Novell exteNd Composer™ Enterprise Server

for Novell exteNd 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、exteNd Composer Enterprise Server を使用して Composer アプリケーションを配備する方法について説明します。したがって、『exteNd Composer ユーザガイド』の補足ガイドです。

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。
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。

章	説明
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、「Extend Application Server の依存関係」	Extend Application Server での配備に固有なデータベースの接続プールの問題について説明します。
付録 B、「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 C、「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 D、「予約語」	Composer で使用されるためコードには使用できないキーワードのリストです。
付録 E、「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。

- URL(Uniform Resource Identifier)
- ファイル名

「斜体」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」フォントは、次の項目を示します。

- メソッド名
- コード例
- 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 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 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 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 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 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 Connects 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 eXtensible Markup Language (XML) 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。本質的に 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でき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、ウェブ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。

サーバとは

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。サーバは、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) との統合ができる「アプリケーションサーバに合わせたフレームワーク」の両方を提供します。

このガイドでは、Composer アプリケーションを Extend 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 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、標準的な JDBC コンポーネントでは、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。

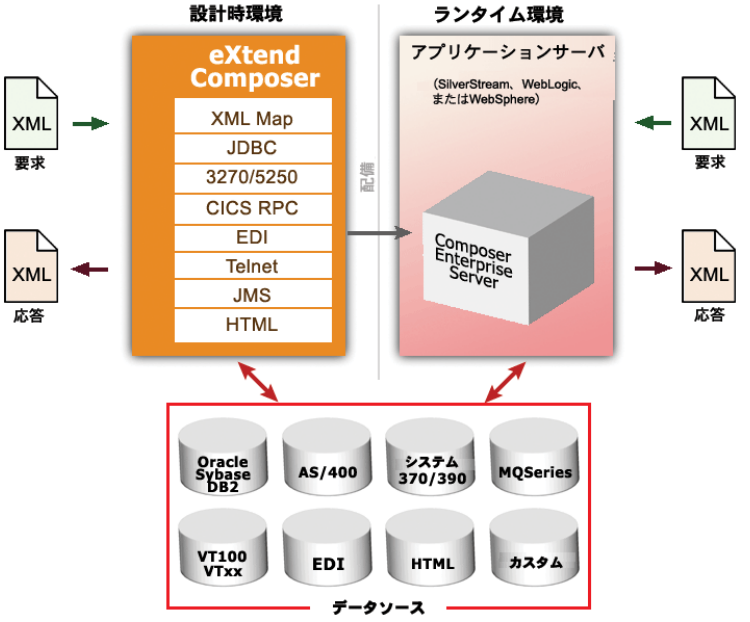
「サービス」は通常、複数のコンポーネントを構成したり、イベント処理、ルーティング、ロギングと例外処理などの重要なフロー管理作業を実行したりします。一般的なサービスには、単数または複数の XML ドキュメントの受信、洗練されたドキュメントマップおよび変換の実行、JDBC データソースからの追加情報の収集、トランザクションの実行、エラー条件の処理、状況依存型の電子メール通知の送信、および元のリクエスタへ単数または複数の XML 応答ドキュメントの返信を実行するコンポーネントが含まれます。サービスは、eXtend の「配備の単位」であり、アプリケーションサーバ上のサーバレットまたは EJB からイン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)。SOAP にラップされた形式で Web サービスを配備することもできます。

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 JMS Connect 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スイートとどのように適合するは、次の図のとおりです。



付随する図に示すように、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運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 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間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 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の3270Connect、および 5250Connect 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 の CICS RPC and JMS Connects 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 の HTML Connect 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 の JDBC Connect 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 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 Composer Enterprise Server の概要

サーバ仕様

exteNd Composer Enterprise Server は、ランタイムエンジンおよび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
- 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 Composer Enterprise 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
- 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および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 には、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。

- プロジェクトのリソースを 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 (必要な場合)
- 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およびコンパイルする

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 ファイルに保存され、次回［Deployment Wizard］を呼び出す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、最初の［Deployment Wizard］スクリーン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配備を実行するには、生成したメタデータを適切な SilverCmd ファイルとともに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。この方法は一般的に、サーバの管理者にアプリケーションの配備の責任がある、大規模インストレーションで使用されます。Composer がすべての必要な配備ファイルを生成すると、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。

追加のオプションとして、exteNd Developer Workbench アプリケーションを通じて、配備オブジェクトを exteNd Application Server に手動でロードできます。

配備オプションの詳細については、第 4 章「プロジェクトの配備」を参照してください。

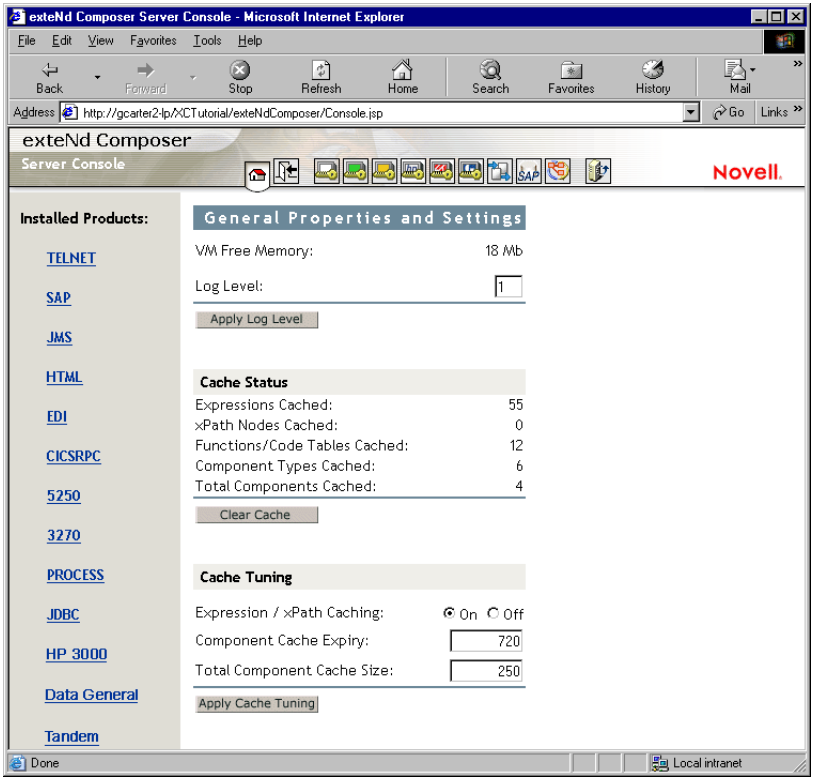
Composer Enterprise Server の管理制御

プロジェクトの配備後、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のを確認してください。
2. Web ブラウザを起動します。
3. Composer アプリケーションサーバの提供元であるホストアドレスおよびポートまで移動します (たとえば http:\\localhost:80)。選択肢のリストがあるマスタコンソールウィンドウが表示されます。

4. マスタコンソールウィンドウで、**exteNd Composer** リンクを選択します。新しいコンソール画面が表示されます。
5. コンソールリンクをクリックします。Composer Enterprise Server のメインコンソールが表示されます。



6 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。

7 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。

8 説明に従って(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 Server 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` 要素の内部で、サーバに割り当てたいRAMの最初および最大量を反映するように設定を変更します。(標準 JVM コマンドラインオプションがあります。JVM 起動パラメータと設定方法の詳細については、Sun 独自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) を使用してこれを実現できます。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、次のことを考慮します。

- 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)
- 使用するトリガオブジェクトモデル (EJB またはサーブレット)。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およびXSLスタイルシート)にアクセスできるようにするかどうか
- 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の実行のためだけにインスタンス化する必要のあるスタンドアロンエンティティとして配備できます。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。eXtend の柔軟性により、サービスをどのようにするかは配備時に決定できます。これは、配備されるサービスは、XML のソースを理解しなくても、XML ドキュメントで機能するためです (詳細については、第 4 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

Novell eXtend では、Web サービスと JMS サービスという 2 種類のサービスを作成できます (JMS は Java Messaging Service の略で、メッセージ指向ミドルウェアと Java アプリケーションとの統合用の Sun 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 サービスがキューで「リスニング」を行うと、着信メッセージにより、サービスの `onMessage()` ハンドラが起動され、サービスが呼び出されます。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：この JMS サービスタイプの説明が該当するのは、eXtend JMS Connect を購入してインストールした場合だけです。このコネクタがない場合、JMS サービスを作成することはできません。

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、インスタンス化され、eXtend Composer Enterprise Server 内で必要なすべての手順 (たとえば、入力パラメータの XML 文字列への変換、必要な HTTP 要求の実行、システムイベントの記録など) を実行します。Deployment Wizard は、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いたサービストリガの作成を支援します。

たとえば、パラメータを持つ Web ページ要求またはフォームからデータをマップしたり、取引パートナーの URI から情報を受け取る必要があることもあります。Deployment Wizard では、このような処理の実行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 Composer Enterprise 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 C「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 ファイルパッケージングおよび CLASSPATH 仕様が必要になります。**xcs-all.jar** ファイルには、eXtend Composer Enterprise 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。

統合アプリケーション配備のオプションの詳細については、第 5 章 「Composer サービスへのプログラムによるアクセス」を参照してください。

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、サーブレットでは、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。

プールとは

前述の通り、アプリケーションサーバを介した提供サービスの 1 つに、データベース接続プールがあります。Extend Application Server では、これらのプールは、データベース名で識別されます。Composer プロジェクトをアプリケーションサーバに配備する場合、配備されるサービスで、トランザクション管理がどのように使用されるか理解する必要があります (詳細については、第 6 章 「トランザクション管理」を参照してください)。また、サービスで実行されるデータベースアクセスについても理解する必要があります。

Extend Application Server の接続プールを利用するには、ターゲットデータベースの接続リソースにプール名を指定する必要があります。

コネクタ

データベース以外のリソースへの接続に対して、eXtend Composer Enterprise Server は、アプリケーションサーバの接続を向上させる接続プール機能を提供しています。eXtend Composer Enterprise Server のコネクタ固有の接続プールは、個々の eXtend コンソールを介して構成および管理できます。

接続プールを利用するには、配備される接続の各タイプ (3270、5250、CICS RPC、JMS など) の Connect 関連ガイドを参照してください。

プロキシサービス

サービスがプロキシサーバ内で実行される場合、**xconfig.xml** ファイルで特定の設定を変更する必要があります。設計時に、[**Tools/Configuration**] ダイアログボックスを使用して、eXtend Composer でのプロジェクトのプロキシサーバ設定を修正できます (詳細については、『eXtend Composer ユーザガイド』を参照してください)。すでにアプリケーションを配備している場合、または Composer を使用せずにプロキシ設定の最終的な変更をする必要がある場合、テキストエディタで **xconfig.xml** ファイルを開き、PROXYSERVERINFO タグを参照します。このタグの子要素を使用すると、プロキシ設定を微調整できます (アプリケーションがプロキシサーバ内で実行している場合、USEPROXYSERVER 要素を「オン」に設定します)。

セキュリティの役割

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。

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、サイトから送信したドキュメントを検証するための DTD、またはスキーマファイルがあります。

管理とメンテナンス目的で、専用の JAR でこれらのファイルを準備し、アプリケーションサーバに配備すると、より効率的です。URI は、JAR、およびその公開されたコンテンツに関連付けることができます。

特別な目的の JAR を使用すると、サービスに必要なリソースファイルを効果的に扱うことができます。これは、補助ファイル (およびそれらを使用するサービス) を個別に配備および保守できるからです。特定の目的の JAR を作成する場合、eXtendプロジェクト変数を介してこれらのリソースへのすべての参照を事前に、簡単に計画する必要があります。

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 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 を Extend Application Server CLASSPATH に置くことです。この処理を行う方法は 3 種類あります。

- Windows NT の場合、[**Start**]、[**Settings**]、[**Control Panel**]、[**System**]、[**Environment**] を選択します。UNIX の場合、環境変数を修正して、必要な JAR を示すように AGCLASSPATH 変数を設定します。

  または、

- 該当する $SS_LIB エントリを **agjars.conf** に追加し、JAR ファイルをアプリケーションサーバの **lib** ディレクトリにコピーします。

  または、

- クラスをサーバのアプリケーションデータベースに直接追加します。

4 プロジェクトの配備

exteNd Composer からプロジェクトを配備するときの基本タスクは、次のとおりです。

- 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。たとえば、サービスをスタンドアロンプロセスとして実行するか、別のアプリケーションの一部として実行するか、またはサービスが SOAP サービスか JMS サービスかなどを決定します。
- Composer の Deployment Wizard から配備オブジェクトを作成する
- 配備オブジェクトを exteNd Application Server にインストールする

プロジェクトは、Composer からアプリケーションサーバに直接配備できます。また、Composer プロジェクトを Workbench プロジェクト内のサブプロジェクトとして追加することもできます。アプリケーションサーバに直接配備する場合、JAR を配備します。プロジェクトを Workbench プロジェクトに含める場合、EAR または WAR 形式のいずれかの配備を選択します。

ここでは、JAR 配備を中心に説明します。Composer プロジェクトの Workbench 配備の詳細については、**ComposerAndWorkbench.pdf** マニュアルを参照してください。

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、配備の一部として exteNd Composer 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。サービスを他の Java アプリケーションに統合する場合、exteNd Workbench を使用する必要があります。exteNd Workbench は、［**File**］、［**New**］、［**JSP**］の順に選択し、表示される **Java class**、**EJB**、および **Servlet** ウィザードにあり、様々なトリガに対して骨組みになるコードを自動生成します。

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	注意 :
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、再利用可能なリソースとして配備できます (この章の後半を参照)。
サーブレットまたはEJBサービストリガのクラスファイル	サービスをサーブレットや EJB に接続し、サービス入力を準備し、サービスを開始します。	スタンドアロン運用に必要です。カスタムフレームワークを介してサービスが別のアプリケーションに統合される場合はオプションです。
配備記述子ファイル (サーブレットまたはEJBサービストリガファイルと名前が同じ XML ファイル)	サービストリガJavaクラスをアプリケーションサーバにインストールするときに必要な情報を記述します。サーブレットベースのサービストリガに関連付けられた URI、および EJB の JNDI 名が記述されています。	サーブレットまたは EJB が作成される場合は必須です。Composerにより自動的に生成されます。
構築ファイル	アプリケーションサーバにより構築されるサーブレットをリストします。	自動的に作成されます。
SilverCmd バッチファイル ImportObjects.bat	配備オブジェクトをアプリケーションサーバにインストールする SilverCmd ユーティリティ呼び出しを含みます。	自動的に作成されます。
配備プロファイルファイル **xDeploy.xdp**	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。配備が次に実行されるときに、前の配備情報を exteNd により回復できます。

Composer の Deployment Wizard は、これらすべてのオブジェクトの作成方法を示し、オブジェクトを exteNd Application Server に自動的にインストールします。Deployment Wizard は、いつでも再実行して、使用可能なオプションの一部またはすべてを変更したり、修正されたプロジェクトコンポーネントを再配備できます。

これらのオブジェクトの内容の詳細については、付録 B「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

配備オブジェクトを exteNd Application Server にインストールする方法は 2 種類あります。1 つめは、Deployment Wizard を使用して、ターゲットサーバへのアクセスに必要な情報を提供してから、オブジェクトを自動的にインストールする方法です。2 つめは、exteNd Application Server の SilverCmd ユーティリティまたは Novell exteNd Workbench IDE などのツールを使用して、オブジェクトを手動でインストールする方法です。

Web サービスのサービストリガについて

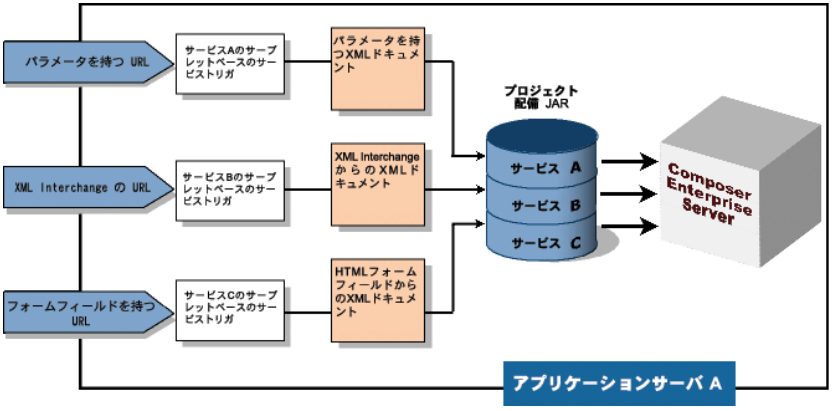
「サービス」は、アプリケーションサーバ環境における実行の基本単位です。サービスを含めすべての exteNd オブジェクトはメタデータの指示として作成され、配備 JAR の中にある XML ファイルに保存されます。配備するとき、サービスは適当なサーバトリガメカニズムに接続する必要があります。Deployment Wizard で、Web サービスの主要な 2 つのサービストリガである Java サーブレットおよび Enterprise Java Beans を生成できます。さらに、exteNd Composer Enterprise Server のフレームワークを使用すると、サービストリガをカスタマイズしたり、手動で作成したりできます。

注記： トリガに関する以下の説明は、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 からサービスが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 (34 ページ「EJB ベースのサービストリガ」を参照)、それによって呼び出されるサーブレットを生成する必要があります。

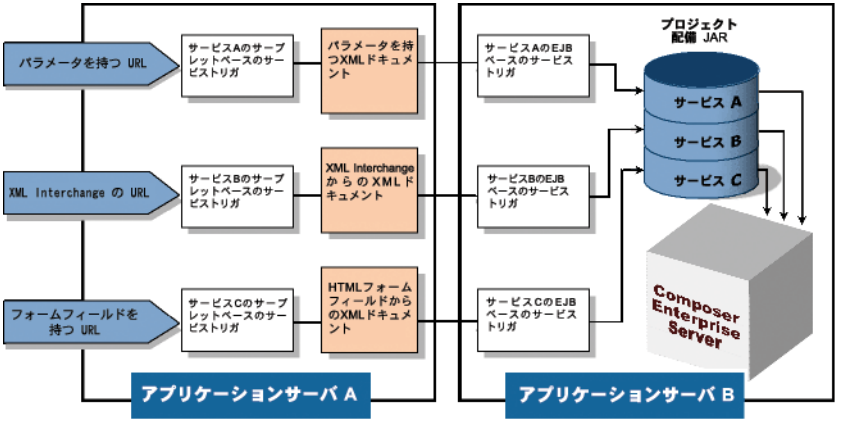
サーブレットベースのサービストリガは、Composer Deployment Wizard により自動的に作成されます。必要な場合は、これらのサーブレットベースのサービストリガを変更したり、exteNd により作成されたサービストリガとは別のものを定義したりできます。次の図では、Deployment Wizard により生成されたサーブレットタイプに関する一般的なシナリオが説明されています。



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。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。



アプリケーションベースのサービストリガ

前に説明したとおり、exteNd 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、exteNd の外でサービストリガオブジェクトを作成するか、exteNd 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指向のいずれのトリガオプションでも、exteNd 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。詳細については、第 5 章「Composer サービスへのプログラムによるアクセス」を参照してください。

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 Composer Enterprise 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、ブラウザコンソールからの管理制御の対象になります。また、JMS サービスは指定のサービスが「複数」のリスナオブジェクトに依存するように配備することもき、結果として性能および帯域幅が向上します。ただし、どのような場合でも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トリガ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 Administrative Console は、そのために提供されています。『*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、Novell 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。

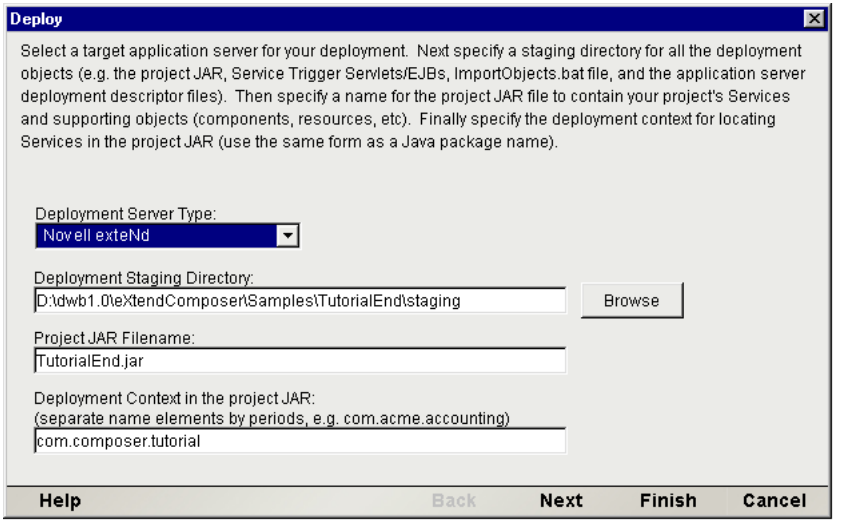
Composer Deployment Wizard の使用

Composer Deployment Wizard では、配備の様々な特性に関する情報が求められる一連のパネルが表示されます。いくつかのパネルはオプションです。これらのパネルには、そのように記されています。その他のファイルは、「入力する必要があります」。必須フィールドに入力していない場合、ウィザードにより警告されます。

Composer Deployment Wizard を使用して、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。ウィザードの最初のパネル (次の図を参照) が表示されます。

一般情報パネル

Deployment Wizard の最初のパネルは次のようになります。



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

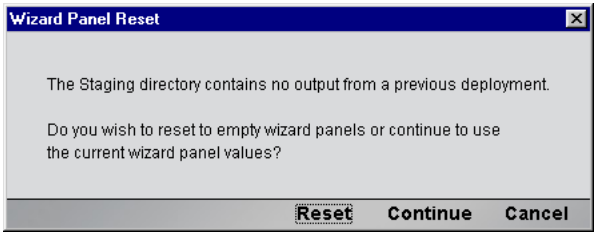
この配備のアプリケーションサーバタイプを［Novell］、［IBM WebSphere］、または［BEA WebLogic］から選択します。これにより、選択されたサーバ環境に固有な配備ファイルが Composer により作成されます。

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を選択するか、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。

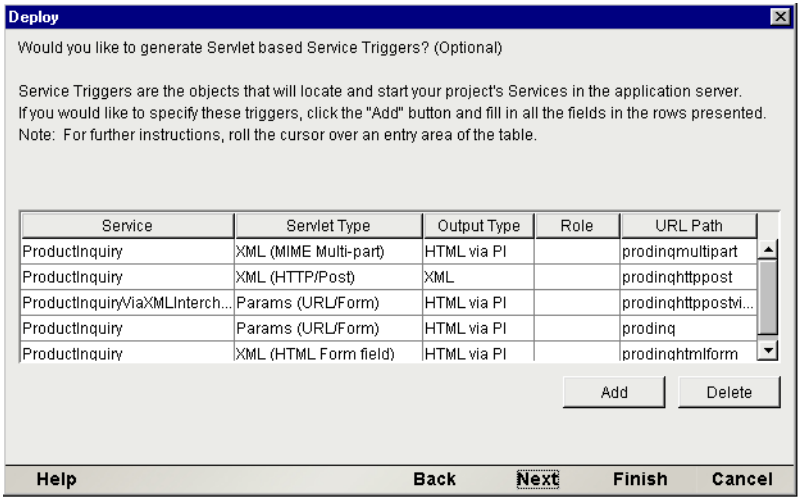
プロジェクト JAR の配備コンテキスト

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、2 つの Composer サービス (プロジェクトが異なる) 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。

注記：コンテキスト文字列に Java 言語キーワードを使用しないでください。予約語の完全なリストについては、付録 B「予約語」を参照してください。

サーブレットベースのサービストリガパネル

Deployment Wizard の 2 番目のパネルは次のようになります。



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、以下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ています。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	メモ
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 (サービスに出力に適した方)。XSL スタイルシート (処理命令を介した) 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

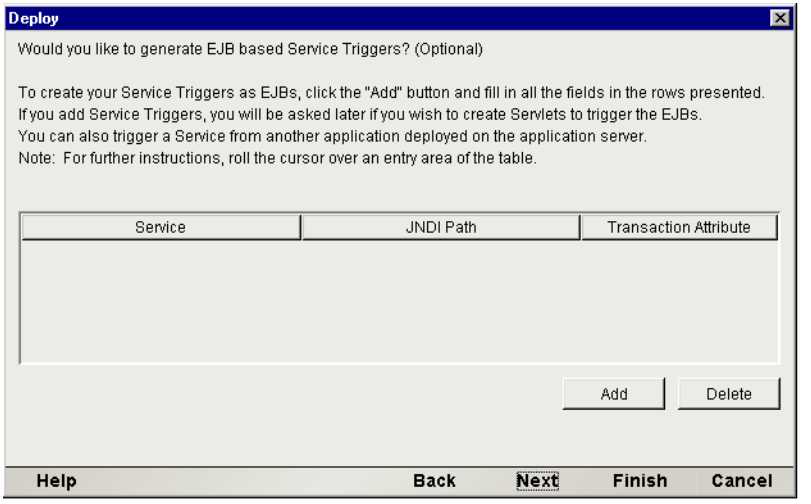
［URI Path］は、このサーブレットに割り当てられている URI の固有な名前を提供します。この名前は、Novell exteNd サーバ名およびデータベース名とともに、このサーブレットにアクセスする完全な URI を形成します。

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/InfoDB/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、以下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「sssw://」、Novell exteNd サーバ名、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。

sssw://stratus/RMI/WebStore/ProductInquiry .

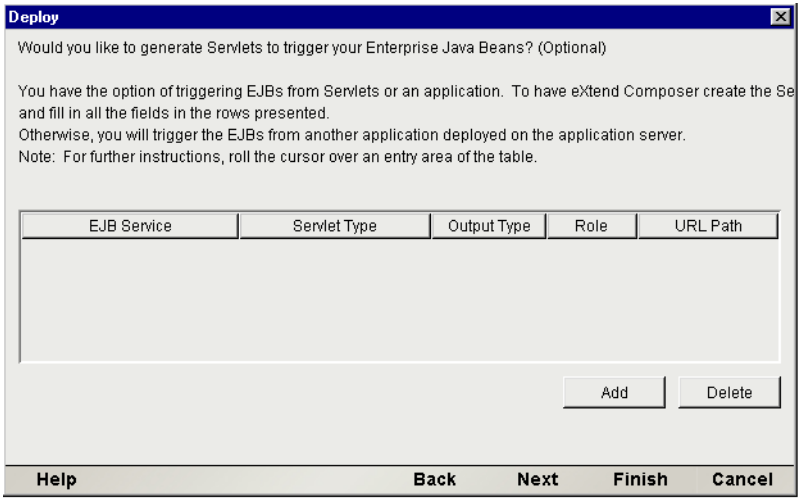
Transaction Attribute

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。

サーブレット -EJB トリガパネル

このパネルは、前のパネルで 1 つ以上のエントリを追加している場合だけ表示されます。追加していない場合、［SOAP Trigger］パネル (次の節を参照) に直接移動します。

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」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、以下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)。スタイルシート (処理命令を介した) 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。

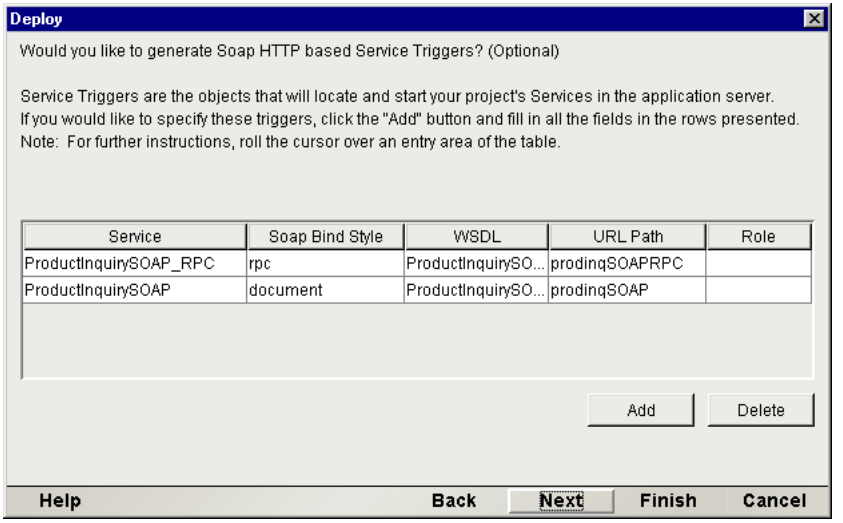
URI Path

[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、Novell exteNd サーバ名およびデータベース名とともに、このサーブレットにアクセス完全な 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/InfoDB/WebStore/ProductInquiry.

SOAP ベースのサービストリガパネル

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適切な［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。

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 エイリアスを指定します。

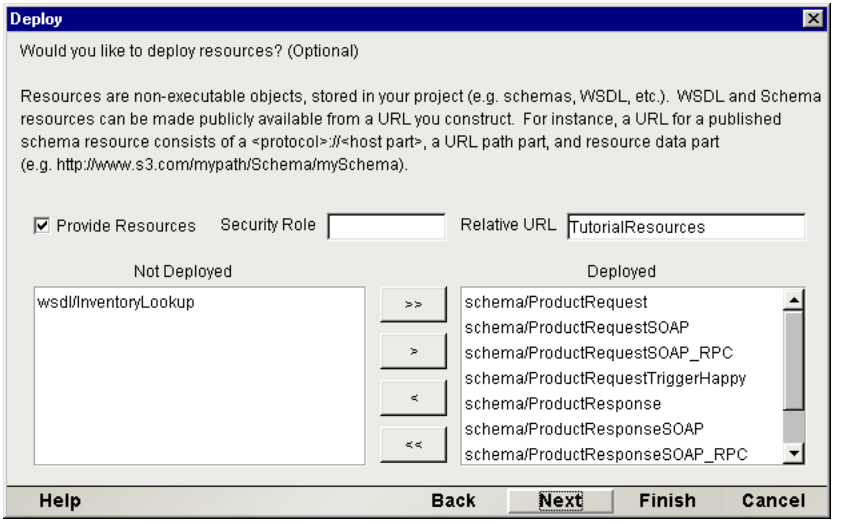
Role

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。

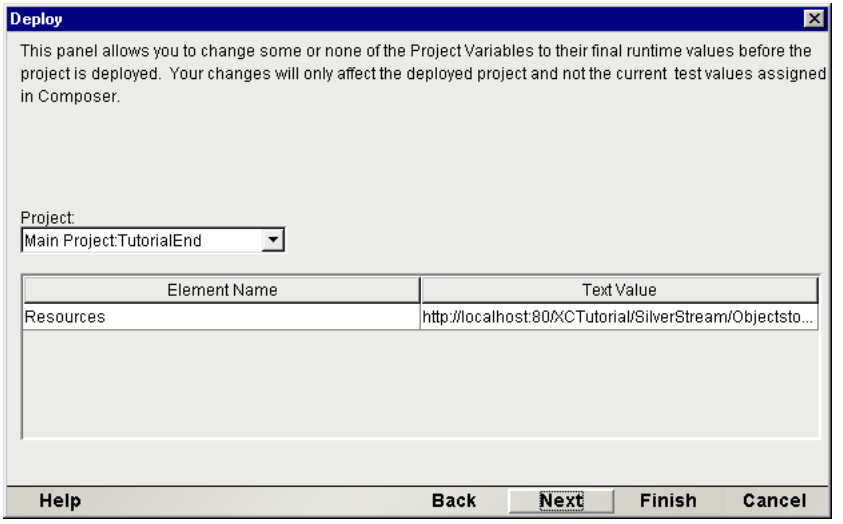
［Relative URL］は、URL のホスト部分に対して相対的です。ホストの部分は、Deployment Wizard の最後のパネル (以下を参照) の上部にあるテキストフィールドに入力する URL と同じです。上記のダイアログボックスでこのルート URL を入力する必要はありません。［Relative URL］で入力する必要のある箇所は、上のピックリストで示されるリソースのリソースディレクトリ (またはサブパス) だけです。たとえば、**http://www.somedomain.com** から提供し、リソースが **/resources/soap** にある場合、これを［Relative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

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。

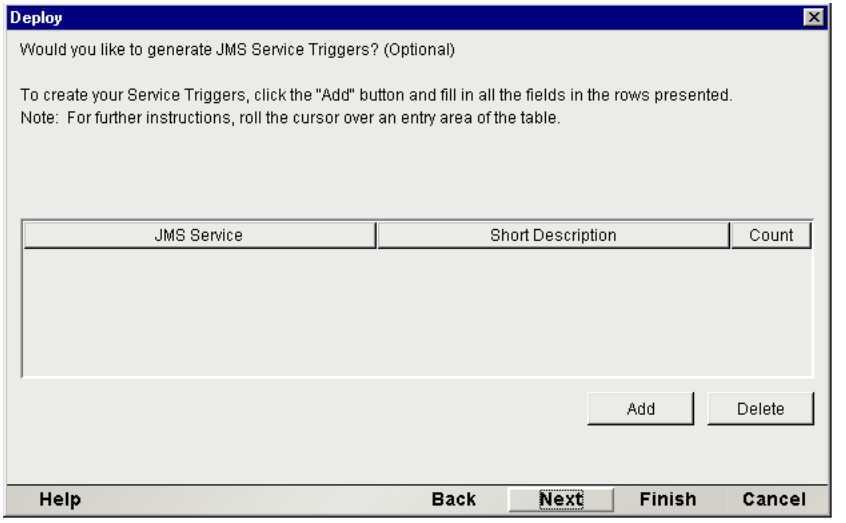
注記： プロジェクト変数は、**PROJECT.xml** ファイル (プロジェクトの **.spf** ファイルと同じディレクトリ) に保存されます。このパネルを使用してプロジェクト変数値を変更すると、サーバに書き込まれた **PROJECT.xml** ファイルが、作業ディレクトリの **PROJECT.xml** ファイルと一致しなくなります。通常、これで問題ありません。



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 (次を参照) が表示されます。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」をいくつか配備するかを示す値を［**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。

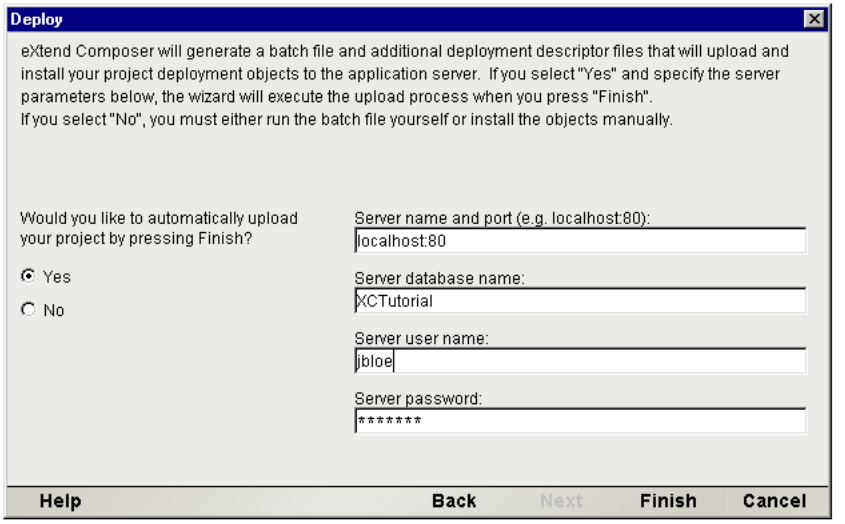


自動インストールパネル

Deployment Wizard の最後のパネルを次に示します。このパネルを使用すると、次の操作を実行できます。

- サーバ名およびポートの設定
- サーバデータベースの設定
- サーバユーザ名およびパスワード情報の指定
- Deployment Wizard により、exteNd 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どうかを決める

このウィザードのすべてのパネルと同様に、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このパネルに表示されます。



Novell exteNd サーバおよびポート

配備オブジェクトを受け入れるNovell exteNdサーバの名前およびポートを入力します。

Novell exteNd データベース名

配備オブジェクトを登録するアプリケーションデータベースを入力します。

ユーザ名およびパスワード

必要な場合、アプリケーションデータベースに対して有効なユーザ名とパスワードを入力します。

配備プロジェクトの管理

配備プロジェクトの管理は、Web ブラウザで表示されるコンソールパネルを介して行います。詳細については、19 ページ「Composer Enterprise Server の管理制御」を参照してください。

SilverCmd を使用した配備オブジェクトのインストール

この節では、Composer プロジェクトを配備するファイルのセットを作成する方法、および exteNd Application Server に装備されている SilverCmd コマンドラインユーティリティを使用した exteNd Application Server でのサービストリガのサポートについて説明します。Composer Deployment Wizard は、通常、プロジェクトを自動的に配備しますが、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もあります。手動での配備については、この節で説明する SilverCmd ユーティリティを使用します。

注記 : 次の説明は、SilverCmd ユーティリティを完全に説明したものではありません。SilverCmd のより完全な説明については、Novell exteNd Docs フォルダの **help/books/ToolsSilverCmd.html** にある、SilverCmd のオンラインリファレンスガイドを参照してください。

extend Application Server は、オブジェクトリポジトリを使用して、そのサーブレット、Web ページ、およびその他のリソースを保存します。コマンドライン管理ユーティリティ SilverCmd は、Novell exteNd IDE 外で作成されたアプリケーションリソースをインストールするために提供されています。SilverCmd は、XML ファイルを使用して、配備エンティティを記述し、ソースコードクラスファイルをインポートするかどうか、または、Novell exteNd 環境内でソースコードを作成するかどうかを定義する簡単な命令を発行します。SilverCmd は、アプリケーション配備に重要な機能を提供するので、Composer アプリケーションの手動配置に使用する理想的なツールとなり、exteNd Application Server へのリソースをサポートします。

SilverCmd バッチおよび記述子ファイルの作成を考慮する理由はいくつかあります。

- 配備 Composer サービスコンポーネントと通信する Java クラスを追加する
- XSL スタイルシート、DTD、および XML スキーマの公開用に XML リソース JAR ファイルを配備する

➢ 配備オブジェクトを Novell exteNd サーバにインストールする

1. 配備関連ファイルをまとめます。
2. 配備関連ファイルの SilverCmd 記述子ファイルを作成します。
3. SilverCmd を使用してオブジェクトをインストールします。

注記 : exteNd Composer では、これらの手順をすべて自動的に実行できます。

SilverCmd 命令

手動配備を開始する最適な方法は、すべての SilverCmd 命令を含むメインバッチファイルを作成して、配備オブジェクトをインポートおよび初期化することです。Composer サービスコンポーネントのサポートに必要な各オブジェクトの SilverCmd 命令を含むファイルを作成します。通常、exteNd Composer によりステージングディレクトリに配備ファイルが作成されると、必要な SilverCmd 命令をすべて含むメインバッチファイルの名前は ImportObjects.bat になります。通常使用できる命令のタイプは、次のとおりです。

- Import Java ソースファイル
- Build Java ソースファイル
- Import Java クラスファイル
- Import Composer プロジェクト JAR ファイル
- Import EJB 配備 JAR ファイル
- Deploy EJB JAR ファイル
- Import XML リソース JAR ファイル

Import Java ソースファイル

このソースファイルには、Composer サービスコンポーネントと通信する、Novell exteNd トリガビジネスオブジェクトの実装が含まれます。トリガオブジェクトのタイプの例は、次のとおりです。

- Servlet
- Invoked
- Data Source
- Table-modified
- Mail
- Scheduled

これら、およびその他のトリガビジネスオブジェクトの完全な説明については、exteNd Application Server Help『*Programmers Guide*』の「*Developing Triggered Business Objects*」を参照してください。

ソースファイルは、インポートされますが、Application Server でコンパイルされるまで使用できません。つまり、ImportObjects.bat ファイルで示される各ソースファイルに対して、作成命令記述子ファイルで同等のエントリが必要です。トリガビジネスオブジェクトのソースコードをインポートする利点は、ソースを Novell exteNd Composer IDE 内から修正または参照できるということです。

これらのエントリでは、トリガビジネスオブジェクトの必須パラメータに値を提供する記述子ファイルが必要です。たとえば、Mail トリガオブジェクトには、メールアカウント名およびオブジェクトの寿命を含む記述子ファイルがあります。また、Servlet トリガオブジェクトには、サーブレットの URL エイリアスのリストおよびオブジェクトの寿命を含む記述子ファイルがあります。

ソースファイルをインポートする通常の構文は、次のとおりです。

```
SilverCmd ImportSource server [:port] database javafile -f
descriptorfile -o
```

ここで、

- `javafile` - Java ソースファイルの名前。
- -f - 入力ファイルを指定します (`descriptorfile` に指定)。
- `descriptorfile` - ソースを実装するトリガビジネスオブジェクトのタイプ固有のパラメータ、および Composer プロジェクト JAR ファイルの関連付けを含むファイルの名前。
- `-o` - ファイルの既存のコピーが、このインポートにより上書きされるかどうかを指定します。

Build Java ソースファイル

exteNd Application Server にインポートされる Java ソースファイルに対して、Java ファイルを使用可能なオブジェクト、つまりクラスファイルにコンパイルするようにサーバに明示的に指定する必要があります。

この命令では、コンパイルするソースコードファイルの名前を含む記述子ファイルが必要です。エントリは、インポートされる各ソースファイルの記述子ファイルにあります。

ソースファイルを作成する通常の構文は、次のとおりです。

```
SilverCmd Build server [:port] database -f descriptorfile
```

ここで、

- `-f descriptorfile` - 作成するオブジェクトのリストを含むファイルの名前。

Import Java クラスファイル

外部コンパイラを使って Novell exteNd トリガビジネスオブジェクトを作成し、ソースなしでクラスファイルだけをインポートする場合に使用されます。

この命令では、クラスがトリガビジネスオブジェクトの場合、記述子ファイルが必要です。記述子ファイルには、ビジネスオブジェクト固有のパラメータが含まれます。また、プロジェクト JAR ファイルとの関連付けも含まれます。

クラスファイルのインポートに使用される通常の構文は、次のとおりです。

```
SilverCmd ImportClass server [:port] database classfile -f
descriptorfile
```

ここで、

- `classfile` - インポートされる Java クラスの名前。
- `-f descriptorfile` - ビジネスオブジェクトの固有パラメータ、および Composer プロジェクト JAR ファイルの関連付けを含むファイルの名前。

Import exteNd プロジェクト JAR ファイル

exteNd 配備プロジェクトを含む JAR ファイルは、SilverCmd を使用して、メタデータファイルとしてインポートすることができます。JAR ファイルが、Novell exteNd Composer IDE 外で作成される場合、JAR のコンテンツは、アプリケーションサーバにエクスポートしなくても表示できます。

この命令の通常の構文は、次のとおりです。

```
SilverCmd ImportMedia server [:port] database jarfile -o
```

ここで、

- `jarfile` - インポートする Composer プロジェクト JAR ファイルの名前。
- `-o` - 同じ名前の JAR ファイルが、このインポートにより上書きされるかどうかを示すオプション。

Import EJB 配備 JAR ファイル

exteNd Composer Enterprise Server フレームワークを使用するカスタム配備 EJB では、EJB 配備 JAR ファイルが必要です。JAR ファイルには、EJB を形成するクラスファイル、および配備記述子 inf/ejb-jar.xml が含まれます。この配備記述子には、すべての EJB を記述するエントリ (次の情報) が含まれます。

- EJB のタイプ : セッションまたはエンティティ Bean
- EJB クラス名
- ホームインタフェース名
- リモートインタフェース名
- ステート管理タイプ : ステートフルまたはステートレス
- トランザクションのタイプ : Bean 管理またはコンテナ管理

配備記述子の内容に関する完全な説明については、exteNd Application Server Help『*Programmers Guide*』の「*Developing Enterprise Beans*」、「*Understanding Deployment Descriptors*」を参照してください。

JAR ファイルのインポートに使用される SilverCmd 命令は、`ImportMedia` です。SilverCmd 記述子は必要ありません。インポートした JAR ファイルに含まれる EJB のいずれかを使用するには、その JAR ファイルを配備する必要があります。

この命令の通常の構文は、次のとおりです。

```
SilverCmd ImportMedia server [:port] database ejbjarfile -o
```

ここで、

- `ejbjarfile` - インポートされる EJB 配備 JAR ファイルの名前。
- `-o` - 同じ名前の JAR ファイルが、このインポートにより上書きされるかどうかを指定します。

Deploy EJB JAR ファイル

exteNd Application Server にインポートされる各 EJB 配備 JAR ファイルに対して、EJB JAR を配備する最後の作業を行います。これを行う SilverCmd 命令は、`DeployEJB` です。

この命令には、配備される各 Bean の JNDI 名を設定するパラメータを含む記述子ファイルが必要です。また、EJB と Composer プロジェクト JAR ファイルの関連を規定することもできます。

この命令の通常の構文は、次のとおりです。

```
SilverCmd DeployEJB server [:port] database ejbjarfile -f
descriptorfile
```

ここで、

- `ejbjarfile` - 配備されるexteNd Application ServerのEJB JARファイルの名前。
- `-f descriptorfile` - EJBの配備に必要なパラメータを含むファイルの名前。

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、およびスタイルシートです。通常、これらのリソースはアプリケーションサーバで公開する必要があります。

XML リソース JAR ファイルの配備

XML リソース JAR では、独自のパス名を使用して、パブリック URI を持つ XML リソースを公開することができます。このメカニズムを利用するには、まず、XML リソース (DTD、XSL など) を含む JAR を作成します。次に、SilverCmd 命令 `DeployJSP` を使用して、リソース JAR ファイルをアップロードします。この命令の構文は、次のとおりです。

```
SilverCmd DeployJSP server [:port] database jarFile -f
descriptorfile
```

ここで、

- `jarFile` : XML リソースを含む JAR。

`-f descriptorfile` : URI の関連付けを含むファイルを指定します (このファイルの構造については、/DTD ディレクトリの **deploy_jsp.DTD** および **deploy_jsp_sample.XML** を参照してください)。

5 Composer サービスへのプログラムによるアクセス

Deployment Wizard(最後の章を参照) を使用した Composer サービスの配備が該当するのは、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です。しかし、通常は、実行可能オブジェクト間での統一が必要な大規模で、複雑になる可能性のある 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、オブジェクト間をしっかりと統合する必要があります。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。

カスタムトリガを作成する方法は 2 種類あります。1 つめは、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 指向コードを生成するため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ングする、またこれらのオブジェクトを生産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 に装備されています) を呼び出すコードを作成する方法です。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。

どの配備方法を選択しても、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を説明してから、Framework API を使用する方法を説明します。その後で、簡単な配備として eXtend の Workbench を使用する方法を示し、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 ページの作成を説明します。

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、Extend Application Server 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 を作成する場合、ファイルはここに生成されます。クラスの詳細については、JavaDoc ファイルを参照してください。簡単な説明が記されています。

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) 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 (32 ページ「配備オブジェクトの作成」を参照してください)。

次の行は、完全修飾のサービス名の例です。

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

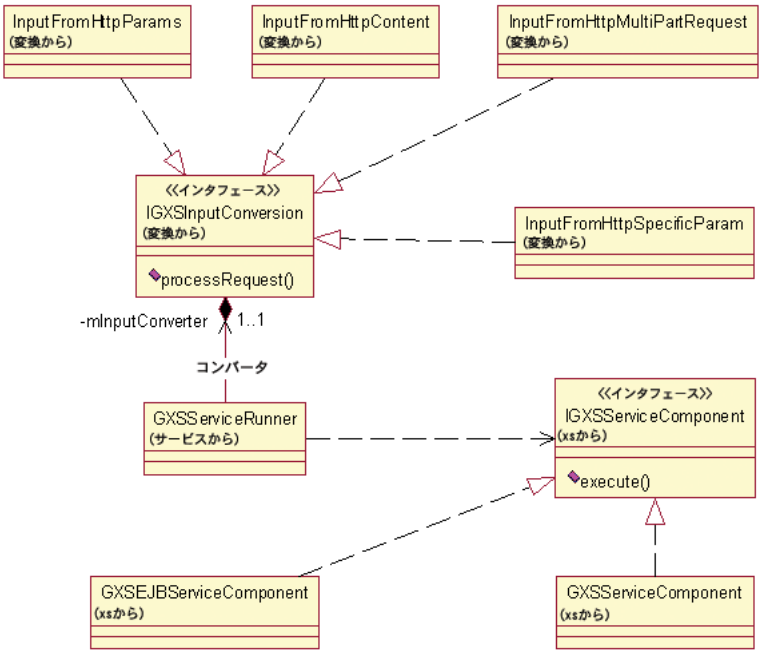
ここで、

- `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 つ以上の Composer サービスが含まれます。サービストリガは、アプリケーションサーバにより受け取られる要求間のブローカーで、Composer サービスを起動します。

各配備メソッドについて、サービスをカスタムアプリケーションに統合できる追加機能が装備された「ヘルパ」クラスが提供されます。

様々なフレームワークコンポーネントの関係を次の図に示します。



サービストリガ拡張子の作成方法

サービストリガは、入力 XML ドキュメントをその環境から受け取り、Composer サービスコンポーネントを実行し、呼び出し側クライアントに出力ドキュメントを返すクラスです。これは、メインのサービストリガです。

➢ Web サービスのサービストリガ拡張を作成する

1. Novell exteNd Workbench またはその他の Java 開発環境を使用して、Composer 配備フレームワークオブジェクトの Java オブジェクトを作成します。
2. Composer がオブジェクト作成に使用されなかった場合、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きます。抽象クラス GXSServiceRunnerEx は、HttpServlet から派生し、次のことに必要なロジックを含みます。

- Composer サービスコンポーネントでは、すべての要求が抽象メソッド getServiceName() を呼び出して実行されるということを認識する
- サービスコンポーネントの実行前に抽象メソッド processRequestEx() を、実行後に processResponse() を呼び出す

フレームワークを拡張し、独自の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作成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、サービス実行側の様々な「ゲッタ」メソッドをプロパティ取得のために呼び出すことができるように、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 (受け取られる入力タイプ、およびサービスが、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める情報は、「サービスプロパティ」に含まれています。次の表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	GXSServiceRunnerEx 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見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の EJB ホームインタフェースのJNDI名	jndiname	getJndiServiceName()
CONVERTER_ CLASS_NAME	HTTP要求をXML ドキュメントに変換するときに使用されるクラス	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かNORMAL かを示します。	xcs_servicetype	getServiceType()
PROVIDER_PARAM	JNDI プロバイダ URI	providerURI	

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 として参照されるかどうかを指定するインジケータ	transform_into_html	getOutputHTMLIndicator()
OUTPUT_XSL	出力ドキュメントが HTML に変換される場合、スタイルシートの URI が提供されます。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だけ必要です。	output_xsl_URI	getOutputXSL()

GXSServiceRunnerEx クラスは、次のメソッドの実装を提供する必要があります。

- 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 getServiceName() メソッド
- メモリ内 DOM の名前を返す getRootName() メソッド
- NORMAL または EJB の文字列値を返し、呼び出されるサービスコンポーネントのタイプを示す getServiceType() メソッド
- IGXSInputConversion インタフェースを実装するクラスの名前を返す getConverterClassName() メソッド
- サービスの出力が HTML の場合 true を返し、XML の場合 false を返す getOutputHTMLIndicator() メソッド

コンバータクラスの例は、次のとおりです。

```
public class myGXSInputFromHttpParams implements IGXSInputConversion
{
   // Attribute that holds the service
// runner for querying parameters.
   IGXSServiceRunner mRunner = null;

   // Constructor to take the IGXSServiceRunner
   // so that the class can retrieve params
   public GXSInputFromHttpParams( IGXSServiceRunner aRunner )
   {
      mRunner = aRunner;
   }

   // the processRequest method should take
   // an HttpServletRequest as
   // a parameter and return an XML doc as a String:
```

```
    public String processRequest( HttpServletRequest aReq )
       throws GXSConversionException
    {
       String lsExpandedDoc = null;
    // ( create DOM . . . )
       return lsExpandedDoc;
    }
}
```

GXSServiceRunnerEx

GXSServiceRunnerEx クラスの抽象メソッド processRequestEx() を簡単に説明します。

```
StringÅmÅn processRequestEx( HttpServletRequest )
```

これは、現在のサーブレット要求を受け取るメソッドです (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受け取ります)。このメソッドは、サーブレットがバインドされる 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。要求の処理時に、XML ドキュメントのバインドの情報がない場合、空の String が返されます。Composer サービスコンポーネントは、入力 XML コンポーネントのように動作します。

カスタムサーブレット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、堅固で柔軟な配備モデルを提供できます (たとえば、サーバの異なる 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 ベースのサービスの場合「EJB」です)。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た場合だけ、インスタンス化できます。

フレームワークを拡張し、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、サービス実行側の様々な「ゲッタ」メソッドをプロパティ取得のために呼び出すことができるように、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ングする必要があります。このようにパッケージングすると、新しいサービストリガテンプレートから継承する簡単な配備時サービストリガを作成できます。この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。

サービストリガを最初から作成する

What Is Available in the Framework

eXtend Composer Enterprise 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すべてのファクトリおよびインタフェースが提供されます。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のインスタンス化を支援するスタティックメソッドのセットが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メイン Java 要素は、`IGXSServiceComponent` 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は、様々なパラメータを受け渡しする 4 つの `execute()` メソッドを提供します。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 サービスが実行され、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、ファクトリクラス `GXSServiceFactory` 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。文字列にサービスの修飾名を提供します。次の例では、**com.acme.inventory** の配備コンテキストで Composer サービス **ProductInquiry** 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;
    }
```

アプリケーションでのサービス EJB の使用

企業要求が増加し、分散アプリケーションが必要になった場合、Composer サービスを EJB として配備することができます。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設定

Composer サービスが Extend Application Server で EJB として配備されているので、開発およびランタイム環境に必要な Novell JAR ファイルが含まれていることを確認してください。必要な JAR ファイルの完全なリストについては、*Extend Application Server Help* の『*Programmers Guide*』の「*Developing External Java Clients*」、「*Deploying Your Client*」を参照してください。

Novell exteNd Application Server は、JBroker for RMI 機能を使用します。Novell プログラムディレクトリ内にある JRE には、アプリケーションで EJB を正常に利用するために必要なプロパティがあります。

EJB ホームおよびリモートインタフェースの取得

EJB リモートインタフェース **IGXSEJBServiceComponent** 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、環境設定「servicename」を追加します。EJB は、環境リファレンスを参照して、バインドする Composer サービスを確認します。また、EJB 配備時に、JNDI 名を EJB に割り当てます。EJB ホームインタフェースのリファレンスの取得に使用されるのは、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。

EJB ホームインタフェースの JNDI 名を指定する場合、Extend Application Server では、「sssw://host/RMI/」という文字列が名前の前に追加されるので注意してください。たとえば、EJB を **main.server** という名前の Extend Application Server に配備し、この EJB の JNDI 名が **com/acme/inventory/ProductInquiry** の場合、完全な修飾 JNDI 名は、**sssw://main.server/RMI/com/acme/inventory/ProductInquiry** になります。

GXSServiceFactory の createService() のようなホームインタフェースが取得されると、create() メソッドが呼び出され、このメソッドにより 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 つめの実行メソッドは、パラメータを受け取らず、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取ります。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参照できないためです。

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.sssw package** に **GXSEJBAccessor** という名前のファクトリクラスがあります。このクラスには、Extend Application Server から EJB のインタフェースを取得する 2 つのメソッドがあります。

1 つは、認証の必要がないサーバ内で使用できるメソッドで、もう 1 つは、ユーザ名およびパスワードの 2 つの追加パラメータを提供するメソッドです。

このファクトリを使用する場合、EJB に割り当てられている JNDI 名を完全に修飾する必要はありません。このファクトリは、指定したパラメータを使用して完全修飾ホスト名を作成します。次の例では、EJB の JNDI 名は、**com/acme/inventory/ProductInquiry** で、Extend Application Server 名は、**main.server** で、ポートは、デフォルトのインストール80(HTTP)または54890(RMI)にあります。

```
import com.sssw.b2b.xs.ejb;
import com.sssw.b2b.xs.sssw.GXSEJBAccessor;

public void doSomeEJBStuff() throws java.rmi.RemoteException
{
    IGXSEJBServiceHome srvcHome = GXSEJBAccessor.getHomeBean(
        “com.sssw.b2b.xs.ejb.IGXSEJBServiceHome”,
        “com/acme/inventory/ProductInquiry”, “main.server”,
        80, 54890 );
    IGXSEJBServiceComponent ejbSrvc = srvcHome.create();
    // Do something with the service component

}
```

eXtend Workbench を使用してカスタムトリガを作成する

eXtend Workbench には、上記のようなコードを作成するときに便利なビルトインウィザードがいくつかあります。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、およびカスタム Java クラスの作成を支援します。これらのウィザードおよびその使用方法についての詳細は、付属マニュアル『**ComposerAndWorkbench.pdf**』を参照してください。この付属マニュアルは、Composer インストールフォルダにあります。

6 トランザクション管理

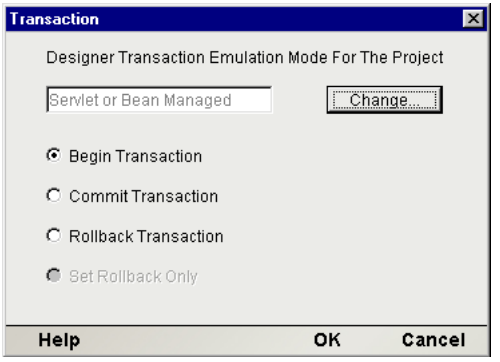
トランザクションを実行する Composer アプリケーションでは、特別な計画および配備が必要です。この章では、トランザクション管理に関する問題について説明します。

exteNd Composer でのトランザクション制御

exteNd Composer では、「トランザクションアクション」は、定義済みの 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 (次の例を参照)。

- ［*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) に含まれています。



Extend Application Server に対するトランザクション配備の考慮事項

前の章で説明したように、Composer プロジェクトの場合、サーブレットと 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 接続プールを使用する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。つまり、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ます。それ以降のロールバックへの影響はありません。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 アクションを使用して)、続いて Commit または Rollback コマンド (該当する方) を使用します。

  2. 接続に対して［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、71 ページ「JDBC トランザクション制御 : ユーザトランザクションを可能にする」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を実行できます。データを多く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がアプリケーションで必要な場合は、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 (通常は EJB) を使用します。

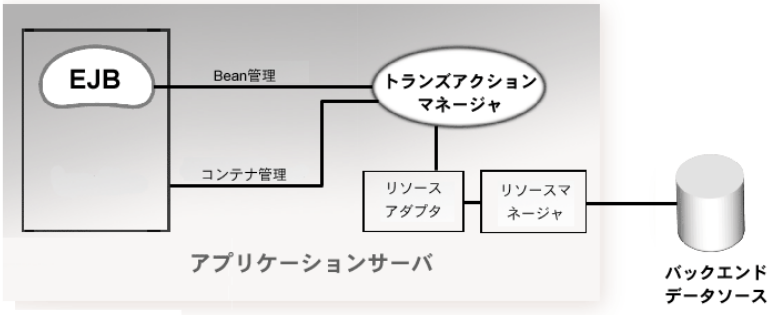
「**コンテナ**」は、EJB が配備され実行される、アプリケーションサーバ提供のコンテキストです。

「**リソースマネージャ**」は、データベースやメッセージキューなど、バックエンドシステムに対する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に対する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提供の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セットアップし、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。



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背面で行われ、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 (例外がスローされた場合は、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、begin、commit、rollback などのトランザクションコマンドを実行できます。これらのコマンドは、Bean 管理として配備された場合にのみ Bean に対して使用できます。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、Bean の配備方法について事前に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の管轄は、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、構築後の EJB のトランザクション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 で他の 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により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をロールバックできます。

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をサポートしています。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 が別の EJB を呼び出すと、そのトランザクションは、2 つめの EJB に渡さ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 6-2

トランザクションのタイプ	動作
サポートなし	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。呼び出し元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、特定の状況においてのみ適切です。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では、バックエンド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。

注記： この時点では、Extend Application Server は、「1 つの」接続プールから複数の接続にまたがる分散トランザクション管理をサポートしています。

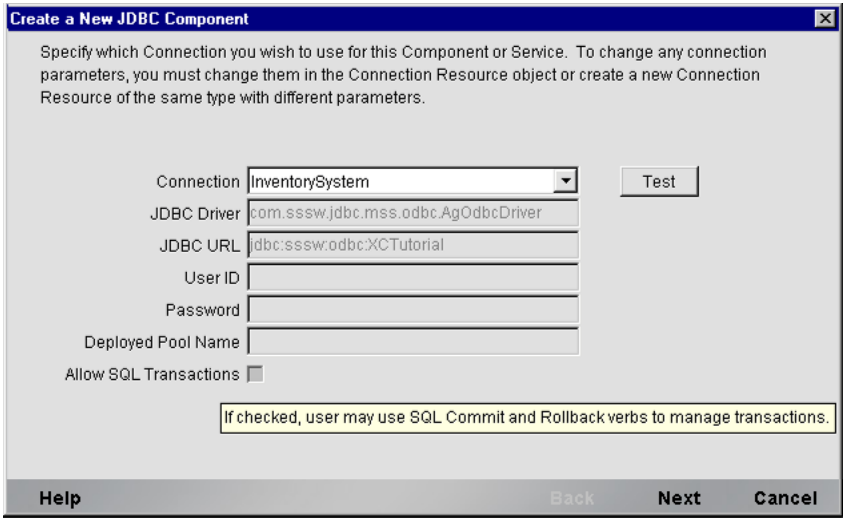
EJB 配備の考慮事項

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、Deployment Wizard の配備モードの選択は、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**］の順に選択して設定) 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 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］というコンテナ管理配備になります。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の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 : ユーザトランザクションを可能にする

手動によるトランザクション制御が必要になることがあります。たとえば、1 つのサービスで 2 つの JDBC コンポーネントが必要な場合、JDBC オブジェクト間での分散トランザクション機能が不足すると、コンテナ管理、Transaction Required モードなどのサービスを配置できなくなります。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： これは高度なオプションであり、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 commitおよびrollbackコマンドが、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アクションがコミットまたはロールバックでない場合、Composer Enterprise Server により JDBC 接続で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 コンポーネントのアクションモデルの最後に、commit 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］チェック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の「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、commit または rollback 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もお勧めします。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、対象の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 exteNd Application Server の依存関係

接続

Novell exteNd 接続プールの使用

exteNd JDBC 接続パネルで接続プールの名前を指定する場合、必ず Novell exteNd の命名規則に従ってください。サーバに追加されるデータベースは、接続プールで使用できます。データベースプールの命名規則は、**Databases/appDBName/DataSource** です。ここで、**appDBName** は、接続プールに使用される exteNd Application Server データベースの名前です。

たとえば、exteNd Application Server に、**ProductionDB** という名前のデータベースが接続されている場合、このプールの正しい修飾名は次のようになります。

Databases/ProductionDB/DataSource

B 配備オブジェクトのコンテンツ

プロジェクト JAR

プロジェクトを配備すると、Composer で作成されたすべての xObject を含む JAR ファイルが作成されます。xObject は、ファイル名の前に 2 つの部分から成るパスが付けられて JAR に保存されます。

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」のすべてのファイルは、プロジェクト JAR に含まれます。

パスの最初の部分は、ユーザが作成するもので、「配備コンテキスト」と呼ばれる固有な名前です。これは、任意に選択できる名前です。配備コンテキストは、同じアプリケーションサーバデータベースにある異なる Composer プロジェクト内で、名前が同じ 2 つの Composer サービスを区別するために使用されます (つまり、配備コンテキストによりネームスペースが区切られます)。

パスの 2 番目の部分は、exteNd により自動的に作成されるもので、ハードディスク上にある元の Composer プロジェクトと同じディレクトリ構造を反映します。Composer プロジェクトのディレクトリ構造は、プロジェクトに対して名前が同じルートディレクトリと、作成された各 xObject タイプ (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で構成されます。たとえば、**Tutorial** という名前の Composer プロジェクトに、**LookupInventory** という名前の JDBC コンポーネントがある場合、ディスクディレクトリまたはファイル構造は次のようになります。

{ 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 (およびディレクトリ名) です。
- **LookupInventory** は、xObject です。

サーブレット

Deployment Wizard で生成される各サーブレット (サーブレットベースのサービストリガまたはサーブレット EJB ベースのサービストリガ) に対して、Java ソースファイルが、関連するサービスの名前で作成されます。また、配備オブジェクトをインストールする SilverCmd バッチファイルにより参照される Novell 配備記述子ファイル (XML) も作成されます。XML 記述子ファイルは、サーブレットの URI エイリアスを指定し、プロジェクト JAR での依存関係を規定します。さらに、エントリが、**BuildServlets.XML** または **BuildEJBs.XML** 記述子ファイルに作成されます。このファイルでは、どの Java ソースファイルがサーバでコンパイルされるかについて詳しく説明されます。

EJB

Deployment Wizard により生成される各 EJB に対して、EJB 配備 JAR の **meta-inf/ejb-jar.xml** という名前のマニフェストファイルにエントリが作成されます。このファイルには、EJB のタイプ (セッションまたはエンティティ)、各 EJB の環境設定、および EJB を構成するクラスが含まれます。また、作成する EJB およびそれらの JNDI 名を指定する **BuildEJBs.XML** 記述子ファイルにもエントリが作成されます。

EJB Jar の名前には、Deployment Wizard で指定した JAR ファイル名が使用されます。配備 EJB JAR ファイルおよびリモートインタフェース EJB JAR(どちらも配備中に Extend Application Server に作成されます) の名前も、プロジェクト JAR ファイル名に基いています。3 つの JAR ファイル名の名前付け規則は、次のとおりです。

- EJB JAR - **EJB-xxxx**
- EJB 配備 JAR - **EJBDeployxxxx**
- EJB リモートインタフェース JAR - **EJBStub-xxxx**

(ここで、**xxxx** は Project JAR ファイルの名前です)

次の例では、プロジェクト JAR ファイル名は **Production.jar** です。

- EJB JAR: **EJB-Production.jar**
- EJB 配備 / 構築 JAR: **EJBDeployProduction.jar**
- EJB リモートインタフェース JAR: **EJBStub-Production.jar**

ImportObjects.bat

このマスタバッチファイルには、Deployment Wizard で作成されたさまざまなソースおよび JAR ファイルをインポート、コンパイル、および配備する SilverCmd 呼び出しがすべて含まれています。

C Deployment Framework API マニュアル

Novell Composer Deployment Framework の Java Doc バージョンは、アプリケーションサーバの exteNd インストールプログラムにより作成される次のインストールディレクトリにあります。

{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 *providerURI* 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 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。

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

XML ドキュメントを参照する指定 URI を **org.apache.xml.Document** に変換するメソッドです。

XSL 機能

XSL を直接処理できるようにするために、メソッドが特定の Framework クラスに追加されました。新しいメソッドによって可能になった機能は、次のとおりです。

- XSL プロセッサインタフェースでは、XML ドキュメント「自体」(XML ドキュメント「および」XSL スタイルシートではなく) 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 に変換できます。
- 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は、XML、「または」XSL 処理命令の結果として作成された HTML を出力できます。

XSL メソッドを持つクラス

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。このメソッドシグニチャは、次のとおりです。

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

com.sssw.b2b.xs.servlet.GXSServiceRunner & com.sssw.b2b.xs.servlet.GXSEJBServiceRunner

```
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_URI* を検索し、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。

D

予約語

Java 言語キーワードは、配備コンテキスト文字列で使用しないでください。次の表に、Java キーワードのリストを示します。

Java キーワード

abstract	boolean	break
byte	case	catch
char	class	const
continue	default	do
double	else	extends
final	finally	float
for	goto	if
implements	import	instanceof
int	interface	long
native	new	package
private	protected	public
return	short	static
strictfp	super	switch
synchronized	this	throw
throws	transient	try
void	volatile	while

E 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。

JNDI

Java Naming and Directory Interface。Java プラットフォームに対する標準の拡張で、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名前付けスキームおよびディレクトリスキーム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 インタフェース。Bean 管理トランザクションはこの API に依存します。

Params (URL/Form)

4 つの標準的な Composer サービストリガタイプの 1 つ。このサーブレットタイプにより、ノードの名前として HTTP URI フォームのパラメータ、テキストとしてそれら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。

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に依存しないプロトコル。

XML (HTML form field)

4 つの標準的な Composer サービストリガタイプの 1 つ。このサーブレットタイプでは、ポストされたフォーム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。サーブレットでは、「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 (HTTP POST)

4 つの標準的な Composer サービストリガタイプの 1 つ。このトリガサーブレットのタイプにより、HTTP POST メソッドで送信された XML ドキュメントが抽出されます。これは、「パラメータ名と値」のペアを含む HTML Form POST とは異なります。この種類の HTTP 伝送のペイロードは、実際には加工されていない XML ドキュメントです。これは、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方法です。

XML (MIME multipart)

4 つの標準的な Composer サービストリガタイプの 1 つ。このサーブレットタイプにより、ファイルの入力タイプがあるフィールドを含むマルチパートのエンコードフォームからサービスの入力ドキュメントが抽出されます。サーブレットでは、「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 メタデータ

Composer で作成された exteNd オブジェクトは、すべて 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 メタデータと呼ばれます。exteNd ランタイムエンジンでは、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。

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「トランザクションに対応する」必要はありません。代わりに、Bean のトランザクション属性は記述子で設定でき、コンテナでは、Bean が機能できるトランザクションに対して適切な制御が行われるようになります。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 (通常はアプリケーションサーバ) によって制御される、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、サーバで接続をキャッシュする方が効果的です。

配備コンテキスト

配備コンテキストは、同じような名前のコンポーネントがあるサービス間でネームスペースが重複するのを避けるために使用できる名前の文字列 (これらの要素はピリオドで区切られます) です。

索引

N

P

S

T

U

W

X

あ

い

う

か

こ

さ

し

す

せ

た

と

は

ふ

ほ

め

や

ゆ

ら

り

ろ