

# Novell exteNd Composer™ Enterprise Server for BEA WebLogic Application Server

4.2

ユーザガイド

[www.novell.com](http://www.novell.com)



Novell®

## 保証と著作権

Copyright ©1999, 2000, 2001, 2002, 2003 SilverStream Software, LLC. All rights reserved.

SilverStream ソフトウェア製品は、SilverStream Software LLC により著作権とすべての権利が保留されています。

SilverStream は SilverStream Software, LLC の登録商標です。Novell は、Novell, Inc. の登録商標です。

ソフトウェアとマニュアルの所有権、および特許、著作権、およびそれに関連するその他のすべての財産権は常に、単独で排他的に SilverStream とそのライセンサーに保留され、当該所有権と矛盾するいかなる行為も行わないものとします。本ソフトウェアは、著作権法と国際条約規定で保護されています。ソフトウェアならびにそのマニュアルからすべての著作権に関する通知とその他の所有権に関する通知を削除してはならず、ソフトウェアとそのマニュアルのすべてのコピーまたは抜粋に当該通知を複製しなければなりません。本ソフトウェアのいかなる所有権も取得するものではありません。

Jakarta-Regexp Copyright ©1999 The Apache Software Foundation. All rights reserved. Ant Copyright ©1999 The Apache Software Foundation. All rights reserved. Xalan Copyright ©1999 The Apache Software Foundation. All rights reserved. Xerces Copyright ©1999-2000 The Apache Software Foundation. All rights reserved. Jakarta-Regexp, Ant, Xalan, Crimson, および Xerces ソフトウェアは、The Apache Software Foundation によりライセンスを付与され、Jakarta-Regexp, Ant, Xalan, Crimson, および Xerces のソースおよびバイナリ形式での再配布および使用は、変更のあるなしにかかわらず、以下の条件が満たされることを前提として許可されます。1. ソースコードの再配布に上記の著作権に関する通知、条件のリスト、および以下の権利放棄に関する通知が記載されていること。2. バイナリ形式の再配布では上記の著作権に関する通知、条件のリスト、および以下の権利放棄に関する通知がマニュアルまたは配布の際に提供されるその他の資料、あるいはその両方に記載されていること。3. エンドユーザの資料には、適宜、以下の通知を再配布の際に含めてください。「この製品には、Apache Software Foundation (<http://www.apache.org/>) により開発されたソフトウェアが含まれています」代わりに、この謝辞をソフトウェア自体に表示し、当該サードパーティに対する謝辞が通常表示される場所に表示することもできます。4. 「The Jakarta Project」、「Jakarta-Regexp」、「Xerces」、「Xalan」、「Ant」、および「Apache Software Foundation」は、書面による事前の許可なく、このソフトウェアから派生する製品を推薦したり、販売促進したりするのに使用してはなりません。書面による許可については、[apache@apache.org](mailto:apache@apache.org) <<mailto:apache@apache.org>> にお問い合わせください。5. 本ソフトウェアから派生する製品は「Apache」と呼ばれてはならず、「Apache」は The Apache Software Foundation の事前の書面による許可なくその名前に使用することはできません。本ソフトウェアは「現状のまま」提供され、いかなる明示的、暗黙の保証も行われるものではありません。販売可能性や特定の目的に対する適合性に対する暗黙の保証も行われません。いかなる場合でも、Apache Software Foundation またはその関係者はいかなる直接的、間接的、偶発的、特別な、免除的、または結果的な損害（代替品やサービスの調達、使用機会、データ、または利益の喪失、または業務の中断などを含む）についても、理論上責任がある場合でも、契約上の責任がある場合でも、厳密な責任、または瑕疵（怠慢などを含む）があった場合でも、ソフトウェアの使用の過程で生じ、当該損害の可能性を助言した場合であっても、責任を持ちません。

Copyright ©2000 Brett McLaughlin & Jason Hunter. All rights reserved. ソースおよびバイナリ形式での再配布および使用は、変更のあるなしにかかわらず、以下の条件が満たされることを前提として許可されます。1. ソースコードの再配布に上記の著作権に関する通知、条件のリスト、および以下の権利放棄に関する通知が記載されていること。2. バイナリ形式の再配布では上記の著作権に関する通知、条件のリスト、および以下の権利放棄に関する通知がマニュアルまたは配布の際に提供されるその他の資料、あるいはその両方に記載されていること。3. 「JDOM」という名前は、書面による事前の許可なく、このソフトウェアから派生する製品を推薦したり、販売促進したりするのに使用してはなりません。書面による許可については、[license@jdom.org](mailto:license@jdom.org) <<mailto:license@jdom.org>> にお問い合わせください。4. 本ソフトウェアから派生する製品は「JDOM」と呼ばれてはならず、「JDOM」は JDOM Project Management ([pm@jdom.org](mailto:pm@jdom.org)) <<mailto:pm@jdom.org>> の事前の書面による許可なくその名前に使用することはできません。本ソフトウェアは「現状のまま」提供され、いかなる明示的、暗黙の保証も行われるものではありません。販売可能性や特定の目的に対する適合性に対する暗黙の保証も行われません。いかなる場合でも、Apache Software Foundation またはその関係者はいかなる直接的、間接的、偶発的、特別な、免除的、または結果的な損害（代替品やサービスの調達、使用機会、データ、または利益の喪失、または業務の中断などを含む）についても、理論上責任がある場合でも、契約上の責任がある場合でも、厳密な責任、または瑕疵（怠慢などを含む）があった場合でも、ソフトウェアの使用の過程で生じ、当該損害の可能性を助言した場合であっても、責任を持ちません。

Sun Microsystems, Inc. Sun, Sun Microsystems, Sun Logo Sun, Sun のロゴ、Sun Microsystems、JavaBeans、Enterprise JavaBeans、JavaServer Pages、Java Naming and Directory Interface、JDK、JDBC、Java、HotJava、HotJava Views、Visual Java、Solaris、NEO、Joe、Netra、NFS、ONC、ONC+、OpenWindows、PC-NFS、SNM、SunNet Manager、Solaris sunburst design、Solstice、SunCore、SolarNet、SunWeb、Sun Workstation、The Network Is The Computer、ToolTalk、Ultra、Ultracomputing、Ultraserver、Where The Network Is Going、SunWorkShop、XView、Java WorkShop、Java Coffee Cup のロゴ、Visual Java、および NetBeans は、米国およびその他の国の Sun Microsystems, Inc. の商標ならびに登録商標です。

Copyright ©2001 Extreme!Lab, Indiana University License. <http://www.extreme.indiana.edu>. 同社により許可が無料で、Indiana University ソフトウェアと関連する Indiana University のドキュメントファイル (「IU Software」) のコピーを取得したすべての人に、制限なく IU Software を取り扱うために付与されます。その際に、IU Software の使用、コピー、変更、マージ、公開、配布、サブライセンス、または販売、あるいはそれらのすべてに関する権利に制限はなく、IU Software が指定した人に以下の条件に基づき権利を付与します。上記の著作権に関する通知とその許可に関する通知は、IU Software のすべてのコピーおよび主要部分に含まれる必要があります。本 IU ソフトウェアは「現状のまま」提供され、いかなる明示的、暗黙の保証も行われるものではありません。販売可能性、特定の目的に対する適合性や権利侵害がないことに対する暗黙の保証も行われません。いかなる場合でも、作成者または著作権所有者は、契約上の責任がある場合でも、厳密な責任、または瑕疵 (怠慢などを含む) があつた場合でも、IU Software に関連して、または IU Software の使用やその他の取引の過程で生じた場合であっても、クレーム、損害、その他の責任について責任を持ちません。

本ソフトウェアは、著作権を持つ SSLava™ Toolkit の一部です。Copyright ©1996-1998 by Phaos Technology Corporation. All Rights Reserved.

Copyright © 1994-2002 W3C® (Massachusetts Institute of Technology, Institut National de Recherche Informatique et en Automatique, Keio University), all Rights Reserved. <http://www.w3.org/consortium/legal>. この W3C の成果物 (ソフトウェア、ドキュメント、またはその他の関連品目を含む) は、以下のライセンスの下で著作権所有者により提供されています。この成果物の取得、使用、またはコピー、あるいはそれらのすべてにより、ライセンシーは以下の条件を読み、理解し、遵守することに合意するものとします。本ソフトウェアとそのドキュメントの使用、コピー、変更、および配布は、変更のあるなしにかかわらず、いかなる目的でも無料または本契約で許可された使用料をもって許可されます。ただし、変更箇所を含む本ソフトウェアとドキュメントのすべてまたはその一部に以下のとおり記述することを前提とします。1. この通知の全文は、再配布物または派生物のユーザが見やすい場所に掲示しなければなりません。2. すべての前もって存在する知的所有権の放棄、通知、または条件。存在しない場合は、以下の形式の短い通知 (ハイパーテキストが望ましい、テキストでも良い) を再配布または派生コードの本文内で使用しなければなりません。「Copyright © [date-of-software] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. <http://www.w3.org/Consortium/Legal/>」3. W3C のファイルに変更または修正を加えた場合はその日付を含む通知。(コードが派生する場所への URI を示すことをお勧めします。) 本ソフトウェアは「現状のまま」提供され、いかなる明示的、暗黙の保証も行われるものではありません。販売可能性、特定の目的に対する適合性やサードパーティの特許、著作権、商標またはそのたの権利を侵害しないことに対する暗黙の保証も行われません。著作権の所有者は本ソフトウェアまたはマニュアルの使用の結果生じる、直接的、間接的、特殊な、または結果的な損害に対していかなる責任も負いません。著作権所有者の名前および商標は、特別な書面による事前の承諾なしにソフトウェアに関する広告や広報に使用してはなりません。本ソフトウェアおよび関連する資料の著作権の所有権は常に、著作権所有者に帰属するものとします。

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

[www.novell.com](http://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](http://www.novell.com/documentation) を参照してください。

# 目次

このガイドについて	7
<b>1 Composer Enterprise Server へようこそ</b>	<b>11</b>
exteNd Composer Enterprise Server とは	11
コンポーネントおよびサービスについて	12
<b>2 サーバ概要</b>	<b>15</b>
サーバ仕様	15
運用ランタイム環境	16
メタデータアーキテクチャ	16
配備オプション	16
Composer Enterprise Server の管理制御	16
キャッシュの調整	18
Least-Recently-Used (LRU) キャッシュアルゴリズム	18
ライセンスのアップデート	20
<b>3 配備の計画</b>	<b>21</b>
サービスのインスタンス化	21
Web サービスと JMS サービス	22
スタンドアロン配備	22
統合アプリケーションの配備	23
接続プール	23
データベース接続プール	23
Connect 用リソースプール	24
セキュリティ役割	24
トランザクション制御	24
XML リソースの公開	24
追加の Java クラス	25
<b>4 プロジェクトの配備</b>	<b>27</b>
配備プロセス	27
はじめに	27
ランタイム環境の決定	27
配備オブジェクトの作成	28
配備オブジェクトのインストール	28
ブラウザベースの配備	29
Deployment Manager について	33
Web サービスのサービストリガについて	35
サブレットベースのサービストリガ	35
EJB ベースのサービストリガ	36
アプリケーションベースのサービストリガ	37

SOAP ベースのサービストリガ	37
JMS サービストリガ	38
Composer Deployment Wizard の使用	38
一般情報パネル	39
サーブレットベースのサービストリガパネル	40
EJB ベースのサービストリガパネル	43
EJB トリガパネル	44
SOAP ベースのトリガパネル	45
リソース配備パネル	47
プロジェクト変数リマッピングパネル	48
JMS サービストリガパネル	48
自動インストーパネル	49
手動による配備オブジェクトのインストール	50
<b>5 配備フレームワークの使用</b>	<b>51</b>
Composer のサービストリガ階層の拡張	52
サービストリガ拡張子の作成方法	53
サーブレットベースのサービストリガの作成	53
GXSServiceRunnerEx	55
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	55
サービストリガの新規作成	56
アプリケーションでのサービス EJB の使用	58
<b>6 トランザクションの管理</b>	<b>61</b>
exteNd でのトランザクション制御	61
WebLogic Application Server に対するトランザクション配備の考慮事項	62
サーブレット配備の考慮事項	62
EJB 配備	62
EJB 配備の考慮事項	65
JDBC トランザクション制御 - User Transactions の許可	65
WebLogic での JTS 対応データソースのセットアップ	67
参照	68
<b>A Deployment Framework API マニュアル</b>	<b>69</b>
クラスおよびインタフェース	69
インタフェース	70
ファクトリクラス	71
XSL 機能	72
クラス	72
<b>B 配備オブジェクトの内容</b>	<b>75</b>
プロジェクト JAR	75
<b>C Server 用語集</b>	<b>77</b>

# 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 BEA WebLogic Application Server に配備する方法について説明します。このドキュメントは、『exteNd Composer ユーザガイド』に付属しています。

## 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や、ランタイム環境で Composer サービスの配備および管理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 のアプリケーション構築例および Web サービスに関する J2EE の概念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 WebLogic の配備の概念についても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、「サーバの概要」	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 Deployment Wizard の使用方法について説明します。

章	説明
第 5 章、「配備フレームワークの使用」	非標準配備に対して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について説明します。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、「トランザクション管理」	アプリケーションのトランザクションに関する側面を制御するためのオプションについて説明します。
付録 A、「Deployment Framework API マニュアル」	カスタムトリガの作成およびプログラムによるサービスの呼び出しのために使用可能なクラスとメソッドについて説明します。
付録 B、「配備オブジェクトのコンテンツ」	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コンテンツ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

## 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、および配備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 Connects** - 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企業情報ソースの XML 対応を可能にする製品ファミリーです。

このガイドでは、exteNd Composer Enterprise Server について説明します。exteNd Composer の詳細については、『exteNd Composer ユーザガイド』を参照してください。

## 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あり、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。サーバでは、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、フェイルオーバーとセキュリティ)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」の両方が装備されます。そしてフレームワークでは、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における他のプログラムとの統合方法に関連するサービスについて、配備された Composer サービスのサーバ側でのカスタマイズ機能も提供されます。

このガイドでは、Composer アプリケーションを WebLogic Application Server に配備するという前提で解説します。

## 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が基礎になっています。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少細分化された、(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。

「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、および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、1 つまたは複数の 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メッセージの送信、または元のリクエスタへ 1 つまたは複数の 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 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。

Extend バージョン 2.5 では、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で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のみ該当します。

次の図には、サーバとツールである exteNd スイートとの対応が示されています。

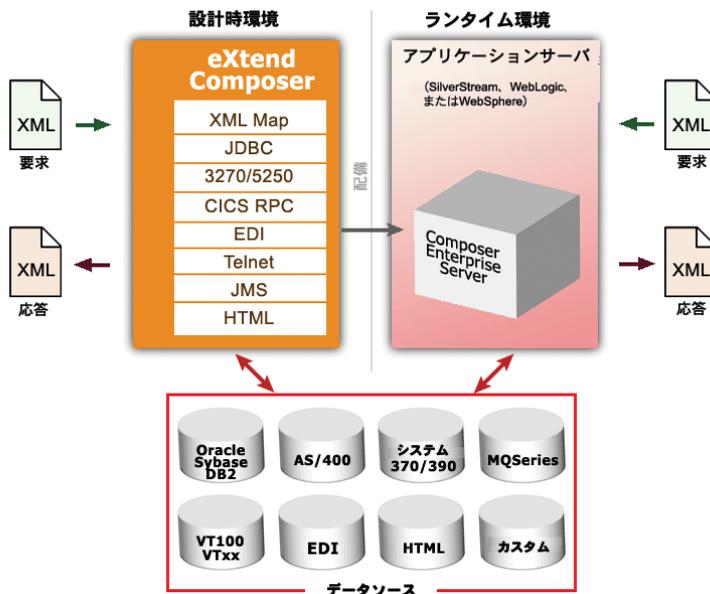


図 1-1

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を、それらが 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配備します。exteNd Connect では、企業ビジネスシステムが XML 対応になり、アプリケーションの操作によるランタイム管理が可能になります。

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

イベントの繰り返しや構成要素、条件、エラー、ログ、カスタム ECMAScript 関数のグループ化などを含む、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。

- ◆ 端末データインター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

## サーバ概要

### サーバ仕様

exteNd Composer Enterprise 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で構成される、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 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
- ◆ 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 ラップ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。

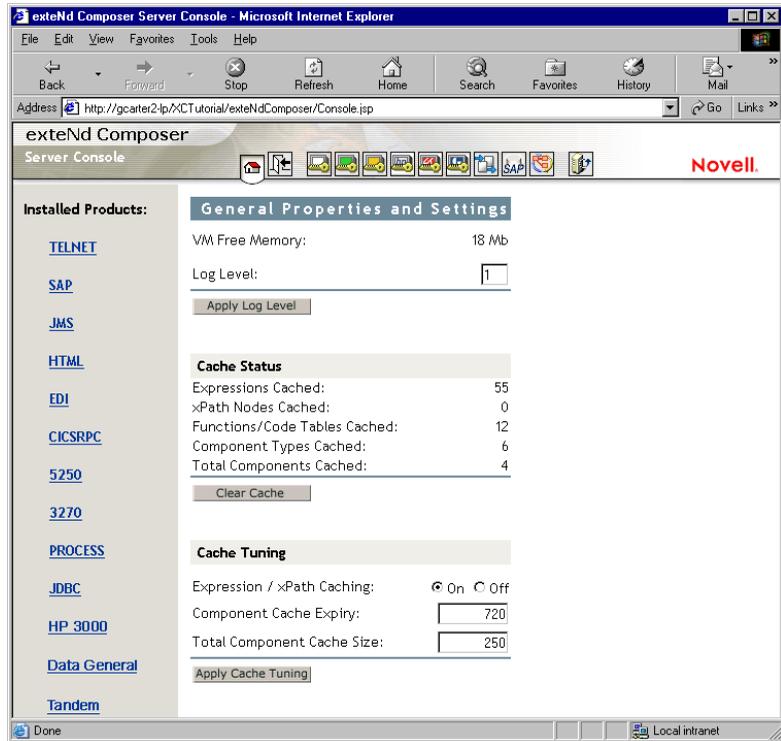
オプションとして手動で配備するには、生成した配備 JAR ファイルをステージングディレクトリに格納するよう Composer を設定します。一般的に、この方法はアプリケーションの配備を担当するサーバの管理者が大規模なアプリケーションサーバをインストールする場合に使用されます。Composer ですべての必要な配備ファイルが生成されると、管理者は IT 組織によって定義された標準を使用してアプリケーションサーバにインストールできます。

## 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ページしたい場合は、[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 Cache 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ページされます。「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 (メモリの圧縮とページ) 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」を探する必要があります。

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。既定値は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残す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(キャッシュブルーナ)により強制されます。ブルーナは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。ブルーナは、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)ページされ、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、次のことを考慮します。

- ◆ サービスをインスタンス化する方法 (例: パラメータを持つ受信URIに対するインスタンス化、他のプログラムによるインスタンス化)
- ◆ 接続要件 (接続プールを必要とするかどうか)
- ◆ トランザクション制御が必要であるかどうか (トランザクションモデルが「コンテナ」管理、または「Bean」管理かどうか)
- ◆ 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。配備されるサービスは、XML のソースが分からない場合でも XML ドキュメントで実行されるため、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

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：** 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 exteNd Composer Enterprise Server のコンテキスト内でインスタンス化され、必要な手順をすべて実行します ( 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 入力に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、[16 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の配備

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 A](#)、「[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があるサービスを配備する場合、適切な 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 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に対するオプションの詳細については、[第 1 章](#)、「[配備フレームワークの使用](#)」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限に抑えるために、exteNd Composer Enterprise 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。WebLogic Application Server では、これらのプールは、**WebLogicdatasource\_name** の形式を受け入れる JNDI 名によって特定されます ( ここで、**datasource\_name** は、WebLogic データソースの名前で置き換える必要があります)。

WebLogic 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)。

## Connect 用リソースプール

exteNd Composer Enterprise 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 Composer Enterprise Server の接続プールは、個別の exteNd コンポーネント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、コネクタ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、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、配備によって起こりうる影響をすべて設計時に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 配備が必要と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。

WebLogic では、サーブレットおよび Web パスの両方が Web アプリケーションに関連付けられます。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。

## 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となる場合があります。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 WebLogic Server で使用 ( 参照 ) できるようにする必要があります。

次のいずれかの方法で、サーバ CLASSPATH にカスタム Java クラスの JAR ファイルを追加することができます。

- ◆ **setenv** コマンドファイルに JAR ファイルを追加する。  
**setEnv** コマンドファイルで WLISERVERCP 環境変数に JAR ファイルを追加すると、マシンにインストールした WebLogic Integration ドメインすべてに対して使用できるようになります。**startWeblogic** を実行するたびに、サーバを起動する Java コマンドを発行する前に **setEnv** コマンドが呼び出されます。
- ◆ **startWeblogic** コマンドファイルに JAR ファイルを追加する。  
**startWeblogic** コマンドファイルで指定した CLASSPATH に JAR ファイルを追加すると、そのコマンドファイルによって起動した特定のサーバのみで、Java クラスを使用できます。



# 4

## プロジェクトの配備

### 配備プロセス

Composer プロジェクトを WebLogic 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 WebLogic 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 サンプル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、このサンプル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（[第 1 章、「配備フレームワークの使用」](#)を参照）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、個別に公開する必要があります。公開する形式の例は、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 )。次の配備時に、exteNd で配備情報を復元できます。

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 WebLogic 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 B, 「配備オブジェクトの内容」](#) を参照してください。

## 配備オブジェクトのインストール

WebLogic ノードに配備オブジェクトをインストールするには、3 つのオプションがあります。

- ◆ (設計時) Composer 内で、生成される JAR ファイルを、Deployment Wizard によって WebLogic ノードで実行されているサーブレット (Composer Server 製品インストールの一部) である Deployment Manager に自動的に送信するようにする。
- ◆ Web ブラウザを使用して Composer 配備コンソール移動し、ブラウザのセッション内で配備を実行する (次の節の説明を参照してください)。
- ◆ (*xcdeploy.bat* ユーティリティを使用して) コマンドラインモードで Deployment Manager を実行することで、オブジェクトを手動でインストールする。45 ページ「手動による配備オブジェクトのインストール」を参照してください。

## ブラウザベースの配備

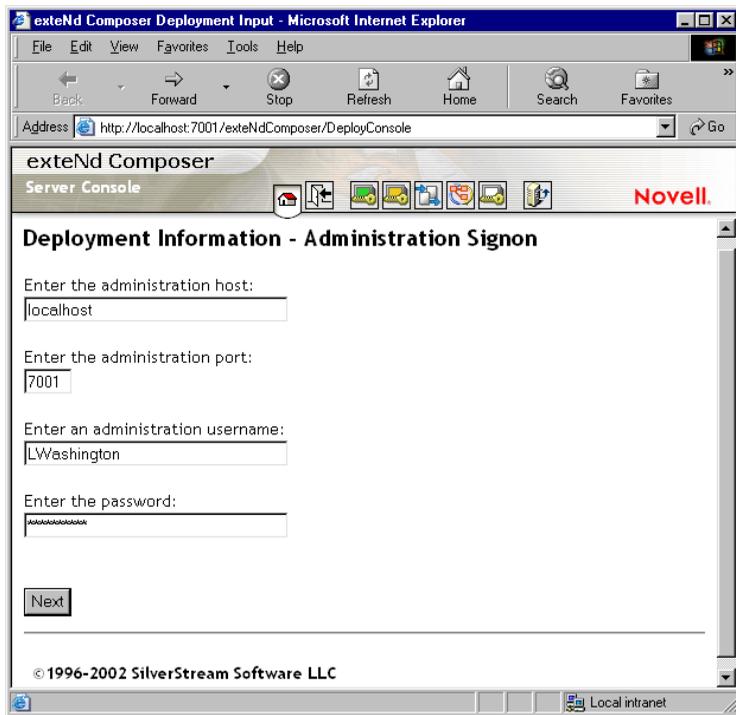
(前に説明したように) 配備オプションの1つとして、Composer 配備コンソールを使用して、ブラウザセッション内で配備プロセスを初期化、管理できるようにします。この方法の利点は、設計時と配備時の操作を明確に区分できる点です。(Composer の設計側エディタで作業している) アプリケーション開発者は、配備可能なプロジェクト JAR を作成して、ネットワーク上のステージングディレクトリにポストできます。その後しばらくして、サーバ管理者または配備専門家が、ここで説明するコンソールを使用した方法によって JAR を取得、配備できます。

### ➤ 配備コンソールを使用する

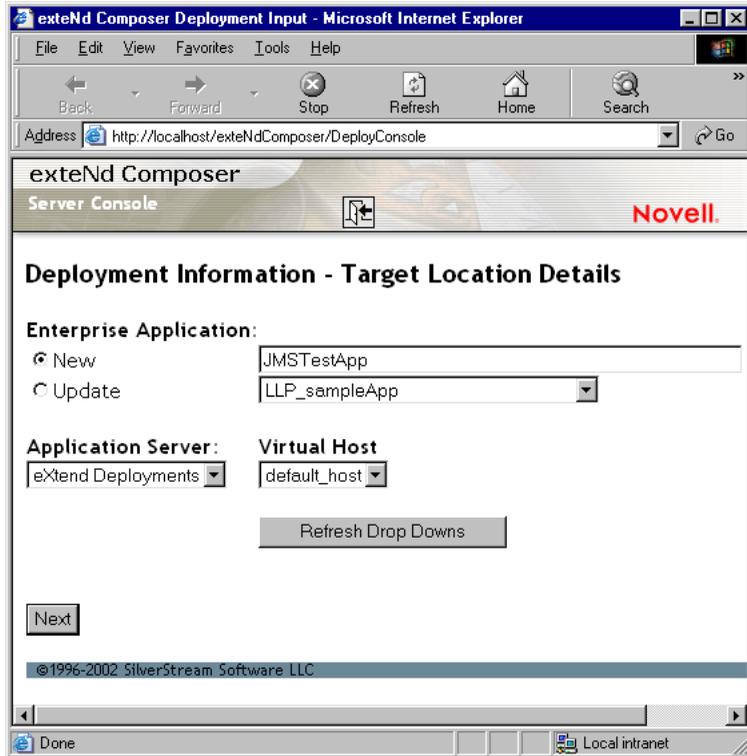
- 1 Composer Enterprise Server がインストール済みおよび動作中で、アプリケーションサーバが実行中であることを確認してください。
- 2 Web ブラウザを起動して、メイン Composer サーバコンソールに移動します(「サーバ管理コンソールにアクセスする」の説明を参照してください)。
- 3 ツールバーの **[Deploy from Browser]** ボタンをクリックします。



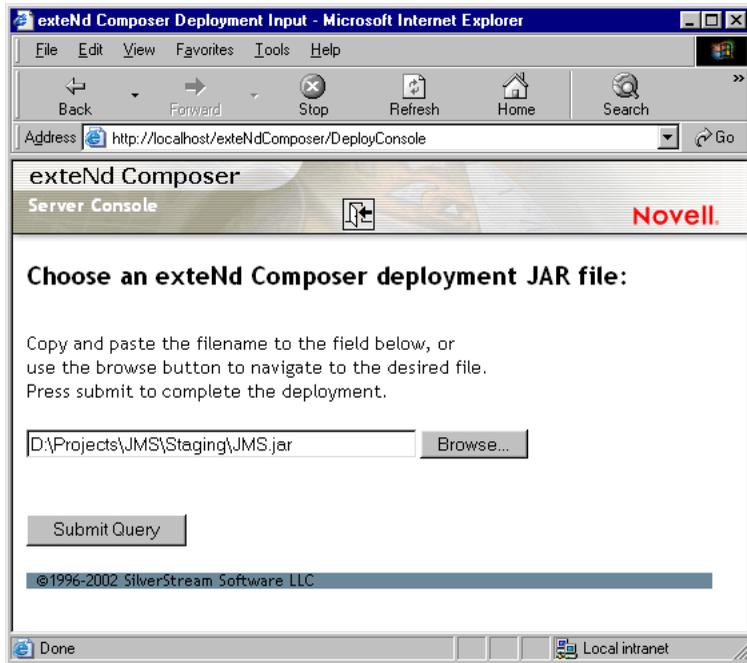
- 4 表示される画面(次の図を参照)で、必要な情報を入力します。



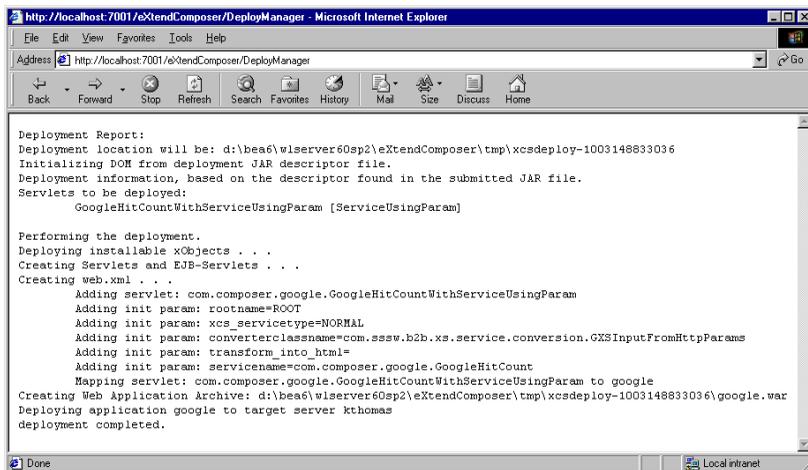
- 5 [Next] ボタンをクリックします。新しい画面が表示されます。



- 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、配備された項目を詳細に示したレポートがブラウザに生成されます (次を参照)。



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 Composer Enterprise Server にインストールされた **xcs-all-wlxx.jar** ファイル (**xx** には、WebLogic アプリケーションサーバのバージョン番号が入ります) に含まれるサブルーットである 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インストールされます。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)。

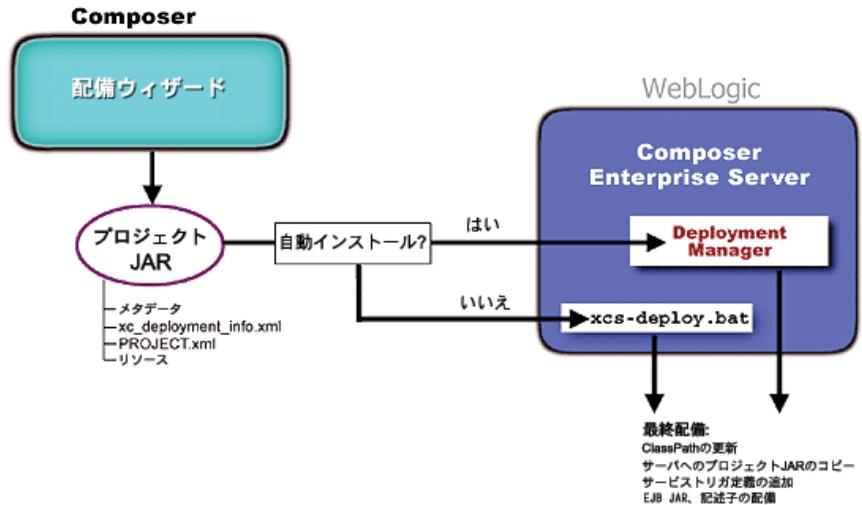


図 4-1

Deployment Manager の URI は、次のとおりです。

**http://<hostname>/<Virtual\_host\_path>/exteNdComposer/DeployConsole**

ここで、

- ◆ **hostname** は、WebLogic 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 ( 該当する場合 ) を表すパスです。

たとえば、次の WebLogic 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 ( パス = '/' )

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

**http://accounting/exteNdComposer/DeployConsole**

**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 ( 次の「Composer Deployment Wizard の使用」を参照)。配備オブジェクトを手動でインストールする場合は、xcs-deploy ユーティリティを使用すると、その操作を実行できます ( 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 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。

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 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 (36 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、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 (第章「配備フレームワークの使用」を参照)。図では、Deployment Wizard によって生成されるサーブレットタイプの一般的なシナリオについて説明します。

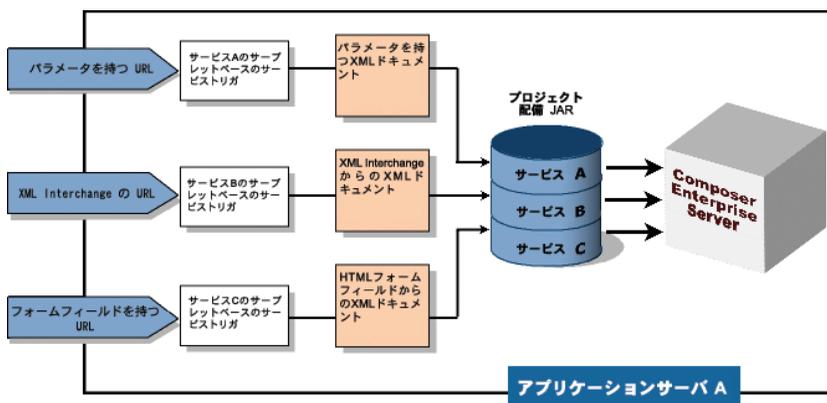


図 4-2

## 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 )。

**注記：** exteNd Composer から EJB を配備するために、(WebLogic サーバプロセスを起動するユーザに対して) WebLogic プロセス UID??PATH 環境変数で Java SDK bin パスが必要です。

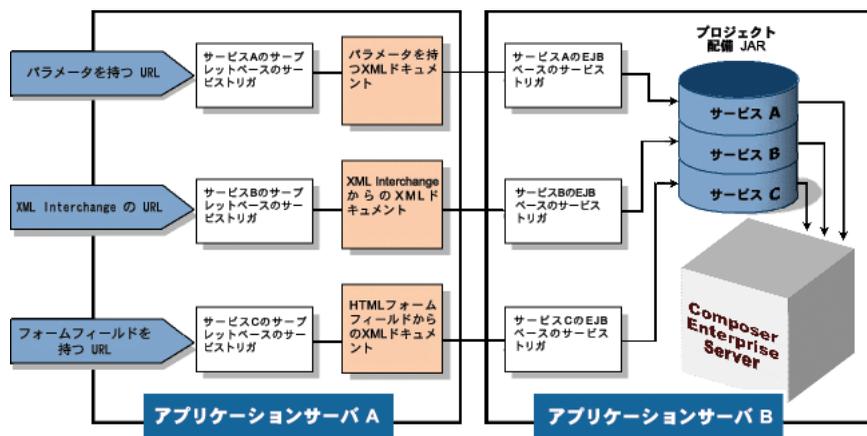


図 4-3

## アプリケーションベースのサービストリガ

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、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、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 1 章、「[配備フレームワークの使用](#)」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。詳細については、次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。その他のパネルには、入力必須のフィールドがあります。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が表示されます。

`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が表示されます (次の図を参照)。

## 一般情報パネル

Select a target application server for your deployment. Next specify a staging directory for all the deployment objects (e.g. the project JAR, Service Trigger Servlets/EJBs, ImportObjects.bat file, and the application server deployment descriptor files). Then specify a name for the project JAR file to contain your project's Services and supporting objects (components, resources, etc). Finally specify the deployment context for locating Services in the project JAR (use the same form as a Java package name).

Deployment Server Type:  
WebLogic

Deployment Staging Directory:  
D:\dwb1.0\extendComposer\Samples\TutorialEnd\staging

Project JAR Filename:  
TutorialEnd.jar

Deployment Context in the project JAR:  
(separate name elements by periods, e.g. com.acme.accounting)  
com.composer.tutorial

Help Back Next Finish Cancel

このパネルを使用して、配備の一般情報を指定します (次に詳しく説明します)。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

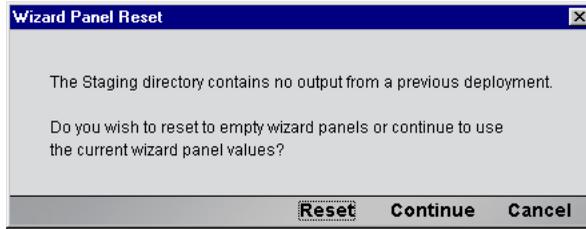
この配備のアプリケーションサーバタイプを [Novell]、[IBM WebSphere]、または [BEA WebLogic] から選択します。これにより、選択されたサーバ環境に固有な配備ファイルが Composer により作成されます。

### 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の (新しい) 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。

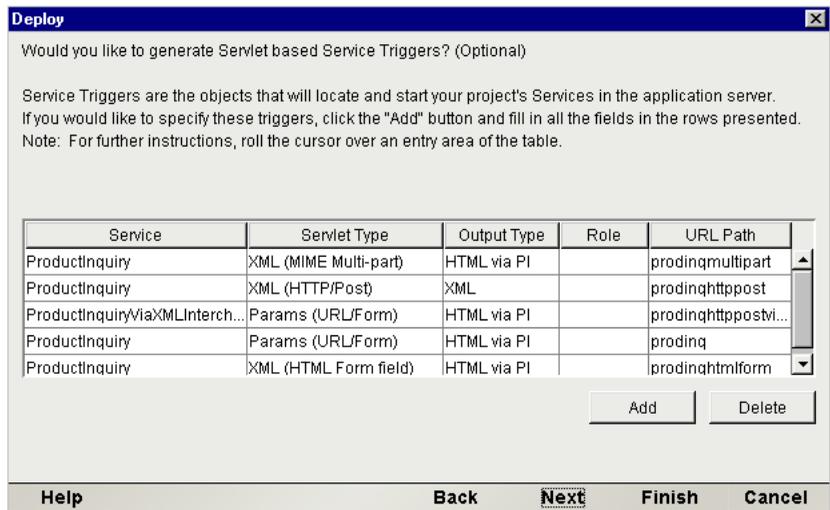
## プロジェクト JAR ファイル名

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징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」入カ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カ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カ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]

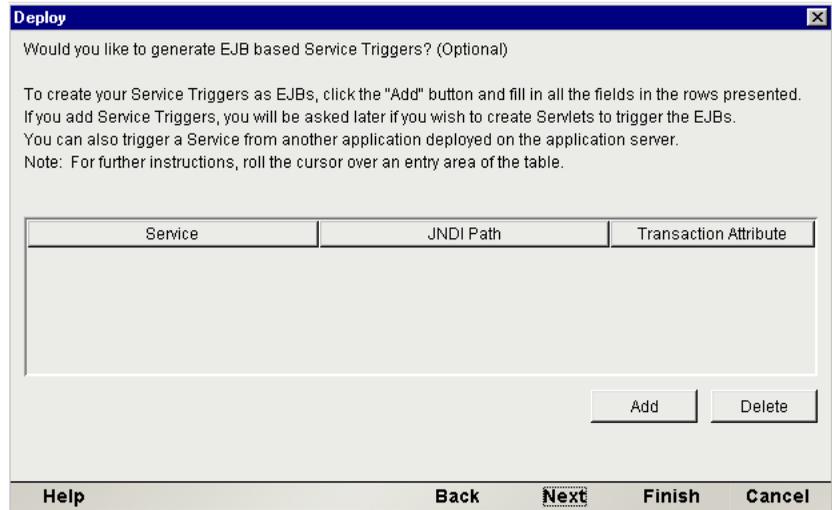
[URI Path] は、このサーブレットに割り当てられている URI の固有な名前を提供します。この名前は、WebLogic サーバ名およびプロジェクト JAR 名とともに、プロジェクトに含まれるすべてのサーブレットのURIコンテキストパスとなります。

たとえば、[URI Path] が「WebStore/ProductInquiry」と指定されていて、サーバ名が「stratus」となる場合、完全な URI は次のようになります。

**http://stratus/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。



The image shows a 'Deploy' dialog box with a title bar containing a close button (X). The main text asks 'Would you like to generate EJB based Service Triggers? (Optional)'. Below this, there is explanatory text: 'To create your Service Triggers as EJBs, click the "Add" button and fill in all the fields in the rows presented. If you add Service Triggers, you will be asked later if you wish to create Servlets to trigger the EJBs. You can also trigger a Service from another application deployed on the application server. Note: For further instructions, roll the cursor over an entry area of the table.' A table with three columns is shown: 'Service', 'JNDI Path', and 'Transaction Attribute'. The table is currently empty. Below the table are 'Add' and 'Delete' buttons. At the bottom of the dialog, there are navigation buttons: 'Help', 'Back', 'Next', 'Finish', and 'Cancel'.

Service	JNDI Path	Transaction Attribute
---------	-----------	-----------------------

Buttons: Add, Delete

Navigation: Help, Back, Next, Finish, Cancel

### [Service]

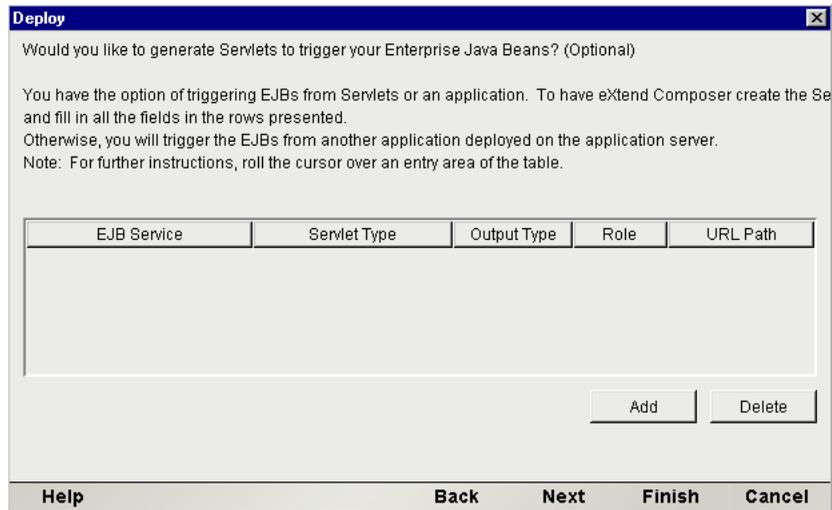
[Service]フィールドをクリックすると表示されるドロップダウンリストからサービスを選択します。このリストには、プロジェクトに含まれるサービスの名前が表示されます。

### [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。



The image shows a 'Deploy' dialog box with the following content:

Would you like to generate Servlets to trigger your Enterprise Java Beans? (Optional)

You have the option of triggering EJBs from Servlets or an application. To have extend Composer create the Servlets and fill in all the fields in the rows presented.

Otherwise, you will trigger the EJBs from another application deployed on the application server.

Note: For further instructions, roll the cursor over an entry area of the table.

EJB Service	Servlet Type	Output Type	Role	URL Path

Buttons: Add, Delete

Footer: Help, Back, Next, Finish, Cancel

### [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ブグループを作成し、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。

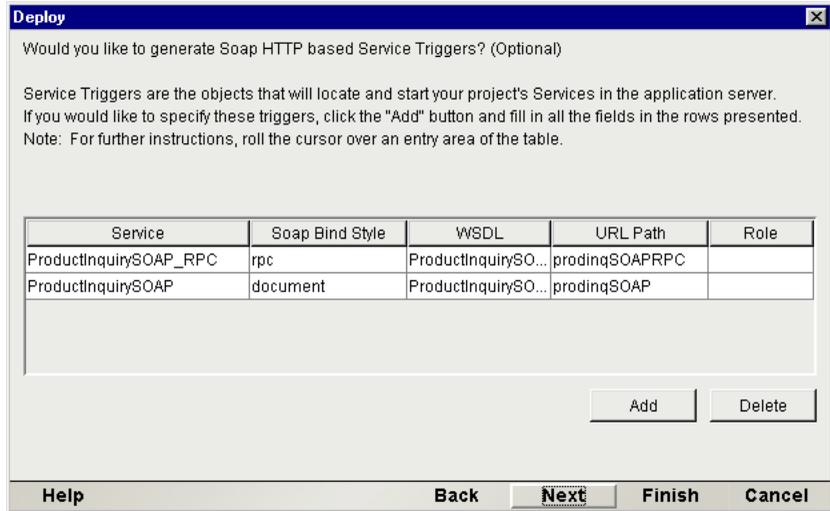
## [URI Path]

[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、WebLogic サーバ名および Novell データベース名とともに、このサ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] が「**WebStore/ProductInquiry**」と指定されていて、サーバ名が「**stratus**」となる場合、完全な URI は次のようになります。

**http://stratus/WebStore/ProductInquiry**

## SOAP ベースのトリガパネル

サ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レ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。

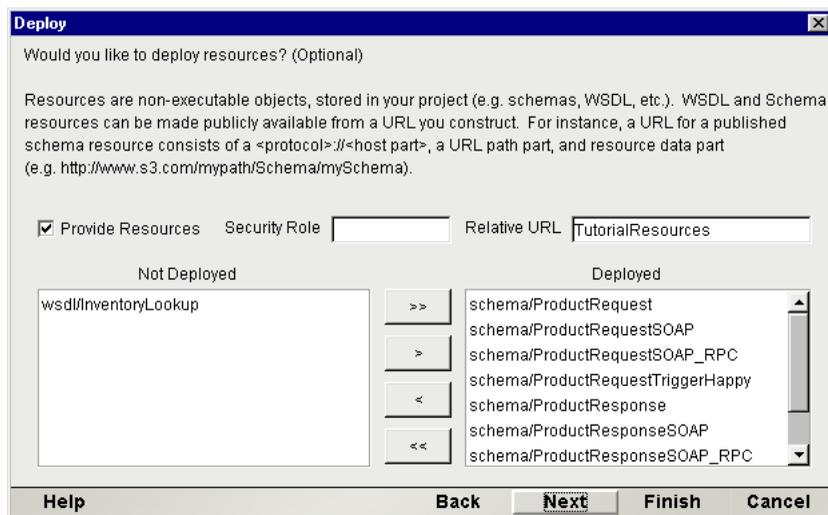
### [Role]

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、リソースチェーンを参照する 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。

## 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** ファイルと一致なくなります。通常、これで問題ありません。

Deploy

This panel allows you to change some or none of the Project Variables to their final runtime values before the project is deployed. Your changes will only affect the deployed project and not the current test values assigned in Composer.

Project:  
Main ProjectTutorialEnd

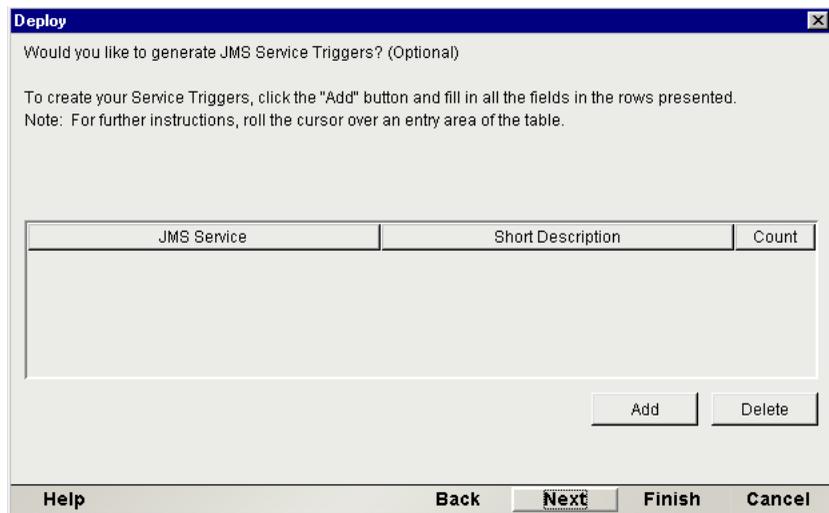
Element Name	Text Value
Resources	http://localhost:80/XCTutorial/SilverStream/Objectsto...

Help Back Next Finish Cancel

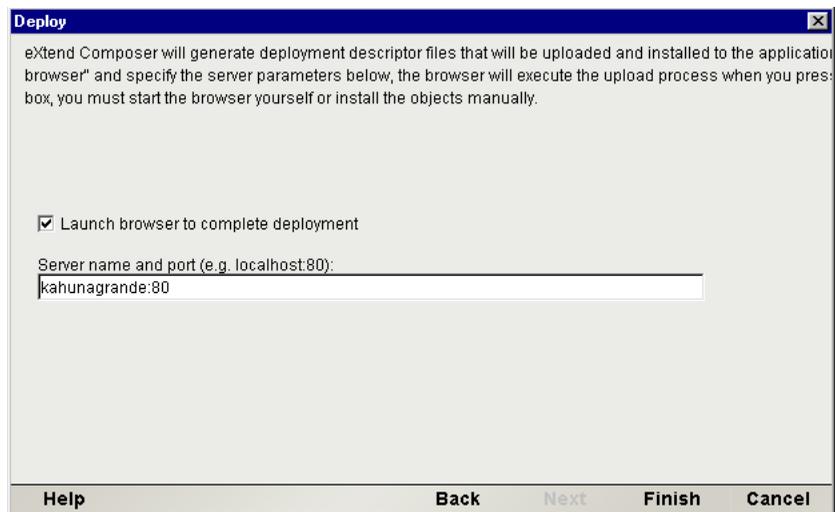
## JMS サービストリガパネル

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。終了すると、[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



JAR をステージングディレクトリ (一番最初のウィザードパネルで指定したディレクトリ) に配置するだけの場合、単純に [Finish] をクリックします。

プロジェクトをアプリケーションサーバに即時に配備する場合は、[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、配備コンソール画面が表示されます。この章の前半の「ブラウザベースの配備」を参照してください。

## 手動による配備オブジェクトのインストール

プロジェクトを手動で配備するには、コマンドラインから GXSWLDeployHandler クラスを実行する必要があります。これを正常に実行するには、次の 4 つをクラスパスに含める必要があります。

- 1 exteNdComposer\lib ディレクトリ
- 2 exteNd Composer サーバ jar
- 3 WL の **weblogicaux.jar**
- 4 WL のクラスディレクトリ

2 つのシステムプロパティ (つまり、weblogic.system.home および weblogic.class.path) を設定する必要があります。この設定例は、次のとおりです。(独自のディレクトリおよび jar ファイル名を反映するように適切に代入して) コマンドラインから次の 2 つのコマンドを実行します。

```
set
WL_CLASSPATH=c:\weblogic\classes;c:\weblogic\lib\weblogicaux.jar;c:\weblogic\exteNdComposer\lib;c:\weblogic\exteNdComposer\lib\xcs-all-w151.jar;

java -cp %WL_CLASSPATH% -Dweblogic.class.path=%WL_CLASSPATH% -Dweblogic.system.home=c:\weblogic\com.sssw.b2b.xs.deploy.wl.GXSWLDeployHandler -jar D:\tmp\staging\test2.jar
```

# 5

## 配備フレームワークの使用

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 (Novell exteNd 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化する、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。

いずれ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。そのため、この章では、フレームワークアーキテクチャについて説明し、フレームワーク API を使用して希望とする配備 / 呼び出し目標を達成する方法についても説明します。

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の簡単な配備として、exteNd の Workbench を使用する方法の詳細については、別のガイド『exteNd Workbench での Composer プロジェクトの使用』(Composer マニュアルスイートの一部) を参照してください。

## 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。ここでは、簡単な概要を示します。

## フレームワークオブジェクト内で使用する完全修飾サービス名の作成方法

フレームワークオブジェクト(サブレットなど)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。

次の行は、完全修飾サービス名の例です。

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

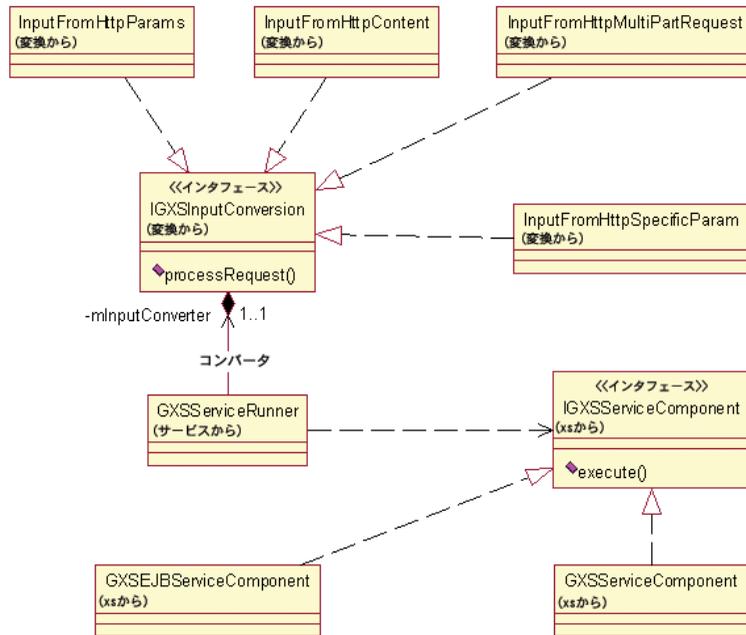
ここで、

- ◆ `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。

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子の作成方法

### ➤ Web サービスの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 開発環境のいずれかを使用して、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 オブジェクト作成に 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。

## サブレットベースのサービストリガの作成

サブレットの階層は、HttpServlet から派生した 1 つの抽象クラスに基づきます。抽象クラス GXSServiceRunner 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は、抽象メソッド `getServiceName()` の呼び出しによってすべての要求が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 `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または `IGXSInputConversion` インタフェースを実装するコンバータオブジェクト、あるいはそ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では、`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では、サービス実行側のさまざまな「ゲッター」メソッドを呼び出してプロパティを取得できるよう、`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 (新しいアークテクチャでは、受け取られる入力タイプ、およびサーブレットまたは EJB のどちらからサービスが呼び出されるか決定する情報は、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	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を使用されるクラス	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	出力ドキュメントを HTML に変換する場合、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だけ必要。	output_xsl_URI	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、`servicename` パラメータを追加して、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 `processRequestEx()` メソッドの説明は、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 )。複数の入力ドキュメントを持つクラスに対して使用できるため、メソッドによって、サーブレットがバインドされた `Composer` サービスコンポーネントの実行時に使用する XML ドキュメントを含む `String` の「配列」が返されます。要求の処理時に、XML ドキュメントのバインド情報がない場合、空の `String` 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、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 ヘルパー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。

## サービストリガの新規作成

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促す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主要な Java 要素は、IGXSServiceComponent と呼ばれるインタフェースで、**com.ssw.b2b.xs package** にあります。このインタフェースは、さまざまなパラメータを受け渡す 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指定します。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 GXSEException
{
    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ために前に説明した手順をカプセル化し、簡単に使用できる **JavaBean** が提供されています。**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ーメソッド。
- ◆ 入力ドキュメントに対するセッターメソッド。
- ◆ 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ーメソッド。

**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 GXSEException
{
    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

WebLogic 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、ホスト名およびクライアントがアクセスするネームサーバのポートが指定されます。プロパティの値は、`t3://<hostname>:<port>` の形式となります。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** であり、登録されたポートが **900** の場合、プロパティの値は `t3://stratus:900` となります。

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が指定されます。EJB では、プロパティの値は、`weblogic.jndi.WLInitialContextFactory` です。

### 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()` 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 つ目の実行メソッドは、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.WebLogic** パッケージに **GXSWLEJBAccessor** 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、WebLogic 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あります。

次の例では、EJB の JNDI 名は、**com/acme/inventory/ProductInquiry** で、WebLogic ネームサーバホスト名は、**main.server** で、ポートは、デフォルトのインストール 900 にあります。

```
import com.sssw.b2b.xs.ejb;
import com.sssw.b2b.xs.WebLogic.GXSWLEJBAccessor;
import java.rmi.RemoteException;

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



# 6

## トランザクションの管理

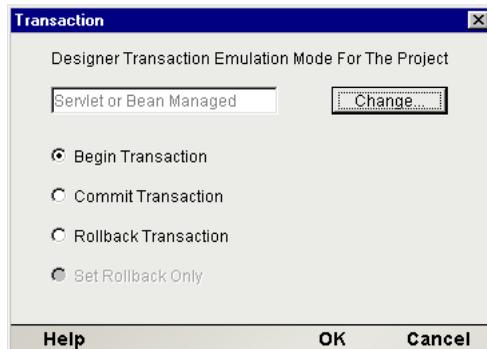
トランザクションを実行する Composer アプリケーションには、特別な計画と配備が必要です。この章では、トランザクション管理に関連する問題について説明します。

### exteNd でのトランザクション制御

exteNd Composer では、「Transaction アクション」により、次のような定義された Java Transaction API (JTA) の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。

- ◆ [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。



# WebLogic 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. 接続について [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、[65 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。アプリケーションで、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 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。

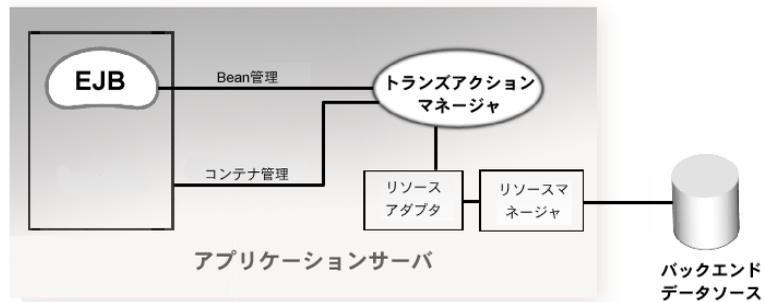
「リソースマネージャ」は、データベースまたはメッセージキューなど、バックエンドシステムのインタフェースです。

「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がありますが、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 に対して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 [Transaction 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 Composer Enterprise 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。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。

## WebLogic での JTS 対応データソースのセットアップ

処理されるコンポーネントを使用する前に、WebLogic で JTS (Java Transaction Service) 対応データソースをセットアップする手順には注意が必要です。次の手順は WebLogic 5.x に対応しています。

### JTS データソースのセットアップ手順

まず、weblogic.properties ファイルで、データソース (DataSource) を定義します。

接続プール「myConnectionPool」の JTS 対応データソース「myJtsDataSource」を定義するには、次の行を weblogic.properties ファイルに追加します。

```
weblogic.jdbc.TXDataSource.myJtsDataSource=myConnectionPool
```

接続プール「myConnectionPool」の JTS 対応でないデータソース「myNonJtsDataSource」を定義するには、次の行を weblogic.properties ファイルに追加します。

```
weblogic.jdbc.DataSource.myNonJtsDataSource=myConnectionPool
```

1 つの接続プールを使用する複数のデータソース定義できます。そうすると、同じデータベースを共有する JTS 対応および JTS 非対応両方のデータソースオブジェクトを定義できます。

次の行を weblogic.properties ファイルに追加することにより接続プールを定義します。

```
weblogic.jdbc.connectionPool.myConnectionPool=\
  URI=jdbc20:weblogic:oracle,\
  driver=weblogic.jdbc20.oci.Driver,\
  loginDelaySecs=1,\
  initialCapacity=4,\
  maxCapacity=10,\
  capacityIncrement=2,\
  allowShrinking=true,\
  shrinkPeriodMins=15,\
  refreshTestMinutes=10,\
  testTable=dual,\
  props=user=SCOTT;password=tiger;server=DEMO
```

この例は、WebLogicjDriver for Oracle を使用する Oracle データベースに対する接続プールを基にしています。斜体で表示されている URI およびクラス名を、DBMS への接続に使用されている JDBC ドライバのものに置き換えてください。

## 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/ejb>

# A

## Deployment Framework API マニュアル

Novell exteNd 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 install directory}\Docslapi`

exteNd の最新バージョンのフレームワークのクラスおよびメソッドの詳細については、JavaDoc 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。実装されるインタフェースが必要とするすべてのメソッドを提供しますが、連続不可能パラメータを持つ実行メソッド (文字列を除くすべて) には、このクラスは入力パラメータを文字列に変換してから EJB に対して実行します。このクラスをインスタンス化できるのは、GXSServiceFactory ヘルパクラスを使用した場合だけです。

#### **com.sssw.b2b.xs.service.conversion.IGXSIInputConversion**

これは、HttpServletRequest から文字列形式の XML ドキュメントに変換するために必要なメソッドを定義するインタフェースです。実装クラスで必要となるメソッドは 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。定義すると、変換クラスで、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 (つまり、URI パラメータまたは送信されるフォームフィールドのいずれかとして指定されるパラメータ) 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したルート名が使用されます。

## **com.sssw.b2b.xs.service.conversion.GXSInputFromHttpContent**

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が予想されます) のコンテンツを取得します。SOAP サーバおよび `exteNd` アクションである 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、マルチパート / 要求の HTML フォームタイプを使用して XML ファイルを受け取ります。特定のファイルのパラメータを参照し、XML 入力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、その 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.GXSServiceRunnerEx**

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ロジックを提供したり、入力 XML ドキュメントを提供したり、出力 XML ドキュメントを返したりする際に使用されます。

## **com.sssw.b2b.xs.service.conversion.GXSConversionException**

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

## **インタフェース**

### **com.sssw.b2b.xs.IGXSServiceRunner**

このインタフェースには、重要なパブリック変数が多数含まれます。

```
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、サーブレット要求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 URI 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。

## ファクトリクラス

### 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、これが **Composer** サービスコンポーネントの通常のクラス表記か、または **EJB** 表記であるか判断するために変更されます。デフォルトは通常のクラス表記です。**IGXSServiceComponent** のタイプ固有バージョンは、インスタンス化され、呼び出し側に返されます。

```
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記を作成します。システムプロパティ `SERVICE_NAME` を使用して、`IGXSServiceComponent` が表す 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前提としています。

```
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記を作成します。このメソッドでは、JNDI コンテキストルックアップが正しく機能するように環境が設定されていることが前提となっています。つまり、このメソッドをサーバ環境外から呼び出す場合そ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` を受け取るので、`IGXSServiceRunner` で `getServiceProperty()` というメソッドを呼び出すことで、JNDI 名を参照します。

## XSL 機能

XSL 処理を円滑化するために、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。これらのメソッドでは、次の機能を実行できます。

- ◆ XSL プロセッサインタフェースで、(XML ドキュメントおよび XSL スタイルシートの両方ではなく)XML ドキュメントのみを受け入れ、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、ターゲットドキュメントをメモリ内 DOM に変換する。
- ◆ Composer サービスコンポーネントで、XSL 処理命令の結果として作成される XML または HTML を出力する。

## クラス

### `com.sssw.b2b.xs.GXSServiceFactory`

XML ドキュメントを参照する指定の 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は、次のとおりです。

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

## com.sssw.b2b.xs.IGXSSSLProcessor

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の XSL 処理命令を使用して出力ドキュメントに変換するメソッドが含まれます。メソッドシグニチャは、次のとおりです。

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

## 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、デフォルトの実装は NULL を返します。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 ドキュメントにある必要があります。

```
public boolean doTransformOutputToHTML()
```

このメソッドは、初期化パラメータ *transform\_into\_html* を参照し、関連す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、変換が発生すると、このメソッドはオーバーロード状態になります。

```
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により変換が発生すると、このメソッドはオーバーロード状態になります。



# B

## 配備オブジェクトの内容

### 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 つの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。

**{parent directory of project}\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。

# 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えます（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しません。

## 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

## A

[Allow SQL Transactions] 62, 66  
API マニュアル 69

## C

CLASSPATH 23, 50, 55  
COBOL/CICS Procedure Division 14  
Connect  
    接続プール 24  
CONVERTER\_CLASS\_NAME 54

## D

Deployment Manager 27, 28  
DTD/スキーマファイル 24

## E

EJB  
    EJB ホームインタフェースを取得するファク  
        トリ 59  
    EJB ホームおよびリモートインタフェースの  
        取得 58  
    アプリケーション 62  
    コンテナ 62  
    コンテナ管理トランザクションの管轄 64  
    コンテナ管理トランザクションのタイプ 64  
    トランザクションマネージャ 63  
    配備 62  
    配備の考慮事項 65  
    リソースマネージャ 63  
EJB (Enterprise Java Bean) 36  
[EJB Service] 44  
[EJB Service Triggers] パネル 65  
EJB トリガパネル 44  
    [EJB Service] 44  
    [URL Path] 45  
    サブレットタイプ 44  
EJB 配備 62  
EJB ベースのサービストリガ 36

    カスタムの作成 55  
EJB ベースのサービストリガパネル  
    サービス 43  
EJB ホーム 58  
exteNd Deployment Wizard 38

## G

GXSEJBServiceRunner 56  
GXSServiceComponentBean 57  
GXSServiceFactory 56  
GXSServiceRunner 55  
GXSWLDeployHandler 50

## H

HTML\_INDICATOR 55

## I

IGXSEJBServiceHome 58  
IGXSServiceComponent 56  
IGXSServiceRunner 54  
INITIAL\_CONTEXT\_FACTORY 58

## J

Java SDK 36  
Java Transactions API 61, 62  
Java クラス 25  
Java の XML 対応 14  
JDBC 14  
JDBC インタフェースを使用したデータベースの  
    XML 対応 14  
JDBC 接続プール 62  
JDBC トランザクション制御 65  
    ユーザトランザクションの許可 65  
JMS サービスタイプ 12  
JNDI 23  
JNDI\_NAME 54  
JSP 51

## O

OUTPUT\_XSL 55

## P

PARAM\_NAME 54  
processRequest() 54  
PROJECT.xml 48, 75  
PROVIDER\_PARAM 54

## R

ROOT\_NAME 54

## S

SERVICE\_NAME 54  
SERVICE\_TYPE 54  
[Service]、サブプレットベースのサービストリガ  
パネル 41  
SOAP 15  
SOAP サービストリガ 37  
SQL、トランザクション制御 65

## T

Transaction アクション 61

## U

UID 36  
[URL Path] 42

## W

weblogicaux.jar 50  
weblogic.class.path 50  
weblogic.system.home 50  
Web サービス 12

## X

X/Open XA プロトコル 63  
xc\_deployment\_info.xml 16, 75  
xconfig.xml 23  
xcs-all.jar 23, 33  
xDeploy.xdp 28  
XML メタデータ、定義 78  
XML リソースの公開 24  
XSL 55  
XSL スタイルシート 24

## あ

アプリケーションサーバ  
トランザクション配備の考慮事項 62  
アプリケーションベースのサービストリガ 37

## い

異種ドキュメントマップ 13  
一般情報パネル 39, 40  
配備サーバタイプ 39  
配備ステージングディレクトリ 39  
プロジェクト JAR ファイル名 40

## う

運用ランタイム環境 16

## か

開発  
設定 58  
開発およびランタイム環境の設定 58  
カスタムサブプレットベースのサービストリガの  
作成 55

## こ

コンテナ管理トランザクションの管轄 64  
コンテナ管理トランザクションのタイプ 64, 65  
コンパイルインタフェース 54  
コンポーネント、概要 12

## さ

- サーバ
  - 概要 15
  - 仕様 15
- サービス
  - インスタンス化 21
  - 概要 12
- サービス EJB、アプリケーションでの使用 58
- サービストリガ
  - EJB ベース 36
  - EJB ベースパネル 43
  - exteNd サービストリガ階層の拡張 52
  - アプリケーションベース 37
  - 概要 35
  - サービストリガ拡張子の作成方法 53
  - サブレットベース 35
  - サブレットベースの作成 53
  - サブレットベースパネル 40
  - 新規作成 56
  - 定義 79
- サービストリガ拡張子 53
- サービス、EJB ベース 43
- サブレットタイプ 41, 44
- サブレット配備 62
- サブレットベースのサービストリガ 35, 53
  - string processRequest(HttpServletRequest) 55
  - カスタムの作成 55
- サブレットベースのサービストリガパネル 41
  - [URL Path] 42

## し

- 自動インストールパネル 49
- 手動配備 50

## す

- スクリーンスクレーピング 14
- ステージングディレクトリ 16, 39

## せ

- 接続プール 23, 62
  - Connect 24

## た

- 端末データ 14

## と

- 統合アプリケーションの配備 23
- 統合制御処理 13
- トランザクション
  - コンテナ管理 65
  - 属性、設定 44
- トランザクション管理
  - サブレット配備の考慮事項 62
  - トランザクション配備の考慮事項 62
- トランザクションの伝達 64
- トランザクションマネージャ 61

## は

- 配備
  - EJB 62
    - exteNd ウィザードの使用 38
    - オブジェクトのインストール 28
    - オブジェクトの作成 28
    - オプション 16
    - 計画 21
    - サブレットの考慮事項 62
    - 手動 50
    - 統合アプリケーション 23
  - 配備ウィザード
    - [EJB Service Triggers] パネル 43
    - EJB トリガパネル 44
    - 一般情報パネル 39
    - サブレットベースのサービストリガパネル 40
    - 自動インストールパネル 49
    - 使用 38
  - 配備オブジェクト
    - インストール 28
    - 作成 28
    - 内容 75
    - プロジェクト JAR 75
  - 配備オブジェクトのインストール 28
  - 配備オブジェクトの作成 28
  - 配備コンテキスト 75
  - 配備サーバタイプ 39
  - 配備ステージングディレクトリ 39
  - 配備フレームワーク

使用 51  
配備フレームワーク API マニュアル 69  
配備プロセス 27

## ふ

プロジェクト JAR 40, 75  
プロジェクト JAR の配備コンテキスト 40  
プロジェクト JAR ファイル名 40  
プロジェクト変数 48  
プロジェクト、配備 27

## ほ

ホームインタフェース 58  
ホストアプリケーションの XML 対応  
    端末データインタフェースの使用 13  
    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使用 14  
ポストされた形式 42

## め

メタデータアーキテクチャ 16  
メッセージキュー 12

## や

役割 24

## ら

ランタイム環境  
    決定 27  
    設定 58  
ランタイム環境の決定 27

## り

リソース、XML の公開 24