェクトのすべてのサービスがリストに表示されます)。

Servlet Type

サーブレットタイプは、Params (URL/Form)、XML (MIME multipart)、XML (HTML form field)、および XML (HTTP POST) の 4 種類あり、Composer サービスでデータを送受信する方法を表します。それぞれのサーブレットタイプは、入力データを受け入れる方法が異なります。相違点について、次の表で簡単に説明します。

サーブレットタイプ	データの受け入れ方法	注記
Params (URL/Form)	URL エンコード形式で URI に追加された URI パラメータ	このサーブレットにより、ノードの名前として HTTP URI 形式のパラメータ、テキストとしてその値を使用するメモリ内 XML ドキュメントが作成されます。1 つのパラメータに対して複数の値を持つことができますが、複数の入力ドキュメントが作成されるわけではありません。
XML (HTTP POST)	ポストされたXMLドキュメントのコンテンツ (XML Interchange アクション - Post など)	このサーブレットにより、HTTP POST メソッドで送信された XML ドキュメントが取得されます。HTML Form POST は parameter name/value の対を含む点がこのサーブレットと異なります。この種類のHTTP接続の実際のペイロードは、加工されていない XML ドキュメントです。取引パートナーと XML ドキュメントを交換する場合に便利なメソッドです。
XML (MIME multipart)	HTTP multipart/form データポストメソッドからのXMLファイル(これにより、ユーザはファイルシステムを検索したり、XML ファイルを選択して送信したりできます)。	このサーブレットにより、「file」入力タイプのフィールドを含むマルチパートのエンコード形式のデータからサービスの入力ドキュメントが抽出されます。サーブレットは、「xmlfile」と呼ばれる XML ファイルを含むフィールド名を予期し、このパラメータが最初に発生した場合にドキュメントを抽出します。

XML (HTML form field)	XMLが内部にポストされている形式のファイル。XMLが「xmlfile」とラベルのついたフィールドの中に存在する必要があります。	このサーブレットでは、ポストされた形式のフィールドからサービスの入力ドキュメントが抽出されます。 サーブレットは、「xmlfile」と呼ばれるXMLファイルを含むフィールド名を予期し、このパラメータが最初に発生した場合にドキュメントを抽出します。

Output Type

プルダウンメニューから、［**XML**］または［**HTML via PI**］を選択します (サービスの出力に適した方)。(処理命令を介した)XSL スタイルシートの結果として HTML を XML データに出力する場合、［**HTML via PI**］を使用します。このオプションを使用すると、exteNd により、出力に適した MIME タイプが設定されます。

Role

サービストリガへのアクセスを許可する J2EE 役割の名前を入力します。 入力した役割名にマップされるグループの個人またはメンバーだけが、サービストリガにアクセスできます。役割の名前に一致するサーバグループを作成し、役割のアクセス機能を利用できるようにする必要があります。

セキュリティ役割の定義はオプションです。アクセス制限が必要ない場合、［Role］フィールドは空白のままにしてください。

URI Path

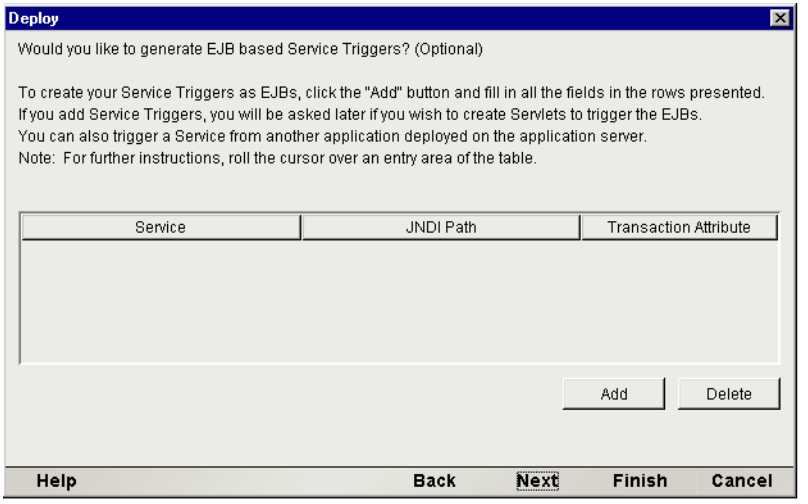
［URI Path］は、このサーブレットに割り当てられている URI の固有な名前を提供します。この名前は、Websphere サーバノード名および Websphere Web アプリケーション名とともに、このサーブレットにアクセスする完全な URI を形成します。

たとえば、［URI Path］に「**WebStore/ProductInquiry**」が指定されていて、サーバ名が「**stratus**」で、Web アプリケーション名が「**InfoDB**」の場合、完全な URI は次のようになります。

http://stratus/InfoApp/WebStore/ProductInquiry

EJB ベースのサービストリガパネル

このパネルを使用して、EJB ベースのサービストリガで配備する Web サービスを選択します。同じステージングディレクトリを使用して、プロジェクトがすでに配備されている場合、最新の配備で使用された値が表示されます。配備される各 EJB ベースのサービストリガに対して、［**Add**］ボタンをクリックして、次のパラメータを指定します。



Service

［Service］フィールドをクリックすると表示されるドロップダウンリストからサービスを選択します。このリストには、プロジェクトに含まれるサービスの名前が示されます。

JNDI Path

［JNDI Path］は、EJB の検索に必要な固有な名前を提供します。この名前は、イニシャルコンテキスト文字列「iiop://」、WebSphere サーバノード名、文字列「RMI」とともに、この EJB を検索する完全な JNDI 複合名を形成します。たとえば、［JNDI Path］に「**WebStore/ProductInquiry**」が指定されていて、サーバ名が「**stratus**」で、JNDI サービスプロバイダが「**RMI**」の場合、完全な JNDI 名は次のようになります。

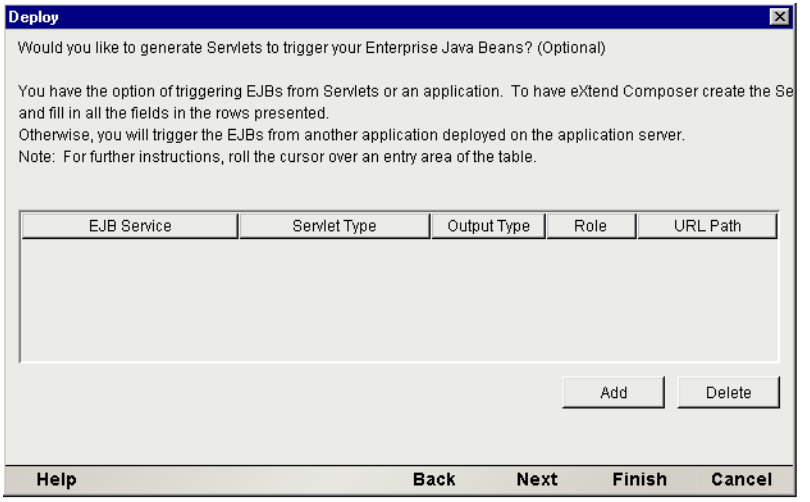
iiop://stratus/RMI/WebStore/ProductInquiry

Transaction Attribute

必要な場合、プルダウンリストから、コンテナに適用する EJB のトランザクション属性を選択します。必要ない場合、［Not Supported］を選択します。

EJB トリガパネル

このパネルを使用して、EJB ベースのサービストリガのサーブレット呼び出しを設定します。「サービストリガについて」の節で説明したとおり、サーブレット-EJB ベースのサービストリガは、2 つのオブジェクトを使用して、URI 関連付けのトリガ機能、XML への入力変換、およびサービスの起動を行います。EJB は、サービスの起動を行い、フロントエンドのサーブレットは、URI 関連付け、入力変換、および EJB の起動を行います。同じステージングディレクトリを使用して、プロジェクトがすでに配備されている場合、最新の配備で使用された値が次のパネルに表示されます。配備される各サーブレット-EJB ベースのサービストリガに対して、［**Add**］ボタンをクリックして、次のパラメータを指定します。



EJB Service

前のパネルで定義した EJB ベースのサービストリガが、各行の最初のフィールドに表示されます。フィールドをクリックすると表示されるドロップダウンリストを使用して、JNDI パスにより EJB ベースのサービストリガを選択します。

Servlet Type

サーブレットタイプは、Params (URL/Form)、XML (MIME multipart)、XML (HTML form field)、および XML (HTTP POST) の 4 種類あります。それぞれのサーブレットタイプは、サービスへの入力データを受け入れる方法が異なります。相違点については、前の節「サーブレットベースのサービストリガ」の表で簡単に説明しています。

Output Type

プルダウンメニューから、[**XML**] または [**HTML via PI**] を選択します (サービスの出力に適した方)。(処理命令を介した) スタイルシートの結果として HTML を XML データに出力する場合、[**HTML via PI**] を使用します。このオプションを使用すると、exteNd により、出力に適した MIME タイプが設定されます。

Role

サービストリガへのアクセスを許可する J2EE 役割の名前を入力します。 入力した役割名にマップされるグループの個人またはメンバーだけが、サービストリガにアクセスできます。役割の名前に一致するサーバグループを作成し、役割のアクセス機能を利用できるようにする必要があります。

セキュリティ役割の定義はオプションです。アクセス制限が必要ない場合、[Role] フィールドは空白のままにしてください。

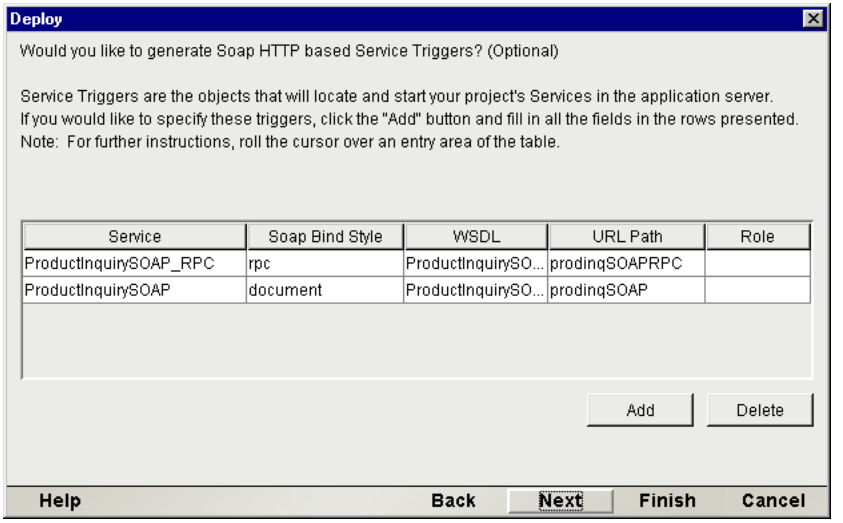
URI Path

[URI Path] は、この EJB トリガサーブレットに割り当てられている URI の固有な名前を提供します。この名前は、Websphere サーバノード名および Websphere Web アプリケーション名とともに、このサーブレットにアクセスする完全な URI を形成します。たとえば、[URI Path] に「**WebStore/ProductInquiry**」が指定されていて、サーバ名が「**stratus**」で、データベース名が「**InfoDB**」の場合、完全な URI は次のようになります。

http://stratus/InfoApp/WebStore/ProductInquiry

SOAP ベースのトリガパネル

サーブレットベーストリガで、着信データをラップする SOAP を「認識」させる場合、このパネルを使用します。



Service

［Add］ボタンをクリックして、サービスをリストに追加します。［Service］フィールドの各行に対して、フィールドをクリックすると表示されるドロップダウンリストからサービスを選択します。

SOAP Bind Style

SOAP では、RPC スタイルまたはドキュメントスタイルのメッセージが可能です。ドロップダウンメニューから［rpc］または［document］を選択します。これらのオプションの詳細については、http://www.w3.org/TR/SOAP を参照してください。

Body/Namespace

SOPA 本文に含まれるメッセージに割り当てられるネームスペースプリフィックスを指定できます。

WSDL

このフィールドでは、ドロップダウンから、サービスに関連する WSDL オブジェクトを選択します。

URL Path

このフィールドでは、生成されるサーブレットを参照するURL別名を指定します。

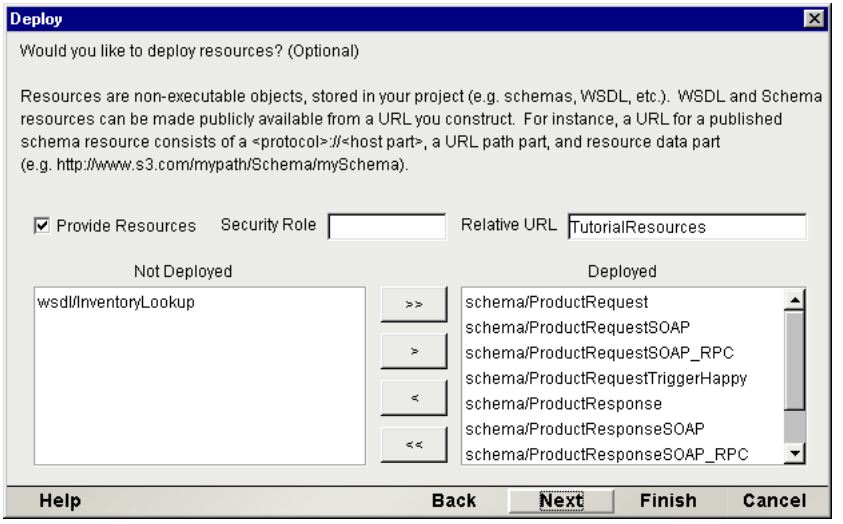
Role

サービストリガへのアクセスを許可する J2EE 役割の名前を入力します。入力した役割名にマップされるグループの個人またはメンバーだけが、サービストリガにアクセスできます。役割の名前に一致するサーバグループを作成し、役割のアクセス機能を利用できるようにする必要があります。

セキュリティ役割の定義はオプションです。アクセス制限が必要ない場合、［Role］フィールドは空白のままにしてください。

リソース配備パネル

パブリック URI を介してアクセスできるようにプロジェクトリソース (WSDL や XSD ファイルなど) を配備する場合、このパネルを使用します。



最初に、［**Provide Resources**］チェックボックスをオンにします。［**Relative URL**］フィールドが使用できるようになります。このフィールドでは、リソースチェインを参照する URI フラグメント (ホストパス名に相対的) を入力します。次に、配備するリソースを左側のリストから選択し、ダイアログボックスの中央にある転送ボタンを使用して、右側のリストに移します。

オプションで、サービストリガへのアクセスを許可する J2EE 役割の名前を入力します。

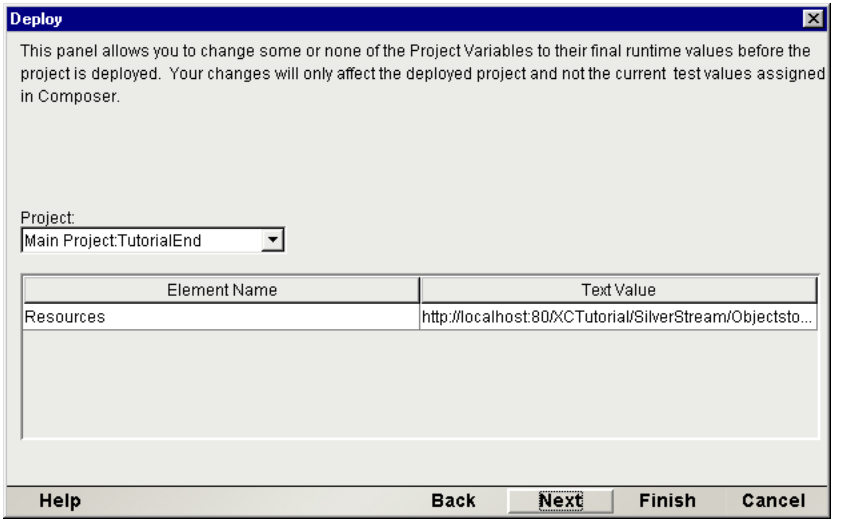
［Relative URL］は、URL のホスト部分に対して相対的です。ホストの部分は、Deployment Wizard の最後のパネル (次を参照) の上部にあるテキストフィールドに入力する URL と同じです。*前のダイアログボックスでこのルート URL を入力する必要はありません。*［Relative URL］で入力する必要のある箇所は、前の選択リストで示されるリソースのリソースディレクトリ (またはサブパス) だけです。たとえば、**http://www.somedomain.com** から提供し、リソースが **/resources/soap** にある場合、これを［Relative URL］に入力します。**wsdl/SOAP-WSDL** をパブリックリソース (前のスクリーンショットを参照) として使用する場合、次の箇所で使用できます。

http://www.somedomain.com/resources/soap/wsdl/SOAP-WSDL

プロジェクト変数リマッピングパネル

プロジェクト変数の値を配備目的で変更する場合、このパネルを使用します。たとえば、設計時に、配備ドライブまたはディレクトリにあるワークファイルを参照するプロジェクト変数を定義します。その後、最終的な配備段階で、実際の配備リソースのサーバの位置に基いて、これらのプロジェクト変数に新しい値を割り当てます。このパネルでは、このような操作を行うことができます。

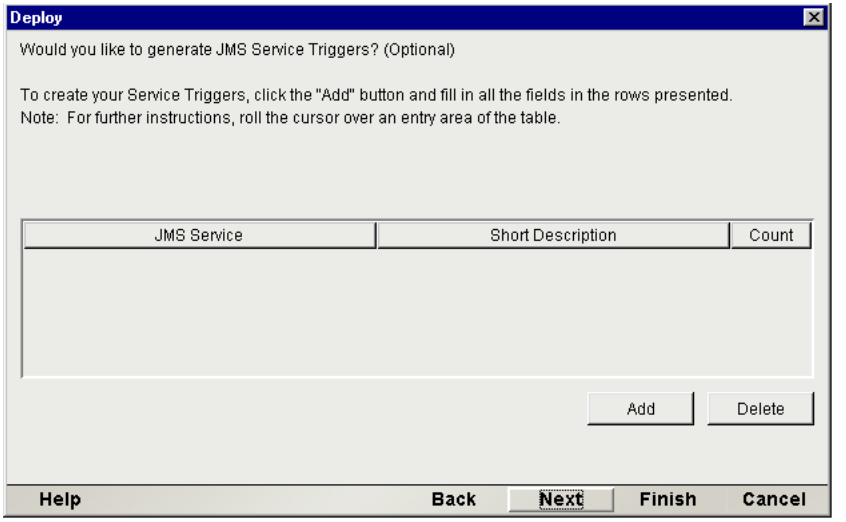
注記： プロジェクト変数は、**PROJECT.xml** ファイル (プロジェクトの **.spf** ファイルと同じディレクトリ) に保存されます。このパネルを使用してプロジェクト変数値を変更すると、サーバに書き込まれた **PROJECT.xml** ファイルが、作業ディレクトリの **PROJECT.xml** ファイルと一致しなくなります。通常、これで問題ありません。



JMS サービストリガパネル

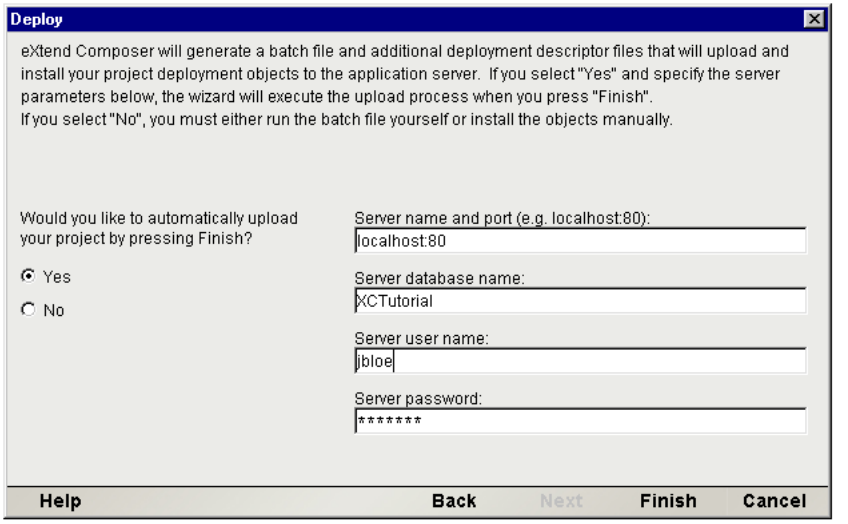
JMS Connect をインストールしていて、プロジェクトに 1 つ以上の JMS サービスが含まれている場合、JMS サービストリガパネルが表示されます (次を参照)。JMS サービスを配備する場合、［**Add**］ボタンをクリックして、［**JMS Service**］のプルダウンメニューから［JMS Service］を選択します。次に、その右側にテキスト記述を入力します。「同時リスナ」を配備する数を示す値を［*Count*］に入力します。

配備する JMS サービスの数だけ、［**Add**］ボタンを繰り返しクリックします。終了後、［**Next**］をクリックして、Deployment Wizard の最後のパネルに移動します。



自動インストールパネル

このパネルを使用して、Deployment Wizard が WebSphere Application Server に対して、配備オブジェクトを自動的にアップロードおよびインストールするかどうか定義します。Composer によって最終的な配備の詳細が処理されるよう選択すると (これはデフォルトの動作で、ダイアログボックスのチェックボックスがオンになっています)、サーブレットがサーバで起動し、配備を完了する一連のブラウザ画面が表示されます (次を参照)。チェックボックスをオフにすると、ステージング領域からアプリケーションサーバに、手動でオブジェクトを配備します。



配備が実行されるサーバ名とポートを入力し、「Launch browser to complete deployment」というラベルが付いたチェックボックスをオンにします。次に、［**Finish**］をクリックします。Web ブラウザがロードされ、コンソール画面が表示されます。

手動による配備オブジェクトのインストール

この節では、*xcs-deploy* ユーティリティを使用して、Composer の配備 JAR ファイルを WebSphere に配備する方法を説明します。

注記： *xcs-deploy* **ユーティリティは、**exteNd Server **インストールの /bin ディレクトリにあります。**

Composer Deployment Wizard は、配備 JAR ファイルを自動的に生成し、必要に応じて (前ページで説明したプロセスを通じて) 自動インストールを実行します。しかし、プロジェクトを手動で配備しなければならない特別な状況が発生することがあります。手動での配備については、この節で説明する *xcs-deploy* ユーティリティを使用します。

➢ exteNd JAR ファイルを手動で配備する

1. exteNd Composer を実行して、Deployment Wizard でプロジェクトをサーバにアップロードしたくないと指定し(最終パネル)、すべての手順を完了します。
2. 生成された配備 JAR ファイルを、ステージングディレクトリ (Deployment Wizard の最初のパネルで指定済み) から WebSphere サーバのディレクトリにコピーします。

3 ノード名、およびそのノード上の、配備 JAR ファイルのコンテンツを配備する Application Server を記録しておいてください (この情報は、*xcs-deploy* ユーティリティの引数に使用します)。

4 Application Server内の、サーブレットサービストリガを配備したいWebアプリケーションを記録しておいてください(この情報は*xcs-deploy*ユーティリティの引数に使用します)。

5 Application Server内の、EJBサービストリガを配備したいEJB Containerを記録しておいてください (この情報は *xcs-deploy* ユーティリティの引数に使用します)。

6 配備 JAR ファイル名、ノード名、Application Server、Web Application、および EJB Container をパラメータとして指定して、*xcs-deploy* ユーティリティを実行します。使用命令構文は、次のとおりです。

```
xcs-deploy -jar <JAR Name> -adminNodeName <Node name> -appServer
<Server Name> -webApp <Web Application Name> -ejbContainer <EJB
Container Name>
```

ここで、

`-jar` は、プロジェクト JAR の名前です。

`-adminNodeName` は、WebSphere の管理サーバのノード名 (`localhost` ではない) です。

`-appServer` は、配備先の Application Server の名前です。

`-webApp` は、Web Application 名です。

`-ejbContainer` は、EJB が配備される EJB コンテナの名前です (適用可能な場合)。

例 :

```
xcs-deploy -jar D:\staging\new-ws\ws-srvc.jar -adminNodeName
lovejoy -appServer "Default Server" -ejbContainer "Default
Container" -webApp xcs_services
```

注記 : 複数の単語のパラメータ値は、`-appServer` 値 `"Default Server"` のように、引用符で囲む必要があります。

注記 : スイッチ (およびそのパラメータ) の順序は関係ありません。

7 WebSphere Administrative Console を実行します。

8 コンポーネントが配備されたApplication Serverを探します。Application Server を停止してから再起動すると、変更内容が有効になります。

The *xcs-deploy* ユーティリティは、次のアップデートを自動的に実行します。

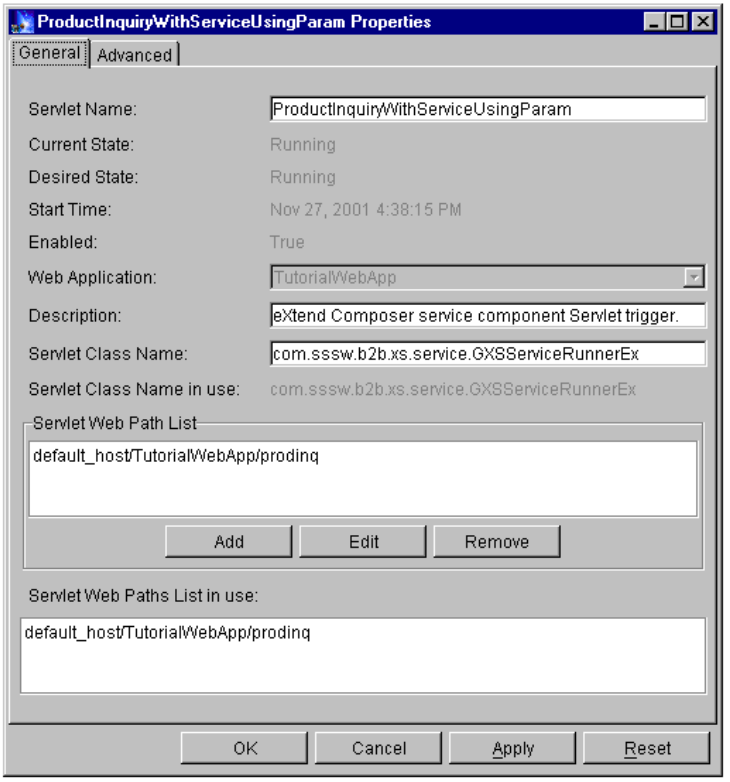
- Application Server の CLASSPATH を、配備 JAR ファイルの場所にアップデートします。
- Web Application 内の、すべてのサーブレットおよび EJB-Servlet ベースのサービストリガの定義を追加します。
- すべての EJB ベースのサービストリガと EJB Container(適用可能な場合) の説明を含む、EJB の JAR ファイルを作成し配備します。
- プロジェクト JAR を、ステージングディレクトリからターゲットサーバにコピーします。

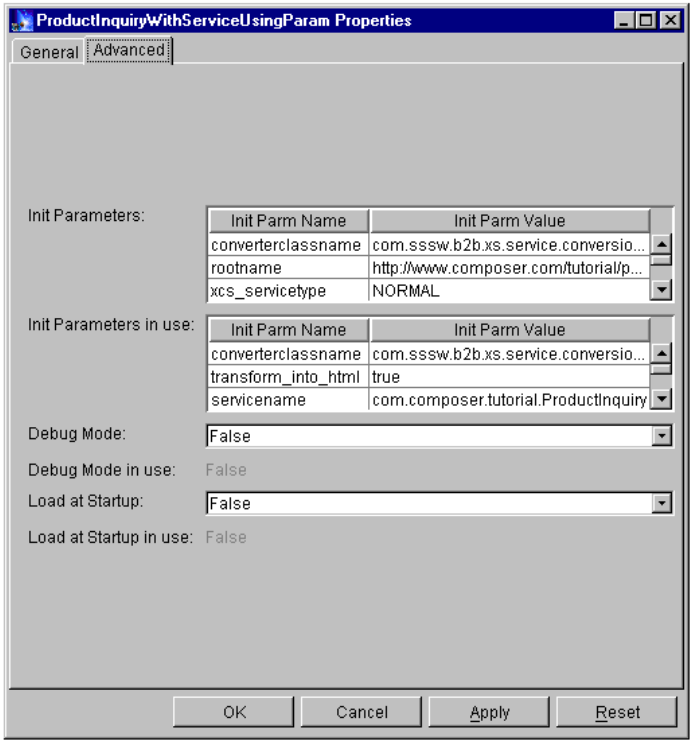
サーブレットの定義

exteNd で生成されたサーブレット名は、eployment Wizard でそのサービスのために選択した、サービスコンポーネントの名前、および D サービストリガのタイプに基づいています。次に示す画面の、

SimpleProductInquiryWithServiceUsingParam

と名前が付いたサーブレットベースのトリガは、サービス名、**SimpleProductInquiry**、およびサーブレットのタイプ : **Params (URL/Form)** に基づいて生成されました。.


ProductInquiryWithServiceUsingParam Properties
General
Advanced
Servlet Name:
ProductInquiryWithServiceUsingParam
Current State:
Running
Desired State:
Running
Start Time:
Nov 27, 2001 4:38:15 PM
Enabled:
True
Web Application:
TutorialWebApp
Description:
eXtend Composer service component Servlet trigger.
Servlet Class Name:
com.sssw.b2b.xs.service.GXSServiceRunnerEx
Servlet Class Name in use:
com.sssw.b2b.xs.service.GXSServiceRunnerEx
Servlet Web Path List
default_host/TutorialWebApp/prodinq
Add
Edit
Remove
Servlet Web Paths List in use:
default_host/TutorialWebApp/prodinq
OK
Cancel
Apply
Reset



Administrative Console の「Edit Servlet Attributes」ダイアログボックス (次を参照) の Advanced ペインでは、初期化時に、サーブレットを配備されたサービスコンポーネントにバインドすることができる、初期化パラメータを検査することができます。

EJB-Servlet の定義

EJB-Servlet 名は、Deployment Wizard で指定したサーブレットタイプ (**ServiceUsingParam** など) とともに、EJB サービスコンポーネントの JNDI 名から生成されます。

EJB-Servlet サービストリガの初期化プロパティ (Administrative Console の「Edit Servlet Attributes」ダイアログボックスの［Advanced］ペインに表示) には、ランタイム時にサーブレットを EJB にバインドするために必要なパラメータが含まれます。デフォルトでは、EJB-Servlet トリガは、サーブレットと同じノード上の EJB を参照するように設定されています。さまざまな初期化パラメータを使用すると、トリガを、異なるネームサーバ、および (または) 異なるコンテキストファクトリのエンジンに参照させることができます。ネームサーバに参照するためのパラメータには `providerurl`、コンテキストファクトリのエンジンに参照するためのパラメータには `contextfactory` という名前が付いています。「Edit Servlet Attributes」パネルで、これらのパラメータの値を入力します (前の図を参照)。

`providerurl` の値は、EJB を取得するためにトリガがアクセスするネームサーバのホスト名およびポートを表します。値の形式は以下のとおりです。

iiop://<hostname>:<port>

たとえば、EJB が、**stratus** というホスト名および **900** というデフォルトのポートを持つネームサーバにある場合、`providerurl` の値は次のようになります。

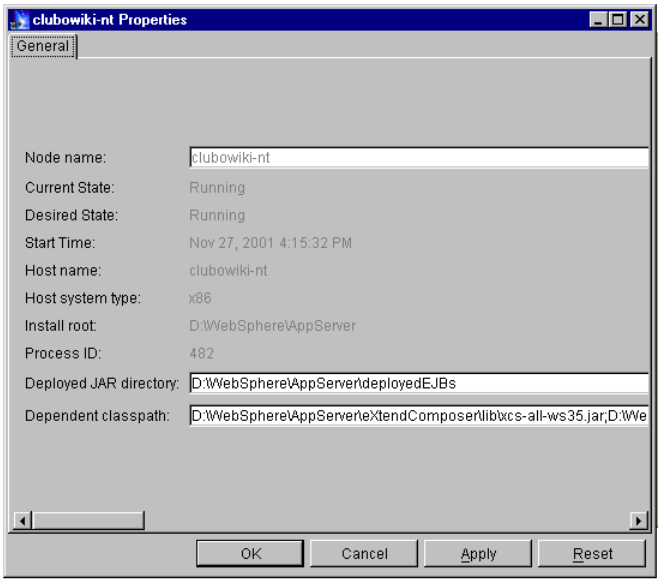
iiop://stratus:900 or iiop://stratus

通常、WebSphere Advanced Edition サーバで配備された EJB にアクセスする場合、2 番目のプロパティ、`contextfactory` の値は変更する必要はありません。デフォルトでは、このプロパティの値の設定は、**com.ibm.ejs.ns.jndi.CNInitialContextFactory** です。IBM Component Broker で配備された EJB にアクセスする場合は、プロパティの値を **com.ibm.ejb.cb.runtime.CBCtxFactory** と指定する必要があります。

EJB の定義

EJB の生成には、2 段階の手順があります。まず、*xcs-deploy* ユーティリティは、EJB として配備された各 Composer サービスの連続した記述子クラスを含む、EJB 配備 JAR ファイルを作成します。EJB JAR ファイルの名前は、値「EJB-」が付いた Composer 配備 JAR ファイルの名前です。たとえば、Composer 配備 JAR ファイルの名前が **AccountingService.jar** である場合、対応する EJB JAR ファイルは **EJB-AccountingServices.jar** になります。

次に、*xcs-deploy* ユーティリティは、配備された各 EJB サービスコンポーネントのための EJB コンテナに、エントリを追加します。このアクションによって、WebSphere は、ノードのプロパティで指定されたディレクトリ「Deployed JAR directory」(次の図を参照) にある配備された EJB JAR に、必要なスタブコードを生成します。



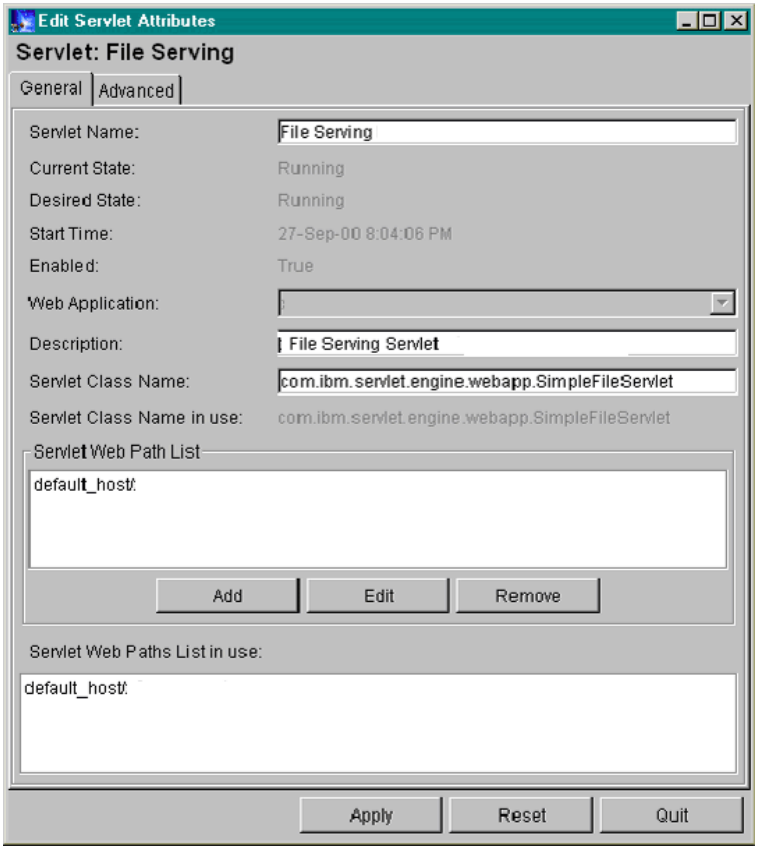
注記： EJB を異なる WebShere ノードに配備する必要がある場合は、生成された EJB 配備 JAR ファイルを、新しいノードにコピーします。そして、新しいノードの Administrative Console から、JAR ファイルの EJB を EJB コンテナに追加します。

Application Server での XML リソースの公開

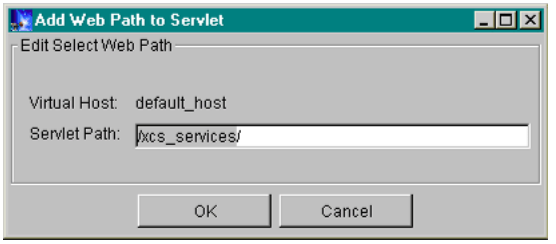
XML リソースは、プロジェクトに関連付けられている DTD、スキーマ、および XSL スタイルシートです。通常、これらのリソースは、URI を通じてアプリケーションサーバで公開する必要があります。. WebSphere では、サービストリガが置かれている同じ Web Applicatioins に、リソースを追加することができます。各 Web Application には、サーブレットパスおよび Web パスの両方があります。「Web パス」は、Web Application を含む物理的なディレクトリ、および Web Application の URI パス間の、仮想マップです。

Web Application を設定する

各 Web Application は Web パスを持ち、Web パスを使用して「リソース」を提供しますが、リソースの要求を Web パスからの検索に変換するメカニズムが存在することを確認する必要があります。IBM WebSphere には、このメカニズムを提供する File Serving Servlet があります。



サーブレットの定義を Web Application に追加する場合は、クラス名に **com.ibm.servlet.engine.webapp.SimpleFileServlet** と入力します。また、Web パスを入力する場合は、デフォルトの **/<web application path>/** を受け入れます。この「*web application path*」は、Web Application に関連するパスになっています。



リソースファイルを配備する

リソースファイルをユーザから使用可能にするには、あとは Web Application に関連する Web ディレクトリにコピーするだけです。Web Application のルートから作成されるディレクトリは、公開されるリソースの URI に反映される必要があります。

たとえば、Web Application に、次に示す Web ディレクトリを持つ **/xcs_services** という URI パスがあるとします。

/WebSphere/AppServer/hosts/default_host/webapp/xcs_services

productinquiry.xsl という名前の付いた XSL スタイルシートを、**/WebSphere/AppServer/hosts/default_host/webapp/xcs_services/style** ディレクトリに配置すると、そのスタイルシートの URI は次のようになります。

http://hostname/xcs_services/style/productinquiry.xsl

5 Composer サービスへのプログラム的なアクセス

Deployment Wizard (最後の章を参照) を使用した Composer サービスの配備は、ほぼスタンドアロン状態で「ストレートスルー処理」機能を実行する Composer サービスに HTTP を介してネットワークアクセスできるようにする場合に適しています。しかし、多くの場合は、実行可能なオブジェクト間で特別な調整を必要とする、大規模で、複雑な可能性のある J2EE プロジェクトに Composer サービスを作成します。このようなプロジェクトでは、パッケージング (WAR ファイル §EAR ファイル) およびランタイム実行の点から、オブジェクト間に密接な統合性が必要となることがあります。このような統合を実現するには、Composer サービスを呼び出すためのカスタムトリガオブジェクトを作成すると役に立ちます。

カスタムトリガは、主に 2 種類の方法で作成できます。1 つ目は、exteNd Workbench を使用して、Composer サービスを呼び出すことができるサーブレットコード、JSP ページ、またはカスタム Java クラスを作成する方法です(Workbench には、Composer 指向コード生成専用のウィザードが組み込まれています。また、Workbench では、コードを WAR、EAR、および JAR ファイルにパッケージ化し、これらのオブジェクトを運用環境に配備するための柔軟なオプションが提供されています)。カスタムトリガコードを配備するもう 1 つの方法は、任意の Java IDE を使用して、Composer の「フレームワークオブジェクト」(Composer Enterprise Server に含まれています) を呼び出すコードを作成することです。フレームワークオブジェクトを使用すると、トリガコードの作成を低レベルで実現できます。

いずれの配備方法を選択する場合でも、Composer サービスをインスタンス化および呼び出すメカニズムを理解しておくことは重要です。したがって、最初にフレームワークアーキテクチャと Framework API を使用する方法について説明します。その後、Composer サービスを呼び出すことができるカスタム Java コードおよび JSP ページを作成するために、簡単な配備を支援するものとして exteNd の Workbench を使用する方法について説明します。

Composer のサービストリガ階層の拡張

exteNd の配備オブジェクトを手動で簡単に作成できるように、Novell では、Composer サービスのサービストリガの作成に使用できるフレームワークのセットを提供しています。フレームワークファイルは、Composer Enterprise インストールの **\docs\api\com\sssw\b2b\xs** パスにあります。また、JavaDoc ファイルもこのパスにあり、独自の JavaDoc を作成した場合、ファイルはここに生成されます。クラス記述の詳細については、JavaDoc ファイルを参照してください。ここでは、簡単な概要のみを示します。

フレームワークオブジェクト内のサービスコンポーネント名

フレームワークオブジェクト (サーブレットなど) 内でサービスコンポーネント名を参照する場合、配備コンテキストとサービスコンポーネント名を結び付けます。

完全修飾サービス名の例は、次のとおりです。

```
com.yourcompany.project.ProductInquiry
```

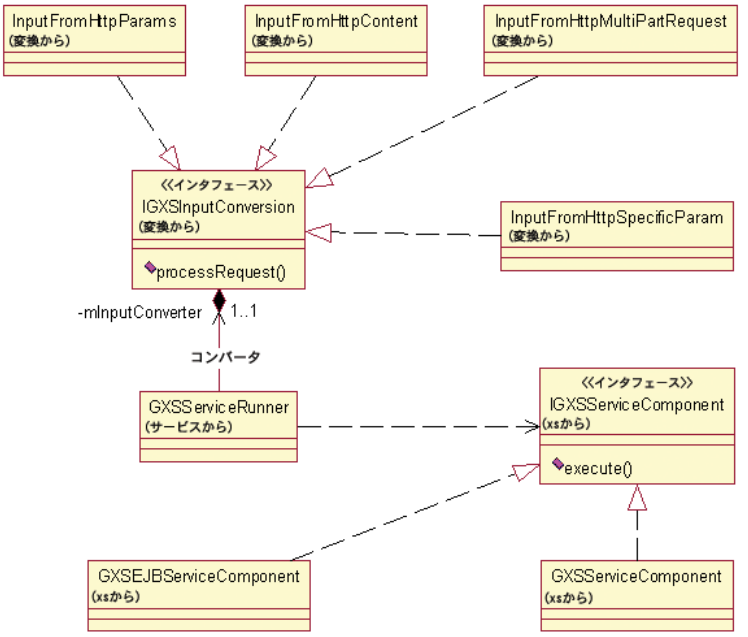
ここで、

- `com.yourcompany.project` は、配備中に指定した配備コンテキストです。
- `ProductInquiry` は、Composer サービスコンポーネント名です。

配備 JAR には、1 つまたは複数の Composer サービスが含まれます。サービストリガは、Composer サービスを開始するためにアプリケーションサーバによって受信されるリクエスト間のブローカーです。

各配備メソッドに対しては、さらにサービスをカスタムアプリケーションに統合できるようにする「ヘルパ」クラスが提供されます。

さまざまなフレームワークコンポーネントの関係は、次の図のとおりです。



サービストリガ拡張の作成方法

➢ **Web サービスに対してサービストリガ拡張を作成する**

1. exteNd Workbench またはその他の Java 開発環境を使用して、Composer Deployment Framework オブジェクトから派生した Java オブジェクトを作成します。
2. オブジェクトの作成に Composer を使用しなかった場合は、オブジェクトをインポートします。
3. オブジェクトをコンパイルします。

exteNd フレームワークには、Java サーブレットに基づいて、サービストリガの基本タイプに基本機能を提供する抽象クラスがあります。

サーブレットベースのサービストリガの作成

サーブレットの階層は、`HttpServlet` から派生した 1 つの抽象クラスに基づきます。`GXSServiceRunner` という抽象クラスは、`HttpServlet` から派生し、次の操作を実行するために必要な論理を含みます。

- Composer サービスコンポーネントで抽象メソッド `getServiceName()` の呼び出しによってすべてのリクエストが実行されることを認識する。

- サービスコンポーネントの実行前に抽象メソッド `processRequest()` を、実行後に `processResponse()` を呼び出す。

フレームワークを拡張して独自のトリガタイプを作成するには、`GXSServiceRunner` に準拠するオブジェクトと `IGXSInputConversion` インタフェースを実装するコンバータオブジェクトの両方を作成する必要があります。2 番目のオブジェクトは、`HttpServletRequest` を XML 入力ドキュメントに変換できます。

サービス実行クラスは、抽象メソッド `processRequest()` を実装する必要があります。

また、コンバータインタフェースを実装するクラスは、サービス実行側のさまざまな「ゲッタ」メソッドを呼び出してプロパティを取得できるように、`IGXSServiceRunner` をパラメータとして受け取るコンストラクタを定義する必要もあります (新しいアークテクチャでは、どの入力タイプが受け入れられ、サービスがサーブレットまたは EJB のどちらから呼び出されるかを決定する情報が、サービスプロパティに含まれています。詳細については、表 5-1 を参照してください)。

サービスプロパティ名	説明	初期化パラメータ	GXSServiceRunner メソッド
SERVICE_NAME	Composer サービスコンポーネントの名前	servicename	getServiceName()
ROOT_NAME	入力ドキュメントとみなされるルートノード	rootname	getRootName()
JNDI_NAME	Composer サービスコンポーネントに対する EJB ホームインタフェースの JNDI 名	jndiname	getJndiServiceName()
CONVERTER_CLASS_NAME	HTTP リクエストを XML ドキュメントに変換するために使用されるべきクラス	converterclassname	getConverterClassName()
PARAM_NAME	入力 XML ドキュメントを含むパラメータの名前	xcs_paramname	getxcsParamName()

SERVICE_TYPE	サービスコンポーネントリファレンスがEJB または NORMAL であるかどうか	xcs_servicetype	getServiceType()
PROVIDER_PARAM	JNDI プロバイダ URI	providerurl	
CONTEXT_FACTORY	JNDI コンテキストファクトリ	contextfactory	
HTML_INDICATOR	出力ドキュメントがHTMLとして表示されるかどうかを指定するために使用されるインジケータ	transform_into_html	getOutputHTMLIndicator()
OUTPUT_XSL	出力ドキュメントをHTMLに変換する場合にスタイルシートのURIを提供する。XSL処理命令が出力XML ドキュメントに組み込まれていない場合にだけ必要。	output_xsl_url	getOutputXSL()

JAR ファイルで新しいサービストリガをパッケージ化すると、トリガを利用するアプリケーションサーバまたはWebアプリケーションのCLASSPATHを更新します。サーブレット定義を作成して新しいサービストリガのインスタンスを配備する場合は、サーブレットのクラス名をトリガクラスの名前に設定します。初期化パラメータでは、`servicename` パラメータを追加し、値を Composer サービスコンポーネントの修飾名に設定します。

GXSServiceRunnerEx

`GXSServiceRunnerEx`クラスの抽象メソッド`processRequestEx()`の簡単な説明は、次のとおりです。

String[] processRequestEx(HttpServletRequest)

これは、現在のサーブレットリクエストを受け取るメソッドです (その結果、要求側クライアントによりサーブレットに渡されるパラメータおよびその他のデータを保持します)。このメソッドは、サーブレットがバインドされている Composer サービスコンポーネントを実行するときに使用される XML ドキュメントを含む String 配列を返す必要があります。リクエストの処理中に、XML ドキュメントを作成する情報がない場合は、単に `null` が返され、Composer サービスコンポーネントは、入力 XML ドキュメントがなかった場合のように動作します。

カスタムサーブレット -EJB ベースのサービストリガの作成

EJB では、接続プール、トランザクション、および持続性をアプリケーションサーバで自動的に管理できます。前に説明したとおり、サーブレットベースおよび EJB ベースのサービストリガは結合して、強力で柔軟な配備モデルを提供できます (たとえば、異なるサーバ上にある exteNd Web サービスおよびサーブレットトリガ)。

EJB ベースのサーブレットトリガの開始点は、`GXSServiceRunner` です。これには、SERVICE_TYPE プロパティを検査するゲッタメソッドが含まれます (Bean ベースサービスの場合は「EJB」になります)。サービスコンポーネントは、`GXSEJBService` です。これは、`GXSServiceFactory` ヘルパクラスを使用してのみインスタンス化できます。

フレームワークを拡張して独自の EJB トリガタイプを作成するには、`HttpServletRequest` を XML 入力ドキュメントに変換できるコンバータオブジェクトを提供する必要があります。このためには、`IGXSInputConversion` インタフェースを実装するクラスを作成します。クラスは、抽象メソッド `processRequest()` を実装する必要があります。また、このインタフェースを実装するクラスは、サービス実行側のさまざまな「ゲッタ」メソッドを呼び出してプロパティを取得できるように、`IGXSServiceRunner` をパラメータとして受け取るコンストラクタを定義する必要もあります。

クラスは、別々のライブラリにパッケージ化する必要があります。このようにパッケージ化すると、新しいサービストリガテンプレートから継承する単純な配備時サービストリガを作成できます。このような配備時サービストリガでは、EJB ベースのサーブレットを Composer サービスコンポーネントにバインドする `getJndiServiceName()` メソッドの実装だけを提供する必要があります。

サービストリガの新規作成

exteNd Server フレームワークでは、Composer サービスを Java クラスに統合できるようにするファクトリおよびインタフェースがすべて提供されます。このフレームワークでは、Composer サービスを表す簡単なインタフェースが提供されます。また、Composer サービスを表すオブジェクトをインスタンス化できるようにする静的メソッドのセットもファクトリクラス内に提供されます。

exteNd サービスインタフェース : IGXSServiceComponent

Composer サービスの実行を可能にする主な Java 要素は、`IGXSServiceComponent` というインタフェースです (**com.sssw.b2b.xs package** にあります)。このインタフェースにより、さまざまなパラメータを受け付けて返す 4 つの `execute()` メソッドが提供されます。インタフェースが 3 つの形式のいずれかの入力ドキュメントでインスタンス化されると、Composer サービスを実行でき、その出力ドキュメントが返されます。

exteNd サービスファクトリ : GXSServiceFactory

フレームワーク内には、`GXSServiceFactory` というファクトリクラスがあります。このクラスを使用すると、Composer サービスの作成が簡単になります。また、Composer サービスの入力および出力ドキュメントをより柔軟に扱うことができる変換メソッドがプログラマに対していくつか提供されます。

新しいサービスインタフェースをインスタンス化するには、ファクトリの `createService()` メソッドを呼び出します。文字列でサービスの修飾名を指定します。次の例では、**com.acme.inventory** の配備コンテキストで **ProductInquiry** という Composer サービスを実行します。

```
public void doSomething() throws GXSException
{
    String serviceName = "com.acme.inventory.ProductInquiry";
    String inputDoc = "";
    String outputDoc = null;
    IGXSServiceComponent myService = null;

    // set the inputDoc to a valid XML document format
    // that your service expects.

    myService = GXSServiceFactory.createService( serviceName );
    outputDoc = myService.execute( inputDoc );

    // Do something with the output document.
    return;
}
```

exteNd サービス Bean: GXSServiceComponentBean

サービスコンポーネントをインスタンス化する手順 (前に説明) をカプセル化する JavaBean を使用するために、簡単な方法が与えられています。Bean では、次のメソッドが提供されます。

- サービスコンポーネント名に対するセッタメソッド
- 入力ドキュメントに対するセッタメソッド
- 名前が付けられたサービスコンポーネントを実行するメソッド
- 最後の実行から出力ドキュメントを取得するゲッタメソッド

Java Bean を使用したメソッドの例は、次のとおりです。

```
public void doSomethingWithABean() throws GXSException
{
String serviceName = "com.acme.inventory.ProductInquiry";
String inputDoc = "";
String outputDoc = null;
GXSServiceComponentBean myBean = new GXSServiceComponentBean();

// set the inputDoc to a valid XML document format
// that your service expects.

myBean.setServiceName( serviceName );
myBean.setInputXMLDoc( inputDoc );

myBean.execute();
myBean.getOutputXMLDoc();

// Do something with the output document.
return;
}
```

アプリケーションでのサービス EJB の使用

企業での要件が増大し、分散アプリケーションが必要になった場合は、Composer サービスを EJB として配備しなければならない可能性があります。アプリケーションを開発して、これらのリモートサービスを使用する場合、`IGXSServiceComponent` ではなく EJB リモートサービスを使用します。

開発およびランタイム環境のセットアップ

Composer サービスは WebSpherer ノードで EJB として配備されるため、開発およびランタイム環境に必要な WebSphere JAR ファイルが含まれていることを確認します。必要な JAR ファイルの完全なリストについては、WebSphere Advanced Server ドキュメントの「*How Do I, Deploy Enterprise Beans, In Advanced Edition*」を参照してください。

WebSphere では、JNDI ホームインタフェースを検索するために 2 つのプロパティに依存しています。最初のプロパティ `javax.naming.Context.PROVIDER_URI` では、EJB クライアントがアクセスするネームサーバのホスト名とポートを指定します。プロパティの値は、**iiop://<hostname>:<port>** という形式になります。たとえば、EJB クライアントがアクセスするネームサーバのホスト名が **stratus** で、登録されたポートが **900** の場合、プロパティの値は **iiop://stratus:900** になります。

2 つ目のプロパティ **javax.naming.Context.INITIAL_CONTEXT_FACTORY** では、EJB クライアントで使用しなければならないネームサービスを指定します。WebSphere Advanced Edition にある EJB の場合、プロパティの値は **com.ibm.ejs.ns.jndi.CNInitialContextFactory** です。

EJB のホームインタフェースおよびリモートインタフェースの取得

IGXSEJBServiceComponent という EJB リモートインタフェースは、**com.sssw.b2b.xs.ejb package**にあります。Composerサービスを EJBとして配備する場合は、「servicename」という環境設定を追加します。EJB では、環境リファレンスを検索して、バインドされる Composer サービスを確認します。また、EJB 配備時には、JNDI 名を EJB に割り当てます。EJB のホームインタフェースに対するリファレンスを取得するために使用するのは、修飾 Composer サービス名ではなく、この名前です。Composer サービスの EJB ホームインタフェースの名前は、**IGXSEJBServiceHome** です。

GXSServiceFactory の `createService()` メソッドのようなホームインタフェースが取得されると、`create()` というメソッドが呼び出され、その後、EJB のリモートインタフェースが返されます。リモートインタフェースには、2 つの実行メソッドが含まれます。1 つ目の実行メソッドは、パラメータを受け付けず、Composer サービスで入力ドキュメントが必要ない場合に使用します。もう 1 つの実行メソッドは、XML ドキュメントを文字列として受け付けます。`IGXSServiceComponent` で使用できるリーダまたはドキュメントバージョンは、EJB リモートインタフェースでは使用できないので注意してください。これは、このタイプはシリアル化できず、リモートメソッドで表示できないためです。

EJB ホームインタフェースを取得するファクトリ

作業を簡単にするため、**com.sssw.b2b.xs.websphere** パッケージに **GXSWSEJBAccessor** というファクトリクラスがあります。このクラスには、WebSphere Administration Server から EJB のホームインタフェースを取得するメソッドが含まれています。

次の例では、EJB の JNDI 名は **com/acme/inventory/ProductInquiry**、WebSphere ネームサーバのホスト名は **main.server**、ポートは 900 (インストール時のデフォルト) です。

```
import com.sssw.b2b.xs.ejb;
import com.sssw.b2b.xs.websphere.GXSWSEJBAccessor;
import java.rmi.RemoteException;
```

```
public void doSomeEJBStuff() throws RemoteException
{
    IGXSEJBServiceHome srvcHome = GXSWSEJBAccessor.getHomeBean(
        "com.sssw.b2b.xs.ejb.IGXSEJBServiceHome",
        "com/acme/inventory/ProductInquiry", "main.server",
        900);
    IGXSEJBServiceComponent ejbSrvc = srvcHome.create();
    // Do something with the service component

}
```

exteNd Workbench を使用したカスタムトリガの作成

exteNd Workbench には、前に説明した種類のコードを作成するときに役立つ可能性のあるウィザードがいくつも組み込まれています。たとえば、Workbench は、Composer サービスをインスタンス化および使用する JSP ページ、カスタムサーブレット、Director ePortal クラス、およびカスタム Java クラスを作成できるようにします。これらのウィザードの詳細については、Composer マニュアルのフォルダにある付属ガイドファイル **ComposerAndWorkbench.pdf** を参照してください。

6 トランザクションの管理

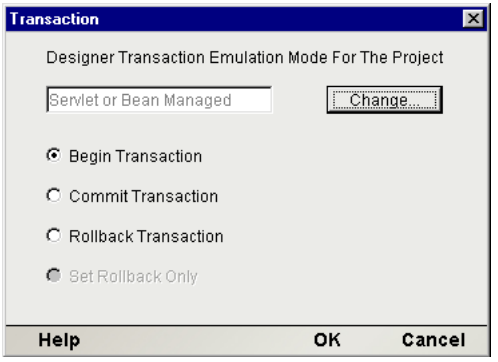
トランザクションを実行する Composer アプリケーションには、特別な計画と配備が必要です。この章では、トランザクション管理に関連する問題について説明します。

exteNd Composer でのトランザクション制御

exteNd Composer では、「Transaction アクション」は、定義された Java Transaction API (JTA) サーバ側トランザクションコマンドを呼び出すことができます。たとえば、次のようになります。

- ［*Begin Transaction*］、［*Commit Transaction*］、および［*Rollback Transaction*］コマンドは、サーブレットとしてまたは bean 管理 EJB として配備されるプロジェクトで使用できます。
- ［*Set Rollback Only*］コマンドは、コンテナ管理 EJB として配備されるプロジェクトで使用できます。

これらの選択肢は、状況に応じて有効または無効になり、Composer で新しい Transaction アクションを作成すると表示される［Transaction］ダイアログボックス (次の図を参照) から使用できます。



WebSphere Application Server についての Transaction 配備の考慮事項

前の章で説明したように、Composer プロジェクトの場合、サーブレットおよび Enterprise Java Bean (EJB) という 2 つのサービストリガメカニズムが内蔵されています。Java Transaction API (JTA) におけるトランザクションの定義方法の結果として、各メカニズムはトランザクション制御に重要な意味を持ちます。

サーブレット配備の考慮事項

JDBC 接続プールを使用したサーブレット配備は、問い合わせのみのサービスなど、複雑なトランザクション動作が必要でない場合にお勧めします。サーブレット配備には、主に次のような制限があります。

- 宣言型のトランザクション制御は使用できません。必要な場合は、代わりに EJB 配備を使用してください。
- デフォルトでは、接続プールからサーブレットへの JDBC 接続は、自動コミットが「オン」の状態に設定されています。Update、Delete、Insert の各ステートメントの後、トランザクションは自動的にデータベースにコミットされるということです。それ以降のロールバックは影響ありません。この動作を変更するには、次の 2 つの方法があります。

  1. Begin Transaction コマンドを発行し (Transaction アクションを使用)、続いて Commit または Rollback コマンド (該当する方) を使用します。

  2. 接続で「Allow SQL Transactions」チェックボックスをオンにします。詳細については、73 ページ「JDBC トランザクション制御 - User Transactions の許可」を参照してください。

  注記：ネストされたトランザクションは使用できませんが、連続するトランザクションは使用できます。

EJB 配備

Composer サービスを EJB として配備すると、柔軟性の高いトランザクション管理ができます。アプリケーションが、データをいくつかのバックエンドシステムに更新しなければならない分散トランザクション環境を必要とする場合は、EJB 配備をお勧めします。EJB 配備における exteNd の特性を説明する前に、EJB の仕様に示されているトランザクションに関する配備オプションを確認します。次のような定義を知っておくと理解しやすくなります。

「**アプリケーション**」は、トランザクションサービス (通常 EJB) を使用します。

「**コンテナ**」は、EJB が配備され実行されるコンテキストであり、アプリケーションサーバによって提供されます。

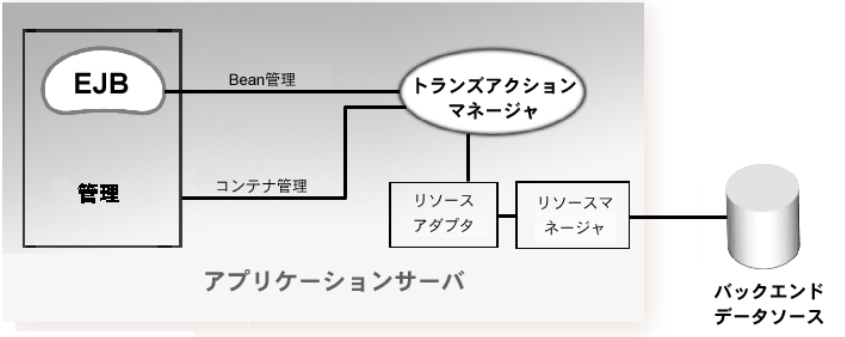
「**リソースマネージャ**」は、データベースまたはメッセージキューなど、バックエンドシステムのインタフェースです。

「**リソースアダプタ**」は、JDBC ドライバなど、リソースマネージャのインタフェースです。

「**トランザクションマネージャ**」は、アプリケーションサーバによって提供されるオブジェクトであり、すべてのプレーヤ間にトランザクションを設定し、トランザクションの流れを制御します。通常、高レベルの呼び出しが低レベルのトランザクションの呼び出し、標準の X/Open XA プロトコルにマップされます。

次の図を参照してください。

図 6-1



ステートフルな Bean 管理トランザクションはメソッド呼び出しになる場合もありますが、コンテナ管理トランザクションはすべてメソッド呼び出し制です。EJB の資料では、EJB の場合、すべてのトランザクション管理は舞台裏で行われ、アプリケーション開発者にとって重要ではないと説明されることもあります。これらの複雑な 2 段階コミット論理は、実際には自動的に実行されます (例外の場合はロールバックが自動的に実行されます) が、開発者は、必要なアプリケーションの結果が確実に得られるように、EJB トランザクションの管理方法を理解する必要があります。

Bean 管理トランザクションの管轄

EJB を Bean 管理トランザクションとして配備する場合、*UserTransaction* という簡略化されたトランザクションインタフェースを通じて、トランザクションマネージャと間接的に通信する必要があります。UserTransaction により、開始、コミット、およびロールバックなどのトランザクションコマンドを実行できます。Bean 管理として配備された場合にのみ、Bean はこれらのコマンドを使用できます。EJB がコンテナ管理として配備されるときにこれらのコマンドが発行されると、IllegalStateException がスローされます。したがって、開発者は、事前に Bean の配備方法を知っておく必要があります。

コンテナ管理トランザクションの管轄

コンテナ管理トランザクションは、宣言型のトランザクションサポートとも呼ばれ、トランザクションのサポートとして強力かつ柔軟な方法です。アプリケーションのアセンブラでは、構築後 EJB のトランザクションの動作を自由に決定できます。コンテナ管理トランザクションは、EJB により他の EJB を利用して作業を完了する場合に最も役に立ちます。この場合の典型的な例は、いくつかのエンティティ Bean を呼び出してデータベースのさまざまなテーブルを更新する、ステートレスなセッション Bean です。宣言型のトランザクション管理でこれらのトランザクションをリンクすると、コードの複雑さが大幅に減少し、コンポーネントで障害が起きた場合は自動的にトランザクションがロールバックされます。

EJB では、6 つの異なるコンテナ管理トランザクションタイプがサポートされます。6 つを区別する最も重要な違いは、「トランザクションの伝達」の概念です。トランザクションが進行中の EJB が他の EJB を呼び出すと、そのトランザクションが 2 つめの EJB に渡される場合と渡されない場合があります。トランザクションが渡される場合は、続いてトランザクションがロールバックされてから、そのトランザクションの範囲内のすべての EJB で実行された作業がロールバックされます。

コンテナ管理トランザクションのタイプには、次のものがあります。

表 6-1

トランザクションのタイプ	動作
Not Supported	使用できるトランザクションサポートはありません。
Required	トランザクションで呼び出されると、呼び出されたトランザクションが実行され、そうでない場合は作成されます。
Supports	トランザクションで呼び出されると、呼び出されたトランザクションが実行され、そうでない場合はなしで実行されます。
Requires New	常に新しいトランザクションが作成されます。呼び出し側のトランザクションは、これが完了するまで一時停止されます。

Mandatory	トランザクションで呼び出されると、呼び出されたトランザクションが使用され、そうでない場合は例外がスローされます。
Never	トランザクションで呼び出されると、例外がスローされます。

コンテナ管理トランザクションでは、いずれのタイプのコミットも呼び出すことはできません。ユーザは、EJB コンテキストで `setRollbackOnly()` メソッドを呼び出すことによって、ロールバックを開始できます。しかし、この呼び出しは、特定の状況にのみ使用できます。アプリケーションが、Bean 管理 EJB、またはトランザクションサポートのないコンテナ管理 EJB として配備されると、`setRollbackOnly()` への呼び出しの結果は `java.lang.IllegalStateException` になります。

コンテナ管理トランザクションは、異種環境で複雑なトランザクション管理を実行する、非常に強力なメカニズムです。このように複雑な分散環境には、バックエンドのリソースマネージャ、ミドルウェアドライバ、およびアプリケーションサーバからのサポートが必要です。

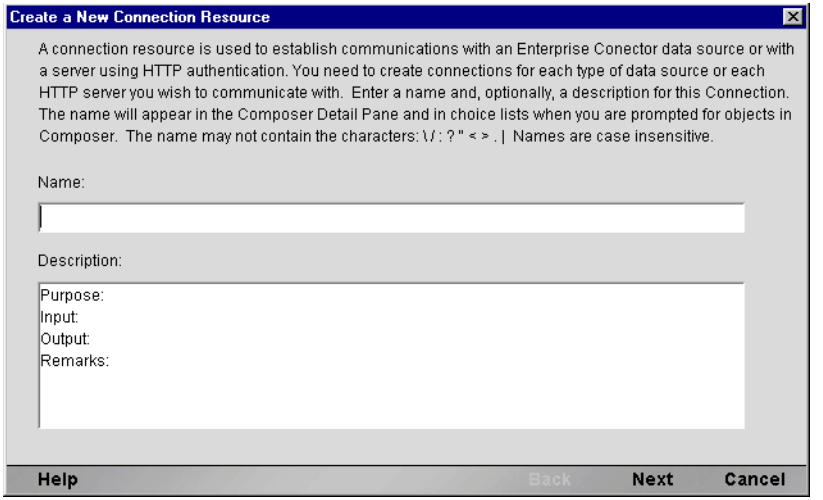
EJB 配備の考慮事項

EJB 配備は、複雑なトランザクションの動作が必要である場合にお勧めします。デフォルトでは、配備モードは Deployment Wizard の現在のトランザクションエミュレーションモード (［Advanced］タブを使用して、［**Tools**］、［**Configuration**］で設定) の選択に基づきます。選択したエミュレーションモードが Bean 管理の EJB 配備である場合は、Deployment Wizard によりこのタイプの配備が作成されます。それ以外の場合はデフォルトの Transaction Not Supported によるコンテナ管理の配備になります。Deployment Wizard の、EJB ベースのサービストリガパネルで［Trnasaction Attribute］フィールドのプルダウンメニューを使用すると、［Transaction Not Supported］から［Mandatory］、［Supports］、［Require New］、または Bean 管理またはコンテナ管理トランザクションに有効な他の選択肢に簡単に変更できます (第 4 章を参照)。

JDBC トランザクション制御 - User Transactions の許可

まれなことですが、トランザクションの手動制御が必要な場合があります。そのような状況のために、exteNd Composer には、ユーザ制御の SQL トランザクションを許可する JDBC 接続コンポーネントに、特別なチェックボックスがあります。

注記： これは拡張オプションであり、SQL プログラミングの詳細を習熟している場合にのみ使用してください。



［Allow SQL Transactions］ボックスをオンにすると、次のことが実行されます。

- JDBC ドライバの自動コミットが「オフ」になります。
- すべてのSQLコミットコマンドおよびロールバックコマンドは、同等のJDBC接続の呼び出しに変換されます。
- JDBC コンポーネントの最後の Execute SQL Action が、コミットまたはロールバックでない場合、exteNd Server により JDBC 接続にロールバックが実行されます。この動作は、接続プールを使用する場合に必要です。プールに接続を返す場合、プールマネージャにはクリーンな接続が返される必要があります。変更がコミットされていない未完了の接続を返すと、テーブルのロック、およびトランザクションスコープの不一致など、不必要な結果が発生します。この状況を修正するため、exteNd は、未完成の接続を検出し、ロールバックの発行によってクリーンアップします。そのため、ロールバックの方がコミットよりも多少安全です。例外が発生し、コンポーネントが完全に実行されない場合、ロールバックによりデータベースは実行前の状態に復元されます。この動作では、JDBC コンポーネントアクションモデルの最後に、コミットを明示的に発行することが重要です。
- 自動コミットフラグの状態は、接続がプールに戻るすぐ前に、トランザクションの最後に回復されます。

［Allow SQL Transactions］ボックスをオンにする場合、Composer サービスをサーブレットとして、またはコンテナ管理における Not Supported トランザクションモードの EJB として配備することをお勧めします。さらに、コミットまたはロールバックを、JDBC コンポーネントの最後の SQL ステートメントとして、発行することを強くお勧めします。また、Try/On Error ブロックで、JDBC コンポーネントのアクション全体を囲み、例外を探すこともお勧めします。

注記： データベースドライバは反応が異なる場合があるので、アプリケーションを配備した状態でテストし、トランザクションの動作を検証してください。

参照

EJB ホームページ - **http://java.sun.com/products/ejb**

JTA ホームページ - **http://java.sun.com/products/jta**

A 配備オブジェクトのコンテンツ

プロジェクト JAR

Deployment Wizard によって、配備時に Composer で作成したプロジェクトに関連するすべてのxObjectを含むJARファイル、(プロジェクト変数を含む)**PROJECT.xml** ファイル、および **xc_deployment_info.xml** と呼ばれるファイルが作成されます。

注記： プロジェクトディレクトリ (およびネストされたサブディレクトリ !) 内のすべてのファイルは、プロジェクト JAR に含まれます。

xObject は、xObject ファイルの名前に付けられた 2 部分のパスから構成される命名規則を使用して、JAR に保存されます。

パスの前部は、「配備コンテキスト」と呼ばれる固有の名前です。これには、任意の名前を使用できます。完全に重複が防止されます。つまり、配備コンテキストは、同じアプリケーションサーバのデータベース内に存在する異なる Composer プロジェクトで同じ名前となる 2 つの Composer サービスを識別するために使用されます。

パスの後部は、ハードディスク上の元の Composer オブジェクトと同じディレクトリ構造となります。Composer プロジェクトのディレクトリ構造は、プロジェクトの名前であるルートディレクトリに加え、作成された各 xObject タイプ (つまり、JDBC、マップ、接続、関数、スクリプト、サービス、コードテーブルなど) に対するサブディレクトリから構成されます。**LookupInventory** という名前の付いた JDBC コンポーネントを使用した、チュートリアルと呼ばれる Composer プロジェクトについて考えてみます。ディスクのディレクトリ構造またはファイル構造には、次のものが含まれます。

{ プロジェクトのペアレントディレクトリ **}\Tutorial\JDBC\LookupInventory.XML**

パスの最後の部分は、xObject の名前です。

例 :

com.yourcompany.project.jdbc.LookupInventory

ここで、

- **com.yourcompany.project** は、配備コンテキストです。
- **jdbc** は、オブジェクトタイプです。
- **LookupInventory** は、xObject です。

B Deployment Framework API マニュアル

Composer Deployment Framework の Java Doc バージョンは、アプリケーションサーバでexteNdインストールプログラムにより作成されるインストールディレクトリ (次を参照) にあります。

{exteNd インストールディレクトリ }\Docs\api

exteNd の最新バージョンにおけるすべての Framework のクラスとメソッドの詳細については、Java Doc マニュアルを参照してください。

クラスおよびインタフェース

com.sssw.b2b.xs.GXSEJBService:

これは、サービスコンポーネントの EJB バージョンです。このクラスは、実装するインタフェースで必要なすべてのメソッドを提供します。ただし、シリアル化できないパラメータ (String 以外すべて) を持つ実行メソッドの場合は、入力パラメータを String に変換してから、EJB に対して実行します。このクラスは、`GXSServiceFactory` ヘルパクラスを使用してのみインスタンス化できます。

com.sssw.b2b.xs.service.conversion.IGXSInputConversion:

これは、`HttpServletRequest` から String フォームの XML ドキュメントへの変換に必要なメソッドを定義するインタフェースです。実装を行うクラスが提供しなければならないメソッドは、`processRequest()` です。また、このインタフェースを実装するクラスは、`IGXSServiceRunner` をパラメータとして受け取るコンストラクタを定義する必要があります。これにより、変換クラスでは、処理の実行に必要なプロパティを取得できるようになります。

com.sssw.b2b.xs.service.conversion.GXSInputFromHttpParams:

これは、`HttpServletRequest` パラメータ (URI パラメータまたは送信されたフォームフィールドとして指定されたパラメータ) を XML ドキュメントに変換します。この XML ドキュメントでは、サービスの定義に使用されたルート名を使用します。

com.sssw.b2b.xs.service.conversion.GXSInputFromHttpContent:

これは、指定された `HttpServletRequest` から InputStream を開き、要求バッファ (XML ドキュメントであることを期待する) のコンテンツを取得します。SOAP サーバと、exteNd アクションの XML Interchange では、この形式で XML ドキュメントを提供できます。

com.sssw.b2b.xs.service.conversion.- GXSInputFromHttpMultiPartRequest:

このクラスは、「multipart/request」の HTML フォームタイプを使用して XML ファイルを受信します。また、特定のファイルパラメータを検索し、入力 XML ドキュメントとして使用します。ファイルの mime タイプが text または xml でない場合、このクラスは、XML ドキュメントを作成し、この XML ドキュメント内の CDATA セクションにファイルのコンテンツを配置します。

com.sssw.b2b.xs.service.conversion.- GXSInputFromHttpSpecificParam:

このクラスは、特定の HttpServletRequest パラメータのコンテンツを受け取り、入力 XML ドキュメントとして使用します。

com.sssw.b2b.xs.service.GXSServiceRunner:

このクラスは、サービスコンポーネントをインスタンス化する処理論理を提供した後、入力 XML ドキュメントを提供し、出力 XML ドキュメントを返すために使用されます。

com.sssw.b2b.xs.service.conversion.GXSConversionException:

このクラスは、`processResponse()` メソッドが失敗したことを示すために、変換ヘルパクラスによって使用されます。

com.sssw.b2b.xs.IGXSServiceRunner:

```
public final static String JNDI_NAME:
```

この文字列は、EJB のホームインタフェースの JNDI 名に対してサービスプロパティを要求するときに使用されます。値 *jndiname* は、サーブレット初期化パラメータの名前になります。

```
public final static String CONVERTER_CLASS_NAME:
```

この文字列は、コンバータクラス名のサービスプロパティを要求するときに使用されます。値 *converterclassname* は、サーブレット初期化パラメータの名前になります。

```
public final static String PARAM_NAME:
```

この文字列は、サーブレットリクエストパラメータ名のサービスプロパティを要求するときに使用されます。値 *xcs_paramname* は、サーブレット初期化パラメータの名前になります。

```
public final static String SERVICE_TYPE:
```

この文字列は、Composer サービスコンポーネントのインスタンス化されたタイプのサービスプロパティを要求するときに使用されます。値 *xcs_servicetype* は、サーブレット初期化パラメータの名前になります。

```
public final static String PROVIDER_PARAM:
```

この文字列は、JNDI プロバイダの URI のサービスプロパティを要求するときに使用されます。値 *providerurl* は、サーブレットの初期化パラメータの名前になります。

```
public final static String CONTEXT_FACTORY:
```

この文字列は、JNDI コンテキストファクトリのサービスプロパティを要求するときに使用されます。値 *contextfactory* は、サーブレット初期化パラメータの名前になります。

com.sssw.b2b.xs.GXSServiceFactory:

```
public static IGXSServiceComponent createService(
IGXSServiceRunner):
```

このメソッドは、システムプロパティ SERVICE_TYPE を取得して、Composer サービスコンポーネントの通常のクラス表現または EJB 表現であるかを判断するように修正されました。デフォルトは通常のクラス表現です。IGXSServiceComponent のタイプ固有のバージョンは、インスタンス化され、呼び出し元に返されます。

```
private static IGXSServiceComponent createNormalService(
IGXSServiceRunner ):
```

このメソッドは、Composer サービスコンポーネントの通常の表現を作成します。また、システムプロパティ SERVICE_NAME を使用して、IGXSServiceComponent が表している Composer サービスコンポーネントの名前を取得します。

```
public static IGXSServiceComponent createEJBService(
InitialContext aContext, String aJNDIName ):
```

このメソッドは、GXSEJBServiceComponent をインスタンス化します。また、JNDI コンテキストルックアップが正常に機能するように環境が設定されていることを想定します。

```
private static IGXSServiceComponent createEJBService(
IGXSServiceRunner )
```

このメソッドは、EJB として配備されているサービスコンポーネントの表現を作成します。また、JNDI コンテキストルックアップが正しく機能できるように環境が設定されていることを想定します。つまり、このメソッドをサーバ環境外から呼び出すユーザは、この環境が有効であることを確認する必要があります。このメソッドは `IGXSServiceRunner` を受け取るため、`IGXSServiceRunner` で `getServiceProperty()` というメソッドを呼び出すことによって JNDI 名を検索します。

XSL 機能

XSL 処理の向上を可能にする新しいメソッドが、特定のフレームワーククラスに追加されました。新しいメソッドでは、次のことが可能です。

- XSL プロセッサインタフェースで、XML ドキュメントのみ (XML ドキュメントおよび XSL スタイルシートの両方ではない) を受け入れ、XML に組み込まれている処理命令を使用して、XML を HTML に変換する
- URI が XML または XSL ドキュメント、あるいはその両方を参照する場合、GXSServiceFactory オブジェクトでは、ターゲットドキュメントをメモリ内 DOM に変換できます。
- Composer サービスコンポーネントで、XML、または XSL 処理命令の結果として作成される HTML を出力する

XSL メソッドを持つクラス

com.sssw.b2b.xs.GXSServiceFactory

XML ドキュメントを参照する指定 URI を **org.apache.xml.Document** に変換する新しいメソッドが含まれます。メソッドシグニチャを次に示します。

```
public static Document convertURIToDom( URI src ) throws GXSException
```

com.sssw.b2b.xs.IGXSXSLProcessor

1 つのドキュメントを入力として受け取り、そのドキュメント内にある XSL 処理命令を使用して出力ドキュメントに変換する新しいメソッドが含まれます。メソッドシグニチャを次に示します。

```
public String process( Document aXMLDoc ) throws GXSException
```

com.sssw.b2b.xs.servlet.GXSServiceRunnerEx & com.sssw.b2b.xs.servlet.GXSEJBServiceRunnerEx:

```
processResponse()
```

このメソッドは、出力ドキュメントが HTML に変換されるべきかどうかを示すブール値を取得します。呼び出されるメソッドは、`doTransformOutputToHTML()` です。インジケータが true の場合、XSL スタイルシートの URI を返すオプションのメソッドが呼び出されます。呼び出されるメソッドは `getOutputXSL()` で、デフォルトの実装により NULL が返されます。デフォルトの実装が使用される場合、XSL 処理命令が出力 XML ドキュメントに必要です。

```
public boolean doTransformOutputToHTML()
```

このメソッドは、初期化パラメータ *transform_into_html* を検索し、関連付けられているブール値を返します。初期化パラメータがサポートされていない環境では、変換が行われると、このメソッドは過負荷になります。

```
public String getOutputXSL()
```

このメソッドは、初期化パラメータ *output_xsl_url* を参照し、その値を返します。初期化パラメータがサポートされていない環境では、正しいスタイルシートで変換が行われると、このメソッドは過負荷になります。

C Server 用語集

Bean 管理トランザクション

Enterprise Java Bean がそれ自身のトランザクション境界を決定することを、「Bean 管理」トランザクション制御を実行するといいます (別の手段として、コンテナ管理トランザクションがあります)。Bean 管理モデルでは、プログラマは、トランザクション論理に対して低レベルの制御を行うことができますが、コードやプログラムが複雑になります。

EJB コンテナ

EJB コンテナは、Enterprise Beans を含めたり運用したりするのに使用される、ランタイムのリソースです。

JNDI

Java Naming and Directory Interface。Java プラットフォームの標準拡張で、ファイルシステムやサーバドメインに渡って存在する可能性のある複数の命名およびディレクトリ計画に対して統一されたインタフェースを提供します。

JTA

Java Transactions API。分散トランザクションシステムに関連するトランザクションマネージャとパーティ間での標準 Java インターフェイス。Bean 管理トランザクションはこの API に依存します。

SOAP (Simple Object Access Protocol)

トランスポート層として HTTP を、またペイロードの表現に XML を使用する、オブジェクトのリモート呼び出し用のプラットフォーム独立プロトコル。

Web アプリケーション

Web アプリケーションは、同じサーブレットコンテキストを共有する複数のサーブレットであり、単体として管理することもできます。Web アプリケーションはサーブレットを含み、Web アプリケーション自体はサーブレットエンジンに含まれます。

XML メタデータ

Composer で作成されたすべての exteNd オブジェクトは、XML ファイルとして保存されます。これらのファイルのオブジェクトデータおよび処理命令は、XML メタデータと見なされます。exteNd ランタイムエンジンは、このメタデータを処理して、XML 統合サービスを実行します。

アプリケーションサーバ

アプリケーションサーバは、EJB コンテナと Enterprise Beans を含みます。サーブレットエンジン (アプリケーションサーバにつき 1 つ)、およびそのチャイルドを含むこともできます。WebSphere Application Servers は「ノード」によって含まれます。ランタイム「アプリケーションサーバ」は、IBM WebSphere Application Server 製品とは異なります。製品は、インストールされた各マシンに、単数または複数のアプリケーションサーバプロセスを含むことができます。

仮想ホスト

仮想ホストは、サーブレットエンジンから渡された HTML 要求を受け取る、単数または複数の Web アプリケーションのリストを管理するサーブレットホストです。

コンテナ管理トランザクション

宣言型トランザクション制御の別名もあるコンテナ管理トランザクションモデルでは、トランザクションの管理責任が EJB からコンテナに移ります。このトランザクションモデルを使用する EJB では、内部コードレベルで「トランザクションを意識」する必要はありません。その代わり、Bean のトランザクション属性は記述子に設定できるので、コンテナでは、Bean が機能するトランザクションに対して、適切なコントロールが使用される必要があります。コンテナ管理モデルを使用すると、コードを簡略化でき、また信頼性も向上します。

サービストリガ

サービストリガとは、Composer からプロジェクトを配備する際に作成される Java サーブレットまたは Enterprise Java Bean です。これにより、サービスが exteNd.Server に送信され、実行されます。また、サービストリガは URI と関連付けられており、サービスへの入力 (このサービスによってトリガされる) として着信データを XML ドキュメントに変換します。

接続プール

管理プロセス (通常、アプリケーションサーバ) のコントロール下で、プロセス間で共有できるデータベース接続のグループ。データベース接続を開いたり、閉じたりすると、パフォーマンスコストが高くなるので、サーバによる接続のキャッシュは効果的です。

ノード

WebSphere の管理ドメイン内の物理マシン。ノードは、単数または複数のアプリケーションサーバを含むことができます (前の説明を参照)。

配備コンテキスト

配備コンテキストは、名前の似たコンポーネントによるサービス間でのネームスペース競合を避けるために使用できる名前文字列 (これらの要素はピリオドで区切られます) です。

索引

し

す

せ

た

と

は

ふ

ほ

め

や

ら

り